۱۳۷۷

پیتر ایبل

برنامه نویسی و زبان اسمبلی کامپیوترهای شخصی

ویرایش چهارم

۱۹۹۸

ترجمهٔ :

مهندس فرزانه کیمیایی

ابیل، پیتر، ۱۹۳۲-
Abel , Peter

برنامه‌نویسی و زبان اسمبلی کامپیوترهای شخصی / پیتر ابیل ؛ ترجمهٔ فرزانهٔ کیمیایی .- مشهد: انتشارات نما؛ دروش، ۱۳۷۸ .
۵۱۹ ص.: جدول، نمودار.
بهاء: /۲۰۰۰۰ ریال
ISBN : 964-6283-17-9
فهرست نویسی براساس اطلاعات فیپا (فهرست نویسی پیش از انتشار)
عنوان اصلی :
IBM PC Assembly Language and Programming (1998)
واژه‌نامه.
۱.اسمبلر (زبان برنامه نویسی کامپیوتر) ۲.کامپیوتر آی. بی. ام ۳۶۰- برنامه نویسی.
۳.کامپیوتر آی. بی. ام ۳۷۰- برنامه نویسی. الف.کیمیایی، فرزانه. مترجم.
ب. عنوان. ج. عنوان.
QA ۷۶/۷۳/ ۵ الف ۰۰۵/۱۳۶ ۵ الف

انتشارات نما - مشهد، تلفن ۸۰۷۹۹۵ انتشارات دُروش

پیتر ابیل
برنامه نویسی و زبان اسمبلی کامپیوترهای شخصی (ویرایش چهارم)
مهندس فرزانه کیمیایی
چاپ اول ۱۳۷۸
تیراژ ۳۰۰۰ نسخه
چاپ و صحافی چاپخانهٔ سعید
طرح جلد گرافیک نیما ۸۰۷۹۹۵
۵۲۰ صفحه وزیری
قیمت ۲۰۰۰ تومان

شابک ۹ - ۱۷ - ۶۲۸۳ - ۹۶۴ ISBN : 964-6283-17-9

فهرست

بخش ث ورودی / خروجی پیشرفته

بخش ج برنامه نویسی پیشرفته

مقدمهٔ مؤلف

قلب کامپیوتر شخصی یک ریز پردازنده است که برای محاسبات، منطق و کنترل با تجهیزات کامپیوتر کار می‌کند. منشاء ریز پردازنده به سال ۱۹۶۰ زمانی که پژوهشگران مدارات مجتمع (IC) را اختراع نمودند، باز می‌گردد. آنها اجزای مختلف الکترونیکی را در یک جزء تنها روی یک تراشه سیلیکان ترکیب کردند. سازندگان ، این تراشه کوچک را در یک دستگاه شبیه به هزار پا قرار دادند و آن را به یک سیستم تابعی مرتبط نمودند. در اوایل سال‌های ۱۹۷۰، اینتل تراشهٔ 8008 را که اولین ریز پردازندهٔ تولید شده بود، معرفی نمود.

مقارن سال ۱۹۷۴، ریز پردازندهٔ 8008 به 8080 با کاربردی همه منظوره توسعه یافت. در سال ۱۹۷۸ ، شرکت اینتل ریز پردازندهٔ 8086 را به عنوان محصول سوم، تولید نمود که از برخی جهات با 8080 سازگاری داشت، ضمن اینکه طراحی آن به طور قابل توجهی بهبود یافت. شرکت اینتل یک نوع از 8086 با تدارک یک طرح ساده‌تر و سازگاری با ورودی - خروجی موجود، پردازندهٔ 8088 را توسعه داد که IBM آن را در سال ۱۹۸۱ برای کامپیوتر شخصی‌اش در نظر گرفت.

یک نوع بهبود یافتهٔ 8088 ، ریز پردازندهٔ 80188 است و انواع بهبود یافتهٔ 8086، ریز پردازنده‌های 80186، 80286، 80386، 80486، پنتیوم (یا 586)، پنتیوم پرو (یا 6X86) می‌باشند که هر یک عملیات اضافی و قدرت پردازش بیشتری دارند.

هر خانوادهٔ پردازشگرها مجموعه دستورات خاص خود را داراهستند که برای هدایت عملیاتی از قبیل پردازش ورودی از صفحه کلید، نمایش داده روی یک صفحه و انجام محاسبات، استفاده می‌شوند. این مجموعه دستورات با عنوان زبان ماشین سیستم شناخته شده و (همانطور که خواهید دید) برای توسعهٔ برنامه‌ها بسیار پیچیده و مبهم هستند. سازندگان نرم‌افزار، زبان اسمبلی را برای خانواده پردازشگرها مهیا نمودند، که دستورات متنوع را در کد سمبولیک قابل فهم‌تری بیان می‌کند.

زبانهای سطح بالا مانند C و BASIC برای حذف تکنیک‌های یک کامپیوتر خاص، طراحی شده‌اند در حالیکه زبان اسمبلی برای خانوادهٔ خاص از پردازشگرها طراحی شده است. استفاده از زبان اسمبلی مزیت‌های زیر را دارد:

- یک برنامهٔ نوشته شده به زبان اسمبلی به میزان قابل توجهی حافظه و زمان اجرای کمتری به نسبت برنامهٔ زبان سطوح بالا نیاز دارد.
- زبان اسمبلی برای یک برنامه نویس توان انجام وظایف تکنیکی سطح بالا و مشکلی، را ایجاد می‌کند که در یک زبان سطح بالا ممکن نخواهد بود.
- دانش زبان اسمبلی اطلاعاتی راجع به ساختمان ماشین فراهم می‌سازد که هیچ زبان سطح بالایی فراهم نمی‌کند.
- اگر چه اکثر متخصصین نرم‌افزار سیستم‌های کاربردی جدید را به زبانهای سطوح بالا می‌نویسند، لیکن یک تجربهٔ

عمومی بازنویسی روتینهای حساس پردازش به زبان اسمبلی است.

- برنامه‌های مقیم در حافظه (که در زمان اجرا دیگر برنامه‌ها در حافظه باقی می‌مانند) و روتینهای **سرویس وقفه** (که ورودی خروجی را دستکاری می‌کنند) اغلب به زبان اسمبلی کدنویسی می‌شود. موارد زیر برای **یادگیری زبان** اسمبلی PC ضروری است:
- دستیابی به کامپیوتر شخصی IBM یا هر مدل سازگار با آن.
- یک کپی از سیستم عامل DOS (ترجیحاً آخرین نسخه) و آشنایی با نحوهٔ استفاده از آن.
- کپی از یک برنامه مترجم اسمبلر (ترجیحاً آخرین نسخه). عموماً توسط مایکروسافت، بورلند و سیستم‌های SLR ارائه می‌شود.

موارد زیر برای یادگیری اسمبلی PC ضروری نیست:

- تسلط به زبان برنامه‌نویسی. اگر چه چنین دانش ممکن است به شما در کسب مفاهیم برنامه نویسی کمک کند ولی به هیچ عنوان اساسی نیست.
- دانش قبلی از الکترونیک یا برق. این کتاب اطلاعات ضروری راجع به سازمان کامپیوترهای شخصی لازم برای برنامه‌نویس اسمبلی را ارائه می‌دهد.

سیستم‌های عامل

هدف از یک سیستم عامل عبارت است از (۱) برقراری امکان دستور دادن کاربر به کامپیوتر که چه عملی را انجام دهد (مثل اجرای یک برنامهٔ خاص) و (۲) همچنین تدارک وسایلی برای ذخیره و دستیابی اطلاعات روی دیسک، می‌باشد.

سیستم عامل پایه برای کامپیوترهای شخصی و سازگار با آن MSDOS متعلق به مایکروسافت می‌باشد که با عنوان PC - DOS در کامپیوترهای شخصی IBM شناخته می‌شود. هر نسخهٔ DOS ضمیمه‌های اضافی تدارک دیده است که قابلیت‌های PC را افزایش می‌دهد.

یادگیری پیچیدگیهای زبان اسمبلی در یک سیستم عامل ساده مانند DOS ساده‌تر از تلاش برای استفاده از آن در محیط OS/2 یا Windows است. داخل DOS راحت‌تر می‌توانید تجربه کنید تا اینکه مراحل را در سیستم‌های پیشرفته پیگیری کنید.

محور اصلی کتاب

هدف اصلی این کتاب کمک به خوانندگان برای یادگیری برنامه نویسی اسمبلی است. بدین منظور، ابتدا، جنبه‌های سادهٔ سخت‌افزار، و زبان بررسی شده و سپس دستورات مورد نیاز معرفی می‌شوند، به همین جهت در متن بر وضوح مثالهای برنامه تاکید شده است. بنابراین مثالها از دستورات و روشهایی استفاده می‌کنند که درک آنها ساده‌ترند - گرچه برنامه نویسان خبره اغلب مسائل مشابه را با کدهایی خردمندانه‌تر - اما مبهم - حل می‌کنند.

در اغلب برنامه‌ها دستورات ماکرو حذف شده است (در فصل ۲۲ مورد بحث قرار می‌گیرد)، اگر چه برنامه نویسان خبره بطور وسیع از ماکروها استفاده می‌کنند، اما کاربری آن‌ها در این کتاب مانع از یادگیری اصول زبان خواهد شد. وقتی این اصول را فراگرفتید، می‌توانید با تکنیک‌های هوشمندانهٔ این خبرگان سازگار شوید.

روش مطالعه

این کتاب طوری طراحی شده است که می‌توانید به عنوان یک کتاب خودآموز و همچنین به عنوان مرجع دائمی از آن استفاده کنید. برای ایجاد کارآیی بیشتر از کامپیوتر شخصی و نرم‌افزار، هر فصل را به طور دقیق بررسی کنید و هر

موردی که واضح نیست، مجدداً مطالعه کنید. برنامه‌های مثال را وارد کرده و آنها را به ماجولهای اجرایی تبدیل و اجرا کنید. همچنین تمرینات آخر هر فصل را انجام دهید. ۹ فصل ابتدای کتاب اساس موضوع کتاب و زبان اسمبلی را ارائه می‌دهد. بعد از مطالعهٔ این فصول، می‌توانید فصل‌های ۱۲، ۱۳، ۱۵، ۱۶، ۲۰، ۲۱ یا ۲۲ را پیگیری کنید. فصل‌های ۲۵، ۲۶ و ۲۷ به عنوان مرجع هستند. فصل‌های مرتبط، بشرح زیرند:

- ۹ تا ۱۱ (مربوط به عملیات صفحه نمایش و صفحه کلید)
- ۱۳ و ۱۴ (مربوط به عملیات محاسباتی)
- ۱۶ تا ۱۹ (مربوط به پردازش دیسک)
- ۲۳ و ۲۴ (مربوط به مدیریت حافظه و زیر برنامه‌ها)

وقتی این کتاب را به اتمام برسانید، قادر به اعمال زیر خواهید بود :

- درک سخت‌افزار کامپیوتر شخصی.
- درک کد زبان ماشین و قالب مبنای شانزده.
- درک مراحل اسمبل، پیوند و اجرای برنامه‌ها.
- نوشتن برنامه‌هایی به زبان اسمبلی برای دستکاری صفحه نمایش، عملیات محاسباتی، تبدیل قالب‌های ASCII و دودویی ، انجام جستجوها و مرتب سازی جداول و انجام ورودی خروجی دیسک.
- پیگیری اجرای ماشین به عنوان کمک برای اشکال زدایی برنامه.
- نوشتن دستورات ماکرو برای کدنویسی سریع‌تر.
- پیوند برنامه‌های اسمبلی شده مجزا در یک برنامه اجرایی.

آموختن زبان اسمبلی و به کار اندازی برنامه‌هایتان یک تجربهٔ جالب و مهیج است. در قبال تلاش و زمانی که صرف می‌کنید، مسلماً پاداش‌های با ارزشی دریافت می‌کنید.

نکات ویرایش چهارم

ویرایش چهارم بازتاب تعداد بسیاری از پیشرفت‌ها به نسبت ویرایش‌های قبل است:

- نکات بیشتری از ریزپردازنده‌های 80486 اینتل و بعد از آن .
- مثال‌های برنامه بیشتر و تمرین بیشتر.
- معرفی عملیات وقفه‌های اخیر.
- متضمن مواردی از نسخه‌های اخیر اسمبلرها.
- سازمان دهی قابل توجه و بازبینی توضیحات متن.
- بازنویسی و افزودن پرسش‌های انتهای هر فصل.

خوانندگان ویرایش سوم توجه دارند که محتویات فصل ۲۵ و ۲۶ در این ویرایش ترکیب شده‌اند. همچنین، دستورات پرش شرطی فصل ۲۷ ("مجموعه دستورات PC") در یک جدول سازماندهی و خلاصه شده‌اند.

سپاسگذاری

مؤلف از کمک و همکاری تمام کسانی که در ارائه پیشنهادات و مرور کتاب و تصحیح آخرین ویرایش‌ها کمک و مساعدت نمودند تشکر و قدر دانی می نماید.

مقدمهٔ مترجم

با یک بررسی اجمالی در تاریخچه برنامه‌نویسی کامپیوتر به این نتیجه می‌رسیم که بهینه‌سازی کدها و داده‌ها همواره یکی از مهمترین اهداف برنامه‌نویسان و طراحان نرم‌افزار بوده است. زیرا با کاهش دستورالعمل‌های موجود در یک برنامه، غالباً سرعت آن افزایش یافته و حجم فضایی که روی رسانه‌های ذخیره‌سازی اشغال می‌کند، کاهش می‌یابد.

در این راستا زبان‌های برنامه‌نویسی نقش موثری در میزان بهینگی کد کمپایل شده برنامه‌ها دارند. زبان اسمبلی به خاطر ویژگی‌های منحصر به فردی که در برنامه‌نویسی سطح ماشین دارد نیاز اصلی طراحان نرم‌افزار را در مورد بهینه‌سازی کد و داده‌ها و افزایش بهره‌وری در کاربرد سخت‌افزار، برآورد می‌سازد.

پیتر ایبل از جمله مؤلفانی است که کتب آموزشی وی در زمینه برنامه نویسی سیستم و زبان اسمبلی به عنوان متون پایه در دانشگاه‌های سراسر دنیا (به ویژه آمریکا) مورد استناد و استفاده قرار می‌گیرد. متن حاضر برگردان ویرایش چهارم (و بین‌المللی[1]) کتاب اسمبلی پیتر ایبل می‌باشد. این کتاب نیاز دانشجویان الکترونیک و کامپیوتر را برای واحد دانشگاهی برنامه‌نویسی به زبان اسمبلی برآورده ساخته و مرجع خوبی برای علاقه‌مندان به این زبان می‌باشد. آنچه در این کتاب می‌خوانید عبارتست از :

اصول برنامه‌نویسی به زبان اسمبلی، نوشتن برنامه‌های COM ، تکنیک‌های مدیریت و آدرس‌دهی حافظه، مدیریت رسانه‌های ذخیره‌سازی، طراحی ماکروها، وقفه و دستورات پیش پردازنده و . . .

در پایان از همسرم آقای مهندس سید کمال‌الدین شاهچراغی به خاطر تشویق، همراهی و رهنمودهای ارزنده‌ای که در ویرایش ادبی کتاب حاضر ارائه کردند و همچنین خانم درورودی به خاطر تایپ و حروفچینی این کتاب سپاسگزارم.

کتاب حاضر کمال مطلوب نیست، لذا از پیشنهادات و انتقادات عزیزانی که موشکافانه آن را مطالعه می‌کنند استقبال کرده و پیشاپیش از ایشان تشکر می‌کنم.

فرزانه کیمیایی
پائیز ۱۳۷۷

۱- International Edition

بخش الف - اصول سخت‌افزار و نرم‌افزار کامپیوترهای شخصی

فصل ۱

ویژگیهای اصلی سخت افزار کامپیوترهای شخصی

هدف : توضیح اصول بنیادین سخت‌افزار میکرو کامپیوترها و سازمان برنامه

مقدمه

نوشتن یک برنامه به زبان اسمبلی مستلزم داشتن اطلاع از سخت‌افزار کامپیوتر (معماری) و جزئیات مجموعه دستورات آن، می‌باشد. در این فصل توضیحی راجع به بنیاد سخت‌افزار - بیت‌ها، بایت‌ها، ثباتها، حافظه، پردازشگر و گذرگاه داده‌ها - داده خواهد شد. در فصول بعدی، مجموعه دستورات و نحوه استفاده از آنها کاملاً توضیح داده می‌شود.

عناصر ***سخت‌افزار*** داخلی یک کامپیوتر، پردازشگر، حافظه و ثباتها می‌باشد و عناصر سخت‌افزار خارجی، دستگاههای ورودی / خروجی یک کامپیوتر مانند صفحه کلید، صفحه نمایش و دیسکها است. ***نرم‌افزار*** شامل برنامه‌ها و فایلهای داده ذخیره شده بر روی دیسک است. برای اجرای یک برنامه، سیستم آن را از روی دیسک بر روی حافظه داخلی کپی می‌کند. (حافظه داخلی همان حافظه‌ای است که در مشخصات کامپیوتر عنوان می‌شود مثلاً یک کامپیوتر ۱۶ مگابایت حافظه دارد) پردازشگر دستورات برنامه را اجرا می‌کند و ثباتها محاسبات درخواستی، انتقال داده‌ها و آدرسدهی را انجام می‌دهند.

بیتها و بایتها

اساس ساختمان بلوکهای حافظه کامپیوتر، بیت می‌باشد. فعال یا غیر فعال بودن بیت‌ها را با یک و صفر نشان می‌دهیم. اطلاعات ارائه شده با یک بیت بسیار محدود است ولی قابلیت‌هایی که مجموعه‌ای از بیت‌ها می‌توانند ارائه کنند، شگفت‌انگیز می‌باشد.

بایت‌ها

یک گروه نه تایی از بیت‌ها یک بایت نامیده می‌شود که این کمیت در واقع عنصر اصلی حافظه اصلی و جانبی می‌باشد.[1] هر بایت شامل ۸ بیت برای داده و یک بیت برای توازن می‌باشد.

0	0	0	0	0	0	0	0	1
بیت‌های داده								توازن

۱- منظور از حافظه اصلی، حافظه RAM ثبات‌ها و حافظه نهان (cache) می‌باشد و حافظه جانبی به دیسک سخت، فلاپی و انواع رسانه‌های ذخیره سازی اطلاق می‌شود.

۸ بیت داده بنیاد محاسبات دودویی و بیان کاراکترها مثل حرف A یا علامت (*) را فراهم می‌سازد. با ۸ بیت می‌توان $(۲^۸)$ ۲۵۶ حالت مختلف، از حالت همه بیت‌ها خاموش (0000 0000) تا همه بیت‌ها روشن (1111 1111) ایجاد نمود. برای مثال، نمایش حرف A به شکل 01000001 و نمایش علامت (*) به شکل 00101010 می‌باشد (نیازی به حفظ این کدها نیست). بر طبق قانون توازن، مقدار بیت‌های روشن یک بایت همیشه فرد می‌باشد. از آنجایی که حرف A دو عدد بیت روشن دارد، کامپیوترهایی با توازن فرد بیت توازن این حرف را روشن قرار می‌دهند. (01000001-1) و همینطور برای علامت ستاره که ۳ عدد بیت روشن دارد، پردازشگرهایی با توازن فرد بیت توازن این حرف را خاموش می‌نمایند (00101010-0). وقتی یک دستور، یک بایت اطلاعات را برای ذخیره در حافظه داخلی ارسال می‌کند، پردازشگر تمام بیت‌ها را برای توازن بررسی می‌کند. اگر توازن زوج باشد، سیستم فرض می‌کند یک بیت خراب شده و پیغام خطا را نشان می‌دهد.

چگونه یک کامپیوتر ارزش بیتی 01000001 را که بیانگر حرف A می‌باشد می‌فهمد؟ وقتی حرف A را از صفحه کلید فشار می‌دهید سیستم سیگنالی را از آن کلید دریافت می‌کند که بوسیله آن عدد 01000001 در حافظه ذخیره می‌شود.این بایت را می‌توانید به هرجای حافظه منتقل کرده یا روی صفحه نمایش و چاپگر آن را چاپ کنید. همانطور که برای حرف A نشان داده شده بیت‌های یک بایت از راست به چپ و از 0 تا 7 به صورت زیر شماره‌گذاری می‌شوند :

شماره بیت :	۷	۶	۵	۴	۳	۲	۱	۰
محتویات بیت (A) :	0	1	0	0	0	0	0	1

بایت‌های مرتبط

یک برنامه می‌تواند با یک یا چند بایت وابسته به هم به عنوان یک داده مشخص نظیر زمان یا فاصله، برخورد کند. یک گروه از بایت‌ها که مقدار خاص را تعریف می‌کنند یک ***عنصر داده*** یا ***فیلد*** نامیده می‌شود. کامپیوترهای شخصی چندین نوع داده را پشتیبانی می‌کنند مثلاً :

- ***کلمه.*** داده ۲ بایتی (۱۶ بیتی)
- ***دوکلمه.*** داده ۴ بایتی (۳۲ بیتی)
- ***چهارکلمه.*** داده ۸ بایتی (۶۴ بیتی)
- ***پاراگراف.*** داده ۱۶ بایتی (۱۲۸ بیتی)
- ***کیلوبایت.*** (KB) عدد $۲^{۱۰}$ معادل ۱۰۲۴ برای مقدار ۱۲ کیلو بایت در نظر گرفته می‌شود. بنابراین ۶۴۰K حافظه یعنی ۶۵۵۳۶ = ۶۴۰ × ۱۰۲۴ بایت.
- ***مگابایت.*** (MB) عدد $۲^{۲۰}$ معادل ۵۷۶ و ۱۰۴۸ یا یک مگابایت.

بیت‌ها در یک کلمه از 0 تا 15 از راست به چپ شماره‌گذاری می‌شوند که در زیر این موضوع را برای حرف 'PC' مشاهده می‌کنید. حرف 'P' در سمت چپ و حرف 'C' در سمت راست قرار دارد.

محتویات بیت (PC) :	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	1
شماره بیت :	۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰

هر بایت یک ***آدرس*** منحصر بفرد در حافظه دارد. اولین بایت در آدرس صفر، یعنی کمترین موقعیت حافظه و دومین بایت در آدرس 1 و الی آخر...

اعداد دودویی

از آنجاکه یک کامپیوتر فقط می‌تواند بین بیت 0 و 1 تمایز قائل شود، بنابراین کامپیوتر در سیستم عددی مبنای دو، یا سیستم دودویی، کار می‌کند. در حقیقت یک بیت نام خودش را از "Binary digIT" (رقم دودویی) می‌گیرد.

مجموعه‌ای از بیت‌ها می‌توانند نشان دهنده هر مقدار عددی باشند. ارزش یک عدد دودویی بر اساس موقعیت نسبی هر بیت و وجود بیت‌های یک می‌باشد. عدد هشت بیتی زیر دارای ارزش یک در تمام بیت‌هاست.

مقدار بیت :	۱	۱	۱	۱	۱	۱	۱	۱
ارزش مکانی :	۱	۲	۴	۸	۱۶	۳۲	۶۴	۱۲۸
شماره بیت :	۰	۱	۲	۳	۴	۵	۶	۷

سمت راست‌ترین بیت دارای ارزش ۱ ، رقم بعدی دارای ارزش $۲^۱$ و رقم بعدی دارای ارزش $(۲^۲)۴$ می‌باشد و الی آخر ... مجموع کل بیت‌ها در این حالت عبارت است از: $(۱ - ۲^۸) ۲۵۵ = ۱۲۸ + ۰۰۰ + ۴ + ۲ + ۱$

اما صبر کنید مگر 01000001 نماد حرف A نیست؟ بله هست مجموعه بیت‌های 01000001 هم می‌تواند بیانگر عدد ۶۵ = ۶۴ + ۱ باشد و هم نشان دهندهٔ حرف A . این موضوع را در زیر مشاهده می‌کنید.

مقدار بیت :	۱	۰	۰	۰	۰	۰	۱	۰
ارزش مکانی :	۱	۲	۴	۸	۱۶	۳۲	۶۴	۱۲۸

اگر یک برنامه یک عنصر داده را برای اهداف محاسباتی تعریف کند آنگاه ارزش 01000001 بیانگر یک عدد دودویی معادل عدد دهدهی ۶۵ می‌باشد.

اگر یک برنامه یک عنصر داده را برای اهداف توصیفی نظیر یک عنوان تعریف کند آنگاه ارزش 01000001 بیانگر نماد حرف A یا کاراکتر حروف الفبا محسوب خواهد شد.

وقتی برنامه نویسی را شروع کنید این تمایز واضح‌تر خواهد شد زیرا شما منظور هر عنصر داده را تعریف خواهید کرد. یک عدد دودویی تنها به هشت بیت محدود نمی‌شود. یک پردازشگر که از ۱۶ بیت (یا ۳۲ بیت) استفاده می‌کند با اعداد ۱۶ بیتی یا (۳۲ بیتی) سر و کار دارد. برای ۱۶ بیت، $۲^{۱۶}-۱$ مقدار از ۰ تا ۶۵۵۳۵ و برای ۳۲ بیت، $۲^{۳۲} - ۱$ مقدار از ۰ تا ۴/۲۹۶/۹۶۷/۲۹۵ مهیا است.

محاسبات دودویی

از آنجا که یک میکروکامپیوتر اعمال حسابی را فقط در قالب دودویی انجام می‌دهد برنامه‌نویس اسمبلی باید با قالب دودویی و جمع آن آشنایی پیدا کند.

۱	۱	۰	۰
+۱	+۱	+۱	+۰
+۱	۱۰	۱	۰
۱۱			

به رقم نقلی دو جمع آخر توجه کنید. حال دو مقدار 01000001 و 00101010 را با هم جمع می‌کنیم. آیا دو نماد حرف A و نماد علامت (*) را با هم جمع می‌کنیم؟ نه. این مقادیر اکنون بیانگر مقادیر دهدهی ۶۵و۴۲ می‌باشند.

دودویی	دهدهی
۰۱۰۰۰۰۰۱	۶۵
+۰۰۱۰۱۰۱۰	+۴۲
۰۱۱۰۱۰۱۱	۱۰۷

برای حصول اطمینان از برابری حاصل جمع (01101011) با عدد ۱۰۷ ارزش بیتی، بیت‌های یک را با هم جمع کنید. مثال دیگری مطرح می‌کنیم.

دودویی	دهدهی
۰۰۱۱۱۱۰۰	۶۰
+۰۰۱۱۰۱۰۱	+۵۳
۰۱۱۱۰۰۰۱	۱۱۳

برای اطمینان بررسی کنید که آیا جمع دودویی واقعاً برابر ۱۱۳ است.

اعداد دودویی منفی

اعداد دودویی که آخرین بیت سمت چپ آنها مقدار صفر دارد مثبت در نظر گرفته می‌شوند. یک عدد منفی یک بیت با ارزش ۱ در آخرین بیت سمت چپ خود دارد. اما تبدیل یک عدد به منفی آن به سادگی تبدیل آخرین بیت به مقدار ۱ نیست. به عنوان مثال نمی‌توان با تبدیل (۶۵+) 01000001 به 11000001 آن را منفی کرد. بلکه یک عدد منفی به صورت **متمم** دو نمایش داده می‌شود. قانون منفی کردن یک عدد چنین بیان می‌شود: ***همه مقادیر بیت‌ها معکوس شوند و سپس با یک جمع شوند.***

عدد ۶۵+	۰۱۰۰۰۰۰۱
معکوس بیت‌ها	۱۰۱۱۱۱۱۰
جمع با یک	۱
عدد ۶۵-	۱۰۱۱۱۱۱۱

برای محاسبه قدر مطلق یک عدد دودویی منفی مجدداً باید متمم دو آن را محاسبه کنید.

عدد ۶۵-	۱۰۱۱۱۱۱۱
معکوس بیت‌ها	۰۱۰۰۰۰۰۰
جمع با یک	۱
عدد ۶۵+	۰۱۰۰۰۰۰۱

برای مشخص کردن صحت این عمل باید حاصل جمع ۶۵- و ۶۵+ صفر شود.

۰۱۰۰۰۰۰۱	+۶۵
+۱۰۱۱۱۱۱۱	-۶۵
۰۰۰۰۰۰۰۰ (۱)	۰۰

در مجموع همه بیت‌ها صفر شده‌اند تنها یک بیت نقلی داریم که چون رقم نقلی خروجی محاسبه نمی‌شود حاصل درست می‌باشد.

برای انجام عمل تفریق عدد تفریق شونده را متمم دو می‌کنیم و با عدد اول جمع می‌نماییم. به عنوان مثال برای تفریق ۴۲+ از ۶۵+ متمم دو ۴۲+، (00101010) می‌شود (11010110) و باهم جمع می‌کنیم.

۰۱۰۰۰۰۰۱	۶۵
+۱۱۰۱۰۱۱۰	+(-۴۲)
۰۰۰۱۰۱۱۱ (۱)	۲۳

حاصل ۲۳ است و یک رقم نقلی خروجی نیز دارد. اگر هنوز هم توجیه متمم ۲ واضح نیست به این مثال توجه کنید: چه مقداری باید به مقدار دودویی 00000001 اضافه کنیم تا حاصل جمع صفر شود؟ جواب بر حسب اعداد دهدهی ۱- خواهد بود و به صورت دودویی فقط حاصل جمع 11111111 با 00000001 صفر خواهد شد. بنابراین ۱- و ۱+ را با هم جمع می‌کنیم:

۰۰۰۰۰۰۰۱	۱
+۱۱۱۱۱۱۱۱	+(-۱)
۰۰۰۰۰۰۰۰ (۱)	نتیجه

رقم نقلی 1 نادیده گرفته می‌شود. طبق محاسبه می‌بینید که عدد 11111111 معادل با نتیجه ۱- دهدهی می‌باشد به شکل الگویی زیر که کاهش مقادیر اعداد دودویی را نشان می‌دهد توجه کنید :

در حقیقت در اعداد منفی بیت صفر مشخص کننده ارزش آن است: ارزش مکانی هر بیت صفر را به صورتی که آن بیت 1 بوده تصور کنید و مقادیر را جمع نمایید و به حاصل یک را بیفزائید. اهمیت حساب دودویی و اعداد منفی را در مورد اعمال حسابی در فصل‌های ۱۲ و ۱۳ درمی یابید.

+3	۰۰۰۰۰۰۱۱
+2	۰۰۰۰۰۰۱۰
+1	۰۰۰۰۰۰۰۱
0	۰۰۰۰۰۰۰۰
-1	۱۱۱۱۱۱۱۱
-2	۱۱۱۱۱۱۱۰
-3	۱۱۱۱۱۱۰۱

نمایش مبنای شانزده

اگر چه یک بایت می‌تواند ۲۵۶ ترکیب مختلف را داشته باشد، امکان نمایش یا چاپ همه آنها مانند کاراکترهای ASCII وجود ندارد. (مثل ترکیبات بیتی که برای اعمال خاص مانند Tab، Enter، Formfeed، Escape در نظر گرفته شده‌اند) در نتیجه طراحان کامپیوتر یک روش کوتاه‌تر برای بیان اعداد دودویی و تقسیم هر بایت به دو نیم و بیان هر نیم بایت در نظر گرفته‌اند. فرض کنید می‌خواهید اطلاعات دودویی ۴ بایت مجاور (دو کلمه‌ای) حافظه را ببینید. این چهار بایت هم به شکل دودویی و هم دهدهی نمایش داده شده است:

دودویی	0101	1001	0011	0101	1011	1001	1100	1110
دهدهی	5	9	3	5	11	6	12	14

با توجه به اینکه اعداد ۱۱، ۱۲ و ۱۴ دو رقمی هستند، سیستم عدد نویسی را توسعه داده و فرض می‌کنیم A=۱۰، B=۱۱، C=۱۲، D=۱۳ حال ارقام از ۰ تا F هستند و چون ۱۶ تا هستند، نمایش مبنای ۱۶ شناخته شده‌اند. حال مقادیر چهار بایت فوق را با اعداد مبنای شانزده چنین می‌توان نوشت:

دودویی	0101	1001	0011	0101	1011	1001	1100	1110
مبنای شانزده	5	9	3	5	B	9	C	E

در شکل ۱-۱ اعداد ۰ تا ۱۵ را به همراه معادل‌های دودویی و مبنای شانزده آنها مشاهده می‌کنید.

زبان اسمبلی بطور قابل توجهی از مبنای شانزده استفاده می‌کند. یک برنامه اسمبلی شده، تمام آدرسها کدهای ماشین و محتویات ثباتها را به صورت مبنای شانزده نشان می‌دهد. به منظور اشکال زدایی برنامه از برنامه DEBUG مربوط به سیستم عامل می‌توانید استفاده کنید که تمامی آدرسها و محتویات بایتها را در مبنای شانزده نشان می‌دهد.

جدول صفحهٔ بعد اعداد ۰ تا ۱۵ و مقادیر دودویی و مبنای شانزده آن را نشان می‌دهد.

مبنای شانزده	دهدهی	دودویی
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

شکل ۱-۱ نمایش مبنای دو، ده و شانزده

از محاسبات مبنای شانزده زیاد استفاده خواهید کرد. بخاطر بسپارید که عدد بعد از F در مبنای شانزده 10 شانزدهی می‌باشد که مقدار دهدهی آن ۱۶ می‌باشد. مثالهای زیر محاسباتی ساده در مبنای شانزده می‌باشند.

7	6	F	F	10	38	FF
+3	+7	+1	+F	+30	+18	+1
A	D	10	1E	40	50	100

بخاطر بسپارید 40 مبنای شانزده معادل 64 دهدهی و 100 مبنای شانزده معادل 256 دهدهی و 1000 مبنای شانزده، 4096 مبنای ده می‌باشد. در مثال 50 = 18 + 38 به یاد بسپارید که 8 + 8 مساوی 10 است.

برای تشخیص اعداد مبنای شانزده در انتهای عدد همیشه 'H' قرار داده می‌شود. در زبان اسمبلی عدد Hex همیشه با رقم 0 تا 9 باید آغاز شود مثلاً عدد B8H را به صورت 0B8H نمایش می‌دهیم. برای مقادیر دودویی پس از عدد 'B' می‌گذاریم مثل 01001100B و اعداد دهدهی فقط با خود عدد مشخص خواهد شد. ضمیمه A چگونگی تبدیل عدد مبنای شانزده به دهدهی و بالعکس را شرح می‌دهد.

کد اسکی ASCII

به منظور استاندارد سازی بیان اطلاعات، سازندگان کامپیوتر کد اسکی را پذیرفتند تا انتقال اطلاعات بین دستگاههای کامپیوترهای مختلف ممکن باشد. ۸ بیت کد اسکی که کامپیوترهای شخصی استفاده می‌کنند شامل نمادهایی برای الفبای خارجی نیز می‌باشد. به عنوان مثال می‌بینیم که ترکیب بیت‌های 01000001 (41 hex) بر حرف A دلالت دارد ضمیمه B لیست ۲۵۶ کاراکتر کد اسکی را نشان می‌دهد و در فصل ۸ نشان می‌دهد که چگونه آنها را نمایش می‌دهیم.

پردازشگر

یک عنصر مهم سخت‌افزار کامپیوتری شخصی واحد سیستم است که شامل برد سیستم، منبع تغذیه و Slot جهت بردهای انتخابی است. عناصر اساسی برد سیستم یک پردازشگر اینتل (یا مشابه)، حافظه فقط خواندنی (ROM) و حافظه‌هایی با دسترس تصادفی (RAM) می‌باشد. مغز کامپیوتر شخصی پردازشگری از خانواده 8086 اینتل است که تمام اعمال اجرای دستورات و پردازش داده توسط آن انجام می‌گیرد.

پردازشگر از نظر سرعت، میزان حافظه، ثباتها و گذرگاه داده متفاوتند. گذرگاه داده، داده‌ها را بین پردازشگر و حافظه و دستگاههای خارجی منتقل می‌کند. در حقیقت مدیریت ترافیک داده‌ها بر عهده گذرگاه داده‌ها است.

8086/80186. این دو پردازشگر با 8088/80188 مشابهند ولی گذرگاه داده ۱۶ بیتی دارند و سریعتر عمل می‌کنند. 80186 مانند 8086 است اما تعداد دستورات بیشتری دارد.

80286. این پردازشگر از مدلهای ما قبل خود سریعتر عمل می‌کند و می‌تواند تا ۱۶ میلیون بایت را آدرسدهی کند. این پردازشگر هم درحالت‌حقیقی و هم درحالت محافظت‌شده (چندمنظوره) عمل‌می‌کند (اجرای چندکار دریک لحظه)

80386. این پردازشگر ثباتهای ۳۲ بیتی و گذرگاه داده ۳۲ بیتی دارد و تا ۴ بیلیون بایت حافظه را می‌تواند آدرسدهی کند. هم در حالت حقیقی و هم در حالت محافظت شده عمل می‌کند.

80486. این پردازشگر ثباتهای ۳۲ بیتی و گذرگاه داده ۳۲ بیتی دارد اگر چه بعضی کامپیوترهای مشابه گذرگاه داده ۱۶ بیتی دارند) و برای قابلیت‌های پیشرفته طراحی شده است و در حالت حقیقی و حالت محافظت شده عمل می‌کند.

80586. این پردازشگر ثباتهای ۳۲ بیتی و گذرگاه ۶۴ بیتی دارد و بیشتر از یک دستور در هر سیکل ساعت می‌تواند اجرا نماید. (اینتل نام پنتیوم را به این دلیل انتخاب کرد که اعداد می‌توانند حق کپی داشته باشند) پنتیوم پرو (6X86) این پردازشگر در ظرفیت ثباتها و گذرگاه داده پیشرفت بیشتری داشته است. به عنوان مثال وقتی پردازشگرهای قبلی به یک حافظه نهان متصل می‌شوند مقداری تأخیر ایجاد می‌شد در حالیکه این پردازشگر یک حافظه نهان داخلی با گذرگاه ۶۴ بیت دارد.

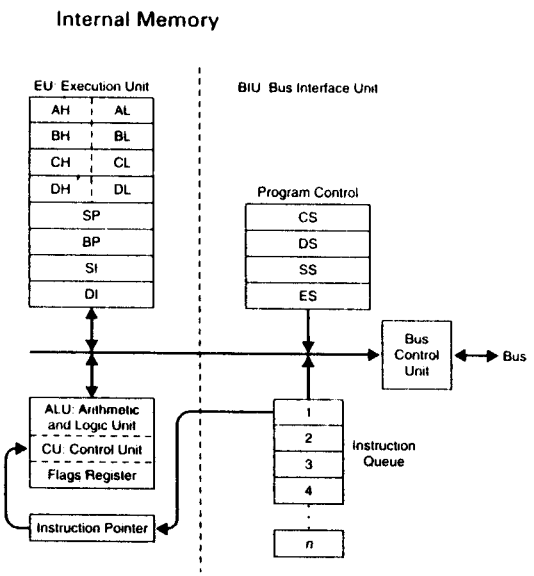
واحد اجرایی[1] و واحد رابط گذرگاه[2]

همانطور که در شکل ۲-۱ مشخص شده است، پردازشگر از نظر منطقی به دو بخش تقسیم شده است. واحد اجرایی و واحد رابط گذرگاه. وظیفه EU اجرای دستورات دریافتی از BIU می‌باشد. واحد اجرایی شامل قسمتهای

1- EU 2- BIU

محاسباتی و واحد منطقی (ALU) و واحد کنترل (CU) و تعدادی ثبات است. این قسمتهای اساسی به منظور اجرای دستورات و عملیات محاسباتی و منطقی فراهم شده‌اند.

مهمترین عمل BIU، مدیریت واحد کنترل گذرگاه و ثباتهای سگمنت و صف دستورالعمل می‌باشد. BIU گذرگاههایی که داده را به EU، حافظه و دستگاههای ورودی - خروجی جانبی ارسال می‌کنند، کنترل می‌کند و ثباتهای سگمنت آدرسهای حافظه را کنترل می‌کنند.



شکل ۲-۱ واحد اجرا و واحد رابط گذرگاه

عمل دیگری از BIU دستیابی به دستورالعمل می‌باشد، زیرا اتمام دستورالعملهای برنامه در حافظه قرار دارد BIU باید دستورالعمل‌ها را از حافظه در یک ***صف دستورالعمل*** قرار دهد که در اندازه، بسته به نوع پردازشگر متفاوتند. این اصول BIU را قادر می‌سازد تا ضمن پیش بینی دستورالعمل‌ها را واکشی نموده بطوریکه همواره صفی از دستوالعمل‌ها برای اجرا آماده باشد.

اگر چه BIU و EU با هم به طور موازی عمل می‌کنند، همواره BIU یک قدم جلوتر از EU است. اگر BIU نیازمند دستیابی به داده‌ها در حافظه با یک دستگاه ورودی، خروجی باشد EU آن را آگاه می‌کند. همچنین EU دستورالعملهای ماشینی را از صف دستورالعمل BIU مطالبه می‌کند.

بالاترین دستورالعمل صف دستورالعمل قابل اجرای جاری می‌باشد. مادامیکه EU در حال اجرای یک دستورالعمل است BIU دستورالعمل بعدی را از حافظه واکشی می‌کند. این واکشی با اجرا در یک زمان اتفاق می‌افتد و سرعت پردازش افزایش می‌یابد.

پردازشگرهای بعد از 80486 خاصیتی دارند به نام ***خط روال یک مرحله‌ای***[1] که به معنای تکمیل اجرای یک دستور قبل از شروع بعدی است.

خط روال شامل روشی است که یک پردازشگر یک دستور را به مراحل ترتیبی با استفاده از ابزارها تقسیم می‌کند پنتیوم خط روال پنج مرحله‌ای دارد و پنتیوم پرو یک ساختار خط روال فوق‌العاده ۱۲ مرحله‌ای دارد.

این اساس آنها را قادر می‌سازد تا دستورات زیادی را به صورت موازی انجام دهند.

یک مشکل در مقابل طراحان هست و آن اینکه پردازشگرها بطور قابل ملاحظه‌ای سریعتر از حافظه‌ها عمل می‌کنند و باید برای تحویل دستورات منتظر بمانند. برای حل این مشکل هر پردازشگر پیشرفته قابلیت بیشتری در ***اجرای دینامیکی*** دارد که شامل سه عنصر اصلی زیاد است :

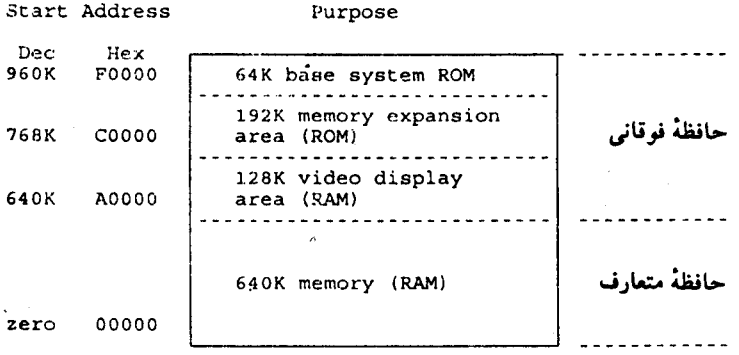
۱) پیش بینی چند شاخه، به واسطه این ویژگی از مجموعه مراحلی که برای پردازش‌های بعدی باید اجرا شوند، تعدادی گزینش می‌شوند.

3- Single - stage pipline

۲) تحلیل جریان داده‌ها، که تحلیل ارتباط بین دستورات را شامل می‌شود.
۳) اجرای نظری، که از حاصل دو عنصر اول برای اجرای نظری دستورات استفاده می‌شود.

حافظه داخلی

دو نوع حافظه در کامپیوترهای شخصی وجود دارد ***حافظه با دسترسی تصادفی*** (RAM) و ***حافظه فقط خواندنی*** (ROM) می‌باشد. بایتهای حافظه به ترتیب از 0 آغاز می‌شود بنابراین هر مکان یک آدرس عددی منحصر بفردی دارد. شکل ۳-۱ یک نقشه حافظه فیزیکی از کامپیوترهای نوع 8086 را نشان می‌دهد در اولین مگابایت حافظه 690K اول RAM پایه است، که برای استفاده فراهم شده است.



شکل ۳-۱ نقشه حافظه اصلی

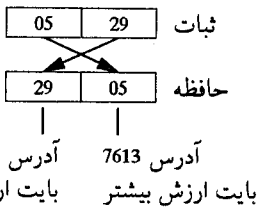
ROM. ROM شامل یک تراشه حافظه است همانطور که از نام کامل آن مشخص می‌شود فقط می‌تواند خوانده شود زیرا دستورات و داده‌ها در داخل این تراشه قابل تفسیر نیستند. ROM پایه سیستم ورودی / خروجی (BIOS) از آدرس 768K آغاز می‌شود و جهت دستکاری ابزارهای ورودی / خروجی مانند کنترل کننده دیسک سخت بکار می‌رود. ROM که از آدرس 960K آغاز می‌شود توابع اصلی کامپیوتر را کنترل می‌کند مانند تست خودش در زمان روشن کردن، الگوی نقطه‌ای برای گرافیک و بارگذار خودکار دیسک، وقتی کامپیوتر را روشن می‌کنید ROM تستهای مختلفی انجام می‌دهد و سیستم خاص خود را از روی دیسک بر روی RAM بارگزاری می‌کند.

RAM. یک برنامه‌نویس اصولاً با RAM سرو کار دارد که بهتر است آن را حافظه نوشتنی خواندنی نامید. RAM جهت صفحه کاریُ برای حافظه موقت و اجرای برنامه در دسترس می‌باشد. از آنجائیکه با خاموش کردن کامپیوتر اطلاعات RAM از بین می‌رود باید از حافظه خارجی جداگانه‌ای جهت نگهداری داده‌ها و برنامه استفاده نمود. وقتی کامپیوتر روشن می‌شود، زیر روال راه‌اندازی ROM قسمتی از سیستم عامل را بر روی RAM بارگذاری می‌کند. آنگاه می‌توانید جهت انجام عملیات مختلف مانند بارگذاری برنامه بر روی RAM از آن استفاده نمائید. برنامه شما در RAM اجرا می‌شود و اصولاً خروجی بر روی صفحه نمایش چاپگر یا دیسک ایجاد می‌کند. وقتی برنامه تمام شد، می‌توانید از سیستم بخواهد تا برنامه دیگری را بر روی RAM بارگذاری نماید، (فعالیتی جهت بازنویسی بر روی برنامه قبلی). توضیح بیشتر اینکه RAM از عبارت عمومی حافظه استفاده می‌کند.

آدرسدهی داده‌ها در حافظه

پردازشگرهای مدلهای مختلف می‌توانند یک یا چند بایت را در هر لحظه از زمان دستیابی نمایند مثلاً عدد ۱۳۱۵ را در نظر بگیرید معادل مبنای شانزده آن 0529H است که به ۲ بایت حافظه یا یک کلمه نیاز دارد. این عدد شامل دو بایت با

ارزش بیشتر 05 و با ارزش کمتر 29 است. سیستم، داده‌ها را در حافظه به ترتیب بایت معکوس ذخیره می‌کند. بایت با ارزش کمتر در آدرس کمتر و بایت با ارزش بیشتر در آدرس بالاتر ذخیره می‌شود. مثلاً پردازشگر 0529H را از ثبات به (یک پردازشگر خاص) آدرسهای حافظه با شماره 7612 و 7613 مثل شکل زیر می‌فرستد:



پردازشگری که داده‌های عددی را در حافظه به ترتیب بایت معکوس ذخیره می‌کند، پردازش اطلاعات را بر همان اساس انجام می‌دهد. وقتی پردازشگر اطلاعات را از حافظه می‌خواند دوباره بایتها را معکوس می‌کند و به نحو صحیح در ثباتها یعنی 0529 ذخیره می‌کند. اگر چه این عملیات بطور خودکار در داخل پردازشگر انجام می‌شود، شما جهت برنامه نویسی و اشکال زدایی برنامه‌های اسمبلی باید از آن آگاهی داشته باشید.

یک برنامه‌نویس اسمبلی باید بین آدرس یک موقعیت حافظه و محتویات آن تفاوت قائل شود. در مثال قبل محتویات آدرس 7612، مقدار 29 و **محتویات** 7613، مقدار 05 است.

آدرسدهی و سگمنت‌ها

یک سگمنت ناحیه‌ای است که از ***مرز یک پاراگراف***، یعنی هر آدرسی از حافظه که بر ۱۶ قابل قسمت باشد شروع می‌شود. اگر چه یک سگمنت می‌تواند از هر جایی از حافظه قرار گیرد و در حالت حقیقی ممکن است تا 64K بایت باشد، ولی فقط نیازمند فضای لازم جهت اجرای برنامه است. تعداد سگمنت‌ها می‌تواند دلخواه باشد، برای آدرسدهی یک سگمنت خاص، کافیست آدرس موجود در ثبات سگمنت مربوطه را تغییر دهیم.

سه سگمنت اصلی عبارتند از: سگمنت ***کد*** - سگمنت ***داده*** و سگمنت ***پشته***.

سگمنت کد

سگمنت کد حاوی دستورالعملهای ماشین جهت اجرا می‌باشد اصولاً اولین دستورالعمل قابل اجرا در شروع این سگمنت قرار دارد و سیستم عامل به این مکان برای اجرای برنامه پیوند می‌خورد. ثبات CS این سگمنت را آدرسدهی می‌کند. اگر ناحیه کد شما بیشتر از 64K باشد. باید در برنامه بیشتر از یک سگمنت کد تعریف نمائید.

سگمنت داده

سگمنت داده‌ها حاوی داده‌های تعریف شده، ثوابت و نواحی کاری مورد نیاز برنامه است. ثبات DS این سگمنت را آدرسدهی می‌کند.

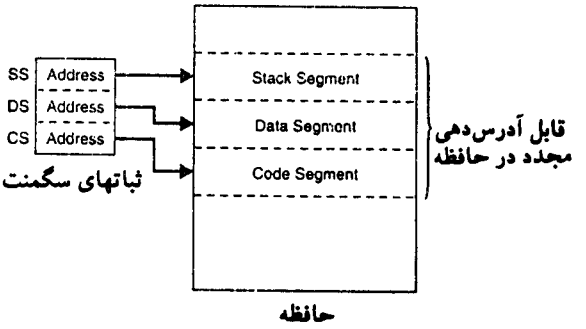
سگمنت پشته

به بیان ساده، پشته حاوی آدرسهای بازگشت، برای برنامه جهت بازگشت به سیستم عامل و برای هر زیر برنامه جهت بازگشت به برنامه اصلی می‌باشد. ثبات این سگمنت SS می‌باشد.

مرز سگمنت‌ها

یک ثبات سگمنت شامل آدرس شروع سگمنت است. شکل ۴-۱ نمای گرافیکی SS، DS، CS و ارتباط بین سگمنت‌های پشته، داده و کد را بیان می‌کند (ثباتها و سگمنت‌ها الزاماً به ترتیبی که نشان داده شده نیستند) یک ثبات سگمنت دیگر ES (سگمنت اضافی است) و در پردازشگرهای 80386 و بالاتر FS، GS با استفاده‌های خاص وجود دارند.

همانطور که قبلاً گفته شد، یک سگمنت ناحیه‌ای است که از مرز پاراگراف آغاز می‌شود و آدرس آن بر عدد ۱۶ قابل تقسیم می‌باشد. ناحیه داده‌ای را در نظر بگیرید که از آدرس 038EH حافظه آغاز می‌شود. چون در همه حالات آخرین عدد سمت راست صفر است طراحان کامپیوتر در نظر گرفتند که نیازی به ذخیره این رقم نیست. بنابراین 038E0H به شکل 038EH در نظر گرفته می‌شود با صفری که در سمت راست آن می‌دانیم وجود دارد و هر زمان لازم باشد به این شکل 038E [0] نشان می‌دهیم.

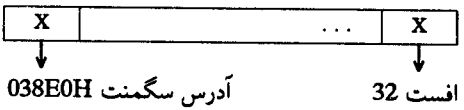


شکل ۴-۱ سگمنت‌ها و ثباتها

افست سگمنت‌ها

داخل یک برنامه تمام موقعیتهای حافظه با یک آدرس شروع سگمنت در ارتباطند. فواصل داخلی هر سگمنت، افست نام دارد. یک افست ۱۶ بیتی می‌تواند از 0000H تا FFFFH مقدار بگیرد (از 0 تا ۶۵۵۳۵) بنابراین اولین بایت در افست 00 در دومین بایت در افست 1 و الی آخر - و نهایتاً در افست ۶۵۵۳۵ قرار دارد. برای مراجعه به هر مکان در سگمنت، پردازشگر ترکیبی از آدرس سگمنت در ثبات سگمنت و مقدار افست را بدست می‌آورد.

در نظر بگیرید که ثبات سگمنت از مکان 038E0H آغاز می‌شود. ثبات DS حاوی آدرس سگمنت داده است یعنی 038E[0] و یک دستور مکانی را ارجاع می‌دهد که افست 0032H در داخل سگمنت است.



بنابراین مکان حقیقی حافظه 03912H که توسط این دستور به آن مراجعه می‌شود.

آدرس سگمنت DS :	038E0H
افست:	+0032H
آدرس حقیقی :	03912H

توجه کنید که برنامه ممکن است شامل چندین سگمنت باشد. که از هر جایی از حافظه آغاز می‌شوند و در اندازه متفاوتند و به هر ترتیبی ممکن است قرار گرفته باشند.

ظرفیت آدرسدهی

پردازشگرهای مختلف اینتل که توسط کامپیوترهای شخصی استفاده می‌شوند ظرفیتهای آدرسدهی متفاوتی فراهم می‌سازند.

آدرسدهی 8086/8088. ثباتهای پردازشگر 8086/8088 ۱۶ بیتی هستند و ۱۶ بیت را در اختیار می‌گذارند. از آنجائیکه آدرس یک سگمنت از مرز پاراگراف آغاز می‌شود (قابل تقسیم بر ۱۶ یا ۱۰ مبنای شانزده) چهار بیت سمت راست که صفر است در نظر گرفته نمی‌شود. حال FFFF [0]H تا 1,048,560 را بایت را آدرسدهی می‌کند. اگر مطمئن نیستید، هر رقم مبنای شانزده F را به 1111 دودویی تبدیل کنید. در نهایت چهار بیت 0 در سمت راست ارزش دودویی

مربوطه قرار دهید و ارزش مکانی همه یک‌ها را جمع کنید.

آدرسدهی 80286. در حالت حقیقی پردازشگر 80286 آدرس دهی مشابه با 8086 دارد. ولی در حالت محافظت شده پردازشگر از ۲۴ بیت برای آدرسدهی استفاده می‌کند و بنابراین [0] FFFF تا ۱۶ میلیون بایت را می‌تواند آدرسدهی کند. ثبات سگمنت به مانند انتخاب کننده آدرس سگمنت ۲۴ بیتی از حافظه عمل می‌کند و این مقدار با آدرس افست ۱۶ بیتی جمع می‌شود.

ثبات سگمنت :	۱۶ بیت [0000]
آدرس سگمنت :	۲۴ بیت

آدرسدهی پنتیوم 80386/80486. در حالت حقیقی، این پردازشگرها مانند 8086 آدرسدهی می‌کنند. در حالت محافظت شده، این پردازشگرها از ۴۸ بیت برای آدرسدهی استفاده می‌کنند که برای آدرسدهی سگمنت‌ها تا ۴ بیلیون بایت امکان دارد. ثبات سگمنت ۱۶ بیتی به آدرس سگمنت ۳۲ بیتی دست می‌یابد و این مقدار با مقداری ۳۲ بیتی آدرس افست جمع می‌شود :

ثبات سگمنت :	۱۶ بیت [0000]
آدرس سگمنت :	۲۴ بیت

ثباتها

ثباتهای پردازشگر جهت کنترل دستورات که اجرا می‌شوند، دستیابی به آدرس حافظه و قابلیت فراهم سازی محاسبات استفاده می‌شوند. برخی ثباتها قابل آدرسدهی هستند مانند CS ، DS ، SS. بیتها در ثبات از راست به چپ شماره‌گذاری می‌شوند و با صفر آغاز می‌شوند مثل:

۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹ ۱۰ ۱۱ ۱۲ ۱۳ ۱۴ ۱۵ ...

ثباتهای سگمنت

هر ثبات سگمنت با طول ۱۶ بیت برای آدرسدهی یک ناحیه از حافظه بکار می‌رود که به عنوان سگمنت جاری شناخته می‌شود. چون یک سگمنت از مرز پاراگراف شروع می‌شود همیشه ۴ بیت صفر در سمت راست آن در نظر گرفته می‌شود.

ثبات CS. ثبات CS دارای آدرس شروع سگمنت کد می‌باشد. این آدرس بعلاوه مقدار افست در اشاره‌گر دستورالعمل (IP) مشخص کننده آدرس دستورالعملی است که جهت اجرا، واکشی می‌شود. دربرنامه‌نویسی عادی مراجعه به CS ضروری نیست.

ثبات DS. ثبات DS دارای آدرس شروع سگمنت داده‌ها می‌باشد. به بیان ساده‌تر این آدرس بعلاوه یک افست در یک دستورالعمل سبب ارجاع به یک مکان مشخص از سگمنت داده‌ها می‌شود.

ثبات SS. با این ثبات می‌توان یک پشته در حافظه مهیا نمود که یک برنامه برای ذخیره موقت آدرس و داده‌ها از آن استفاده می‌کند. سیستم، آدرس شروع پشته را در ثبات SS می‌نویسد. آدرس پشته بعلاوه یک افست در ثبات اشاره‌گر پشته (SP) کلمه جاری در پشته را آدرسدهی می‌کند. در برنامه نویسی عادی رجوع به SS ضروری نیست.

ثبات ES. برخی اعمال رشته‌ای، ثبات سگمنت اضافی را برای دستکاری آدرسدهی حافظه استفاده می‌کنند. در این متن، ثبات ES با ثبات DI مرتبط شده است. اگر ثبات ES مورد نیاز باشد باید توسط برنامه اسمبلی ارزشدهی شود.

ثباتهای FS و GS. ثباتهای سگمنت اضافی دیگری که در 80386 و بالاتر در نظر گرفته شده است.

ثباتهای اشاره‌گر

سه ثبات اشاره‌گر، IP ، SP و BP هستند.

ثبات اشاره‌گر دستورات (IP). ثبات ۱۶ بیتی IP حاوی آدرس افست دستور بعدی جهت اجرا می‌باشد. ثبات IP با

ثبات CS در ارتباطند بطوریکه بر دستور جاری سگمنت کد اجرایی قرار دارند در برنامه‌نویسی معمولی به این ثبات رجوع نمی‌کنید. جهت امتحان برنامه، زمانی که از برنامه، DEBUG استفاده می‌کنید می‌توانید مقدار این ثبات را تغییر بدهید. پردازشگرهای 80386 و بالاتر یک ثبات IP، ۳۲ بیتی به نام EIP دارند.

در مثال بعد، ثبات CS حاوی 39B4[0]H است و IP حاوی 514H می‌باشد. برای یافتن دستور بعدی جهت اجرا، پردازشگر آدرس CS و افست IP را ترکیب می‌کند.

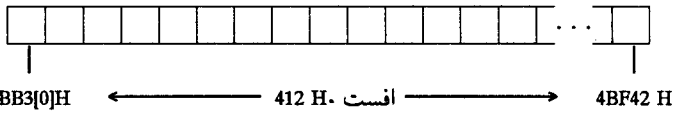
آدرس سگمنت در CS	39B40 H
افست IP	+ 514 H
آدرس دستور بعدی	3A054 H

ثبات پشته SP و اشاره‌گر پایه BP با ثبات SS در ارتباطند و به سیستم اجازه می‌دهند تا داده‌ها را در سگمنت پشته دستیابی کند.

ثبات اشاره‌گر پشته (SP). ثبات ۱۶ بیتی SP یک مقدار افست را فراهم می‌کند که وقتی با ثبات SS مرتبط بشود به کلمه جاری که باید در پشته پردازش شود، پردازشگر 80386 و بالاتر، یک ثبات پشته ۳۲ بیتی به نام ESP دارند که سیستم بطور خودکار این ثبات را دستکاری می‌کند.

در مثال بعد، ثبات SS حاوی آدرس سگمنت 4BB3[0] H و ثبات SP حاوی افست 412H است برای یافتن آدرس کلمه جاری در پشته، پردازشگر آدرس SS و افست IP را ترکیب می‌کند.

آدرس سگمنت در SS	4BB30 H
بعلاوه افست در IP	+ 412 H
آدرس پشته	4BF42 H



ثبات اشاره گر پایه (BP). ثبات اشاره‌گر پایه ارجاع به پارامترها، شامل داده‌ها و آدرسهایی که از یک برنامه بر پشته عبور می‌کنند را ساده می‌نماید. پردازشگر آدرس SS و افست BP را ترکیب می‌کند. پردازشگرها 80386 و بالاتر یک ثبات اشاره‌گر پایه ۳۲ بیتی به نام EBP دارند.

ثباتهای عمومی

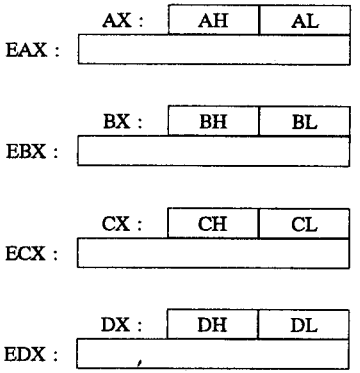
AX ، BX ، CX ، DX ثباتهای عمومی هستند که برای سیستم ، مبنای کاری محسوب می‌شوند. این ثباتها در اینکه می‌توانید آنها را به عنوان یک کلمه یا یک قسمت یک بایتی آدرسدهی کنید منحصر بفردند. بایت سمت چپ قسمت بالا رتبه و بایت سمت راست قسمت پایین رتبه است. برای مثال ثبات AX شامل قسمتهای AH (بالا رتبه) و AL (پایین رتبه) است و هر قسمت با نام آن قابل دسترسی می‌باشد. پردازشگر 80386 و بالاتر همه این ثباتهای عمومی را دارند بعلاوه یک نسخه ۳۲ بیتی توسعه یافته آنها شامل: EAX ، EBX ، ECX ، EDX

دستورات اسمبلی زیر مقدار صفر را در ثباتهای AX ، BH ، ECX به ترتیب قرار می‌دهد :

```
MOV   AX,00
MOV   BH,00
MOV   ECX,00
```

ثبات AX. ثبات AX به انباشتگر اولیه معروف است و برای اعمال ورودی، خروجی ، برخی اعمال رشته‌ای و اعمال حسابی استفاده می‌شود. مثلاً اعمال ضرب ، تقسیم و انتقال ، از AX استفاده می‌کنند. بعضی از دستورالعمل‌ها در صورتی که از ثبات AX استفاده کنند، کد کوچکتری تولید می‌کنند به نسبت آنکه از دیگر ثباتها استفاده نمایند.

ثبات BX. ثبات BX به ثبات پایه معروف است چون تنها ثبات عمومی است که می‌تواند به عنوان یک شاخص برای توسعه دادن آدرسدهی استفاده‌شود. به منظور محاسبات نیز استفاده می‌شود.



ثبات CX. ثبات CX به ثبات شمارش معروف است. ثبات CX برای کنترل کردن تعداد دفعات تکرار یک حلقه مورد نیاز می‌باشد. همچنین حاوی تعداد بیتهایی است که باید به چپ یا راست انتقال داده شود. ثبات CX به منظور محاسبات نیز استفاده می‌شود.

ثبات DX. ثبات DX به ثبات داده‌ها معروف است. برخی اعمال ورودی، خروجی نیازمند استفاده از آن هستند و اعمال ضرب و تقسیم که مستلزم استفاده از مقادیر بزرگ‌اند از جفت ثبات AX,DX استفاده می‌کنند.

هر یک از ثباتهای عمومی می‌توانند برای اعمال جمع و تفریق هشت بیتی یا شانزده بیتی یا ۳۲ بیتی استفاده شوند.

ثباتهای شاخص

هر دو ثبات DI و SI برای آدرسدهی شاخص و استفاده در جمع و تفریق قابل دسترسی هستند.

ثبات SI. این ثبات به‌عنوان شاخص مبدأ شناخته‌می‌شود و برای برخی اعمال رشته‌ای مورد نیاز است. در این متن ثبات SI باثبات DS مرتبط است. پردازشگرهای 80386 و بالاتر یک ثبات توسعه‌یافته ۳۲بیتی به نام ESI را پشتیبانی می‌کنند.

ثبات DI. این ثبات به عنوان شاخص مقصد شناخته می‌شود و برای اعمال رشته‌ای مورد نیاز قرار می‌گیرد. در این مورد ثبات DI با ثبات ES مرتبط می‌باشد. پردازشگرهای 80386 و بالاتر یک ثبات توسعه یافته ۳۲ بیتی به نام EDI را پشتیبانی می‌کنند.

ثبات پرچم

نه بیت از ۱۶ بیت ثبات پرچم در بین همه پردازشگرهای خانواده 8086 مشترکند و بر حالت جاری کامپیوتر و نتیجه اجرای برنامه دلالت دارند. بسیاری از دستورالعمل‌ها، از جمله دستورات مقایسه و محاسبه، وضعیت پرچم را تغییر می‌دهند. اساساً بیت‌های پرچم به صورت زیر هستند:

- OF. (سرریز) بعد از محاسبات مشخص کننده سرریز از بیت مرتبه بالا می‌باشد.
- DF. (جهت) تعیین کننده جهت چپ یا راست برای انتقال یا مقایسه داده‌های رشته‌ای می‌باشند.
- IF. (وقفه) تعیین می‌کند که یک وقفه ناتوان است.
- TF. (تله) به پردازشگر اجازه می‌دهد تا دستورات برنامه به صورت تک مرحله‌ای اجرا شود. برنامه‌های اشکال زدا مانند **DEBUG** بیت T را یک می‌کنند تا هر بار یک دستور اجرا شود و بتوان تأثیر آن بر روی پرچم یا حافظه را امتحان نمود.
- SF. (علامت) حاوی علامت نتیجه عمل حسابی می‌باشد (۰ = مثبت و ۱ = منفی)
- ZF. (صفر) تعیین کننده نتیجه عمل حسابی یا منطقی است (۰ = نتیجه غیر صفر و ۱ = نتیجه صفر)
- AF. (رقم نقلی کمکی) دارای یک رقم نقلی خروجی از بیت سه در داده‌های ۸ بیتی در عمل حسابی می‌باشد.
- PF. (توازن) تعیین کننده توازن فرد یا زوج بایت پایین رتبه داده در عملیات می‌باشد.
- CF (رقم نقلی) حاوی رقم نقلی از بیت بالا رتبه بعد از عمل حسابی است. همچنین حاوی آخرین بیت عمل جابجایی یا چرخش می‌باشد.

پرچمهایی که در ثبات پرچم قرار دارند در موقعیت‌های زیر قرار گرفته‌اند (نیازی به حفظ آنها نیست)

پرچم :						O	D	I	T	S	Z		A		P		C
شماره بیت :	۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰	

پرچم‌هایی که اغلب در برنامه نویسی اسمبلی با اعمال حسابی و مقایسه در ارتباط هستند عبارتند از: OF، SF، ZF، CF پرچم D جهت اعمال رشته‌ای استفاده می‌شود. پردازشگرهای 80286 و بالاتر پرچمهایی جهت اهداف داخلی دارند که در حالت محافظت شده استفاده می‌شوند. پردازشگرهای 80386 و بالاتر یک ثبات پرچم توسعه یافته ۳۲ بیتی دارند. فصل ۸ شامل توضیحات بیشتری راجع به جزئیات ثبات پرچم می‌باشد.

نکات کلیدی

- یک پردازشگر بین صفر (خاموش) و یک (روشن) تفاوت قائل است و محاسبات را فقط بر روی قالب دودویی انجام می‌دهد.
- مقدار یک عدد دودویی با مجموع ارزش بیتهای یک آن محاسبه می‌شود به عنوان مثال مقدار دودویی 1101 برابر است با $۱۳ = ۲^۳ + ۲^۲ + ۲^۱ + ۲^۰$
- یک عدد دودویی منفی با متمم دو، توسط معکوس نمودن بیتهای عدد مثبت و جمع آن با یک بیان می‌شود.
- یک مکان تنها از حافظه، یک بایت است که از ۸ بیت داده و یک بیت توازن تشکیل شده است. ۲ بایت مجاور یک کلمه را تشکیل می‌دهند و ۴ بایت مجاور تشکیل یک دو کلمه‌ای می‌دهند.
- مقدار K معادل $۲^{۱۰}$ یا ۱۰۲۴ بایت است.
- قالب مبنای شانزده یک نشان گذاری مختصر مهم برای نمایش گروهی از چهار بیت است. ارقام ۰ تا ۹ و A تا F نشان دهنده مقادیر دودویی 0000 تا 1111 هستند.
- نمایش داده‌های کاراکتری به صورت قالب کد استاندارد امریکایی برای تبادل اطلاعات (ASCII) می‌باشد.
- قلب کامپیوتر شخصی یک پردازشگر است. پردازشگر، داده‌های عددی را در حافظه به صورت بایت معکوس ذخیره می‌کند.
- دو نوع حافظه داخلی ROM ، RAM می‌باشند.
- یک برنامه زبان اسمبلی شامل یک یا چند سگمنت می‌باشد. یک سگمنت پشته برای دستیابی به آدرسهای بازگشت، سگمنت داده برای تعریف داده‌ها و ناحیه کاری و سگمنت کد برای دستورات اجرایی است. آدرس داخل یک سگمنت با یک افست مرتبط با آدرس شروع سگمنت بیان می‌شود.
- ثباتهای CS ، DS ، SS آدرسهای سگمنت کد، داده و پشته را فراهم می‌سازند.
- ثبات IP شامل آدرس افست دستور بعدی جهت اجرا می‌باشد.
- ثباتهای اشاره‌گر SP ، BP با ثبات SS در ارتباطند و به سیستم اجازه می‌دهند تا داده‌ها در سگمنت پشته را دستیابی نماید.
- ثباتهای عمومی AX ، BX ، CX ، DX ثباتهای مبنای کاری سیستم می‌باشند. بایت سمت چپ قسمت بالا رتبه و بایت سمت راست قسمت پایین رتبه است. ثبات AX (انباشتگر اولیه) جهت انجام محاسبات ورودی / خروجی استفاده می‌شود. BX (ثبات پایه) برای شاخص دهی جهت توسعه آدرس استفاده می‌شود. ثبات CX با عنوان ثبات اشاره‌گر و DX به عنوان ثبات داده شناخته می‌شود.
- ثباتهای شاخص SI ، DI جهت توسعه آدرسدهی و جمع و تفریق بکار می‌رود. این ثباتها برای اعمال رشته‌ای مورد نیاز است.
- ثبات پرچم بر وضعیت جاری پردازشگر و نتیجه برای دستورات دلالت دارد.

پرسش‌ها

۱-۱. (الف) بلوک ساختار مبنا در حافظه کامپیوتر چیست؟ (ب) دو شرط مربوط به آن چیست ؟

۱-۲. (الف) مجموعه‌ای از نه عنصر پرسش ۱-۱ چه نامیده می‌شود؟ (ب) به چه منظور ۸ تا از این عنصر استفاده می‌شود و منظور از نهمین عنصر چیست ؟

۱-۳. طول عناصر داده زیر چیست؟ (الف) کلمه، (ب) دو کلمه، (ج) پاراگراف، (د) کیلو بایت

۱-۴. اعداد دهدهی زیر را به قالب دودویی آن تبدیل نمایید. (الف) ۵، (ب) ۱۳، (ج) ۲۳، (د) ۲۹، (ه) ۳۱

۱-۵. اعداد دودویی زیر را با هم جمع نمایید.

الف)	ب)	ج)	د)
00011101	00101110	00011111	01010101
00000101	00111001	00000001	00111111

۱-۶. مکمل ۲ اعداد زیر را تعیین کنید (الف) 00100110، (ب) 0011101، (ج) 01111000، (د) 00000000

۱-۷. اعداد منفی دودویی زیر را به مقدار مثبت آن تبدیل نمایید. (الف) 11000100، (ب) 10111011 (ج)11111100، (د) 11111111.

۱-۸. مبنای شانزده مقادیر زیر را بیابید: (الف) حرف W (ASCII)، (ب) عدد ۹ (ASCII)، (ج) عدد دودویی 01010111، (د) عدد دودویی 01101111.

۱-۹. اعداد مبنای شانزده زیر را با هم جمع نمائید.

الف)	ب)	ج)	د)	ه)
29AS	62FC	7889	DBCD	FCBB
+0033	+0009	+0777	+35B5	0BAF

۱-۱۰. مبنای شانزده اعداد دهدهی زیر را تعیین نمائید. در ضمیمه A روش تبدیل توضیح داده شده است. برای تست صحت عمل، عدد مبنای شانزده را به دودویی تبدیل کنید و بیتهای یک را با هم جمع کنید.

۱-۱۱. ترکیبهای بیتی کاراکترهای ASCII زیر را تعیین کنید از ضمیمه B جهت راهنمایی استفاده کنید:
(الف) Q ، (ب) q ، (ج) ؟ ، (د) 8 ، (ه)$ ، (و) ویرگول.

۱-۱۲. کار اصلی پردازشگر چیست ؟

۱-۱۳. دو نوع حافظه در کامپیوترهای شخصی کدامند و منظور اصلی از آنها چیست ؟

۱-۱۴. نشان دهید مقادیر زیر چگونه توسط کامپیوتر ذخیره می‌شوند. (الف) ۱۲۳۴ مبنای شانزده (ب) 01C3B5 مبنای شانزده.

۱-۱۵. هر یک از عبارتهای زیر را توضیح دهید: (الف) سگمنت ، (ب) افست ، (ج) مرز آدرسها

۱-۱۶. (الف) سه نوع سگمنت چیست؟ (ب) حداکثر اندازه سگمنت چیست؟ (ج) مرز آدرس شروع سگمنت چیست ؟

۱-۱۷. منظور از هر یک از ثباتهای سگمنت چیست ؟

۱-۱۸. توضیح دهید که کدام ثبات برای منظورهای زیر استفاده می‌شود: (الف) آدرسدهی ثبات‌ها، (ب) افست دستوری که اجرا می‌شود، (ج) جمع و تفریق، (د) ضرب و تقسیم، (ه) شمارشگر حلقه‌ها، (و) دلالت بر حاصل صفر.

۱-۱۹. ثبات EDX و اندازه مکان DH ، DL ، DX در داخل آن را نشان دهید.

۱-۲۰. دستورات اسمبلی برای انتقال مقدار 36 به هر یک از ثباتهای زیر را بنویسید؟
(الف) BX، (ب) BH، (ج) BL، (د) EBX.

۱-۲۱. دستورات اسمبلی برای جمع مقدار 36 با هر یک از ثباتهای زیر را بنویسید؟
(الف) DX، (ب) DH، (ج) DL، (د) EDX .

فصل ۲

موارد ضروری برای استفاده از نرم‌افزار

هدف: توضیح محیط نرم‌افزارهای عمومی

مقدمه

در این فصل، راجع به محیط نرم افزارهای کامپیوترهای شخصی : عملیات سیستم عامل و اجزاء اصلی آن توضیح می‌دهیم. فرایند راه اندازی (چگونگی بارگذاری سیستم وقتی کامپیوتر روشن می‌شود) را بررسی کرده و نحوه بارگذاری یک برنامه جهت اجراء، نحوه استفاده از پشته و نحوه آدرس دهی داده در سگمنت داده توسط دستورالعمل از سگمنت کد را تشریح خواهیم کرد.

این فصل توضیحات پایه راجع به سخت‌افزار و نرم‌افزار کامپیوترهای شخصی را کامل می‌کند و شما را برای مطالعه فصل ۳ قادر می‌سازد، که آنگاه شما می‌توانید برنامه‌های ساده‌ای در حافظه وارد کنید و آنها را مرحله به مرحله اجرا نمائید.

ویژگیهای سیستم عامل

سیستم عامل عموماً دسترسی مستقل از ابزار را به منابع کامپیوتر برای برخی از ابزارها مانند صفحه کلید، صفحه نمایش و دیسک درایوها فراهم می‌سازد. منظور از مستقل از ابزار این است که نیازی نیست تا هر ابزار بطور مجزا آدرسدهی شود زیرا سیستم عملیات ورودی / خروجی را بدون در نظر گرفتن تقاضای برنامه‌های در حال اجرا انجام می‌دهد. توابع DOS که در این کتاب در مورد آن بحث می‌کنیم بشرح زیرند :

- مدیریت فایل. DOS فایلها و شاخه‌ها را بر روی دیسکهای سیستم نگهداری می‌کند. برنامه‌ها ایجاد می‌شوند و فایلها بروز رسانی می‌شوند، اما سیستم جهت مدیریت موقعیت آنها بر روی دیسک جوابگو است.
- ورودی / خروجی. برنامه‌ها در خواست ورود داده از سیستم دارند یا انتقال همین داده به سیستم توسط وقفه‌ها را درخواست می‌کنند و برنامه نویس از کد نویسی در سطح پایین ورودی خروجی آسوده است.
- بارگذاری برنامه. وقتی کاربر یا برنامه‌ای تقاضای اجرای یک برنامه را دارد، بارگذار برنامه مراحلی را طی می‌کند که شامل دستیابی به برنامه بر روی دیسک، قرار دادن آن بر روی حافظه و مقدار دهی اولیه جهت اجرا است.
- مدیریت حافظه. وقتی بارگذار برنامه، یک برنامه را بر روی حافظه جهت اجرا بارگذاری کرد، فضای بزرگ مکفی در حافظه جهت کد برنامه و داده‌های آن اشغال می‌کند. برنامه‌ها می‌توانند داده‌ها را در داخل ناحیه خودشان پردازش نمایند، می‌توانند حافظه اضافی را آزاد کنند یا می‌توانند حافظه اضافی بگیرند.
- دستکاری وقفه‌ها. سیستم به کاربران مجوز نصب برنامه‌های مقیم در حافظه می‌دهد، این برنامه‌ها با الحاق خود به سیستم وقفه قادر به انجام عملیاتی خاص می‌شوند.

سازمان سیستم عامل

سه عنصر اساسی تشکیل دهنده MS DOS عبارتنداز:IO.SYS،MSDOS.SYS،COMMAND.COM

- IO.SYSتوابع مقدار دهی اولیه در زمان راه اندازی را انجام می‌دهد و همچنین شامل توابع مهم ورودی / خروجی و راه انداز ابزارها است که مکمل پشتیبان ورودی خروجی اصلی در حافظه BIOS است. این اجزاء بر روی دیسک در یک فایل سیستمی پنهان به نام IBMBIO.COM ذخیره می‌شود.
- MSDOS.SYS مانند هسته سیستم فعالیت می‌کند و این فعالیت بر مدیریت فایلها، مدیریت حافظه و ورودی / خروجی متمرکز شده است. این اجزاء بر روی دیسک در یک فایل سیستمی پنهان به نام IBMDOS.COM ذخیره می‌شود.

COMMAND.COM یک پردازشگر فرمان یا لایه‌ای است که مانند رابطی بین کاربر و سیستم عامل فعالیت می‌کند این پردازشگر فرمان اعلان کاربر را نمایش می‌دهد، صفحه کلید را چک می‌کند و دستورات کاربر مثل حذف فایل یا بارگذاری برنامه جهت اجرا را پردازش می‌کند.

فرآیند راه اندازی

روشن کردن یک کامپیوتر "راه اندازی سرد" نامیده می‌شود. در این حالت پردازشگر حالت تنظیم دوباره را فعال می‌کند، همه حافظه را پاک می‌کند، یک تست توازن حافظه انجام می‌دهد و ثبات CS آدرس سگمنت FFFF[0]H و به ثبات IP مقدار صفر می‌دهد. اولین دستوری که قابل اجرا است در آدرس جفت ثبات CS:IP یعنی FFFF0H قرار دارد که همان آدرس ورود به BIOS است. روتین BIOS که از موقعیت FFFF0H آغاز می‌شود پورتهای مختلف را برای تشخیص و مقداردهی اولیه ابزارهایی که به کامپیوتر متصل هستند بررسی می‌کند.

دو ناحیه داده‌ای که BIOS برقرار می‌کند عبارتند از :

۱ - یک ***جدول برداری وقفه‌ها***، که از موقعیت 0 در حافظه پائینی آغاز می‌شود و شامل آدرس وقفه‌هایی که رخ می‌دهد می‌باشد.

۲ - یک ***ناحیه داده*** **BIOS** در موقعیت [0]40 ، که با ابزارهای متصل به کامپیوتر سر و کار دارد.

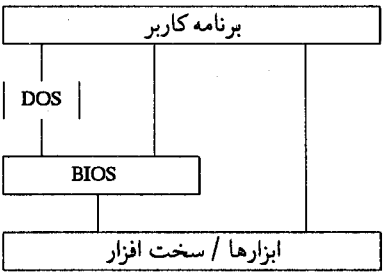
BIOS مشخص می‌کند که آیا دیسک شامل فایلهای سیستم توضیح داده شده می‌باشد و اگر باشد فایلهای بارگذار را از روی دیسک بر می‌دارد. این برنامه فایلهای سیستم IO.SYS ، MSDOS.SYS را از روی دیسک بر روی حافظه بارگذاری می‌کند و کنترل را به نقطه ورودی IO.SYS منتقل می‌نماید. نقطه‌ای که شامل راه‌اندازهای ابزار و کدهای ویژه سخت‌افزاری دیگر می‌باشد. IO.SYS خودش را از حافظه آزاد می‌کند و کنترل را به MSDOS.SYS منتقل می‌نماید. این ماجول داخلی DOS و جدول وقفه‌های DOS را مقدار دهی اولیه می‌نماید. همچنین فایل config.sys را می‌خواند و دستورات آن را اجرا می‌نماید. در انتها MSDOS.SYS کنترل را به COMMAND.COM می‌دهد که فایل AUTOEXEC.BAT را اجرا نماید، اعلان را نمایش دهد و صفحه کلید را جهت ورودی بررسی نماید.

در اینجا، حافظه متعارف تا 640K در شکل ۱-۲ نمایش داده شده است. تحت مدیریت حافظه قسمتی از سیستم در حافظه بالایی مقیم می‌شود.

رابط ورودی / خروجی

BIOS شامل مجموعه‌ای از روتین‌هایی در RAM جهت فراهم ساختن پشتیبانی ابزارها است. BIOS، ابزارهای متصل را تست و مقدار دهی می‌کند و سرویسهایی جهت استفاده در خواندن و نوشتن از ابزارها فراهم می‌سازد. یک وظیفه DOS، رابطه با BIOS هنگام دستیابی به وسایل می‌باشد. وقتی یک برنامه کاربر سرویس ورودی خروجی از

سیستم عامل تقاضا می‌کند، سیستم عامل این تقاضا را به BIOS می‌فرستد که در این حالت ابزار مورد تقاضا دستیابی می‌شود. گاهی، یک برنامه مستقیماً از BIOS تقاضا می‌کند، مخصوصاً سرویس‌های صفحه کلید و صفحه نمایش. در بقیه مواقع اگر چه به ندرت - یک برنامه می‌تواند از طریق DOS و BIOS برای دسترسی به ابزار راه فرعی را طی کند. شکل ۲-۲ این مسیر متغییر را نشان می‌دهد.



شکل ۲-۲ رابط ورودی - خروجی

شکل ۱-۲ جدول حافظه متعارف

بارگذار برنامه سیستم

در نوع برنامه اجرای، EXE ، COM هستند. **یک برنامه** COM شامل یک سگمنت است که هم کد، هم داده و هم پشته در آن قرار گرفته‌اند. یک برنامه COM برای برنامه‌های سودمند کوچک یا برنامه‌های مقیم مفید است. (برنامه مقیم، در حافظه نصب می‌شود و زمانی که برنامه‌های دیگر اجرا می‌شوند نیز در دسترس هستند). **یک برنامه** EXE شامل سگمنت‌های جدا برای کد، داده و پشته است و این روش برای بیشتر برنامه‌های مهم استفاده می‌شود. این کتاب از هر دو نوع برنامه استفاده می‌کند. وقتی شما از سیستم بخواهید که یک برنامه EXE. از دیسک را بر روی حافظه جهت اجرا بارگذاری نماید، بارگذار مراحل زیر را انجام می‌دهد:

۱) دستیابی برنامه EXE. از دیسک.
۲) ساختن ۲۵۶ بایت (100H) پیشوند سگمنت برنامه (PSP) بر روی مرز پاراگراف حافظه داخلی در دسترس.
۳) ذخیره برنامه در حافظه بلافاصله بعد از PSP .
۴) بارگذاری آدرس PSP در ثباتهای ES ، DS .
۵) بارگذاری آدرس سگمنت کد در ثبات CS و تنظیم ثبات IP به آدرس اولین دستور در سگمنت کد. (اغلب صفر).
۶) بارگذاری آدرس پشته در ثبات SS و تنظیم ثبات SP با اندازه پشته.
۷) انتقالی کنترل به برنامه جهت اجرا که با اولین دستور در سگمنت کد آغاز می‌شود.

در روش ذکر شده ، بارگذار برنامه به نحو صحیحی ثباتهای CS:IP ، SS:SP را مقدار دهی می‌کند. اما بخاطر داشته باشید که بارگذار برنامه آدرس PSP را در ثبات DS ، ES ذخیره می‌کند اگر چه برنامه شما به آدرس سگمنت داده در این ثباتها نیاز دارد. در نتیجه برنامه EXE. شما DS را با آدرس سگمنت داده، همانطور که در فصل ۴ خواهید دید، مقدار دهی می‌کند. ما سگمنت‌های پشته و کد و داده را بررسی می‌نماییم.

پشته

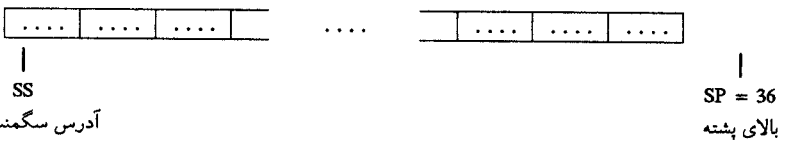
برنامه‌های EXE. ، COM. نیاز به ناحیه‌ای در برنامه به نام **پشته** دارند. منظور از پشته فراهم سازی فضایی برای ذخیره سازی موقتی اقلام داده و آدرس‌ها است.

بارگذار برنامه به طور اتوماتیک پشته برنامه COM. را تعریف می‌کند در حالیکه در برنامه EXE. شما باید صریحاً

پشته را تعریف نمائید. هر عنصر داده در پشته یک کلمه (۲ بایت) است. بارگذار برنامه ثبات SS را مقدار دهی می‌کند که حاوی آدرس شروع پشته است. در ابتدا، ثبات SP شامل اندازه پشته است مقداری که به آخرین بایت انتهای پشته اشاره می‌کند. پشته با بقیه سگمنت‌ها در روش ذخیره سازی داده تفاوت دارد. پشته، داده‌ها را از بالاترین موقعیت سگمنت آغاز می‌کند و داده‌ها را به سمت پایین حافظه ذخیره می‌کند. دستورات PUSH ، POP دو دستوری هستند که محتویات ثبات SP را تغییر می‌دهند و برای ذخیره سازی داده در پشته و بازیابی آن بکار می‌روند. دستور PUSH با کاهش عدد ۲ از SP و ذخیره کردن (یا انجام PUSH) یک مقدار در آنجا انجام می‌شود. POP با بازیابی یک مقدار از پشته و اضافه کردن ۲ به SP و اشاره به موقعیت کلمه ذخیره شده بعدی انجام می‌شود.

مثالهای بعدی PUSH ، POP کردن مقادیر ثباتهای AX ، BX بر روی پشته و سپس به ترتیب POP کردن آنها از روی پشته به ثباتها را نشان می‌دهد، در نظر بگیرید که AX حاوی مقدار مبنای شانزده 026B و BX حاوی مقدار مبنای شانزده 04E3 و SP حاوی مقدار 36 است. (آدرس سگمنت در SS در اینجا در نظر گرفته نشده است).

۱) ابتدا پشته خالی است و مانند شکل زیر به نظر می‌رسد.



۲) PUSH AX : ۲ از SP (34) کم می‌شود و محتویات AX ، 026B در پشته ذخیره می‌شود. به خاطر بسپارید که عملیات ترتیب بایت‌های ذخیره شده را معکوس می‌نماید بنابراین 026B خواهد شد 6B02

۳) PUSH BX : ۲ را از SP کم می‌کند (32) و محتویات BX ، OPE 3 در پشته به شکل 304 E ذخیره می‌شود.

۴) POP BX : بازیابی کلمه‌ای که SP در پشته به آن اشاره می‌کند (E304) به داخل ثبات BX و ۲ را به SP می‌افزاید (34). حال BX شامل 04E3 با بایت صحیح ذخیره شده است. حال پشته چنین خواهد بود.

۵) POP AX : بازیابی کلمه‌ای که SP در پشته به آن اشاره می‌کند (6B02) به داخل ثبات AX و ۲ را به SP می‌افزاید (36). حال AX حاوی 026B با بایت‌های صحیح است. حال پشته چنین خواهد بود.

بخاطر داشته باشید که دستور POP به ترتیب معکوس از دستورات PUSH کد می‌شود. در مثال ثباتهای BX AX، بر روی پشته قرار گرفتند در حالیکه BX و AX از پشته برداشته شدند.

همچنین مقادیری که بر روی پشته قرار گرفتند هنوز در آنجا هستند اگر چه SP دیگر به آنجا اشاره نمی‌کند. شما همیشه

باید از مقداری که توسط برنامه بر روی پشته قرار می‌گیرد و مطابقت آن با مقداری که برداشته می‌شود مطمئن شوید. اگر چه این مطلب ساده به نظر می‌رسد ولی یک خطا می‌تواند موجب قطع برنامه شود. همچنین برای یک برنامه EXE. پشته‌ای که تعریف می‌کنید باید به اندازه کافی برای تمام مقادیری که ممکن است بر روی آن قرار گیرد، بزرگ می‌شود.

دستورات دیگری که مقادیر را بر روی پشته قرار می‌دهند و از روی آن برمی دارند عبارتند از:

- PUSHF ، POPF : ذخیره و بازیابی حالتهای ثبات پرچم.
- PUSHA ، POPA : (مدلهای 80286 و بالاتر): ذخیره و بازیابی کلیه ثباتهای عمومی

آدرس دهی دستورات و داده

یک برنامه نویس زبان اسمبلی یک برنامه را به کدهای سمبولیک می‌نویسد و از اسمبلر برای تبدیل آن به کد ماشین استفاده می‌کند. برای اجرای برنامه، سیستم کد ماشین را به حافظه بارگذاری می‌کند. هر دستور شامل یک عملیات است مانند انتقال، جمع یا بازگشت. بسته به نوع عمل، یک عملیات یک یا چند عملوند دارد که دادهٔ پردازش عملیات را منتقل می‌کنند.

همانطور که گفته شد، برای یک برنامه EXE. ثبات CS آدرس شروع سگمنت کد برنامه را فراهم می‌کند و ثبات DS آدرس شروع سگمنت داده را مشخص می‌کند. سگمنت کد ، شامل دستوراتی است که باید اجرا شود در حالیکه سگمنت داده شامل داده‌هایی است که دستورات به آن مراجعه می‌کنند. ثبات IP بر افست دستور جاری در سگمنت کد دلالت دارد که باید اجرا شود. یک عملوند دستور بر آدرس افست سگمنت داده که به آن مراجعه می‌شود دلالت دارد.

مثالی را در نظر بگیرید که بارگذار برنامه، یک برنامه EXE. را بر روی حافظه بارگذاری می‌کند که از آدرس 05BE0H آغاز می‌شود. بارگذار بر طبق آن ثبات CS را با آدرس قطعه یعنی 05BE[0]H تنظیم می‌کند. برنامه هم اکنون اجرا را آغاز می‌کند و IP شامل تفاوت مکان 0023H است.

CS:IP با هم آدرس دستور بعدی برای اجرا را مشخص می‌کنند به این شکل :

```
آدرس سگمنت CS :  50E0H
افست IP :        +0023H
آدرس دستور :     5C03
```

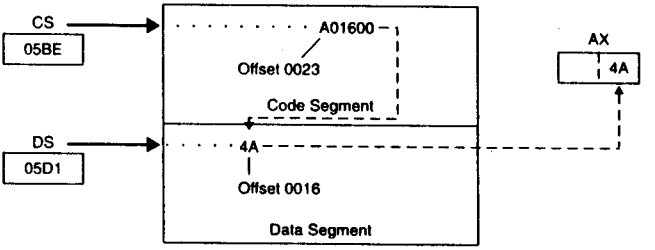
حال دستوری که در آدرس 05C03H آغاز می‌شود یک بایت در حافظه را به ثبات AL کپی می‌کند، بایتی که در افست 0016H از سگمنت کد قرار دارد. در اینجا کد ماشین و کد سمبولیک این دستور ذکر شده :

```
A01600    MOV    AL,[0016]
|
موقعیت 05C03H
```

آدرس حافظه 05C03H شامل بایت اول (A0) دستوری که پردازش آن در دسترس است، می‌باشد بایت دوم و سوم شامل مقدار افست است البته به صورت بایت معکوس (0016 به صورت 1600 ذخیره می‌شود) بارگذار ثبات DS را با آدرس سگمنت 05D1 [0]H مقدار دهی می‌نماید. برای دسترسی به داده، پردازشگر موقعیت آن را با استفاده از آدرس ثبات سگمنت DS بعلاوه افست (0016 H) ذکر شده در عملوند دستور مشخص می‌کند. از آنجائیکه DS حاوی 05D1[0]H است آدرس حقیقی داده ارجاع شده چنین است :

```
آدرس سگمنت DS :  5D10H
افست سگمنت :     0016H
آدرس عنصر داده :  5D26H
```

اگر آدرس 05D26H حاوی 4AH باشد، پردازشگر 4AH از آدرس 05D26H اقتباس می‌کند و در ثبات AL کپی می‌نماید همانطور که در شکل ۳-۲ نشان داده شده است.



شکل ۳-۲ سگمنت‌ها و تفاوت مکان‌ها

زمانی که پردازشگر، هر بایت از دستورالعمل را واکشی می‌کند. ثبات IP را یکی افزایش می‌دهد. از آنجائیکه IP حاوی 23H است و کد ماشین دستورالعمل ، ۳ بایت می‌باشد حال ثبات IP حاوی 26H خواهد بود، که افست دستور بعدی است. پردازشگر هم اکنون آمادهٔ اجرای دستور بعدی است ، که بار دیگر آدرس سگمنت در CS را (05BE0H) را بعلاوه افست IP (0026H) نماید که همان 05CO6H می‌باشد.

یک دستور ممکن است بیشتر از یک بایت را در یک زمان دستیابی کند. برای مثال یک دستور محتویات ثبات AX (0248H) را در دو بایت مجاور در سگمنت داده که با افست 0016H ذخیره می‌کند. کد سمبولیک این دستور چنین است: MOV [0016] ، AX. عملوند [0016] در کروشه (یک عملگر شاخص‌دهی) نشانگر یک موقعیت حافظه است تا از عدد ۱۶ تشخیص داده شود. پردازشگر، دو بایت AX را به ترتیب بایت معکوس بارگذاری می‌کند.

محتویات بایت‌ها : 48 02

افست در سگمنت داده : 0016 0017

دستور دیگر MOV AX ، [0016] است که دو بایت را بعداً در داخل ثبات AX بازیابی می‌کند. این عملیات بایت‌ها را در AX به صورت 0248 معکوس (و تصحیح) می‌کند.

عملوندهای دستورات

ویژگی عملوندهای دستور استفاده از نامهای معمولی، نامهایی که در کروشه هستند و اعداد معمولی است. در مثال زیر، WORDX ، DW را یک کلمه (۲ بایت) تعریف می‌کند :

```
WORDX DW   0          ;  تعریف WORDX به صورت کلمه
      ...
      MOV  CX,WORDX   ;  انتقال محتویات WORDX به CX
      MOV  CX,25      ;  انتقال مقدار ۲۵ به CX
      MOV  DX,CX      ;  انتقال DX به CX
      MOV  CX [DX]    ;  انتقال محتویات موقعیت آدرس‌دهی شده توسط DX
```

اولین MOV داده را بین حافظه و ثبات مبادله می‌کند. دومین MOV داده بلافصل را به یک ثبات منتقل می‌کند. سومین MOV داده را بین ثباتها منتقل می‌کند.

کروشه در چهارمین MOV یک ***عملگر شاخص*** تعریف می‌کند یعنی: استفاده از آدرس افست DX (با آدرس سگمنت در DS ترکیب می‌شود مثل DS : DX) تا در موقعیت کلمه حافظه قرار گیرد و محتویات آن را به CX منتقل نماید. افست این دستور را با سومین MOV مقایسه کنید که محتویات DX را به CX منتقل می‌کند.

نکات کلیدی

- سه جزء اصلی سیستم عامل IO.SYS ، MSDOS.SYS و COMMAND.COM است. روشن کردن کامپیوتر یک "راه‌اندازی سرد" است. پردازشگر یک حال تنظیم دوباره ایجاد می‌کند. همه موقعیتهای حافظه را صفر می‌کند، توازن حافظه را چک می‌کند و ثبات CS و ثبات IP را به نقطه ورود به BIOS در ROM تنظیم می‌نماید.
- COM. ، EXE. دو نوع برنامه هستند.
- وقتی شما از سیستم می‌خواهید تا یک برنامه EXE. را جهت اجرا بارگذاری کند، بارگذار برنامه، ۲۵۶ بایت (100H) PSP در مرز پاراگراف حافظه می‌سازد و برنامه را بلافاصله بعد از PSP ذخیره می‌کند. سپس آدرس PSP را در ثباتهای DS و ES قرار می‌دهد، آدرس سگمنت کد را در CS قرار می‌دهد. IP را با آدرس اولین دستور از سگمنت کد مقدار دهی می‌کند و آدرس پشته را در SS قرار می‌دهد و اندازه پشته را در SP قرار می‌دهد. در انتها بارگذار، کنترل را به برنامه جهت اجرا منتقل می‌کند.
- منظور از پشته فراهم سازی فضای حافظه موقتی جهت آدرسها و داده‌ها است. هر داده در پشته یک کلمه است.
- بارگذاربرنامه، پشته‌برای برنامهٔ COM. تعریف‌می‌کند، درحالیکه برای‌برنامه EXE. شمابایدصریحاًپشته تعریف‌کنید.
- از آنجائیکه پردازشگر با واکشی هر بایت از دستور، ثبات IP را یکی می‌افزاید پس ثبات IP حاوی افست دستور بعدی است.

پرسش‌ها

۲-۱. ۵ عمل اصلی سیستم عامل را معین کنید.

۲-۲. سه جزء اصلی سیستم عامل چیست و منظور از ایجاد هر کدام را توضیح دهید.

۲-۳. مراحل سیستم در هنگام "راه اندازی سرد" چیست.

۲-۴. بارگذار برنامه ناحیه داده را در جلوی ماجول‌های اجرایی در هنگام اجرای ماجول ساخته و ذخیره می‌کند. (الف) نام این ناحیه داده چیست؟ (ب) اندازه آن چقدر است؟

۲-۵. بارگذار برنامه وقتی یک برنامه EXE. را جهت اجرا بارگذاری می‌کند ، عملیات خاص انجام می‌دهد. مقادیری که بارگذار ارزشدهی می‌کند (الف) در ثبات‌های DS ، ES چیست؟ (ب) در ثبات‌های CS ، IP چیست؟ (ج) در ثبات‌های SS ، SP چیست؟

۲-۶. منظور از پشته را توضیح دهید.

۲-۷. توضیح دهید که پشته چگونه تعریف می‌شود برای (الف) یک برنامه COM.، (ب) یک برنامه EXE. (چه کسی، چرا پشته را تعریف می‌کند؟)

۲-۸. (الف) در ابتدا بالای پشته کجاست و چطور آدرسدهی می‌شود؟، (ب) اندازه هر ورودی به پشته چقدر است؟

۲-۹. در طی اجرای یک برنامه، CS حاوی 6C3A[0] و SS حاوی 6C62[0] و IP حاوی 42H و SP حاوی 36H است (مقادیر، همه به شکل معمولی نشان داده شده‌اند نه با ترتیب بایت معکوس) آدرس‌های زیر را محاسبه کنید: (الف) دستوری که می‌خواهد اجرا بشود و (ب) بالای (موقعیت جاری) پشته.

۲-۱۰. در طی اجرای یک برنامه، CS حاوی 74A5[0]، SS حاوی 752B[0]،IP حاوی 54H و SP حاوی 24H است (مقادیر همه به شکل معمولی نشان داده شده‌اند نه با ترتیب بایت معکوس) آدرس‌های زیر را محاسبه کنید. (الف) دستوری که می‌خواهد اجرا بشود و (ب) بالای (موقعیت جاری) پشته.

۲-۱۱. آدرس داده‌های ارجاع شده برای حالتهای زیر را محاسبه کنید. (الف) DS حاوی 4034[0] و یک دستور که داده را از حافظه به AL منتقل می‌کند. چنین است، A02B04 (A0 یعنی انتقال). (ب) DS حاوی 5B4[0] است و یک دستور که داده را از حافظه به AL منتقل می‌کند بدین شکل A03A01.

فصل ۳

اجرای دستورات کامپیوتر

هدف : مقدمه‌ای بر ورود و اجرای برنامه‌ها در حافظه

مقدمه

این فصل از یک برنامه DOS بنام DEBUG برای دیدن حافظه برای ورود برنامه‌ها در حافظه و پیگیری اجرای آنها استفاده‌می‌کند واین متن چگونگی‌ورود برنامه‌ها را بطورمستقیم به‌حافظه درسگمنت کد و فراهم‌سازی‌توضیح هرمرحله اجرایی را مشخص می‌سازد. اگرچه برنامه‌های اشکال‌زدایی پیشرفته‌تری نیزمانند CODEVIEW و TurboDebugger وجود دارند، ما از DEBUG استفاده می‌کنیم زیرا برای استفاده ساده‌تر است و در دسترس همگان می‌باشد.

در تمرین‌های اولیه، شما محتویات نواحی خاص از حافظه را بررسی خواهید کرد. برنامه مثال اول، از تعریف داده‌های بلافصل داخل دستورات برای بارگذاری داده‌ها در ثبات‌ها و انجام محاسبات استفاده می‌کند برنامه مثال دوم از داده‌های تعریف شده مجزا در برنامه‌های اجرایی استفاده می‌کند. پیگیری این دستورات وقتی آنها اجرا می‌شوند اطلاع از عمل پردازشگر و قانون ثباتها فراهم می‌سازد.

شما می‌توانید بدون هیچ دانش قبلی از زبان اسمبلی یا حتی برنامه نویسی شروع کنید. همه آنچه که شما نیاز دارید یک کامپیوتر شخصی سری اینتل است و دیسکی که سیستم عامل DOS را داشته باشد. البته با فرض اینکه با دستورات سیستم و انتخاب دیسک درایو و فایلها آشنا باشید.

استفاده از برنامه DEBUG

سیستم DOS با برنامه‌ای به نام DEBUG همراه است که برای تست و اشکال زدایی برنامه‌های اجرایی بکار می‌رود یک ویژگی DEBUG نمایش همه کد برنامه و داده‌ها به شکل شانزده است و هر داده‌ای هم که شما وارد حافظه می‌کنید باید در مبنای شانزده باشد. DEBUG همچنین از حالت تک مرحله‌ای استفاده می‌کند که به شما امکان اجرای یک برنامه را به شکل یک دستور در هر لحظه می‌دهد و بنابراین می‌توانید تأثیر هر دستور بر حافظه و ثباتها را ببینید.

فرامین DEBUG

مجموعه فرمانهای DEBUG به شما امکان اجرای تعدادی از عملیات کاربردی را می‌دهد. فرمانهایی که در این جا بدان اشاره می‌کنیم بشرح زیرند :

A دستورات سمبولیک اسمبلی درکد ماشین
D نمایش محتویات یک ناحیه از حافظه
E ورود داده به حافظه، با شروع از یک موقعیت خاص
G اجرای برنامه اجرایی در حافظه
N نامگذاری یک برنامه
P اقدام یا اجرای مجموعه‌ای از دستورات مرتبط

Q خروج از مجموعه DEBUG
T پیگیری اجرای یک دستور
W نوشتن یک برنامه بر روی دیسک
R نمایش محتویات یک یا چند ثبات
U تبدیل کد ماشین به کد سمبولیک

ضمیمه E توضیح کاملی از فرامین DEBUG را فراهم می‌سازد.

قانون فرامین DEBUG

در اینجا قوانین اساسی در ارتباط با نحوه استفاده از DEBUG ذکر شده است.

- DEBUG فرقی بین حروف بزرگ و کوچک قائل نیست پس شما هر طور که بخواهید فرامین را وارد نمائید.
- فقط وقتی فاصله بگذارید که لازم است پارامترها در فرامین جدا شوند.
- سگمنت و تفاوت مکان را با یک (:) مشخص کنید، به شکل تفاوت مکان: سگمنت.
- DEBUG فرض می‌کند که همهٔ اعداد در قالب مبنای شانزده هستند.

۳ مثال زیر از فرمان D در DEBUG برای نمایش ناحیه‌ای از حافظه که از تفاوت مکان 200H در سگمنت داده آغاز می‌شود استفاده می‌کند.

D DS:200 (فرمان D با حرف بزرگ است و فاصله بعد از آن هست)
DDS:200 (فرمان D با حرف بزرگ است و فاصله بعد از آن نیست)
dds:200 (فرمان D با حرف کوچک است و فاصله بعد از آن نیست)

نمایش DEBUG

فرمان D محتویات ناحیه داده خواسته شده را بر روی صفحه نمایش می‌دهد. نمایش شامل سه قسمت است.

۱- در سمت چپ آدرس سمت چپ‌ترین بایت نمایش داده شده است. برای مثال. تفاوت مکان: سگمنت.

۲- ناحیه عریض وسط بیان مبنای شانزده ناحیه نمایش داده شده است.

۳- سمت راست بیان ASCII بایتهای نمایش داده شده است که شامل کاراکترهای قابل نمایش است و به شما برای تفسیر ناحیه مبنای شانزده کمک می‌کند.

این عملیات ۸ خط داده را نمایش می‌دهد و هر خط شامل ۱۶ بایت است (۳۲ رقم مبنای شانزده) برای ۱۲۸ بایت که با آدرسی که شما در فرمان D مشخص کردید آغاز می‌شود. شکل دیاگرامی آن چنین است.

```
Address  |<-------- Hexadecimal representation -------->|<--ASCII-->|
xxxx:xx10  xx ..... xx-xx ..... xx x.....x
xxxx:xx20  xx ..... xx-xx ..... xx x.....x
xxxx:xx30  xx ..... xx-xx ..... xx x.....x
...
xxxx:xx80  xx ..... xx-xx ..... xx x.....x
```

آدرس سمت چپ فقط به سمت چپ‌ترین بایت اشاره دارد ، به شکل تفاوت مکان: سگمنت، شما برای تعیین موقعیت هر کدام از بایتهای دیگر باید در طول خط بایتها را بشمارید. ناحیه نمایش مبنای شانزده دو کاراکتر مبنای شانزده را برای هر بایت نمایش می‌دهد که بین بایتها یک فاصله برای خوانایی بیشتر وجود دارد. در ضمن یک خط فاصله بین ۸ بایت دوم و ۸ بایت اول را برای خوانایی بیشتر، مجزا می‌سازد. بنابراین اگر شما می‌خواهید موقعیت بایتی با تفاوت مکان XXBH را ببینید، با XX10H شروع کنید و سه بایت به سمت راست بشمارید.

این کتاب از DEBUG زیاد استفاده می‌کند و جزئیات فرامین آن را هر جا که لازم باشد توضیح می‌دهد.

اجرای DEBUG

برای راه‌اندازی DEBUG، سیستم را بر روی شاخه‌ای از دیسک‌سخت که شامل DUBUG است تنظیم کنید یا در درایو پیش فرض دیسکی که DEBUG دارد قرار دهید. برای شروع برنامه، DEBUG را تایپ کنید و کلید Enter را بزنید. DEBUG از روی دیسک بر روی حافظه بارگذاری می‌شود. وقتی DEBUG اعلان بدهد یک (-) برروی صفحه ظاهرخواهد شد و حالا DEBUG آماده‌پذیرفتن فرامین شماست.حال‌از DEBUG برای‌امتحان حافظه استفاده‌می‌کنیم.

دیدن موقعیت‌های حافظه

در اولین پنج تمرین، شما باید از DEBUG برای دیدن محتویات موقعیت انتخابی حافظه استفاده کنید. تنها فرمانی که این تمرینها با آن کار می‌کند فرمان D است که ۸ خط ۱۶ بایتی را هم به صورت مبنای شانزده و بیان ASCII داده‌ها نمایش می‌دهد.

چک کردن تجهیزات سیستم

ابتدا بیایید آنچه را که BIOS از تجهیزات نصب شده تشخیص می‌دهد ببینید. یک کلمه وضعیت تجهیزات در ناحیه داده BIOS است که اخبار اصلی از ابزارهای نصب شده است. این کلمه در موقعیت 410H-411H قرار دارد با استفاده از DEBUG می‌توان توسط آدرس دو قسمتی محتویات آن را ببینید. 40 برای آدرس سگمنت (آخرین صفر مقروض است) و 10 برای افست از آدرس سگمنت منظور شده است تفسیر آدرس 40:10 مانند سگمنت 40[0]H بعلاوه افست 10H است. فرمان زیر را دقیقاً مانند آنچه که می‌بینید تایپ کنید: (کلید Enter را بفشارید) D 40:10

چیزی که نمایش داده می‌شود چنین آغاز می‌شود :

0040:0010 ×× ×× .. .. ..

بگوییم که ۲ بایت در کلمه وضعیت تجهیزات شامل مقادیر مبنای شانزده 63 و 44. برای تفسیر آن شما باید بایتها را معکوس کنید (4463) و آنها را به شکل دودویی تبدیل نمایید.

بیت:	۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰
دودویی :	0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	1

در زیر توضیح بیت‌ها از چپ به راست آمده است :

بیت‌ها ابزار

۱۴ و ۱۵ تعداد درگاه‌های چاپگر موازی متصل شده = ۱ (دودویی 01)

۹ - ۱۱ تعداد درگاه‌های سریال متصل شده = ۲ (دودویی 010)

۶ و ۷ تعداد ابزارهای دیسکت = ۲ (که 1 = 00 ، 2 = 01 ، 3 = 10 ، 4 = 11)

۴ و ۵ حالت‌اولیه ویدئو = ۱۵ (در حالیکه ۰۱=۲۵×۴۰ رنگی ، در حالیکه ۱۰=۲۵×۸۰ و ۱۱=۲۵×۸۰ تک‌رنگ)

۱ = کمک پردازشگر ریاضی وجود دارد

۰ = دیسک درایو موجود است.

بیتهایی که ذکر نشده استفاده نشده‌اند. می‌توانید برای تمرین بعدی در DEBUG باشید یا Q را برای خروج بزنید.

چک کردن اندازه حافظه

مرحله بعدی بررسی میزان حافظه پایه است که BIOS فکر می‌کند شما نصب کرده‌اید. بسته به مدل کامپیوتر این مقدار ممکن است، بر پایه تنظیم سوئیچهای داخلی باشد و حتی از میزان حافظه واقعی کمتر باشد. مقداری ناحیه داده BIOS در موقعیت 413H و 414H قرار دارد که شما می‌توانید آن را چنین تفسیر کنید :

سگمنت 40[0]H بعلاوه افست 13H و 14H . فرمان زیر را دقیقاً مانند آنچه که می‌بینید تایپ کنید.

(کلید Enter را بفشارید) D 40:13

آنچه که نمایش داده می‌شود چنین آغاز می‌شود: 0040:0013 .. .. .. xx xx .. ..

اولین دو بایت نمایش داده شده در تفاوت مکان 0013H، تعداد کیلوبایت اندازه حافظه در مبنای شانزده به شکل بایت معکوس است. برای مثال اگر ناحیه داده شامل 8002 H باشد مقدار صحیح آن 0280H است که به معنی 640K حافظه پایه است.

این دو تمرین‌اولیه محتویات ناحیه‌داده BIOS در حافظه‌پایینی را بررسی‌می‌کند. سیستم این‌مقادیر را وقتی‌کامپیوتر روشن می‌شود مقدار دهی اولیه می‌کند. سه تمرین بعدی داده‌ها ROM BIOS در حافظه بالایی را بررسی می‌کند.

بررسی عدد سریال و حق کپی محفوظ

عدد سریال کامپیوتر در ROM BIOS در موقعیت FE000H درج می‌شود. برای دیدن سگمنت [0]FE00 و تفاوت مکان 0، تایپ کنید.

(کلید Enter را بفشارید) D FE00:0

این عملیات باید عدد سریال ۷ رقمی و در ماشینهای قراردادی یک حق کپی محفوظ، بعد از آن قرار دارد. عدد سریال به صورت اعداد مبنای شانزده دیده می‌شود، در حالیکه حق کپی محفوظ بیشتر از بقیه کاراکترهای ناحیه ASCII سمت راست قابل تشخیص است. حق کپی محفوظ ممکن است بعد از آنچه که نمایش داده شده ادامه داشته باشد، برای دیدن آن، دوباره D را بفشارید و سپس Enter> را بفشارید.

بررسی تاریخ ROM BIOS

تاریخ ساخت ROM BIOS شما به صورت mm/dd/yy در موقعیت FFFF5H ثبت شده است. برای دیدن سگمنت [0]FFFF و تفاوت مکان 5، تایپ کنید. (کلید Enter را بفشارید) D FFFF : 5

دانستن این تاریخ برای دانستن مدل و سن کامپیوتر مفید است.

بررسی مدل ID

بلافاصله بعد از تاریخ ساخت ROM BIOS یک مدل ID یک بایتی در موقعیت FFFFH یا FFFF:E قرار دارد: در اینجا چندین مدل ID ذکر شده است.

کد	مدل
F8	PS/2 مدلهای 70 ، 80
FA	PS/2 مدل 30
FB	PC-XI (۱۹۸۶)
FC	PC-AT (۱۹۸۴)، PC-XT مدل ۲۸۶، PS/2 مدل‌هایی 50 و 60
FE	PC-XT (۱۹۸۲)، سبک (۱۹۸۲)
FF	کامپیوترهای شخصی IBM اصلی

حال که نحوه استفاده از فرمان نمایش را می‌دانید می‌توانید محتویات موقعیتهایی ذخیره سازی را ببینید. همچنین می‌توانید مرحله به مرحله با فشردن D به صورت مکرر در DEBUG ، ۸ خط متوالی بعد از فرمان D قبلی را ببینید. وقتی کاملاً حافظه را وارسی نمودید کلید Q برای خروج از DEBUG است یا می‌توانید با تمرین بعدی ادامه دهید.

مثال ۱ زبان ماشین: داده‌های بلافصل

حال باید از DEBUG برای ورود دو برنامه مستقیماً به حافظه و پیگیری اجرای آن استفاده کنیم، هر دو، دستورات زبان ماشین ساده‌ای هستند که در حافظه اصلی ظاهر می‌شوند و تأثیر اجرای آنها را مشخص می‌کنند. بدین منظور با فرمان E (Enter) از DEBUG آغاز می‌کنیم. در نحوه استفاده از آنها به طور خاص دقت کنید زیرا ورود داده در یک موقعیت اشتباه یا داده‌های نادرست نتایج غیر قابل پیش بینی به بار می‌آورد شما دوست ندارید سبب یک خرابی بشوید اما ممکن است کمی تعجب کنید که داده‌هایی که در DEBUG وارد کرده‌اید از دست بدهید.

اولین برنامه از ***داده‌های بلافصل*** استفاده می‌کند. داده‌هایی که در قسمتی از یک دستور تعریف می‌شوند در طی توضیحات هم کد مبنای شانزده زبان ماشین و برای خواناتر شدن کدهای سمبولیک آن را نشان می‌دهیم. برای اولین دستور کد سمبولیک چنین است. MOV AX,0123 که مقدار 0123 را به ثبات AX کپی می‌کند. (داده‌های بلافصل به صورت بایت معکوس تعریف نمی‌شوند). MOV دستور است، ثبات AX اولین عملوند و مقدار بلافصل 0123H دومین عملوند می‌باشد.

دستورماشین	کد سمبولیک	توضیح
B82301	MOV AX,0123	مقدار 0123 را در AX کپی می‌کند.
052500	ADD AX,0025	مقدار 0025 را با AX جمع می‌کند.
8BDB	MOV BX,AX	محتویات AX را در BX کپی می‌کند.
03D8	ADD BX,AX	محتویات AX را با BX جمع می‌کند.
8BCB	MOV CX,BX	محتویات BX را در CX کپی می‌کند.
2BC8	SUB CX,AX	محتویات AX را از CX تفریق می‌کند.
2BCO	SUB AX,AX	محتویات AX را از AX تفریق می‌کند.
90	NOP	

یک دستور ماشین ممکن است یک ۲ یا ۳ بایت طول داشته باشد. اولین بایت عمل واقعی است و بقیه بایتها که بعد از آن هستند، ***عملوندها*** هستند که یک مقدار بلافصل ، یک ثبات یا یک موقعیت حافظه را برمی‌گردانند. اجرای برنامه با اولین دستور ماشین آغاز می‌شود و مرحله به مرحله در هر دستور یکی پس از دیگری به ترتیب پیش می‌رود. در اینجا به شناخت بیشتری از کد ماشین توجه نداریم، برای مثال در یک حالت کد ماشین (در اولین بایت) برای کپی مقدار B8 و در حالت دیگر کد کپی مقدار 8B است.

ورود و دستورات برنامه

این تمرین را هم مانند قبلی آغازنمایید. دستور DEBUG را تایپ کنید و کلید <Enter> را بفشارید. وقتی DEBUG بطور کامل بارگذاری شد اعلان آن نمایش داده می‌شود (-). برای ورود مستقیم برنامه به حافظه قسمت زبان ماشین نه کد سمبولیک یا توضیح آن را تایپ کنید. پس از فرمان E یک فاصله بگذارید و چنین بنویسید :

E CS:100 B8 23 01 05 25 00 (کلید <Enter> را بفشارید)

CS:100 آدرس آغازین حافظه که داده‌ها باید در آن ذخیره شود را نشان می‌دهد - 100H (256) بایت پس از شروع سگمنت کد (آدرس شروع معمول برای کد ماشین با DEBUG). فرمان E سبب می‌شود تا هر زوج از ارقام مبنای شانزده در یک بایت حافظه از CS:100 تا CS:105 ذخیره شوند.

فرمان E بعدی ۶ بایت را با شروع از CS:106 تا 107 ، 108 ، 109 ، 10A ، 10B ذخیره می‌کند.

E CS:106 8B D8 03 D8 8B CB (سپس <Enter> را بزنید)

فرمان E بعدی ۵ بایت با شروع از CS:10C تا 10D ، 10E ، 10F ، 110 ذخیره می‌کند.

E CS : 10C 2B CB 28 C0 90 (سپس Enter را بزنید)

اگر یک فرمان را اشتباه زدید، آن با مقدار صحیح تکرار کنید.

اجرای دستورات برنامه

حال یک موضوع ساده برای اجرای دستورات قبلی و یکی در هر زمان وجود دارد. شکل ۱-۳ همه مراحل را نشان می‌دهد، فرمانهای E برای زدن کد ماشین استفاده می‌شود. صفحه شما نتایج مشابهی وقتی هر دستور DEBUG را وارد می‌کند باید نشان بدهد. فرمانهای DEBUG که با آن در اینجا کار کردیم R (ثبات) و T (پیگیری) است.

برای دیدن محتویات داخل ثباتها و پرچم‌ها، همانطور که در شکل ۱-۳ می‌بینید فرمان R را بزنید، سپس کلید <Enter> را بفشارید DEBUG محتویات ثباتها را به شکل مبنای شانزده چنین نشان می‌دهد :

```
AX=0000  BX=0000  ...
```

از آنجا که کامپیوترها ساختار متفاوتی دارند، برخی از محتویات ثباتها بر روی صفحه ممکن است با آنچه به شکل ۱-۳ می‌بینید متفاوت باشد. ثبات IP باید به شکل IP = 0100 دیده شود. مبنی بر آنکه دستورات اجرایی 100H بایت بعد از شروع سگمنت کد آغاز می‌شوند (چرا که شما E CS:100 را برای شروع برنامه وارد کردید) ثبات پرچم در شکل ۱-۳ تنظیمات سر ریز، جهت، وقفه، علامت، صفر، رقم نقلی معین، توازن و پرچم رقم نقلی را نشان می‌دهد:

```
NV UP EI PL NA DO N2 NC
```

این تنظیمات به این معنی است، سر ریز وجود ندارد، جهت به بالا (یا راست) وقفه فعال است، علامت مثبت است، صفر نیست، رقم نقلی معین وجود ندارد، توازن فرد است، رقم نقلی وجود ندارد. در اینجا هیچ کدام از این تنظیمات برای ما مهم نیست.

بلافاصله بعد از ثباتها، آنچه که با فرمان R نمایش داده می‌شود، این است که دستور بعدی اجرا خواهد شد. به یاد بسپارید که در شکل ثبات CS شامل 21CI می‌باشد. از آنجائیکه آدرس سگمنت CS شما با این مقدار متفاوت خواهد بود ما آن را با xxxx نمایش می‌دهیم : `xxxx : 0100 B82301  MOV AX,0123`

- xxxx بر شروع سگمنت کد به شکل [0]xxxx دلالت دارد. مقدار 0100:xxxx به معنی افست 100H بایت بعد از آدرس سگمنت CS[0]xxxx است.
- B82301 کد ماشینی است که در CS:100 وارد می‌کنید.
- MOV AX,0123 دستور سمبولیک اسمبلی است که DEBUG از کد ماشین آن را تشخیص می‌دهد. این دستور در حقیقت به معنی فرستادن مقدار بلافصل 0123H به ثبات AX می‌باشد.

DEBUG دستورات ماشین را Unassembled می‌کند بنابراین شما ممکن است براحتی آن را تفسیر نمایید. بعد از این فصل، شما منحصراً دستورات سمبولیک اسمبلی را وارد می‌کنید.

در این جا، دستور MOV اجرا نمی‌شود. بدین منظور، کلید T را بفشارید و سپس <Enter> را وارد کنید. کد ماشین B8 (انتقال به ثبات AX) بعد از آن 2301 آمده است. این عملیات 23 را به قسمت پایین رتبه (AL) از ثبات AX و 01 را به قسمت بالا رتبه (AH) از ثبات AX کپی می‌کند :

	AH	AL
AX:	01	23

DEBUG تأثیر عملیات بر روی ثباتها را نشان می‌دهد. ثبات IP حال شامل 0103H است (مقدار اصلی 0100 بعلاوه ۳ بایت برای دستور کد ماشین است). این مقدار بر موقعیت افست دستور بعدی که باید اجرا شود دلالت دارد.

```
xxxx : 0103  052500  ADD AX,0025
```

برای اجرای دستور ADD، یک T دیگر وارد کنید. این دستور 25H را به قسمت پایین رتبه (AL) از AX و 00H را به قسمت بالا رتبه (AH)، در حقیقت 0025H را با AX جمع می‌کند. AX حال شامل 0148H است و IP شامل 106H است برای دستور بعدی که باید اجرا بشود.

```
xxxx : 0106  8BD8  MOV BX,AX
```

```
-E CS:100 B8 23 01 05 25 00
-E CS:106 8B D6 03 D8 8B CB
-E CS:10C 2B C8 2B C0 90
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0100   NV UP EI PL NZ NA PO NC
21C1:0100 B82301        MOV     AX,0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0103   NV UP EI PL NZ NA PO NC
21C1:0103 052500        ADD     AX,0025
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0106   NV UP EI PL NZ NA PE NC
21C1:0106 8BD8          MOV     BX,AX
-T

AX=0148  BX=0148  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0108   NV UP EI PL NZ NA PE NC
21C1:0108 03D8          ADD     BX,AX
-T

AX=0148  BX=0290  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=010A   NV UP EI PL NZ AC PE NC
21C1:010A 8BCB          MOV     CX,BX
-T

AX=0148  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=010C   NV UP EI PL NZ AC PE NC
21C1:010C 2BC8          SUB     CX,AX
-T

AX=0148  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=010E   NV UP EI PL NZ AC PE NC
21C1:010E 2BC0          SUB     AX,AX
-T

AX=0000  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0110   NV UP EI PL ZR NA PE NC
21C1:0110 90            NOP
```

شکل ۱-۳ پیگیری دستورات ماشین

بار دیگر فرمان T را بزنید. دستور MOV محتویات ثبات AX را به ثبات BX منتقل می‌کند. بخاطر بسپارید که بعد از انتقال BX حاوی 0148H خواهد بود. AX هنوز حاوی 0148H است زیرا MOV کپی می‌کند تا آنکه بیشتر به معنی انتقال واقعی از یک موقعیت به موقعیت دیگر باشد.

حال به ترتیب فرمان T را برای دستورات باقیمانده بزنید. دستور ADD محتویات AX را با BX جمع می‌کند و در BX مقدار 0290H را بدست می‌دهد. سپس برنامه محتویات BX را به CX منتقل (کپی) می‌کند، AX را از CX تفریق می‌کند و AX را از خودش تفریق می‌کند. بعد از این آخرین عمل، پرچم صفر از NZ به ZR تغییر می‌کند به این معنی که نتیجه آخرین عمل صفر است. (تفریق AX از خودش آن را صفر می‌کند) اگر شما این دستورات را بخواهید دوباره اجرا کنید، باید مقدار ثبات IP را با 100H مقدار دهید. در حقیقت، شما این خروج را بارها انجام خواهید داد، این روال در زیر آمده است :

۱- تایپ کنید R IP تا محتویات ثبات IP را ببینید و

۲- مقدار 100 را تایپ کنید و کلید Enter را بزنید.

این روال شما را به شروع برنامه می‌برد، در حالیکه شما می‌توانید مراحل قبلی را تکرار کنید. فرمانهای R ، T را به تعداد لازم بفشارید و پس از هر کدام کلید Enter را بزنید

نمایش محتویات حافظه

اگر شما می‌توانید T را برای آخرین دستور بفشارید ، دستور NOP (هیچ عمل) کاری انجام نمی‌دهد. در عوض برای مشاهده برنامه زبان ماشین در سگمنت کد، چنین برای نمایش در خواست می‌کنیم: D CS : 100

شکل ۲-۳ نتیجه این فرمان را نشان می‌دهد، با ۱۶ بایت (۳۲ رقم مبنای شانزده) از داده‌های نمایش داده شده در هر خط در سمت راست بیان ASCII هر بایت (اگر یک کاراکتر استاندارد باشد) قرار دارد. در حالت کد ماشین، بیان ASCII بی مفهوم است و نادیده گرفته می‌شود. بخش بعدی سمت راست نمایش داده شده را با توضیحات بیشتری نمایش می‌دهد. خط اول نمایش داده شده از افست 100H از سگمنت کد آغاز می‌شود و محتویات موقعیت CS:100 تا CS:10F را نمایش می‌دهد. خط دوم محتویات CS:110 تا CS:11F را بیان می‌کند. اگر چه برنامه در CS:110 پایان می‌یابد اما دستور D بطور اتوماتیک ۸ خط، از CS:100 تا CS:170 را نمایش می‌دهد. برای منظور ما، هر داده بعد از CS:110 بی مفهوم است.

بجز کد ماشین ناحیه CS:100 تا 110 بقیه یکسان هستند، بایتهایی که بعد از آن هستند ممکن است حاوی هر چیزی باشند. اگر چه شکل ۲-۳ نشان می‌دهد که ثباتهای DS ، ES ، SS ، CS حاوی همان آدرس هستند. این مسئله بدین دلیل است که DEBUG با این ناحیه مانند برنامه COM. یعنی کد و داده در یک سگمنت رفتار می‌کند، اگر چه شما باید آنها را در داخل سگمنت مجزا نمایید.

Q را وارد کنید تا برنامه DEBUG خاتمه یابد یا برای تمرین بعدی ادامه دهید.

```
-D CS:100
21C1:0100  B8 23 01 05 25 00 8B D8-03 D8 8B CB 2B C8 2B C0   .#..%.......+.+.
21C1:0110  90 C3 8D 46 14 50 51 52-FF 76 28 E8 74 00 8B E5   ...F.PQR.v(.t...
21C1:0120  B8 01 00 50 FF 76 32 FF-76 30 FF 76 2E FF 76 28   ...P.v2.v0.v..v(
21C1:0130  E8 88 15 8B E5 FF 36 18-12 FF 36 16 12 8B 76 28   ......6...6...v(
21C1:0140  FF 74 3A 89 46 06 E8 22-CE 8B E5 30 E4 3D 0A 00   .t:.F.."...0.=..
21C1:0150  75 32 A1 16 12 2D 01 00-8B 1E 18 12 83 DB 00 53   u2...-.........S
21C1:0160  50 8B 76 28 FF 74 3A A3-16 12 89 1E 18 12 E8 FA   P.v(.t:.........
21C1:0170  CD 8B E5 30 E4 3D 0D 00-74 0A 83 06 16 12 01 83   ...0.=..t.......
-
```

شکل ۲-۳ نمایش مبنای شانزده سگمنت کد

تصحیح یک ورودی

اگر یک مقدار اشتباه در یک برنامه وارد کردید، دوباره فرمان E را وارد کنید و آن را تصحیح نمایید. همچنین برای صرفنظرنمودن از اجرای اولین دستور، ثبات IP را با 0100 تنظیم‌کنید - طبق روالی که قبلاً توضیح داده‌شده : تایپ‌کنید R IP و مقدار 100 را وارد کنید. سپس فرمان R را وارد کنید (بدون DEBUG (IP ثباتها، پرچم‌ها و اولین دستوری که اجرا می‌شود را نشان می‌دهد. می‌توانید از فرمان T برای پیگیری مراحل دستورات استفاده کنید. اگر برنامه شما مجموع را محاسبه کند، باید برخی موقعیتهای حافظه و ثباتها را پاک کنید. اما محتویات ثباتهای CS ، DS ، SP ، SS که جهت منظورهای خاص مشخص شده‌اند، نباید تغییر بدهید.

مثال ۲ زبان ماشین : تعریف داده

مثال قبلی از مقادیر بلافصل تعریف شده در داخل دستورات MOV و ADD استفاده می‌کند. ما بعداً مثال مشابهی را مشخص می‌کنیم که مقادیر (با محتویات) 0123H و 0025H را مانند اقلام داده مجزا در داخل برنامه تعریف می‌کند.

برنامه موقعیتهای حافظه که شامل این مقادیر هستند را دستیابی می‌کند.

کار با این مثال، آگاهی در مورد نحوه دستیابی داده‌ها توسط کامپیوتر از طریق یک آدرس در ثبات DS است افست را به شما می‌دهد. مثال زیر داده‌ها و محتویات آنها را که در افست 0200H است تعریف می‌کند، که بطور واضحی از دستورات آدرس 0100H مجزا هستند.

افست DS	محتویات مبنای شانزده
0200H	2301H
0202H	2500H
0204H	0000H
0206H	2A2A2AH

بخاطر دارید که یک رقم مبنای شانزده نصف بایت را اشغال می‌کند، بنابراین برای مثال، 23H در افست 0200H (اولین بایت) از ناحیه داده ذخیره می‌شود و 01H در افست 0201H (دومین بایت) ذخیره می‌شود. در اینجا دستورات ماشین که چنین اقلام داده را پردازش می‌کند آورده شده است. مقادیر به صورت بایت معکوس وارد شده‌اند، برای مثال 0200 به شکل 0002 است :

دستور	توضیح
A10002	انتقال کلمه (۲ بایت) در افست 0200H از DS به ثبات AX .
03060202	جمع کردن محتویات کلمه (۲ بایت) شروع شده در افست 0202H از DS با ثبات AX.
A30402	انتقال محتویات ثبات AX به کلمه‌ای که در افست 0209H قرار دارد.
90	هیچ عملی را انجام نده.

ممکن است متوجه شده باشید که دو دستور انتقال، کد ماشین متفاوتی دارند: A1 , A3. کد ماشین واقعی به ثباتی که به آن مراجعه می‌کند بستگی دارد و به اندازه داده (بایت یا کلمه)، جهت انتقال داده (از یا به ثبات) و ارجاع داده بلافصل، حافظه یا ثبات بستگی دارد.

```
-E DS:200 23 01 25 00 00 00
-E DS:206 2A 2A 2A
-E CS:100 A1 00 02 03 06 02 02
-E CS:107 A3 04 02 90
-D DS:200,208
21C1:0200  23 01 25 00 00 00 2A 2A-2A                          #.%...***
-D CS:100,10A
21C1:0100  A1 00 02 03 06 02 02 A3-04 02 90                    ..........
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0100   NV UP EI PL NZ NA PO NC
21C1:0100 A10002        MOV     AX,[0200]                      DS:0200=0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0103   NV UP EI PL NZ NA PO NC
21C1:0103 03060202      ADD     AX,[0202]                      DS:0202=0025
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0107   NV UP EI PL NZ NA PE NC
21C1:0107 A30402        MOV     [0204],AX                      DS:0204=0000
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=010A   NV UP EI PL NZ NA PE NC
21C1:010A 90            NOP
-D DS:0200,0208
21C1:0000  23 01 25 00 48 01 2A 2A-2A                          #.%.H.***
-Q
```

شکل ۳-۳ پیگیری دستورات ماشین

تایپ کردن دستورات برنامه و داده

شما می‌توانید از DEBUG به منظور ورود برنامه و دیدن اجرای آن استفاده کنید. ابتدا، از فرمان E برای ورود دستورات با شروع از CS:0100 استفاده می‌کنیم :

```
E CS:100  A1  00  02  03  06  02  02  (<Enter> را بفشارید)
E CS:107  A3  04  02  90  (<Enter> را بفشارید)
```

حال از فرمان E برای تعریف داده‌ها با شروع از DS:0200 بطور اختیاری استفاده می‌کنیم.

```
E DS:0200  23  01  25  00  00  00  (<Enter> را بفشارید)
E DS:02  06  2A  2A  2A  (<Enter> را بفشارید)
```

اولین فرمان E ، سه کلمه (۶ بایت) در شروع ناحیه داده در آدرس 0200 ذخیره می‌کند. شما باید هر یک از این کلمات را به صورت بایت معکوس ذخیره کنید، بنابراین 0123 تبدیل به 2301 و 0025 تبدیل به 2500 خواهد شد. وقتی یک دستور MOV کلمات را به ترتیب دستیابی کند و در داخل ثبات بارگذاری نماید، بایت‌ها را دوباره به شکل غیر معکوس تبدیل می‌کند، بنابراین 2301 تبدیل به 0123 و 2500 تبدیل به 0023 خواهد شد. دومین فرمان E سه علامت (***) را، به صورت 2A2A2A تعریف می‌کند، بنابراین شما می‌توانید آنها را بعداً با استفاده از فرمان D (فرمان نمایش) ببینید. در غیر اینصورت این علامتها منظور خاصی در ناحیهٔ داده ندارند.

شکل ۳-۳ همهٔ مراحل برنامه، شامل دستورات E را نشان می‌دهد. صفحه شما باید نتایج مشابهی را نشان دهد، اگرچه آدرس CS و DS احتمالاً متفاوت خواهد بود. برای امتحان داده‌های ذخیره شده (در DS:200H تا 208H) و دستورات (در CS:100H تا 10AH)، دستورات D زیر را تایپ کنید:

برای دیدن کد: `D CS : 100,10A <Enter>`

برای دیدن داده: `D CS : 200,208 <Enter>`

بررسی کنید که محتویات هر دو ناحیه با آنچه که در شکل ۳-۳ می‌بینید یکسان باشد.

اجرای دستورات برنامه

بعد از تایپ دستورات، می‌توانید آنها را اجرا کنید همانطور که قبلاً انجام دادید. ابتدا مطمئن شوید که IP حاوی 100H است. سپس کلید R را برای دیدن محتویات ثباتها و پرچم و برای نمایش اولین دستور، بفشارید. اگر چه ثبات AX ممکن است حاوی مقداری از مثال قبل باشد، مقدار آن را جایگزین‌کنید. اولین دستور نمایش‌داده‌شده چنین است :

```
xxxx : 0100 A10002 MOV AX,[0200]
```

CS:0100 به اولین دستور A10002 مراجعه می‌کند. مفسر DEBUG این دستور را مانند یک دستور MOV، تفسیر می‌کند که ارجاع آن به اولین موقعیت [0200H] در ناحیه داده است. کروشه مشخص می‌کند که این یک ارجاع به حافظه است نه یک مقدار بلافصل. (یک مقدار بلافصل جهت انتقال 0200H به ثبات AX بدین شکل ظاهر خواهد شد :

```
(MOV  AX,0200).
```

حال کلید T را (برای فرمان پیگیری) بفشارید. دستور MOV AX,[0200] محتویات کلمه‌ای که در افست 0200H قرار دارد به ثبات AX منتقل می‌کند. محتویات 2301H است که عملیات آن را معکوس به شکل 0123H در AX ذخیره می‌کند و جایگزین هر مقدار قبلی در AX می‌گردد. بار دیگر کلید T را بزنید تا دستور بعدی، ADD اجرا گردد. عملیات ، محتویات کلمه را حافظه را در افست 0202 از DS با ثبات AX جمع می‌کند. نتیجه در AX، مجموع 0123H و 0025H یا 0148H خواهد بود.

دستور بعدی چنین است، MOV [0204],AX کلید T را بفشارید تا اجرا شود. این دستور محتویات ثبات AX را به ناحیه داده با افست 204H و 205H از DS به شکل بایت معکوس 4801H ذخیره می‌کند. برای دیدن محتویات تغییر

یافته داده از 200H تا 208H ، تایپ کنید: D DS : 200,208 <Enter>

مقادیر نمایش داده شده چنین باید باشد:

مقادیر ناحیه داده :	23	01	25	00	48	01	2A	2A	2A
افست :	200	201	202	203	204	205	206	207	208

سمت چپ، کد ماشین واقعی که در حافظه ظاهر می‌شود نمایش می‌دهد. سمت راست به شما کمک می‌کند تا کاراکترهای داده را به ساده‌ترین شکل قرار دهید. توجه کنید که این مقادیر مبنای شانزده، در سمت راست صفحه با مقادیر معادل ASCII بیان شده‌اند. بنابراین 23H سمبول (#) و 25H سمبول درصد (٪) است همانطور که 24H علامت ($) است.

از آنجائیکه دستور دیگری برای اجرا وجود ندارد، Q را وارد کنید تا از DEBUG خارج شوید یا با تمرین بعدی ادامه دهید (IP را با 100 مقدار دهی نمایید).

یک مثال زبان اسمبلی

اگر چه تا اینجا مثالهای برنامه به شکل زبان ماشین داده شده‌اند، می‌توانید از DEBUG برای تایپ جملات زبان اسمبلی استفاده نمایید. امکان استفاده از هر دو روش را دارید. حال بیایید فرمانهای AوU از DEBUG را برای ورود جملات اسمبلی به کامپیوتر، امتحان نماییم.

فرمان A (Assemble)

فرمان A (Assemble) به DEBUG می‌گوید تا دستورالعملهای اسمبلی را بپذیرد و آنها را به زبان ماشین تبدیل کند. آدرس شروع را به صورت زیر مقدار دهی نمایید. A100 <Enter>

DEBUG ارزش سگمنت کد و افست را به صورت 0100 : xxxx نمایش می‌دهد. هر دستورالعمل را وارد کرده و پس از آن Enter را بفشارید :

```
MOV  CL,42    <Enter>
MOV  DL,2A    <Enter>
ADD  CL,DL    <Enter>
NOP           <Enter>,<Enter>
```

وقتی برنامه را وارد کردید، <Enter> را دوبار بزنید تا از دستور A خارج شوید. یک <Enter> اضافی، به DEBUG می‌گوید که شما دستور سمبولیک دیگری برای ورود ندارید، DEBUG چنین نمایش خواهد داد.

```
xxxx : 0100   MOV CL,42
xxxx : 0102   MOV DL,2A
xxxx : 0104   ADD CL,DL
xxxx : 0106   NOP
```

می‌بینید که DEBUG مکان شروع هر دستور را تعیین کرده است. اما قبل از اجرای برنامه، اجازه دهید از فرمان U (Unassemble) از DEBUG برای امتحان زبان ماشین تولید شده استفاده کنیم.

فرمان U (Unassemble)

فرمان U (Unassemble) مربوط به DEBUG، کدماشین دستورهای‌اسمبلی را نشان می‌دهد. شما بایستی مکانهای اولین و آخرین دستوری را که می‌خواهید ببینید، که در اینحالت 100 و 106 است، به DEBUG بگویید، تایپ کنید.

U 100,106 <Enter>

در صفحه ستونهایی برای موقعیت، کد ماشین و کد سمبولیک، نمایش داده خواهد شد.

```
xxxx : 0100   B142    MOV  CL,42
xxxx : 0102   B22A    MOV  DL,2A
xxxx : 0104   00D1    ADD  DL,CL
xxxx : 0106   90      NOP
```

حال اجرای برنامه را پیگیری کنید - آنچه که در حقیقت اجرا می‌شود کد ماشین است. برای نمایش محتویات ثباتها و اولین دستور با R شروع کنید و T را برای پیگیری دستورهای بعدی استفاده نمایید وقتی به دستور NOP در موقعیت 106H می‌رسیدی، IP باید حاوی 106H و CL باید حاوی 6CH باشد. با تمرین بعدی ادامه بدهید یا با فشردن Q از اجرا خارج شوید.

حال چگونگی وارد کردن یک برنامه را به صورت زبان ماشین یا زبان اسمبلی می‌توانید ببینید. اگر چه DEBUG همانطور که از نامش پیداست - اشکال زدای برنامه‌ها است - و بیشتر تلاش شما در جهت استفاده از برنامه به زبان اسمبلی خواهد بود، که وابسته به DEBUG نباشد.

استفاده از دستور INT

چهار مثال بعد نحوه درخواست اطلاعات راجع به سیستم را نشان می‌دهد. برای این منظور، باید از دستور INP (Interupt) استفاده کنید که از برنامه شما خارج می‌شود، تک روتین DOS یا BIOS وارد می‌شود، تابع خواسته شده را انجام می‌دهد و به برنامه شما باز می‌گردد. چندین نوع مختلف عمل INT وجود دارد، برخی از آنها یک کد تابعی در ثبات AH جهت انجام عملی ویژه نیاز دارند. علاوه بر فرمان T برای اجرای تک مرحله‌ای، از فرمان P (Proceed) برای اجرای کامل روتین وقفه استفاده می‌کنیم. IP را با مقدار 0100H تنظیم کنید.

دیدن تاریخ جاری

دستوری که برای دستیابی به تاریخ جاری است INT 21H و کد تابع 2AH می‌باشد. در DEBUG فرمان A100 وارد کنید و سپس دستورات اسمبلی زیر را وارد کنید:

```
MOV  AH,2A
INT  21
NOP
```

R را برای دیدن ثباتها و T را برای اجرای MOV تایپ کنید. سپس برای پیش رفتن در روتین وقفه P را تایپ کنید، عملیات در دستور NOP متوقف خواهد شد. ثباتها حاوی این اطلاعات در مبنای شانزده خواهند بود.

```
AL :   روز هفته، که 0 یعنی یک شنبه
CX :   سال (مثلاً 07D0H = 2000)
DH :   ماه (01H تا 0CH)
DL :   روز ماه (01H تا 1FH)
```

با فشردن Q خارج شوید، یا با تمرین بعدی ادامه دهید (IP را با 100 تنظیم کنید).

تعین اندازه حافظه

در این فصل تمرینی ذکر شد که موقعیتهای 413H و 414H را جهت اندازه‌گیری حافظه پایهٔ کامپیوتر شما را، بررسی می‌کرد. BIOS نیز یک روتین وقفه دارد، INT 12H که اندازه حافظه را ارائه می‌کند. در DEBUG فرمان A 100 را وارد کنید و سپس این دستورات را بزنید:

```
INT  21
NOP
```

R را برای نمایش ثباتها و اولین دستور تایپ کنید. دستور، INT 12H کنترل را به روتینی در BIOS که اندازه

حافظه پایه را در ثبات AX ارائه می‌کند. T و سپس (Enter) را برای دیدن اجرای هر دستور BIOS بزنید. (بله، ما از قانون پیگیری یک وقفه تخلف می‌کنیم، اما این عملیات درست کار می‌کند) دستورات واقعی در BIOS شما ممکن است با این تفاوت داشته باشد بسته به ورژنی که نصب کرده‌اید (توضیحات سمت راست از نویسنده است):

```
STI              ;  تنظیم وقفه
PUSH  DS         ;  ذخیره آدرس DS در پشته
MOV   AX,0040    ;  بازیابی اندازه حافظه در
MOV   DS,AX      ;  سگمنت H [0] 40 بعلاوه
MOV   AX,[0013]  ;  افست 0013H
POP   DS         ;  بازیابی آدرس در DS
IRET             ;  بازگشت از وقفه
```

اگر شما این ماجرا در BIOSرا بگذرانید، حال AX شامل اندازه حافظه پایه، در 1K بایت است. آخرین فرمان T از BIOS خارج می‌شود و به DEBUG بر می‌گردد. دستور نمایش داده شده NOP است که شما وارد کرده‌اید. Q را بفشارید تا خارج شوید یا با تمرین بعدی ادامه دهید (و IP را با 100 تنظیم کنید)

استفاده از INT برای نمایش

این مثال که داده‌ها را بر روی صفحه نمایش می‌دهد، چندین نکته جدید دارد. در DEBUG فرمان A100 را وارد کنید و دستورات اسمبلی زیر را تایپ کنید:

```
100  MOV  AH,09
102  MOV  DX,108
105  INT  21
107  NOP
108  DB   'Your name', '$'
```

دو دستور MOV به INT 21H (AH=09) می‌گوید آنچه که از آدرس 108 آغاز می‌شود (DX=108) را نمایش دهد. توجه داشته باشید که آدرس 108 آغاز تعریف نام شماست. دستور DB به معنی تعریف بایت است و کاراکترها در یک کد (' ') قرار دارند بعد از نام شما یک علامت دلار $ در همان کد (' ') قرار دارد که به INT می‌گوید انتهای نمایش است.

R را بفشارید تا ثباتها و اولین دستور را ببینید و فرمان T را برای دو MOV بفشارید.حال P را برای‌اجرای INT بفشارید و نام‌خودتان را می‌بینید. حال Q را بفشارید تا خارج‌شوید یا با تمرین‌های ادامه‌دهید. (IP را با 100 تنظیم‌کنید)

استفاده از INT برای ورودی صفحه کلید

این تمرین که از صفحه کلید کاراکتر می‌پذیرد، نیز شامل چندین نکته تازه است. در DEBUG فرمان A100 را وارد کنید و دستورات اسمبلی زیر را تایپ نمایید.

```
100  MOV  AH,10
102  INT  16
104  JMP  100
106  NOP
```

اولین دستور، MOV به INT 16H می‌گوید تا از صفحه کلید داده بپذیرد (AH = 10) عملیات، کاراکتری که از صفحه کلید می‌پذیرد در ثبات AL تحویل می‌دهد. دستور JMP سبب می‌شود تا پردازشگر مقدار IP را با 100 جایگزین سازد، بنابراین دستور بعدی که اجرا می‌شود MOV در 100H است. بدین منظور می‌توانید یک یا چند دستور را مرتباً اجرا نمایید. R را برای دیدن ثباتها و اولین دستور بفشارید و فرمان T را برای اجرای MOV وارد کنید. وقتی فرمان P را برای INT تایپ کنید، سیستم منتظر می‌شود تا یک کلید را بفشارید. اگر 1 را بفشارید می‌بینید که عملیات مقدار 31H (معادل مبنای شانزده ASCII) را در AL ارائه می‌دهد. T را بفشارید تا دستور JMP اجرا شود و به دستور MOV در 100 برگردید. از T برای اجرای MOV استفاده کنید. وقتی P را برای INT بزنید، سیستم مجدداً منتظر می‌شود تا یک کلید را بفشارید. اگر کلید 2 را بفشارید خواهید دید که عملیات، 23H را در AL ارائه می‌دهد. می‌توانید همینطور ادامه بدهید. Q را برای خروج بفشارید یا با تمرین بعدی ادامه بدهید (IP را با مقدار 100 تنظیم نمایید)

ذخیره یک برنامه از داخل DEBUG

با استفاده از DEBUG می‌توانید یک برنامه را با دو شرط زیر بر روی دیسک ذخیره نمائید :

۱) برای بازیابی برنامه‌ای موجود در دیسک، تغییر آن و ذخیره آن مراحل زیر را پیگیری کنید :

- برنامه را با نام آن فراخوانی کنید : DEBUG n:filename.
- از فرمان D برای دیدن برنامه زبان ماشین و E برای ورود تغییرات استفاده کنید.
- فرمان W (write) را برای نوشتن برنامه وارد کنید.

۲) برای ایجاد برنامه‌های زبان ماشین کوتاه با استفاده از DEBUG که می‌خواهید آنها را ذخیره نمایید، مراحل زیر را پیگیری کنید :

- برنامه DEBUG را فراخوانی کنید.
- از فرمانهای A و E برای وارد کردن برنامه اصلی استفاده کنید.
- برای نامگذاری برنامه تایپ کنید N filname.com توسعه برنامه باید COM. باشد. (فصل ۷ را برای توضیح بیشتر راجع به فایلهای COM ببینید)
- از آنجائیکه فقط شما می‌دانید کجا برنامه خاتمه یافته، در جفت ثبات BX:CX اندازه برنامه را به بایت قراردهید. به این مثال توجه کنید :

توجه کنید که اگر چه کدهای سمبولیک را وارد کرده‌اید، DEBUG کد ماشین تولید می‌کند و آنچه که شما می‌خواهید ذخیره می‌کند. چون آخرین دستور NOP، یک بایتی است اندازهٔ برنامه از 100H تا 106H، یا 7 می‌باشد.

```
xxxx : 0100  MOV  CL,42
xxxx : 0102  MOV  DL,2A
xxxx : 0104  ADD  CL,DL
xxxx : 0106  NOP
```

- ابتدا از R BX برای نمایش BX (قسمت بالای اندازه) استفاده کنید و با 0 آن را مقدار دهید.
- حال از R CX برای نمایش ثبات CX استفاده نمائید. DEBUG به شکل CX nnnn (مقداری که حاوی آن است) پاسخ می‌دهد و شما می‌توانید آن را با اندازه برنامه یعنی ۷ جایگزین کنید.
- کلیدهای (Enter) W را بزنید تا برنامه بر روی دیسک ذخیره شود.

DEBUG یک پیغام نمایش می‌دهد 'Writing 7 bytes' (۷ بایت نوشته شد) اگر تعداد صفرباشد شما در ورود طول برنامه موفق نبودید، دوباره تلاش کنید. به اندازه برنامه بدقت بنگرید، زیرا آخرین دستور باید بیشتر از یک بایت باشد.

استفاده از عملگر اشاره گر

حال بیایید برنامه دیگری که شامل چندین نکته جدید است امتحان کنیم. در این مثال، شما داده‌ها را در بیت ثباتها و موقعیتهای حافظه منتقل و جمع می‌کنید. در اینجا دستورات اسمبلی برای این منظور ذکر شده است.

```
100   MOV   AX,[11A]
103   ADD   AX,[11C]
107   ADD   AX,25
10A   MOV   [11E],AX
10D   MOV   WORD PTR [120],25
113   MOV   BYTE PTR [122],30
118   NOP
119   NOP
11A   DB    14 23
11C   DB    05 00
11E   DB    00 00
120   DB    00 00 00
```

توضیح دستورات در زیر آمده است :

100 : انتقال محتویات موقعیتهای 11AH-11BH به ثبات AX. کروشه به معنی یک آدرس حافظه است نه یک مقدار بلافصل.

103 : جمع محتویات موقعیت حافظه 11CH-11DH با ثبات AX.

107 : جمع مقدار بلافصل 25H با AX.

10A : انتقال محتویات AX به موقعیت حافظه 11EH-11FH.

10D : انتقال مقدار بلافصل 25H به موقعیت‌های 120H-121H حافظه. توجه کنید استفاده از عملگر WORD PTR به DEBUG می‌گوید که 25H را به یک کلمه در حافظه منتقل کند. اگر دستور را به این شکل کد کنید : MOV[120],25 ، DEBUG راهی برای تشخیص طول نخواهد داشت و یک پیغام خطا نشان می‌دهد. اگر چه شما به ندرت نیاز به استفاده از عملگر PTR خواهید داشت، دانستن آن برای مواقع نیاز ضروری است.

113 : انتقال مقدار بلافصل 30H به موقعیت 122H. در اینجا می‌خواهیم یک بایت را منتقل کنیم و عملگر BYTE PTR بر این طول دلالت دارد.

11A : تعریف مقادیر بایتی 14H و DB.23H در اینجا به معنی تعریف بایت(ها) و مجوزی است برای تعریف اقلام داده که برنامه شما (مانند ارجاع در 100) بدان رجوع می‌کند.

11E,120 و 11C : تعریف دیگر مقادیر بایتها برای استفاده در برنامه.

برای وارد کردن برنامه ابتدا تایپ کنید A 100 <Enter> و سپس دستورات سمبولیک را وارد کنید. (البته موقعیت لازم نیست). در انتها یک کلید (Enter) اضافی برای خروج از فرمان A بزنید. برای اجرای برنامه، با وارد کردن R برای دیدن ثباتها و اولین دستور آغاز کنید، سپس چندین فرمان T را وارد کنید. وقتی به NOP در 118 رسیدید از اجرا خارج شوید. فرمان D 110 را برای دیدن تغییرات محتویات AX (233E) و موقعیت‌های 11EH-11FH(3E23) ، 120H-121H(2500) و 122H(30) وارد کنید.

در این فصل شما مطالب زیادی فراگرفتید که با تکرار واضح‌تر خواهد شد.

نکات کلیدی

- برنامه DEBUG برای تست و اشکال زدایی برنامه‌های زبان ماشین و زبان اسمبلی مفید است.
- DEBUG مجموعه فرامینی دارد که برای اجرای عملیات مختلف مانند نمایش، ورود و پیگیری استفاده می‌شود.
- از آنجائیکه DEBUG بین حروف بزرگ و کوچک فرقی قائل نیست، می‌توانید فرامین را به هر شکلی وارد کنید.
- DEBUG همه اعداد را به شکل مبنای شانزده فرض می‌کند.
- اگر یک مقدار نادرست در سگمنت داده یا سگمنت کد وارد کردید، با فرمان E دوباره شکل صحیح آن را وارد کنید.
- برای دوباره آغاز کردن از اجرای اولین دستور، ثبات IP را با 100 مقدار دهید. به این شکل که فرمان R و سپس از آن ثبات مورد نظر را وارد کنید، مثل DEBUG . R IP <Enter> محتویات IP را نشان می‌دهد و منتظر می‌ماند تا یک مقدار وارد کنید. مقدار100 را وارد کنید و سپس <Enter> بزنید.

پرسش‌ها

۱-۳. منظور هر یک از فرمانهای DEBUG زیر را توضیح‌دهید : (الف) D ، (ب) E ، (ج) R ، (د) Q ، (ه) T ، (و) A ، (ز) U ، (ح) P .

۲-۳. چگونه می‌توان اعمال زیر را با فرامین DEBUG انجام داد؟

(الف) نمایش حافظه با شروع از افست 1A5H در سگمنت داده.

(ب) نمایش حافظه با شروع از موقعیت B40H (توجه کنید: این آدرس را به سگمنت و افست مجزا کنید

(ج) مقدار مبنای شانزده 444E41 را در سگمنت داده با شروع از 18AH وارد کنید.

(د) محتویات همه ثباتها را نشان دهید. (ه) محتویات فقط ثبات TP ....

(و) کد سمبولیک در موقعیتهای 100H تا 11AH را به زبان ماشین تبدیل کنید.

۳-۳. دستور زبان ماشین عملیات زیر را بنویسید. (الف) انتقال مبنای شانزده 324B به ثبات AX (ب) جمع مقدار شانزدهی 024B با AX.

۳-۴. فرض کنید که از DEBUG برای وارد کردن فرمان E زیر استفاده کردید :

E SC : 100 B8 36 01 05 25 00

مقدار 36 فرض کنید 54 باید باشد. یک فرمان E دیگر برای تصحیح فقط یک بایت اشتباه کد کنید که 36 را مستقیماً با 54 تعویض کند.

۳-۵. فرض کنید که از DEBUG برای ورود فرمان E زیر استفاده کردید:

E CS : 100 B8 06 20 05 00 30 90

الف) در اینجا سه دستور سمبولیک بیان شده چه هستند؟ (برنامه اول فصل یک سر نخ ارائه می‌دهد).

ب) در اجرای این برنامه، شما می‌بینید که ثبات AX با 5006 در انتها مقدار دارد نه 0650. خطا چیست و چگونه می‌توان آن را تصحیح کرد؟

ج) دستورات صحیح را بنویسید، حال باید از اولین دستور مجدداً اجرا کنید. کدام فرامین DEBUG لازم است.

۳-۶. دستورات زبان ماشین زیر را در نظر بگیرید :

B0 2A D0 E0 B3 12 F6 E3 90

این برنامه چنین اجرا می‌شود:

مقدار مبنای شانزده 2A را به ثبات AL منتقل می‌کند.

مقدار AL را یک بیت به چپ شیفت می‌دهد. (حاصل ۵۴ است).

مقدار مبنای شانزده 12 را به ثبات BL منتقل می‌کند.

AL را در BL ضرب کنید.

از فرمان E در DEBUG برای ورود این برنامه با شروع از CS:100 استفاده کنید. بخاطر بسپارید که این مقادیر در مبنای شانزده هستند. بعد از ورود برنامه فرمان D CS:100 را برای مشاهده آن اجرا کنید و سپس فرمان R و به مقدار لازم فرمان T را برای پیگیری برنامه تا رسیدن به NOP بزنید. حاصل انتهایی در AX چیست؟

۳-۷. از DEBUG برای ورود برنامه زبان ماشین زیر استفاده کنید:

کد ماشین (در 100H) : A0 00 02 D0 E0 F6 26 01 02 A3 02 02 90 A0

داده‌ها (در 200H): 00 00 12 2A

این برنامه چنین اجرا می‌شود:

محتویات یک بایت در DS : 02 00 (2A) به ثبات AL منتقل می‌شود.

محتویات AL یک بیت به چپ شیفت داده می‌شود. (حاصل 54 می‌شود)

A در یک بایت محتویات DS : 0201(12) ضرب می‌شود.

حاصل از AX به کلمه‌ای با شروع از DS : 0202 منتقل می‌شود.

بعد از ورود برنامه، فرمان D را برای دیدن داده‌ها و کد تایپ کنید. سپس فرمان R و فرمان T به مقداری که لازم است تا به فرمان NOP برسید وارد کنید. در اینجا، AX باید حاوی حاصل 05E8H باشد. یک بار دیگر فرمان

D DS: 0200 را وارد کنید و توجه کنید که حاصل ذخیره شده در DS: 0202 مقدار E80SH باید باشد.

۸-۳. برای پرسش ۷-۳، فرامینی کد کنید که برنامه را بر روی دیسک با نام HEXMULT.COM بنویسد.

۹-۳. از فرمان A از DEBUG برای ورود دستورات زیر استفاده کنید :

```
MOV  CX,3B
ADD  CX,1C
SHL  CX,01
SUB  CX,36
NOP
```

این دستورات را unassemble کنید و اجرای آن راتا NOP پیگیری کنید و مقدار BX را بعداز هردستورچک کنید.

۱۰-۳. منظور از دستور INP چیست؟

۱۱-۳. از DEBUG برای ایجاد و اجرای یک برنامه کد عبارت "Out to lunch" را نمایش دهد استفاده کنید از A 100 برای ورود دستورات و A 110 برای عبارت (علامت $ را به یاد بسپارید) استفاده کنید.

راهنمایی : مثال بخش "استفاده از INT برای نمایش" را ببینید.

۱۲-۳. با استفاده از DEBUG برنامه‌ای ایجاد و اجرا کنید که ۳ کاراکتر از صفحه کلید بپذیرد و سپس آنها را نمایش دهد.

الف) با A 100 آغاز کنید.

ب) از INT 16 برای پذیرش کاراکتر در AL استفاده کنید و آن را به موقعیت [120] منتقل کنید.

ج) با استفاده از دومین INT 16 یک کاراکتر دیگر را در AL قرار داده و آن را به موقعیت [12] منتقل کنید.

د) از یک INT 16H دیگر برای پذیرش سومین کاراکتر در AL استفاده کنید و آن را به موقعیت [122] منتقل کنید.

ه) حال با استفاده از INT 21 کاراکترها را نمایش دهید.

و) در انتها از فرمان '$' 123 E برای تعریف یک '$' در انتهای سه کاراکتر ذخیره شده استفاده کنید. راهنمایی: مثال بخش N استفاده از INT برای ورودی صفحه کلید را ببینید (اما شما نیازی به استفاده از دستور JMP در این مثال ندارید).

فصل ۴

موارد ضروری برای کدنویسی در زبان اسمبلی

هدف: بیان مطالبی حاوی موارد ضروری پایه برای کد نویسی در برنامه‌های زبان اسمبلی و تعریف عناصر داده .

مقدمه

در فصل ۳، شما نحوه استفاده از DEBUG برای ورود و اجرای برنامه‌های زبان ماشین را فرا گرفتید. بی‌شک می‌دانید که چقدر مشکل است کد ماشین را، حتّی برای یک برنامه کوچک از حالت رمز در آورد.

بطور حتم هیچ کس مرتباً به زبان ماشین، بجز برای برنامه‌های کوچک کد نویسی نمی‌کند. شما از فرمان A در DEBUG برای ورود برنامه منبع اسمبلی کوچک استفاده می‌کنید و بی‌تردید توجه کردید که بسیار ساده‌تر از فهم کد ماشین است. استفاده از فرمان A راحت‌تر است زیرا در این فصل برنامه بزرگتر را توسعه می‌دهید و نیاز به قابلیت بیشتری در مستند سازی و اصلاح آنها دارید.

شما برنامه‌های اسمبلی را بر طبق مجموعه‌ای از قوانین دقیق، با استفاده از یک ویرایشگر یا پردازشگر کلمه در قالب یک فایل به کامپیوتر وارد کرده آنگاه از یک برنامه مترجم اسمبلی برای خواندن فایل و تبدیل آن به کد ماشین استفاده می‌کنید. در این فصل، ما موارد ضروری بنیادین برای توسعه برنامه‌های اسمبلی را توضیح می‌دهیم. نحوه استفاده از توضیحات، فرمت کد نویسی معمولی، پیش پردازنده‌هایی برای کنترل لیست برنامه اسمبل شده و موارد ضروری برای تعریف سگمنت و زیر روال‌ها. همچنین این فصل سازماندهی عمومی یک برنامه، شامل مقدار دهی اولیه، برنامه و خاتمه اجرای آن را در بر دارد. در انتها، موارد ضروری برای تعریف عناصر داده را توضیح می‌دهیم.

اسمبلرها و کامپایلرها

دو کلاس اصلی زبانهای برنامه نویسی، ***سطح بالا*** و ***سطح پایین*** می‌باشد. برنامه نویسان که به یک زبان سطح بالا مانند C یا BASIC برنامه می‌نویسند از فرامینی قدرتمندی استفاده می‌کنند که هر یک از آنها ممکن است دستورات زبان ماشین زیادی تولید کند. برنامه نویسانی که به زبان سطح پایین اسمبلی برنامه می‌نویسند ، به عبارت دیگر، دستورات سمبولیک کد می‌کنند، هر یک از آنها یک دستور زبان ماشین تولید می‌کند. علیرغم این حقیقت که کد نویسی در یک زبان سطح بالا سودمندتر است، برخی از مزایای کد نویسی به زبان اسمبلی چنین است.

- فراهم سازی کنترل بیشتر بر دستیابی به موارد ضروری سخت‌افزار.
- امکان تولید، ماجول‌های فشرده‌تر و کوچک‌تر.
- اجرای سریع‌تر برنامه‌ها.

یک رویه عملی ترکیب مزایایی هر دو سطح برنامه نویسی است. کد نویسی قسمت عمده یک پروژه با زبان سطح

بالا و کد نویسی ماجولهای حساس و بحرانی (که تاخیر کمتری ایجاد می‌کنند) با زبان اسمبلی.

علیرغم زبانی که برای برنامه نویسی استفاده می‌کنید، برنامه‌هایی که به این زبان نوشته می‌شوند باید برای ماشین به گونه‌ای قابل اجرا ترجمه شوند. یک زبان سطح بالا از برنامه کامپایلر برای ترجمه کد مبدأ به کد ماشین (در اصطلاح تکنیکی، کد مقصد) استفاده می‌کنند. یک زبان سطح پایین با استفاده از اسمبلر چنین ترجمه‌ای را انجام می‌دهد. برنامه الحاقگر (Linker) برای هر دو سطح پایین و بالا پردازش را با تبدیل کد مقصد به زبان ماشین اجرایی تکمیل می‌کند.

توضیحات برنامه

با استفاده از توضیحات یک برنامه، موارد ابهام را می‌توان بهبود بخشید، بخصوص در زبان اسمبلی که منظور از مجموعه‌ای از دستورات اغلب واضح نیست. به عنوان مثال، کاملاً مشهود است که دستور MOV AH,10H مقدار 10H را به ثبات AH منتقل می‌کند، اما دلیل انجام این دستور واضح نیست. یک توضیح با (;) آغاز می‌شود و در جایی که آن را قرار دهید اسمبلر تمام کاراکترهای سمت راست این خط را توضیح فرض می‌کند. یک توضیح می‌تواند شامل هر کاراکتر قابل چاپ، شامل یک فاصله نیز باشد.

یک توضیح می‌تواند در یک خط جدا باشد، مثل این : calcutate productivity ratio ;

و یا در همان خط پس از یک دستور، مثل این : ADD AX,BY ; Accumulate total quantity

چون یک توضیح فقط در لیست برنامه مبدأ اسمبل شده ظاهر می‌شود و کد ماشین تولید نمی‌کند، هر چقدر که مایلید می‌توانید خط توضیح اضافه کنید بدون آنکه براندازه برنامه اسمبل شده یا اجرای آن اثری بگذارد. در این کتاب هم دستورات اسمبلی با حروف بزرگ و همه توضیحات با حروف کوچک نوشته شده‌اند، این رسم فقط برای خواناتر شدن برنامه است. در حالت تکنیکی ، شما در استفاده از حروف کوچک و بزرگ برای دستورات و توضیحات مختارید. روش دیگر برای فراهم‌سازی توضیحات با استفاده از پیش‌پردازنده COMMERT که در فصل ۲۷، توضیح داده‌شده می‌باشد.

کلمات رزرو شده

نامهایی خاص در زبان اسمبلی برای منظورهای خودش ***تعیین*** شده‌اند. که فقط تحت شرایط خاص استفاده می‌شود. بطور خلاصه کلمات رزروی شامل :

- ***دستورات***، مانند MOV و ADD، که عملیات قابل انجام توسط کامپیوتر هستند.
- ***پیش پردازنده‌ها***، مانند END یا SEGMENT، که اطلاعاتی برای اسمبلر فراهم می‌سازند.
- ***عملگرها***، مانند FAR و SIZE که شما در عبارات از آن استفاده می‌کنید.
- ***سمبولهای از پیش تعریف شده***، مانند Data@ و Model@ که اطلاعات را به برنامه شما برمی گردانند. استفاده نادرست از کلمات رزروی سبب‌می‌شود تا اسمبلریک پیغام خطاتولیدکند. ضمیمه C لیستی از کلمات رزروشده‌است.

شناسه‌ها

یک ***شناسه*** (یا سمبول) نامی است که شما برای یک عنصر در برنامه جهت ارجاع فراهم می‌کنید. دو نوع شناسه نام و برچسب هستند:

۱. ***نام*** به آدرس یک عنصر داده مانند COUNTER در زیر اشاره می‌کند : COUNTER DB 0

۲. برچسب که به آدرس یک دستور زیر روال، یا قطعه مانند MATN در عبارت زیر اشاره می‌کند:

MAIN PROC FAR

قانون یکسانی برای فراهم سازی نام‌ها و برچسب‌ها وجود دارد. یک شناسه از کاراکترهای زیر می‌تواند استفاده کند.

نوع : کاراکترهای مجاز

حروف الفبا: A تا Z و a تا z

ارقام: ۰ تا ۹ (اولین کاراکتر نباید باشد)

کاراکترهای خاص: علامت سئوال (؟) خط فاصله (-)، دلار ($)و@ و نقطه (.) (حرف اول نباید باشند.)

اولین کاراکتر یک شناسه باید یک حرف الفبا یا کاراکتر خاص، بجز نقطه، باشد. از آنجائیکه اسمبلر از برخی از کلمات خاص که با سمبل @ آغاز می‌شوند استفاده می‌کند شما باید در استفاده از آنها برای تعاریف خودتان اجتناب نمائید.

بطور پیش فرض، اسمبلر با حروف بزرگ و کوچک یکسان رفتار می‌کند. (ضمیمه D را ببینید که فرمان خطی برای مجبور کردن اسمبلر برای حساسیت به حروف را دارد). بیشترین طول برای یک شناسه ۳۱ کاراکتر برای MASM6.0، ۲۴۷ می‌باشد. مثالهایی از نامهای معتبر چنین است. QTY250 ، TOTAL و $P50 نام ثباتها مانند AX ، BX و DS برای ارجاع به همان ثباتها است.

در این دستور مانند زیر: ADD CX,BX

اسمبلی بطور اتوماتیک می‌دانند که CX و BX به ثباتها اشاره دارد. اما در یک دستور مانند :

MOV REGSAVE,CX

اسمبلر می‌تواند نام REGSAVE فقط اگر به عنوان یک عنصر داده تعریف شده در برنامه باشد، تشخیص بدهد.

دستورات

یک برنامه اسمبلی شامل یک مجموعه از دستورات است. دو نوع از انواع دستورات چنین هستند:

۱- دستورات مانند MOV و ADD که اسمبلر آنها را به کد مقصد تبدیل می‌کند.

۲- پیش پردازنده، که به اسمبلر برای اجرای عملی ویژه پیغام می‌دهد، مانند تعریف یک عنصر داده .

در این جا شکل عمومی یک دستور ذکر شده که کروشه بر قسمت اختیاری دلالت دارد.

[توضیحات]	[عملوند]	دستورالعمل	[شناسه]

یک شناسه (اگر باشد). دستورالعمل و عملوند (اگر باشد) با یک فاصله یا کاراکتر Tab جدا می‌شوند. ۱۳۲ کاراکتر، حداکثر در یک خط می‌تواند وجود داشته باشد (در MASM 6.0 تا 512)، اگر بیشتر برنامه نویسان تا 80 ستون پیش می‌روند چون حداکثر تعداد کاراکتر تطبیق شده در یک صفحه نمایش ۸۰ می‌باشد.

دو مثال از دستورات چنین هستند :

توضیح	عملوند	دستورالعمل	شناسه	
نام، دستورالعمل، عملوند	1	DB	COUNT	پیش پردازنده:
دستورالعمل، عملوند	AX,0	MOV	P30	دستور:

شناسه، دستورالعمل و عملوند از هر ستونی می‌تواند آغاز شود. اما شروع از ستونهایی مشابه برای این ورودیها برنامه را خواناتر می‌سازد. در ضمن، همه برنامه‌های ویرایشگری علامت Tab در هر ۸ موقعیت برای فاصله گذاری بین فیلدها فراهم می‌کنند.

همانطور که قبلاً توضیح داده شد تحت عنوان "شناسه‌ها" اصطلاح ***نام*** برای نام عنصرهای تعریف شده با پیش‌پردازنده است، در حالیکه اصطلاح Lable به نام یک دستور، که از آن بعد می‌توانیم استفاده کنیم فراهم شده است. عملیات، که باید کد نویسی شود، اغلب برای تعریف ناحیه داده‌ها و کد نویسی دستورات، استفاده می‌شود. برای یک عنصر داده یک عملیات مانند DB یا DW یک فیلد، ناحیه کاری، یا یک مقدار ثابت را تعریف می‌کند. برای یک دستور، یک عملیات مانند MOV یا ADD بر انجام یک فعالیت دلالت دارد.

عملوند (اگر باشد) اطلاعات برای عملیات فراهم می‌سازد. برای یک عنصر داده، عملوند مقدار اولیه را تعریف می‌کند. برای مثال، در تعریف زیر از عنصر داده با نام COUNTER ، عملیات DB به معنی تعریف بایت است و محتویات عملوند با مقدار صفر مقدار دهی می‌شود.

توضیح	عملوند	عملیات	نام
تعریف بایت با مقدار (DB);	0	DB	COUNTER

برای یک دستور، یک عملوند، بر محل فعالیت دلالت دارد. عملوند یک دستور می‌تواند یک، دو یا حتی بدون ورودی باشد. اینجا سه مثال آورده شده است.

توضیح	عملوند	عملیات	عملوندها
بازگشت ;	RET	هیچ	
ثبات BX را یکی اضافه کن ;	BX	INC	یک
جمع مقدار ۲۵ با CX ;	CX,25	ADD	دو

پیش پردازنده‌ها

زبان اسمبلی جملاتی دارد که شما را در کنترل روش اسمبل شدن و لیست‌گیری برنامه توانا می‌سازد. این جملات پیش پردازنده نامیده می‌شوند و فقط طی اسمبل شدن برنامه فعالیت می‌کنند و کد ماشین قابل اجرایی تولید نمی‌کنند. اغلب پیش پردازنده‌های معمولی در بخش بعد توضیح داده خواهد شد.

فصل ۲۷ همه پیش پردازنده‌ها را به تفصیل توضیح می‌دهد که شما از آن به عنوان مرجع می‌توانید استفاده کنید.

پیش پردازنده‌های PAGE و TITLE

پیش پردازنده‌های PAGE و TITLE در کنترل شکل لیست‌گیری یک برنامه اسمبل شده به شما کمک می‌کنند. این تنها هدف آنهاست و هیچ تأثیری بر اجرای بقیه برنامه نخواهند داشت.

PAGE در شروع یک برنامه، پیش پردازنده PAGE حداکثر تعداد خط در صفحه لیست و حداکثر تعداد کاراکتر در خط را تعیین می‌کند. شکل کلی آن چنین است.

```
PAGE [Length] [,Width]
```

برای مثال پیش پردازنده PAGE 60,132 در هر صفحه ۶۰ خط ۱۳۲ کاراکتری فراهم می‌سازد. در یک نوع اسمبلر، تعداد خط صفحه از ۱۰ تا ۲۵۵ می‌تواند باشد و تعداد کاراکتر در هر خط از ۶۰ تا۱۳۲ می‌تواند باشد. حذف جمله PAGE سبب می‌شود اسمبلر مقدار PAGE 50,80 در نظر بگیرد.

فرض کنید یک برنامه حداکثر تعداد خط را برای PAGE، ۶۰ تعریف می‌کند. وقتی اسمبلر برنامه اسمبل شده را چاپ می‌کندو ۶۰ خط را لیست کرد، به طور خودکار به شروع صفحه بعدی می‌رود و شمارشگر صفحه را یکی می‌افزاید.

همچنین ممکن است بخواهید در یک لیست برنامه در خط خاصی در صفحه مثلاً در انتهای یک جمله به صفحه بعد بروید. در خط مورد نظر، کد PAGE را بدون هیچ عملوند قرار دهید وقتی به PAGE برسید اسمبلر به شروع صفحه بعد در لیست می‌رود.

TITLE. از پیش پردازنده TITLE برای ایجاد یک تیتر برنامه در خط دوم از هر صفحه لیست برنامه می‌توانید استفاده کنید. ممکن است TITLE را یک بار کد کنید، فقط در شروع برنامه شکل کلی آن چنین است:

```
TITLE text [توضیح]
```

برای عملوند متن (text) تکنیک توصیه شده استفاده از نام برنامه که با آن بر روی دیسک قرار می‌گیرد می‌باشد، برای

مثال، اگر برنامه را ASMSORT نام گذاری کردید، می‌توانید بعد از عنوان تا ۶۰ کاراکتر، عملکرد برنامه را مقابل نام تشریح کنید (در اینجا نیازی به استفاده از ; نیست).

```
TITLE ASMSORT Assebmbly program to sort customer names
```

پیش پردازنده سگمنت

همانطور که در فصل ۲ توضیح داده شد، یک برنامه اسمبلی در قالب EXE. شامل یک یا چند سگمنت می‌باشد. سگمنت پشته، محل ذخیره سازی پشته را تعریف می‌کند، سگمنت داده عناصر داده را تعریف می‌کند و سگمنت کد برای کد اجرایی فراهم شده است. پیش پردازنده‌ای که سگمنت را تعریف می‌کند، SEGMENT و ENDS می‌باشد، که قالب زیر را دارد :

NAME	OPERATION	OPERAND	COMMENT
name	SEGMENT	[options]	;Begin segment
	.		
	.		
	.		
name	ENDS		;End segment

جمله SEGMENT شروع یک سگمنت را تعریف می‌کند. نام سگمنت باید، منحصر به فرد و مطابق با قواعد نامگذاری اسمبلر باشد. جمله ENDS بر انتهای سگمنت دلالت دارد و شامل همان نامی است که در سگمنت SEGMENT گذاشته شد. حداکثر طول یک سگمنت در حالت حقیقی ۶۴K است. عملوند یک جمله SEGMENT ممکن است سه نوع گزینه داشته باشد. مسیر، ترکیب و نوع که، در قالب زیر کد می‌شود:

```
name  SEGMENT  align  combine  'class'
```

نوع مسیر. ورودی ***align*** بر مرزی که سگمنت از آن آغاز می‌شود دلالت دارد. برای نمونه ،PARA، جمله‌ای است که بر روی مرز پاراگراف قرار می‌دهد، بنابراین آدرس شروع اغلب بر ۱۶ یا 10H قابل تقسیم است. حذف عملوند aling سبب می‌شود اسمبلر مقدار PARA را در نظر بگیرید.

نوع ترکیب. ورودی ***Combine*** بر اینکه آیا سگمنت با دیگر سگمنت‌ها در زمان اسمبل شده پیوند برقرار می‌کند، دلالت دارد. (بعداً تحت عنوان پیوند زدن برنامه بحث خواهد شد). انواع ترکیب PUBLIC ، COMMON ، STACK و عبارت AT می‌باشد. برای مثال، غالباً سگمنت پشته چنین تعریف می‌شود:

```
name SEGMENT PARA STACK
```

وقتی از PUBLIC و COMMON استفاده کنید که می‌خواهید برنامه‌های اسمبل شده مجزا را در زمان پیوند زدن ترکیب کنید. در غیر اینصورت، وقتی یک برنامه با برنامه‌های دیگر ترکیب نمی‌شود، می‌توانید این قسمت را حذف کنید یا کد NONE قرار دهید.

انواع Class. ورودی ***Class*** در بین دو آپاستروف که برای گروه سگمنت مرتبط در هنگام پیوند زدن استفاده می‌شود این کتاب از کلاس 'code' برای سگمنت کد (توسط مایکرو سافت توصیه شده است) و 'data' برای سگمنت داده و 'stack' برای سگمنت پشته استفاده می‌کند. مثال بعد یک سگمنت پشته با مسیر (PARA)، ترکیب (STACK) و کلاس نوع ('Stack') تعریف می‌کند.

```
name SEGMENT PARA STACK 'Stack'
```

```
1           page    60,132
2  TITLE    A04ASM1 Skeleton of an .EXE Program
3  ; ---------------------------------------------
4  STACKSG  SEGMENT PARA STACK 'Stack'
5           ...
6  STACKSG  ENDS
7  ; ---------------------------------------------
8  DATASG   SEGMENT PARA 'Data'
9           ...
10 DATASG   ENDS
11 ; ---------------------------------------------
12 CODESG   SEGMENT PARA 'Code'
13 MAIN     PROC    FAR
14          ASSUME  SS:STACKSG,DS:DATASG,CS:CODESG
15          MOV     AX,DATASG          ;Set address of data
16          MOV     DS,AX              ;  segment in DS
17          ...
18          MOV     AX,4C00H           ;End processing
19          INT     21H
20 MAIN     ENDP                       ;End of procedure
21 CODESG   ENDS                       ;End of segment
22          END     MAIN               ;End of program
```

شکل ۱-۴ اسکلت یک برنامه *.EXE*.

پیش پردازنده PROC

سگمنت کد با دستور PROC تعریف شده و حاوی کد اجرایی برنامه می‌باشد این کد می‌تواند شامل یک یا چند زیر روال باشد. سگمنتی که حاوی فقط یک زیر روال است چنین خواهد بود:

NAME	OPERATION	OPERAND	COMMENT
segname	SEGMENT	PARA	
procname	PROC	FAR	;One
	.		;procedure
	.		;within
	.		;the code
procname	ENDP		;segment
segname	ENDS		

نام زیر روال باید منحصر بفرد بوده و بر طبق نامگذاری قراردادی در اسمبلر باشد. عملوند FAR در این حالت با اجرای برنامه مرتبط است، وقتی می‌خواهید یک برنامه را اجرا کنید، بارگذار برنامه از این زیر روال به عنوان نقطه شروع برای اولین دستور اجرایی استفاده می‌کند.

پیش پردازنده ENDP بر انتهای یک زیر روال دلالت دارد و حاوی همان نامی است که در جمله سگمنت باشد، ENDP انتهای یک زیر روال را قبل از ENDS در انتهای سگمنت مشخص می‌سازد. سگمنت کد می‌تواند حاوی هر تعداد از زیر روال که به عنوان زیر برنامه استفاده می‌شود باشد، که هر یک مجموعه جملات PROC و ENDP خودشان را دارند. هر PROC اضافی همیشه با (یا بصورت پیش فرض) عملوند NEAR کد می‌شوند، که در فصل ۷ توضیح داده می‌شود.

جزئی از برنامه در شکل ۱-۴ جمله SEGMENT با انتخاب‌های مختلف را مشخص می‌کند.

پیش پردازنده ASSUME

یک برنامه EXE. از ثبات SS برای آدرس دهی پشته، ثبات DS برای آدرس دهی سگمنت داده و ثبات CS برای

آدرس دهی سگمنت کد استفاده می‌کند. بدین منظور، باید هدف از هر سگمنت را در برنامه برای اسمبلر مشخص نمود. پیش پردازنده لازم ASSUME است ، که در سگمنت کد ، چنین کد می‌شود.

OPERATION	OPERAND
ASSUME	SS:stackname,DS:datasegname,CS:codesegname, . . .

SS : نام پشته به معنی آن است که اسمبلر نام سگمنت پشته را با ثبات SS مرتبط سازد و به طور مشابه برای بقیه عملوندهای نشان داده شده نیز چنین است. عملوندها به هر ترتیبی می‌توانند باشند. ASSUME ممکن است یک ورودی برای ES نیز داشته باشد، مثل این ES : datasegname اگر برنامه شما از ثبات ES استفاده نکند، می‌توانید این ارجاع را حذف کنید یا بدین شکل کد نمایید: ES, NOTHING : (از MASM 6.0 به بعد اسمبلر بطور خودکار یک ASSUME برای سگمنت کد تولید می‌کند.)

مانند بقیه پیش پردازنده‌ها ASSUME نیز یک پیغام اسمبلر برای تبدیل کد سمبولیک به کد ماشین می‌دهد، ممکن است دستوراتی را کد کنید که بطور فیزیکی آدرس را در ثبات سگمنت در زمان اجرا بارگذاری کند.

پیش پردازنده END

همانطور که گفته شد، پیش پردازنده ENDS یک سگمنت را خاتمه می‌دهد و پیش پردازنده ENDP انتهای یک زیر روال را مشخص می‌کند. یک پیش پردازنده ENP برنامه ورودی را خاتمه می‌دهد و به عنوان آخرین دستور ظاهر می‌شود. شکل عمومی آن چنین است:

عملوند	عملیات
[PROG name]	END

اگر قرار باشد برنامه اجرا نشود، آنگاه جای عملوند خالی می‌ماند، برای مثال ممکن است بخواهید فقط تعریف داده‌ها را اسمبل کنید، یا بخواهید برنامه را با ماجولهای دیگر پیوند بزنید. در اغلب برنامه‌ها، در قسمت عملوند فقط نام اولین روالی که در قسمت PROC به صورت FAR تعریف شده است قرار می‌گیرد.

دستوراتی برای مقدار دهی یک برنامه

دو نوع اصلی از برنامه‌های اجرایی EXE. و COM. هستند. ما موارد ضروری برای برنامه‌های EXE. را ابتدا توضیح می‌دهیم و بررسی برنامه‌های COM. را در فصل ۷ انجام می‌دهیم. شکل ۱-۴ اسکلت برنامه‌های EXE. را با سگمنت پشته، داده و کد نشان می‌دهد. در زیر دستورات برنامه را بررسی می‌کنیم.

خط	توضیح
۱	پیش پردازنده PAGE برای ایجاد ۶۰ خط ۱۳۲ ستونی در هر صفحه.
۲	پیش پردازنده TITLE که نام برنامه را با عنوان A04ASMI مشخص می‌کند.
۳	خطوط ۳ و ۷ و ۱۱ توضیحاتی است که سه سگمنت تعریف شده واضح می‌سازد.
۴-۶	این جملات سگمنت پشته، STACKSG را تعریف می‌کند (اما در این مثال محتویات آن نیست)
۸-۱۵	این جملات سگمنت داده، DATASG را تعریف می‌کند (ولی محتویات آن را تعریف نمی‌کند)
۱۲-۲۱	این جملات سگمنت کد CODESG را تعریف می‌کند.
۱۳-۲۰	این جملات تنها زیر روال سگمنت کد را که MAIN نامیده می‌شود تعریف می‌کند. این زیر روال مقدار دهی‌های اولیه عمومی و موارد ضروری برای خروج را در یک برنامه EXE. را مشخص می‌کند. دو مورد ضروری برای مقدار دهی اولیه بشرح زیرند: (۱) آگاهی دادن به اسمبلر که کدام ثبات سگمنت با

سگمنت مرتبط است و (۲) بارگذاری DS با آدرس سگمنت داده.

۱۴ پیش پردازنده ASSUME برای آگاه کردن اسمبلر از ارتباط سگمنت خاص با ثباتهای سگمنت خاص که در این حالت STACKSG با SS و DATASG با DS و CODESG با CS مرتبط است .

```
ASSUME SS:STACKSG,DS:DATASG,CS:CODESG
```

با مرتبط ساختن سگمنت‌ها با ثباتهای سگمنت، اسمبلر می‌تواند آدرس افست برای عناصر پشته را تعیین کند، برای عناصر داده در سگمنت داده و برای دستورات در سگمنت کد مشخص سازد. برای مثال، هر دستور ماشین در سگمنت کد طول خاصی دارد اولین دستور به زبان ماشین باید در افست 0 باشد و اگر ۲ بایت طول دارد، دومین دستور باید در افست ۲ باشد و الی آخر.

۱۵و۱۶ دو دستور که آدرس سگمنت داده را در ثبات DS مقدار دهی می‌کنند:

```
MOV   AX,DATASG      ;  گرفتن آدرس سگمنت داده
MOV   DS,AX          ;  ذخیره سازی آدرس در DS
```

اولین MOV آدرس سگمنت داده را در ثبات AX بارگزاری می‌کند و دومین دستور MOV آدرس را از AX به DS منتقل می‌کند. هر دو دستور لازمند چرا که دستوری نداریم تا مستقیماً یک داده را از حافظه به ثبات سگمنت منتقل کند، شما باید آدرس را از یک ثبات دیگر به ثبات سگمنت منتقل کنید. بنابراین جمله MOV AX,DATASG کاملاً قانونی است. فصل ۵ در مورد مقدار دهی اولیه ثباتهای سگمنت توضیح بیشتری خواهد داد.

۱۹و۱۸ این دو دستور اجرای برنامه را خاتمه می‌دهند و به سیستم عامل برمی‌گردند. بخش بعدی در مورد آنها بیشتر توضیح می‌دهد.

۲۲ جمله END به اسمبلر خاتمه برنامه را اعلام می‌کند و عملوند MAIN نقطه شروع اجرای برنامه را فراهم می‌سازد. MAIN می‌تواند هر نام قابل قبول دیگری برای اسمبلر باشد.

ترتیب تعریف سگمنت‌ها اغلب بی‌اهمیت است. شکل ۱-۴ آنها را چنین تعریف می‌کند.

```
STACKSG    SEGMENT    PARA    STACK    'Stack'
DATASG     SEGMENT    PARA    'Data'
CODESG     SEOMENT    PARA    'Code'
```

توجه کنید که برنامهٔ شکل **به زبان سمبولیک** کد نویسی شده است. برای اجرای آن، باید از یک اسمبلر و لینکر برای تبدیل آن به کد ماشین قابل اجرا مانند یک برنامه EXE. استفاده کنید.

همانطور که در فصل ۲ توضیح داده شد، وقتی بارگذار برنامه یک برنامه EXE. را از دیسک می‌خواند و در حافظه جهت اجرا قرار می‌دهد، ۲۵۶ بایت PSP(100H) در شروع مرز پاراگراف می‌سازد که در دسترس حافظه داخلی است و برنامه بلافاصله بعد از آن ذخیره می‌باشد. سپس بارگذار

- آدرس سگمنت کد را در CS بارگذاری می‌کند،
- آدرس سگمنت پشته را در SS بارگذاری می‌کند و
- آدرس PSP را در DS و ES بارگذاری می‌کند.

بارگذار برنامه ثباتهای CS:IP و SS:SP را مقدار دهی می‌کند اما ثباتهای ES و DS را مقداردهی نمی‌کند. ولی، برنامه شما بطور معمول به آدرس سگمنت داده در DS نیاز دارد (و اغلب ES نیز چنین است). به این ترتیب، باید DS را با آدرس سگمنت داده همانطور که در شکل ۱-۴ نشان داده شد، مقداردهی کنید. حال، حتی اگر این مقدار دهی در این نقطه واضح نباشد، همهٔ برنامه‌های EXE. مراحل مقداردهی یکسانی دارند که هر زمان یک برنامه اسمبلی را کد نویسی می‌کنید، می‌توانید آن را کپی نمائید.

دستوراتی برای خاتمه دادن به اجرای برنامه

INT 21H یک وقفه عمومی DOS است که از کد تابعی ثبات AH جهت مشخص ساختن فعالیتی که باید انجام شود استفاده می‌کند. تعداد زیادی از توابع INT 21H شامل ورودی صفحه کلید، دستکاری صفحه، ورودی خروجی دیسک و خروجی چاپگر است. تابعی که در اینجا با آن کار داریم 4CH است، که INT 21H تشخیص می‌دهد یک تقاضا برای خاتمه اجرای برنامه است. از این عملیات برای بازگرداندن یک کد در AL برای تست کردن مکرر و مرتب در یک فایل دسته‌ای می‌توانید استفاده کنید (بر طبق جمله IF ERROR LEVEL)، بدین شکل:

```
MOV   AH,4CH        ;  تقاضای خاتمه پردازش
MOV   AL,roetcode   ;  کد بازگشتی اختیاری
INT   21H           ;  فراخوانی سرویس وقفه
```

کد بازگشت برای تکمیل معمولی برنامه همیشه صفر است. می‌توانید دو دستور MOV را با یک جمله کد نویسی کنید (همانطور که در شکل ۱-۴ می‌بینید)

```
MOV   AX,4C00H ; درخواست خروجی عادی
```

تابع 4CH از INT 21H جانشین INT 20H می‌شود و تابع 00H از INT 21H برای خاتمه پردازش استفاده می‌شود.

```
            page   60,132
TITLE       A04ASM1 (EXE)  Move and add operations
; ------------------------------------------------
STACKSG     SEGMENT PARA STACK 'Stack'
            DW      32 DUP(0)
STACKSG     ENDS
; ------------------------------------------------
DATASG      SEGMENT PARA 'Data'
FLDD        DW      175
FLDE        DW      150
FLDF        DW      ?
DATASG      ENDS
; ------------------------------------------------
CODESG      SEGMENT PARA 'Code'
MAIN        PROC    FAR
            ASSUME  SS:STACKSG,DS:DATASG,CS:CODESG
            MOV     AX,DATASG       ;Set address of data
            MOV     DS,AX           ;  segment in DS

            MOV     AX,FLDD         ;Move 0175 to AX
            ADD     AX,FLDE         ;Add  0150 to AX
            MOV     FLDF,AX         ;Store sum in FLDF
            MOV     AX,4C00H        ;End processing
            INT     21H
MAIN        ENDP                    ;End of procedure
CODESG      ENDS                    ;End of segment
            END     MAIN            ;End of program
```

شکل ۲-۴ برنامه *EXE*. با سگمنت‌های معمولی

مثالی از یک برنامه مبدأ

شکل ۲-۴ اطلاعات قبل را در یک برنامه مبدأ اسمبلی ساده اما کامل ترکیب می‌کند که دو عنصر داده را با هم در ثبات AX جمع می‌کند. سگمنت‌ها به این روش تعریف شده‌اند :

- STACKSG حاوی یک ورودی، DW (تعریف کلمه) است که ۳۲ کلمه با مقدار صفر را تعریف می‌کند، که فضایی کافی برای برنامه‌های کوچک می‌باشد.
- DATASG سه کلمه FLDD (با مقدار، ۱۷۵)، FLDE (با مقدار ۱۵۰) و FLDF (بدون مقدار) را تعریف می‌کند.

- CODESG حاوی دستورات اجرایی برای برنامه، است اگرچه اولین جمله، ASSUME هیچ کداجرایی تولید نمی‌کند. پیش پردازنده ASSUME این عملیات را اجرا می‌کند :
- تخصیص ثبات SS توسط STACKSG و بنابراین سیستم از آدرس ثبات SS برای آدرسدهی STACKSG استفاده می‌کند.
- تخصیص DATASG به ثبات DS، بنابراین سیستم از آدرس ثبات DS برای آدرسدهی DATASG استفاده می‌کند.
- تخصیص CODESG به ثبات CS، بنابراین سیستم از آدرس ثبات CS برای آدرسدهی CODESG استفاده می‌کند.

وقتی یک برنامه از روی دیسک بر روی حافظه جهت اجرا بارگذاری می‌شود، بارگذار برنامه آدرس واقعی را در ثباتهای SS و CS تنظیم می‌کند، اما همانطورکه نشان داده شد با اولین دو دستور MOV، شما باید ثبات DS (و در صورت ممکن ES) را مقدار دهی کنید.

ما اسمبل کردن، لینک کردن و اجرای این برنامه را در فصل ۵ پیگیری می‌کنیم.

مقدار دهی برای حالت محافظت شده

در حالت محافظت شده تحت پردازشگرهای 80386 و بالاتر، یک برنامه می‌تواند تا ۱۶ مگابایت حافظه را آدرس دهی کند. استفاده از DWORD برای تنظیم سگمنت بر روی آدرس دو کلمه‌ای سرعت دستیابی حافظه برای یک گذرگاه داده ۳۲ بیتی را بالا می‌برد. در مثال بعد، پیش پردازنده 386. به اسمبلر برای پذیرش دستوراتی که در 80386 و ما بعد آن منحصر بفردند آگاهی می‌دهد، استفاده از نوع USE 32 به اسمبلر برای تولید کد صحیح برای حالت، محافظت شده ۳۲ بیتی خبر می‌دهد :

```
.386
segname SEGMENT  DWORD USE32
```

مقدار دهی ثبات سگمنت داده می‌توانید مشابه این باشد، از آنجاکه در این پردازشگر ثبات DS هنوز ۱۶ بیتی است:

```
MOV EAX,DATASG  ; آدرس سگمنت داده را می‌گیرد
MOV DS,AX       ; بارگذاری بخش ۱۶ بیتی
```

دستورات STI ، CLI ، IN و OUT، که در حالت حقیقی در دسترسند، در حالت محافظت شده مجاز نیستند.

پیش پردازندهٔ، سگمنت ساده شده

اسمبلر برخی از تعاریف سگمنت را فراهم می‌سازد. برای استفاده از آنها شما باید مدل حافظه را قبل از تعاریف سگمنت مقدار دهی نمایید. قالب عمومی چنین است :

```
.MODEL memory-model
```

مدل حافظه ممکن‌است TINY، SMALL ، MEDIUM ، COMPACT یا LARGE باشد (مدل‌دیگر HUGE، که در اینجا به آن نیازی نداریم) ضروریات هر مدل بشرح زیر است:

MODEL	NUMBER OF CODE SEGMENTS	NUMBER OF DATA SEGMENTS
TINY	*	*
SMALL	1	1
MEDIUM	More than 1	1
COMPACT	1	More than 1
LARGE	More than 1	More than 1

شما هر کدام از این مدلها را برای یک برنامه بکار ببرید (برنامه‌ای که با برنامه دیگری لینک نشود). مانند TASM4.0 MASM6.0 و مدل TINY بر استفاده از برنامه‌های COM. تاکید دارد که داده‌ها، کد و پشته در یک سگمنت 64 K قرار دارند. مدل SMALL، کد را در یک سگمنت 64K و داده‌ها را در سگمنت 64K دیگر قرار می‌دهد. این مدل برای اغلب مثالهای این‌کتاب مناسب است. پیش پردازنده MODEL. بطور اتوماتیک جملات ASSUME ضروری را تولیدمی‌کند.

قالب‌های عمومی برای پیش پردازنده‌ها که سگمنت پشته، داده و کد را تعریف می‌کنند.

```
.STACK  [اندازه]
.DATA
.CODE   [نام]
```

هر یک از این پیش پردازنده‌ها سبب می‌شود تا اسمبلر جملات SEGMENT لازم و ENDS متعلق به آن را تولید کند. نام‌های سگمنت پیش فرض (که شما تعریف نکرده‌اید) STACK ، DATA و TEXT (برای سگمنت کد) می‌باشند.

همانطور که قالب کد نویسی حکایت می‌کند، می‌توانید نام پیش فرض سگمنت کد را عوض کنید. اندازه پیش فرض پشته ۱۰۲۴ است که می‌توانید آن را تغییر دهید. با استفاده از پیش پردازنده مکان قرارگرفتن سه سگمنت را تعیین کنید. بخاطر بسپارید دستوراتی که استفاده می‌کنید برای تعیین آدرس سگمنت داده در DS چنین است :

```
MOV  AX,@data
MOV  DS,AX
```

شکل ۲-۴ مثالی از استفاده از تعریف مرسوم سگمنت را ارائه می‌دهد. شکل ۳-۴ همان مثال را با استفاده پیش پردازنده‌های سگمنت ساده مثل STACK. ، DATA. و CODE. ارائه می‌دهد در خط چهارم مدل حافظه SMALL و پشته ۶۴ بایت (۳۲ کلمه) تعریف شده است. توجه کنید که اسمبلر جملات SEGMENT و ENDS مرسوم را تعریف نمی‌کند و شما نیز جمله ASSUME را کد نمی‌کنید.

```
        page   60,132
TITLE   A04ASM2 (EXE)  Move and add operations
;-------------------------------------------------------
        .MODEL  SMALL
        .STACK  64                  ;Define stack
        .DATA                       ;Define data
FLDD    DW      175
FLDE    DW      150
FLDF    DW      ?
;-------------------------------------------------------
        .CODE                       ;Define code segment
MAIN    PROC    FAR
        MOV     AX,@data            ;Set address of data
        MOV     DS,AX               ;   segment in DS

        MOV     AX,FLDD             ;Move 0175 to AX
        ADD     AX,FLDE             ;Add  0150 to AX
        MOV     FLDF,AX             ;Store sum in FLDF

        MOV     AX,4C00H            ;End processing
        INT     21H
MAIN    ENDP                        ;End of procedure
        END     MAIN                ;End of program
```

شکل ۳-۴ برنامه منبع *EXE*. با پیش پردازنده‌های سگمنت ساده

پیش پردازنده‌های STARTUP. و EXIT.

MASM 6.0 به منظور ساده سازی شروع و خاتمه برنامه پیش پردازنده‌های STARTUP. و EXIT. را داراست. STARTUP. دستواتی برای مقدار دهی اولیه ثباتهای سگمنت تولید می‌کند، در حالیکه EXIT. دستورات INT 21 با تابع 4CH را جهت خروج از برنامه تولید می‌کند. TASM حالت ایده‌آل از عبارت STARTUPCODE و

EXITCODE استفاده می‌کند. جهت یادگیری زبان اسمبلی تمرینهایی با مجموعهٔ کاملی از دستورات انجام دهید و روشهای کوتاه سازی را به برنامه نویسان با تجربه واگذار کنید.

تعریف داده

همانطور که بحث شده، منظور از سگمنت داده در برنامه‌های EXE. تعریف مقدارهای ثابت، ناحیه کاری و ناحیه ورودی خروجی است. اسمبلر بر اساس مجموعهٔ پیش پردازنده‌هایی که داده را تعریف می‌کنند عناصر داده را با طولهای مختلف تعریف می‌کند. برای مثال، DB یک بایت را تعریف می‌کند وDW یک کلمه را تعریف می‌کند. یک عنصر داده حاوی مقداری *ناشناخته* است (که مقدار دهی نشده باشد) یا ممکن است حاوی یک مقدار ثابت باشد، رشته‌ای از کاراکتر یا مقداری عددی تعریف شود. اینجا فرمت کلی برای تعریف داده‌ها را می‌بینید.

[name]	Dn	expression

نام. برنامه یک عنصر داده را به توسط نام آن بر می‌گرداند. نام یک عنصر اختیاری است و در داخل کروشه قرار دارد. بخش قبلی "جملات" قوانین نامها را ارائه می‌کند.

پیش پردازنده (Dn). پیش پردازنده‌هایی که عناصر داده را تعریف می‌کنند عبارتند از: DB (بایت) ، DW (کلمه) ، DD (دو کلمه) ، DF (کلمه far) ، DQ (۸ کلمه‌ای) و DT (ده کلمه‌ای) هر کدام بر طول عنصر داده‌ای که تعریف می‌کنند دلالت دارند. MASM 6.0 عبارتهای BYTE ، WORD ، DWORD ، FWORD و TWORD را به ترتیب برای این پیش پردازنده‌ها دارا است .

عبارت. عبارت با عملوندهایی که یک مقدار ناشناخته یا یک مقدار ثابت شناخته شده است تعریف می‌شود. برای عنصر ناشناخته عملوند را با یک علامت سئوال مشخص کنید، به این شکل

```
عنصر ناشناخته ; ? DB FLDA
```

در این حالت، وقتی اجرای برنامه آغاز می‌شود، مقدار اولیه FLDA برای شما مشخص نیست. یک تجربه عادی قبل از استفاده از این عنصر انتقال برخی مقادیر در آن است، اما این مقادیر با اندازه تعریف شده باید یکی باشد. شما با استفاده از عملوند یک مقدار ثابت را تعریف می‌کنید بدین شکل:

```
مقدار دهی نام ; 25 DB/BYTE FLDA
```

حال شما آزادید که از مقدار ۲۵ در برنامه استفاده کنید یا حتی می‌توانید این مقدار را تغییر دهید. یک عبارت می‌تواند حاوی چندین مقدار ثابت مجزا با کاما و محدود به طول یک خط باشد، بدین شکل

```
FLDC DB/BXTE 21 , 22 , 23 , 24 , 25 , 26 , ...
```

اسمبلر این مقادیر ثابت را در بایت‌هایی مجاور تعریف می‌کند رجوع به FLDC به اولین بایت مقدار ثابت است یعنی 21 (شمااین بایت را چنین فرض کنید FLDC+0) و رجوع به FLDC+1 به دومین‌بایت است یعنی 22. برای‌مثال دستور:

```
MOV AL,FLDC + 3
```

مقدار 24 (18H) را به ثبات AL منتقل می‌کند. عبارتی که مجوز تکرار یک مقدار ثابت در جمله را می‌دهد به این شکل است :

[نام]	Dn	(عبارت) DUP شماره تکرار

مثال بعدی تکرار را مشخص می‌سازد:

```
DW/WORD   10DUP(?)          ;  ۱۰ کلمه بدون مقدار
DB/BYTE   5DUP(12)          ;  پنج بایت حاوی 0C0C0C0C0C
DB/BXTE   3DUP(5DUP(4))     ;  پانزده تا 4
```

سومین مثال ۵ کپی از رقم ۴ (44444) و سه مرتبه تکرار این مقدار ۱۵ حاصل می‌شود. یک عبارت ممکن است یک رشته کاراکتر یا یک مقدار عددی را تعریف و مقدار دهی نماید.

رشته‌های کاراکتری

رشته‌های کاراکتری برای توصیف داده‌ها مانند نامهای اشخاص و تولید توضیحات مورد استفاده قرار می‌گیرد. رشته‌ها در یک کد تک تعریف می‌شوند مانند 'PC' و یا در داخل کد دوتایی مانند "PC". اسمبلر رشته‌های کاراکتری را مانند کد مقصد به شکل معمولی ASCII بدون علائم نقل قول ذخیره می‌سازد. تحت MASM , DB (بایت) تنها قالبی است که رشته‌های کاراکتری بیشتر از دو کاراکتر با کاراکترهایی در سمت چپ آن به شکل معمولی چپ به راست ذخیره می‌کند (مانند نامها و آدرسها) نتیجتاً DB (یا BYTE) قالب مرسوم برای تعریف داده‌های کاراکتری با هر طولی است.

یک مثال: DB 'Strawberry Jam'

اگر یک رشته حاوی یک علامت نقل قول دوتایی یا یکی است به یکی از دو روش زیر باید تعریف شود:

DB "Crazy sam's CD Emporium" ; علامت نقل قول دوتایی برای رشته

DB 'Crazy sam"s CD Emporium' ; علامت نقل قول تکی برای رشته

ثابتهای عددی

ثابتهای عددی برای تعریف مقادیر محاسباتی و آدرسهای حافظه مورد استفاده قرار می‌گیرد. مقدار ثابت در داخل کد تعریف نمی‌شود. اما بعد از یک **شاخص مبنا** مانند H در مقادیر مبنای شانزده مانند 12H قرار داده می‌شود. برای اغلب پیش پردازنده‌های تعریف داده، اسمبلر اعداد تعریف شده ثابت را به مقدار مبنای شانزده تبدیل می‌کند و بایت‌های تولید شده را در کد مقصد با ترتیب بایت معکوس از راست به چپ ذخیره می‌سازد. چند قالب عددی مختلف در اینجا ذکر شده است .

دودویی. قالب دودویی تعریف ارقام ۰ و ۱ دودویی را اجازه می‌دهد که پس از آن شاخص مبنای B قرار داده می‌شود. استفاده عمومی قالب دودویی در تشخیص مقادیر در دستورات بیتی مانند TEST ، AND ، OR، XOR است .

دهدهی. قالب دهدهی تعریف ارقام ۰ تا ۹ دهدهی را انجام می‌دهد، که پس از آن بطور دلخواه شاخص مبنای D قرار می‌گیرد. مانند ۱۲۵ یا ۱۲۵D. اگر چه اسمبلر مجوز تعریف مقادیر قالب دهدهی برای کدنویسی راحت‌تر در اختیار می‌گذارد در کد obj دودویی آنها را بصورت مبنای شانزده تبدیل می‌کند. برای مثال تعریف مقدار دهدهی ۱۲۵ خواهد شد 7D مبنای شانزده .

مبنای شانزده. قالب مبنای شانزده تعریف ارقام 0 تا F مبنای شانزده را اجازه می‌دهد که پس از آن شاخص مبنای، H قرارداده‌می‌شود. از آنجائیکه اسمبلر هر ارجاعی که با حروف آغازشوند را بعنوان یک نام سمبولیک فرض‌می‌کند، اولین رقم دریک عدد مبنای‌شانزده باید ۰ تا ۹ باشد.مثلاً 3DH یا 0DE8H که‌اسمبلر آنهارا همان 3D و 0DE8 ذخیره‌می‌کند. توجه کنید که دومین عدد مثال بصورت بایت معکوس ذخیره خواهد شد .

از آنجائیکه اسمبلر همه مقادیر عددی را به دودویی تبدیل می‌کند و آنها را با مبنای شانزده نمایش می‌دهد، تعریف 12 دهدهی، C مبنای شانزده و 1100 دودویی همه یک مقدار را تولید می‌کنند: دودویی 00001100 یا 0C مبنای شانزده، بسته به این دارد که مقادیر یک بایت را چگونه ببینید.

از آنجا که حروف D و B که شاخص مبنا هستند، از ارقام مبنای شانزده نیز می‌باشند، سبب ایجاد ابهام خواهند شد. یک راه حل، MASM 6.0 استفاده از T (دهدهی – Ten) و Y (دودویی – Binary) را برای شاخص مبنای اعداد دهدهی و دودویی در نظر گرفته است.

حقیقی. اسمبلر یک مقدار حقیقی داده شده را (ثابت دهدهی یا شانزدهی که شاخص مبنا را داشته باشد) به قالب مفید شناور برای استفاده در یک کمک پردازشگر عددی تبدیل می‌کند. بین کاراکترها و اعداد ثابت تمیز قائل شوید. بعنوان مثال، ثابت کاراکتری که چنین تعریف می‌شود DB '24' دو کاراکتر ASCII تولید می‌کند که در مبنای شانزده چنین بیان می‌شوند 3234H. ثابت عددی که چنین تعریف شده DB 24 یک عدد دودویی تولید می‌کند که در مبنای شانزده چنین بیان می‌شود 18H.

```
                                    Page 60,132
                          TITLE     A04DEFIN (EXE)  Define data directives
                                    .MODEL  SMALL
                                    .DATA
                          ;         DB - Define Byte:
                          ;         ----------------
0000 00                   BYTE1     DB    ?             ;Uninitialized
0001 30                   BYTE2     DB    48            ;Decimal constant
0002 30                   BYTE3     DB    30H           ;Hex constant
0003 7A                   BYTE4     DB    01111010B     ;Binary constant
0004 000C[ 00 ]           BYTE5     DB    12 DUP(0)     ;Twelve zeros
0010 43 6F 6D 70 75 74    BYTE6     DB    'Computer Processors'
     65 72 20 50 72 6F
     63 65 73 73 6F 72
     73                                                 ;Character string
0023 34 37 38 33 35       BYTE7     DB    '47835'       ;Numbers as chars
0028 01 4A 61 6E 02 46    BYTE8     DB    01,'Jan',02,'Feb',03,'Mar'
     65 62 03 4D 61 72
                                                        ;Table of months
                          ;         DW - Define Word:
                          ;         ----------------
0034 FFF0                 WORD1     DW    0FFF0H        ;Hex constant
0036 007A                 WORD2     DW    01111010B     ;Binary constant
0038 0023 R               WORD3     DW    BYTE7         ;Address constant
003A 0002 0004 0006 0007  WORD4     DW    2,4,6,7,9     ;Table of five
     0009                                               ;  constants
0044 0006[ 0000 ]         WORD5     DW    6 DUP(0)      ;Six zeros

                          ;         DD - Define Doubleword:
                          ;         ----------------------
0050 00000000             DWORD1    DD    ?             ;Uninitialized
0054 00007457             DWORD2    DD    29783         ;Decimal value
0058 0000001B 00000038    DWORD3    DD    27, 56        ;Two constants
0060 00000001             DWORD4    DD    BYTE3 - BYTE2 ;Diff betw addresses
0064 00005043             DWORD5    DD    'PC'          ;Character string

                          ;         DQ - Define Quadword:
                          ;         --------------------
0068 0000000000000000     QWORD1    DQ    ?             ;Uninitialized
0070 395E000000000000     QWORD2    DQ    05E39H        ;Hex constant
0078 5774000000000000     QWORD3    DQ    29783         ;Decimal constant

                          ;         DT - Define Tenbytes:
                          ;         --------------------
0080 000000000000000000   TENB1     DT    ?             ;Uninitialized
     00
008A 307829000000000000   TENB2     DT    297830        ;Decimal constant
     00
0094 435000000000000000   TENB3     DT    'PC'          ;Character string
     00
                                    END
```

شکل ۴-۴ تعریف رشته‌های کاراکتری و مقادیر عددی

پیش پردازنده‌هایی برای تعریف داده

پیش پردازنده‌های مرسوم برای تعریف داده‌ها و نام آنها در MASM 6.0 عبارتند از :

DESCRIPTION	CONVENTIONAL DIRECTIVES	MASM 6.0 DIRECTIVES
Byte	DB	BYTE
Word	DW	WORD
Doubleword	DD	DWORD
Farword	DF	FWORD
Quadword	DQ	QWORD
Tenbytes	DT	TBYTE

در این متن از پیش‌پردازنده‌های مرسوم جهت مورد استفاده عموم بدون آن، استفاده می‌شود .

برنامه اسمبل شده شکل ۴-۴ مثالی از پیش‌پردازنده‌هایی که رشته‌های کاراکتری و اعداد ثبات را تعریف می‌کنند ارائه می‌دهد. کد مقصد تولید شده که آن را اجرا می‌کنید، در سمت چپ لیست شده است، توجه کنید که کد مقصد برای مقادیر ناشناخته صفر ظاهر می‌شود. از آنجائیکه برنامه شامل فقط یک سگمنت داده است برای اجرا مناسب نیست.

DB یا BYTE : تعریف بایت

یکی از پیش پردازنده‌هایی که عناصر داده را تعریف می‌کند ـ یکی از پر استفاده‌ترین آنها ـ DB (تعریف بایت) است. یک عبارت عددی DB (یا BYTE) می‌تواند یک یا چند ثابت یک بایتی را تعریف کند. حداکثر یک بایت دو رقم مبنای شانزده است. بزرگترین عدد یک بایتی شانزدهی FF و یا ۲۵۵ است. با سمت چپ‌ترین بیت که بیت علامت باشد، بزرگترین عدد شانزدهی یک بایتی مثبت 7F و همه اعداد بزرگتر 80 تا FF (که بیت علامت یک است) اعداد منفی را بیان می‌کند. در اصطلاح اعداد دهدهی این محدوده 127+ تا 128- می‌باشد. اسمبلر ثابتهای عددی را به کد obj. دودویی تبدیل می‌کند (در مبنای شانزدهی بیان می‌شود). در شکل ۴-۴ ثابتهای عددی BYTE2، BYTE3، BYTE4 و BYTE5 می‌باشند.

عبارت کاراکتری DB حاوی رشته‌ای با هر طول تا انتهای خط ممکن است باشد. برای مثال BYTE7 و BYTE6 را ببینید. در فایل obj. کاراکترهای اسکی هر بایت به صورت چپ به راست قرار می‌گیرد. در این فایل، مقدار 20H بیانگر فضای خالی است.

BYTE8 ترکیبی از ثباتهای رشته‌ای و عددی مناسب برای تعریف یک جدول را نشان می‌دهد.

DW یا WORD : تعریف کلمه

پیش پردازنده DW (یا WORD) عناصر داده‌ای را تعریف می‌کند که یک کلمه (۲ بایت) طول دارند. عبارات عددی DW می‌توانند یک یا چند ثابت یک کلمه‌ای را تعریف کنند. بزرگترین عدد مثبت یک کلمه‌ای عدد 7FFF مبنای شانزده و همه اعداد بزرگتر 8000 تا FFFF (که بیت علامت آنها یک است)، مقادیر منفی را نشان می‌دهند. در مورد اعداد اعشاری، بازهٔ 32767+ تا 32768- تعریف می‌شود.

اسمبلر ثابتهای عددی DW را به کد هدف دودویی (بیان شده با مبنای شانزده) تبدیل می‌کند. اما به ترتیب بایت معکوس ذخیره می‌سازد. به ترتیب مقدار دهدهی تعریف شده با ۱۲۳۴۵ به مقدار شانزدهی و 303 تبدیل و بصورت 3930 ذخیره می‌شود.

در شکل ۴-۴، WORD1 و WORD2 ثابتهای عددی DW را تعریف می‌کنند. WORD3 عملوندهایی برای آدرس تعریف می‌کنند. در این حالت آدرس افست BYTE7. کد هدف تولید شده 0023 (R سمت راست به معنی دوباره جایگزین شده است) و یک تست از شکل نشان می‌دهد که آدرس افست BYTE7 (ستون سمت چپ) 0023 است.

یک عبارت کاراکتری DW، تحت MASM به دو کاراکتر محدود می‌شود. DW برای تعریف رشته‌های کاراکتری محدودیت دارد.

WORD4 جدولی از ۵ ثابت عددی تعریف می‌کند. توجه کنید که طول هر ثابت یک کلمه (۲ بایت) است/

WORD5 جدولی با ۶ مقدار صفر تعریف می‌کند.

DD یا DWORD : تعریف دو کلمه‌ای

پیش پردازنده DD (یا DWORD) عناصری تعریف می‌کند که دو کلمه‌ای هستند (۴ بایت طول دارند). یک عبارت عددی DD ممکن است یک یا چند ثابت تعریف کند، که هر یک حداکثر ۴ بایت (۸ رقم شانزدهی) هستند. بزرگترین عدد شانزدهی مثبت دو کلمه‌ای 7FFF FFFF است و همه اعداد بزرگتر، از 8000000 تا FFFFFFFF (که بیت علامت آنها یک است) مقادیر منفی را بیان می‌کنند. اعداد اعشاری در بازهٔ 2,127,483,647+ تا 2,147,483,648- قرار می‌گیرند.

اسمبلر اعداد ثابت DD را به کد obj. دودویی (بیان شده شانزدهی) تبدیل می‌کند ولی بصورت بایت معکوس آن را ذخیره می‌سازد. در نتیجه، اسمبلر مقدار دهدهی ۱۲۳۴۵۶۷۸ را به 00BC61 4EH تبدیل و بصورت 4E 61 BC00H ذخیره می‌سازد.

در شکل ۴-۴، DWORD2 یک ثابت عددی از نوع DD تعریف می‌کند و DWORD3 دو ثابت عددی تعریف می‌کند. DWORD4 عددی تولید می‌کند که بین دو آدرس تعریف شده تغییر می‌کند ; در این حالت حاصل، طول BYTE2 است.

عبارت کاراکتری DD با MASM به دو کاراکتر محدود می‌شود و کمی شبیه DW خواهد بود. اسمبلر کاراکترها را از راست به صورت ۴ بایت دو کلمه‌ای تنظیم می‌کند همانطور که کد مقصد برای WORD5 نشان داده می‌شود.

DF یا FWORD : تعریف کلمات Far

پیش پردازنده DF (FWORD) یک کلمه Far را با ۱۶ بایت تعریف می‌کند. معمولاً در پردازشگرهای 80386 و ما بعد آن استفاده می‌شود.

DQ یا QWORD : تعریف چهار کلمه

پیش پردازنده DQ (QWORD) عنصری تعریف می‌کند که ۴ کلمه (۸ بایت) طول داشته باشد. یک عبارت عددی DQ ممکن است یک یا چند ثابت را تعریف کند. که هر یک حداکثر ۸ بایت یا ۱۶ رقم شانزدهی باشد. بزرگتر عدد مثبت چهار کلمه‌ای یک 7 و 15 تا F است. عدد شانزدهی 1 و 15 تا 0 مساوی با عدد دهدهی ۹۷، ۸۴۶، ۶۰۶، ۵۰۴، ۹۲۱، ۱۵۲ و ۱ است.

اسمبلر مقادیر عددی DQ و رشته‌های کاراکتری را مانند مقادیر DD و DW دستیابی می‌کند. در شکل ۴-۴، QWORD2 و QWORD3 مقادیر عددی را مشخص می‌سازند.

DT یا TBYTE : تعریف ده بایت

پیش‌پردازنده DT (TBYTE) عناصر داده‌ای را تعریف می‌کند که 10 بایت طول داشته باشد. منظور از آنها تعریف مقادیر عددی BCD فشرده شده است که برای کمک پردازشگرهای عددی که عملیات محاسباتی استاندارد را انجام می‌دهند مفیدتر است. یک عدد BCD در هر بایت با دو رقم هدهی فشرده می‌شود. توجه کنید که DT، بر خلاف دیگر پیش‌پردازنده‌های داده، ثابتهای عددی دهدهی را بیشتر ذخیره می‌کنند تا مقادیر شانزدهی. برای یک ثابت تعریف شده

۱۲۳۴۵۶۷۸، MASM بایتها را به ترتیب معکوس ذخیره می‌کند 00 00 00 00 00 12 33 56 78، اگر چه TASM آن را با ترتیب معمول و تنظیم از راست ذخیره می‌سازد. شکل ۴-۴، DT که برای یک عنصر ناشناخته، یک مقدار عددی و دو ثابت کاراکتری است را مشخص می‌سازد.

```
-d ds:100
0CA3:0100  00 30 30 7A 00 00 00 00-00 00 00 00 00 00 00 00   .00z............
0CA3:0110  43 6F 6D 70 75 74 65 72-20 50 72 6F 63 65 73 73   Computer Process
0CA3:0120  6F 72 73 34 37 38 33 35-01 4A 61 6E 02 46 65 62   ors47835.Jan.Feb
0CA3:0130  03 4D 61 72 F0 FF 7A 00-23 00 02 00 04 00 06 00   .Mar..z.#.......
0CA3:0140  07 00 09 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0CA3:0150  00 00 00 00 57 74 00 00-1B 00 00 00 38 00 00 00   ....Wt......8...
0CA3:0160  01 00 00 00 43 50 00 00-00 00 00 00 00 00 00 00   ....CP..........
0CA3:0170  39 5E 00 00 00 00 00 00-57 74 00 00 00 00 00 00   9^......Wt......
-d
0CA3:0180  00 00 00 00 00 00 00 00-00 00 30 78 29 00 00 00   ..........0x)...
0CA3:0190  00 00 00 00 43 50 00 00-00 00 00 00 00 00 21 DE   ....CP........!.
0CA3:01A0  42 D9 21 DE 42 D9 21 23-41 D9 21 23 41 D9 21 23   B.!.B.!#A.!#A.!#
0CA3:01B0  41 D9 21 23 41 D9 21 23-41 D9 21 23 41 D9 21 23   A.!#A.!#A.!#A.!#
0CA3:01C0  41 D9 21 23 41 D9 21 E2-43 D9 21 E2 43 D9 21 E2   A.!#A.!.C.!.C.!.
0CA3:01D0  43 D9 21 E2 43 D9 21 E2-43 D9 21 E2 43 D9 21 E2   C.!.C.!.C.!.C.!.
0CA3:01E0  43 D9 21 E2 43 D9 21 D8-43 D9 21 D8 43 D9 21 D8   C.!.C.!.C.!.C.!.
0CA3:01F0  43 D9 21 D8 43 D9 21 D8-43 D9 21 D8 43 D9 21 D8   C.!.C.!.C.!.C.!.

           <---------- hexadecimal representation ---------->   <---- ASCII ---->
```

شکل ۵-۴ نمایش سگمنت داده

نمایش سگمنت داده

بخشی از برنامه شکل ۴-۴ شامل فقط یک سگمنت داده‌است. اگر چه اسمبلر هیچ پیغام خطایی نمی‌دهد، در هنگام Link پیغام "Warning:no STACK SEGMENT." را نشان می‌دهد و لینکر پیغام "There were 1 errors detected." را نمایش می‌دهد خبر را ندیده بگیرید و از DEBUG برای دیدن کد مقصد استفاده کنید، مانند شکل ۵-۴.

برنامه را اسمبل و لینگ نموده، با استفاده از DEBUG فایل EXE. را بارگذاری کنید و دستور D DS:100 را تایپ کنید تا داده‌ها نمایش داده شود. سمت راست، نمایش بیان ASCII است مانند "Computer Processor" در حالیکه مقادیر مبنای شانزده در سمت چپ محتویات ذخیره شده واقعی را نشان می‌دهند. آنچه در صفحه نمایش خودتان می‌بینید با شکل ۵-۴ در افست 0100 تا 019D باید یکسان باشد. فقط آدرس سگمنت (0CA3 در شکل) و داده‌هایی که بعد در افست 019D قرار دارند می‌توانند متفاوت باشند.

دلیل آن که شما DS:100 را برای نمایش استفاده می‌کنید این است که بارگذار برنامه DS را با آدرس PSP بارگذاری می‌کند، اما سگمنتْ داده برای این برنامه 100 بایت بعد از آن آدرس آغاز می‌شود، بعداً وقتی از DEBUG برای برنامه‌های EXE. که DS را با آدرس سگمنت داده مقداردهی می‌کند استفاده کنید می‌توانید برای نمایش DS:0 را بزنید.

پیش پردازنده EQU

پیش پردازنده EQU یک داده را تعریف نمی‌کند، مقداری را تعریف می‌کند که در دستورات دیگر، اسمبلر می‌تواند آن را جایگزین سازد. جمله EQU کد شده در سگمنت کد زیر را در نظر بگیرید:

```
FACTOR EQU 12
```

نام، در این حالت FACTOR ممکن است هر نام قابل قبول برای اسمبلر باشد. حال وقتی که FACTOR در یک دستور یا در یک پیش‌پردازنده دیگر ظاهر می‌شود، اسمبلر مقدار ۱۲ را جایگزین می‌سازد.

برای مثال اسمبلر پیش پردازنده زیر را

```
TABLEX DB FACTOR DUP(?)
```

به مقدار معادل آن تبدیل می‌کند

```
TABLEX DB 12 DUP(?)
```

یک دستور ممکن است شامل یک عملوند معادل سازی شده باشد مثل زیر :

```
LIMITX  EQU  25
        ...
        MOV  CX,LIMITX
```

اسمبلر LIMITX را در دستور MOV با مقدار ۲۵ جایگزین می‌کند، بدین ترتیب هنگام اجرای برنامه، عملوند به یک مقدار بلافصل تبدیل می‌شود.

اسمبلر ۲۵ را جایگزین کرد: MOV CX,25

مزیت استفاده از EQU این است که خیلی از جملات ممکن است از این مقدار تعریف شده LIMITX استفاده کنند. اگر مقدار باید تغییر کند فقط جمله EQU را باید تغییر دهید. از یک مقدار معادل سازی فقط در جائیکه تبدیل برای اسبملر معنی داشته باشد می‌توان استفاده کرد. همچنین نامهای سمبولیک را باید به شکل زیر معادل سازی کرد:

```
TOTSALES DW   0
...
TS       EQU  TOTSALES
MPY      EQU  MUL
```

اولین معادل سازی TS را با عنصر TOTSALES تعریف می‌کند. برای هر دستور که حاوی عملوند TS باشد، اسمبلر آن را با آدرس TOTSALES جایگزین می‌سازد. دومین EQU به برنامه مجوز استفاده از کلمه MPY به جای سمبول معمول دستور MUL می‌دهد.

MASM 6.0 یک پیش پردازشگر TEXTEQU برای داده‌های متنی با شکل زیر را داراست.

نام TEXTEQU (متن)

نکات کلیدی

- علامت ; بیانگر ابتدای توضیح در هر خط می‌باشد.
- کلمات رزرو شده در زبان اسمبلی برای منظورهای خودش در شرایط خاص استفاده می‌شوند.
- یک شناسه نامی است که برای عناصر برنامه شما ظاهر می‌شود. دو نوع شناسه، نامها که به آدرس یک عنصر داده رجوع می‌کنند و برچسب که به آدرس یک دستور اشاره می‌کند، می‌باشد .
- یک عملیات که اغلب استفاده می‌شود تعریف ناحیه داده و دستورات کد شده است. یک عملوند اطلاعاتی راجع به فعالیت عملیات فراهم می‌سازد.
- یک برنامه حاوی یک یا تعداد بیشتری سگمنت است که هر کدام از مرز یک پاراگراف آغاز می‌شود .
- پیش‌پردازنده ENDS سگمنت‌را خاتمه‌می‌دهد، ENDP زیرروال را خاتمه‌می‌دهد، END یک‌برنامه‌را خاتمه‌می‌دهد.
- پیش پردازنده ASSUME ثباتهای سگمنت CS ، DS و SS را با نام‌های سگمنت مناسب مرتبط می‌سازد.
- یک برنامه EXE. باید ۳۲ کلمه را برای پشته تعریف کند.
- برای یک برنامه EXE. می‌توانید ثبات DS را با آدرس سگمنت داده مقدار دهی نمائید.
- برای پیش پردازنده‌های سگمنت ساده شده، شما مدل حافظه را قبل از تعریف هر سگمنت مقدار دهی می‌کنید. انتخاب‌هابشرح زیرند: SMALL (یک سگمنت کد و یک سگمنت داده)، MEDIUM (هرتعداد سگمنت کد و یک سگمنت داده)، COMPACT (یک سگمنت کد و هر تعداد سگمنت داده)، LARGE (هر تعداد سگمنت کد و سگمنت داده).

- تابع 4CH از INT 21H دستورات استاندارد برای خروج از برنامه دارد.
- نامهای عناصر داده باید واضح و منحصر بفرد باشند. برای مثال، یک عنصر برای دستمزد کارمندان می‌تواند EMPWAGE نامگذاری گردد .
- DB فرمت پیشنهادی برای تعریف رشته‌های کاراکتری است چراکه رشته‌هایی با طول بیشتر از ۲ را نیز مجاز می‌داند و آنها را به ترتیب معمول چپ به راست تبدیل می‌کند.
- ثباتهای دهدهی و دودویی مقادیر متفاوتی را تولید می‌کنند. فرض کنید تاثیر جمع دهدهی ۲۵ با جمع ۲۵ شانزدهی متفاوت است:

```
ADD   CX,25    ;   جمع با ۲۵
ADD   CX,25H   ;   جمع با ۳۷
```

- DW ، DD و DQ مقادیر عددی درکد هدف را به ترتیب بایت معکوس ذخیره می‌نمایند. عناصر DB برای پردازش نصف ثباتها (مثل AL و BL و غیره)، DW برای ثباتهای کامل (AX و BX و غیره) و DD برای ثباتهای توسعه یافته (EAX و EBX و غیره) استفاده می‌شوند. مقادیر عددی طولانی‌تر دستیابی‌های خاص را نیاز دارند.

پرسش‌ها

۴-۱. تفاوت بین کامپایلرها و اسمبلرها را توضیح دهید.

۴-۲. کلمات رزرو شده در زبان اسمبلی چیست؟ دو مثال بیاورید.

۴-۳. دو نوع شناسه چه هستند ؟

۴-۴. تعیین کنید کدام یک از نامهای زیر معتبر هستند: (الف) CX ، (ب) 25C4 ، (ج) X-$@ ، (د) $25 ، (ه) AT&T. اگر معتبر نیست توضیح دهید .

۴-۵. تفاوت بین پیش‌پردازنده‌ها و دستورات را توضیح دهید و از هر کدام دو مثال بیاورید.

۴-۶. دستوراتی را بنویسید که سبب می‌شود اسمبلر در زمان لیست‌گیری برنامه (الف) به صفحه جدید برود و (ب) یک عنوان در بالای صفحه چاپ نماید.

۴-۷. منظور از هر یک از سه سگمنت توضیح داده شده در این فصل را توضیح دهید.

۴-۸. شکل پیش پردازنده SEGMENT چنین است :

منظور از (الف) align ، (ب) combine ، (ج) 'class' چیست؟

۴-۹. (الف) هدف از یک زیرروال را توضیح دهید. (ب) انتها و ابتدای یک زیرروال چگونه تعریف می‌شود؟ (ج) کی شما یک زیرروال را بصورت FAR تعریف می‌کنید؟ (د) کی یک زیرروال را NEAR تعریف می‌کنید؟

۴-۱۰. جمله END خاص مربوط به انتهای هر کدام از موارد زیر چیست؟ (الف) یک زیرروال، (ب) یک سگمنت، (ج) یک برنامه.

۴-۱۱. تفاوت بین جملاتی که به اسمبلی خاتمه می‌دهند و جملاتی که به اجرا خاتمه می‌دهند را توضیح دهید.

۴-۱۲. نامهای STKSEG ، DATSEG و CDSEG برای سگمنت‌های پشته، داده و کد داده شده است، جمله ASSUME لازم را کد کنید.

۴-۱۳. فرض کنید دستور MOV AX,4C00H با INT 21H استفاده شده است. (الف) توضیح دهید که چه دستوراتی اجرا می‌شوند.(ب) منظور از 4C00H را توضیح دهید.

۴-۱۴. برای پیش پردازنده‌های سگمنت ساده شده، پیش پردازنده MODEL. برای مدلهای TINY ، LARGE، SMALL، MEDIUM، COMPACT فراهم شده است. تحت چه شرطی هر مدل استفاده می‌شود؟

۴-۱۵. طول بایتهای تولید شده با پیش پردازنده‌های داده زیر را مشخص کنید. (الف) DB، (ب) DQ، (ج) DT، (د) DW، (ه) DD.

۴-۱۶. رشته کاراکتری با نام CONAME حاوی "Computer Services" را بعنوان ثابت تعریف کند .

۴-۱۷. مقادیر عددی زیر را با عناصر داده‌ای با نام AREA1 تا AREA5 به ترتیب تعریف کنید:

(الف) یک بایت حاوی مقدار دودویی معادل ۳۵ دهدهی.

(ب) یک DW حاوی مقادیر به ترتیب ۱۲ ، ۱۴ ، ۲۲ ، ۲۸ ، ۳۳ ، ۴۱.

(ج) یک ۲ بایتی حاوی مقداری ناشناخته.

(د) یک بایت حاوی مقدار شانزدهی معادل ۵۸ دهدهی.

(ه) یک ۴ بایت حاوی مقدار شانزدهی معادل ۴۳۶ دهدهی.

۴-۱۸. کد هدف تولید شده برای موارد زیر را نشان دهید: (الف) DB 34 ، (ب) DB '34' ، (ج) DB 4DUP(0)

۴-۱۹. کد هدف شانزدهی اسمبل شده برای موارد زیر را تعیین کنید: (الف) DD 4A25B2H ، (ب) DQ 3BA53DH (ج) DB 35H ، (د) DW 3728H.

فصل ۵

اسمبل کردن، لینک کردن و اجرای یک برنامه

هدف : مراحل اسمبل کردن، لینک کردن و اجرای یک برنامه زبان اسمبلی.

مقدمه

این فصل نحوه ورود زیر روال‌هایی در برنامه‌های زبان اسمبلی و اسمبل کردن، لینک کردن و اجرای آنها را توضیح می‌دهد دستورات سمبولیکی که شما به زبان اسمبلی کد می‌کنید با نام ***برنامه منبع*** شناخته می‌شوند. با استفاده از برنامه اسمبلر، برنامه منبع را به کد ماشین که با عنوان ***برنامه هدف*** شناخته می‌شود ترجمه می‌کنید. در نهایت با استفاده از برنامه لینکر آدرسهای ماشین در برنامه هدف را کامل کرده و یک ***ماجول اجرایی*** تولید می‌شود.

بخش اسمبل کردن چگونگی تقاضای اجرای برنامه اسمبلر را توضیح می‌دهد که تشخیص خطا (شامل هر پیغام خطا) و تولید برنامه هدف را فراهم می‌سازد همچنین جزئیات توضیح در مورد لیست‌گیری اسمبلر و به عبارت معمولی، چگونگی پردازش یک برنامه منبع توسط اسمبلر ، می‌باشد.

بخش لینک کردن نحوه تقاضای اجرای برنامه لینک کننده را توضیح می‌دهد که با آن می‌توانید یک ماجول اجرایی را بسازید در انتها، یک بخش نحوه تقاضای اجرای یک ماجول اجرایی را توضیح می‌دهد.

مهیا نمودن یک برنامه جهت اجرا

شکل ۲-۴ فقط کد منبع یک برنامه را که هنوز در قالب اجرایی نیست را مشخص می‌سازد. برای وارد کردن این برنامه، باید از یک برنامه ویرایشگر یا پردازشگر کلمات که فایلهای ASCII استاندارد تولید می‌کند، استفاده کنید. در فرمان مثال زیر، درایو مناسب برای سیستم خودتان را جایگزین کنید. با بارگذاری برنامه و فایلها در دیسک RAM می‌توانید کارآیی بیشتری حاصل کنید. برنامه ویرایشگر خودتان را فراخوانی کنید، جملات برنامه شکل ۲-۴ را وارد کنید و نام فایل حاصل را A05ASM1.ASM بگذارید. اگر چه میزان فاصله اجرای اسمبلر مهم نیست، برنامه‌ای خواناتر خواهد بود که نام، عملیات، عملوند و توضیحات از یک ستون خاص مرتب شوند اغلب ویرایشگرها از کلید tab که فواصل ۸ کاراکتری ایجاد می‌کند برای تنظیم بهتر ستون‌ها استفاده می‌کنند.

وقتی برنامه را وارد کردید از نظر دقت کد برنامه را امتحان کنید. اگر چه اغلب ویرایشگرها امکان چاپ کردن را دارند، شما ممکن است از برنامه PRINT از DOS استفاده کنید :

```
PRINT n:A05ASM1.ASM <Enter>
```

این برنامه منبع یک فایل متنی است که نمی‌تواند اجرا شود. شما باید ابتدا آن اسمبل و سپس لینک نمایید.

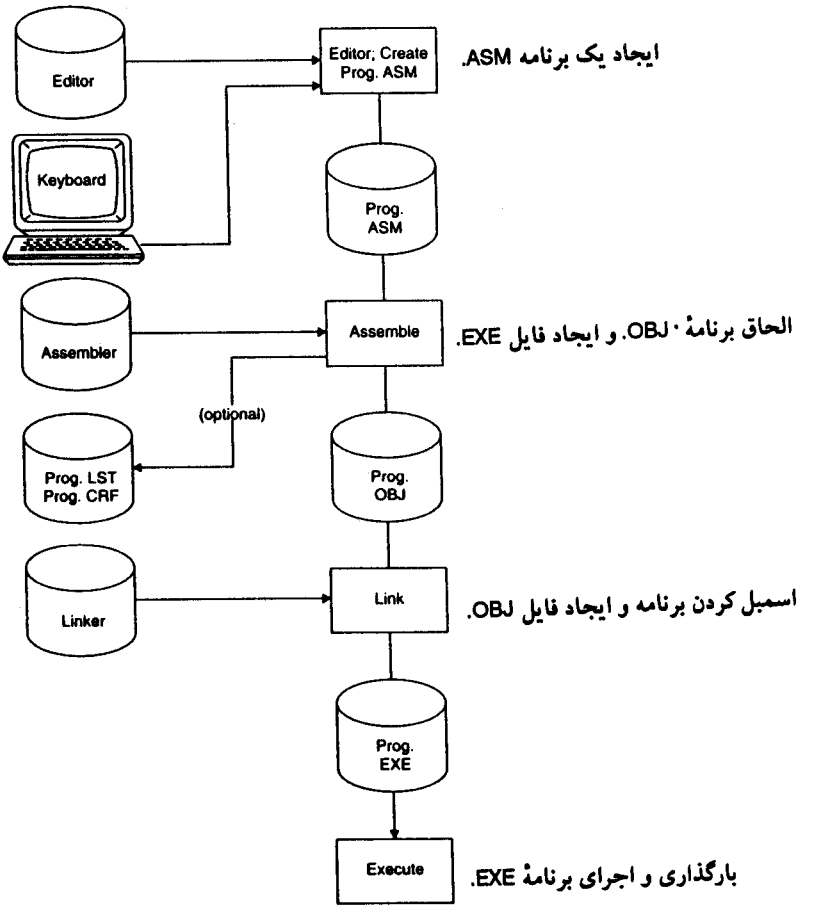
۱) ***مرحله اسمبل کردن*** شامل ترجمه کد منبع به کد هدف و تولید یک فایل (هدف) OBJ. میانی یا ماجول می‌باشد. (مثالهایی از کد ماشین و برنامه منبع در فصول اخیر ارائه شد) یکی از وظایف اسمبلر محاسبه افست برای هر عنصر

داده در سگمنت داده و هر دستور در سگمنت کد می‌باشد. علاوه بر این اسمبلر مقابل ماجول OBJ. تولید شده ، یک عنوان قرار می‌دهد، بخشی از عنوان حاوی اطلاعاتی راجع به آدرس‌های تکمیل نشده است. ماجول OBJ. هنوز به شکل اجرایی نیست.

۲) **مرحله‌لینک‌کردن** شامل تبدیل ماجول OBJ. به یک ماجول کدماشین (اجرایی) EXE. است. وظیفه‌لینک‌کننده شامل تکمیل هر آدرس باز گذاشته شده، توسط اسمبلر و ترکیب برنامه‌های اسمبل‌شده مجزا به یک ماجول اجرایی است.

۳) آخرین مرحله، ***بارگذاری*** برنامه جهت اجرا است. از آنجائیکه بارگذار می‌داند برنامه کجا باید بارگذاری شود، قادر است تا هر آدرس تکمیل شده در عنوان را برقرار کند. بارگذار عنوان را برمی‌دارد و یک پیشوند سگمنت برنامه (PSP) بلافاصله قبل از بارگذاری برنامه در حافظه ایجاد می‌کند.

شکل ۱-۵ یک نما از مراحل اسمبل کردن، لینک کردن و اجرای یک برنامه را نشان می‌دهد.



شکل ۱-۵ مراحل اسمبل، لینک و اجرا

اسمبل کردن یک برنامه منبع

برنامه اسمبل کننده مایکروسافت (بالاتر از نسخه MASM.EXE(5.X است. که برنامه توربو بورلند آن TASM.EXE است. از نسخه 6.0، اسمبلر مایکرو ساخت از فرمان ML استفاده می‌کند، اما برای مطابقت با نسخه‌های

پیشین MASM را نیز می‌پذیرد.

شما فرمان اجرای MASM یا TASM را با یک فرمان خطی، یا توسط اعلان وارد می‌کنید. این بخش نحوه استفاده از فرمان خطی را نشان می‌دهد، ضمیمه D را برای روش اعلان ببینید. شکل کلی برای فرمان خطی اسمبل کردن یک برنامه چنین است : `MASM/TASM [option] source [, Object] [,Listing] [,Cossref]`

option برای جزئیات بیشتر راجع به سطح تنظیم پیغام‌های خبر است که در ضمیمه D توضیح داده شده است پیش فرض اسمبل کننده برای منظور ما بقدر کافی مناسب است.

source نام برنامه منبع مثلاً A05ASM1 را مشخص می‌سازد. اسمبلر قسمت توسعه را ASM. فرض می‌کند پس نیازی به وارد کردن آن نیست. همچنین اگر نام درایو پیش فرض جاری را نمی‌خواهید، می‌توانید نام درایو مورد نظر را وارد کنید.

object برای فایل OBJ. تولید شده فراهم شده است. درایو، زیر شاخه و نام فایل ممکن است یکی باشد یا با آنچه در برنامه منبع است متفاوت باشد.

listing برای یک فایل LST. که حاوی هم کد منبع و کد هدف است فراهم شده است. درایو، زیر شاخه و نام فایل ممکن است با برنامه منبع یکی باشد یا با هم فرق داشته باشند.

crossref یک فایل‌ارجاع متقابل که حاوی سمبولهای استفاده‌شده در برنامه است را فراهم‌می‌سازد، که شمابرای لیست‌گیری ارجاع متقابل می‌توانید از آن استفاده کنید. قسمت توسعه آن CRF. در MASM و XRF. در TASM می‌باشد.

نام فایل منبع را وارد کرده و یک فایل OBJ. که برای لینک کردن یک برنامه به شکل اجرایی لازم است را تقاضا می‌کنید. ممکن است اغلب یک فایل LST نیز بخواهید، مخصوصاً وقتی تشخیص خطا بدهید یا بخواهید کد ماشین تولید شده را امتحان کنید. برای برنامه‌های بزرگی که می‌خواهید ببینید کدام دستور و به چه عنصر داده‌ای رجوع می‌کند، از فایل CRF. استفاده کنید. همچنین، درخواست یک فایل CRF. سبب می‌شود اسمبلر برای فایل LST. شماره خط ایجادکند بطوریکه فایل CRF. به آن اشاره‌نماید. بخش بعد فایلهای LST. و CRF. را با جزئیات‌کامل‌توضیح خواهدداد.

مثال ۱. فایل منبع A05ASM1 را بر روی درایو D مشخص کنید و فقط یک فایل obj تولید کنید. در این حالت ، باید ارجاع به LST. و CRF. را حذف کنید و فرمان زیر را وارد کنید :

```
MASM/TASM D:A05ASM1,D:
```

مثال ۲. فایلهای هدف، لیست و ارجاع متقابل را تولید کنید :

```
MASM/TASM D:A05ASM1,D:,D:,D:
```

اسمبلر جملات منبع را به کد ماشین تبدیل می‌کند و هر پیغام خطا را بر روی صفحه نشان می‌دهد، نوعی از خطاها شامل نامی است که از قاعده نامگذاری تخلف کرده باشد، مانند دستوراتی که خطای نحوی داشته باشند (مثلاً MOVE بجای MOV) یا عملوندی که نام آن تعریف نشده باشد. از آنجائیکه خطاهای ممکن زیاد هستند (۱۰۰ یا بیشتر) و نسخ اسمبل کننده‌های مختلفی وجود دارند، شما می‌توانید به راهنمای اسمبلر خودتان مراجعه کنید. اسمبلر برخی از خطاها را تصحیح می‌کند اما، در هر مورد، شما باید ویرایشگر را بارگذاری نموده برنامه منبع ASM. را تصحیح کرده و دوباره آن را اسمبل کنید.

استفاده از تعاریف سگمنت مرسوم

شکل ۲-۵ لیستی است که اسمبلر MASM تحت نام A05ASM1.LST تهیه نموده است.

عرض خط ۱۳۲ کاراکتر که در ورودی entry مشخص شده است. اگر چاپگر شما بتواند خطوط چاپ را فشرده کند این لیست را می‌تواند چاپ نماید. خیلی از چاپگرهای چکشی سوئیچی دارند که چاپ فشرده را انجام می‌دهد، با استفاده از

محیط‌های ویرایشی نیز می‌توان متون را به صورت فشرده چاپ کرد. روش دیگر استفاده از فرمان MODE در DOS است برای انجام این کار چاپگر را خاموش کرده و فرمان زیر را تایپ کنید MODE LPT1:132,6 و حال چاپ برنامه را انجام دهید.

توجه کنید که بالای لیست، چگونگی فعالیت اسمبلر در پیش پردازنده‌های PAGE و TITLE مشخص شده است. هیچ یک از پیش‌پردازنده‌های SEGMENT ، PROC ، ASSUME و END کد ماشین تولید نمی‌کنند، و فقط پیامی را به اسمبلر ارسال می‌کنند. لیست‌گیری بر اساس مطالب زیر بطور عمودی مرتب شده است :

```
A05ASM1 (EXE)  Move and add operations                    Page   1-1
 1                              page    60,132
 2                      TITLE   A05ASM1 (EXE)  Move and add operations
 3                      ; -----------------------------------------
 4 0000                 STACKSG SEGMENT PARA STACK 'Stack'
 5 0000  0020[                  DW      32 DUP(0)
 6       0000
 7           ]
 8
 9 0040                 STACKSG ENDS
10                      ; -----------------------------------------
11 0000                 DATASG  SEGMENT PARA 'Data'
12 0000  00AF           FLDD    DW      175
13 0002  0096           FLDE    DW      150
14 0004  0000           FLDF    DW      ?
15 0006                 DATASG  ENDS
16                      ; -----------------------------------------
17 0000                 CODESG  SEGMENT PARA 'Code'
18 0000                 MAIN    PROC    FAR
19                              ASSUME  SS:STACKSG,DS:DATASG,CS:CODESG
20 0000  B8 ---- R              MOV     AX,DATASG       ;Set address of data
21 0003  8E D8                  MOV     DS,AX           ;  segment in DS
22
23 0005  A1 0000 R              MOV     AX,FLDD         ;Move 0175 to AX
24 0008  03 06 0002 R           ADD     AX,FLDE         ;Add  0150 to AX
25 000C  A3 0004 R              MOV     FLDF,AX         ;Store sum in FLDF
26 000F  B8 4C00                MOV     AX,4C00H        ;End processing
27 0012  CD 21                  INT     21H
28 0014                 MAIN    ENDP                    ;End of procedure
29 0014                 CODESG  ENDS                    ;End of segment
30                              END     MAIN            ;End of program

Segments and Groups:
                N a m e         Length   Align   Combine   Class
CODESG . . . . . . . . . . .    0014     PARA     NONE     'CODE'
DATASG . . . . . . . . . . .    0006     PARA     NONE     'DATA'
STACKSG  . . . . . . . . . .    0040     PARA     STACK    'STACK'

Symbols:
                N a m e         Type     Value    Attr
MAIN  . . . . . . . . . . .     F PROC   0000     CODESG   Length = 0014

FLDD . . . . . . . . . . . .    L WORD   0000     DATASG
FLDE . . . . . . . . . . . .    L WORD   0002     DATASG
FLDF . . . . . . . . . . . .    L WORD   0004     DATASG

@CPU . . . . . . . . . . . .    TEXT  0101h
@FILENAME  . . . . . . . . .    TEXT  a05asm1
@VERSION . . . . . . . . . .    TEXT  x10

          27 Source Lines
          27 Total  Lines
          15 Symbols
0 Warning Errors
0 Severe  Errors
```

شکل ۲-۵ برنامه اسمبل شده با سگمنت مرسوم

۱. در سمت چپ شماره هر خط قرار دهید.
۲. بخش دوم آدرس مبنای شانزده فیلدهای داده و دستورات قرار دارد.
۳. بخش سوم کد ماشین ترجمه شده به شکل مبنای شانزده را نشان می‌دهد.
۴. بخش سمت راست کد منبع اصلی شما قرار دارد.

برنامه در سه سگمنت سازماندهی شده است. هر سگمنت شامل یک پیش پردازنده SEGMENT است که به اسمبلر می‌گوید تا سگمنت بر آدرسدهی که قابل تقسیم بر ۱۰ شانزدهی است قرار گیرد خود عبارت SEGMENT کد ماشین تولید نمی‌کند. بارگذار برنامه محتویات هر سگمنت را در حافظه ذخیره کرده و آدرس را در یک ثبات سگمنت مقدار دهی می‌کند. شروع سگمنت در افست صفر از این آدرس است.

بخاطر بسپارید که هر سگمنت یک ناحیه مجزا است، با افست خودش برای داده‌ها یا دستورات.

سگمنت پشته

سگمنت پشته حاوی یک پیش پردازنده DW (تعریف کلمه) است که ۳۲ کلمه صفر را تعریف می‌کند. این تعریف ۳۲ کلمه، یک اندازه حقیقی برای پشته است زیرا برنامه‌های بزرگ ممکن است برای انجام اعمال ورودی خروجی به تعداد زیادی وقفه نیاز داشته باشد و زیر برنامه‌های بسیاری را فراخوانی کنند که تمام اینها مشمول استفاده از پشته است. سگمنت پشته در افست 0040H خاتمه می‌یابد، که معادل با مقدار دهدهی ۶۴ (۳۲ کلمه × ۲ بایت) است. اسمبلر ثابت تولید شده را در سمت چپ چنین نمایش می‌دهد 0020[0000]، که 20H (32) کلمه صفر است.

اگر پشته برای تمامی عناصری که بر روی آن گذاشته می‌شود، کوچک باشد، اسمبلر و لینکر، شما را آگاه نمی‌کنند و اجرای برنامه در یک حالت غیر قابل پیش بینی قطع می‌شود.

سگمنت داده

در این برنامه، سه داده از نوع DW در یک سگمنت داده با عنوان DATASG تعریف می‌شوند.

۱. FLDD یک کلمه (۲ بایت) با مقدار ۱۷۵ دهدهی که اسمبلر آن را به 00AFH تبدیل می‌کند.
۲. FLDE یک کلمه با مقدار دهدهی ۱۵۰، که بصورت 0096H اسمبل می‌شود. مقدار ذخیره شده واقعی این دو مقدار ثابت به ترتیب AF00 و 9600 است که می‌توانید با DEBUG آن را چک کنید.
۳. FLDF که به صورت DW با ؟ کد شده که یک کلمه با یک مقدار ناشناخته را تعریف می‌کند. لیست‌گیری محتویات آن را به صورت 0000 نشان می‌دهد.

آدرسهای افست FLDD ، FLDE و FLDF به ترتیب 0000 ، 0002 و 0004 است که در ارتباط با اندازه فیلد خودشان می‌باشد.

سگمنت کد

این برنامه حاوی یک سگمنت کد با عنوان CODESG است که حاوی کد اجرایی برنامه می‌باشد. سه جمله‌ای که آدرس سگمنت داده را برقرار می‌کنند چنین هستند :

```
                        ASSUME SS:STACKSG,DS:DATASG,CS:CODES
0000    B8 ---- R       MOV   AX,DATASG
0003    8E DB           MOV   DS,AX
```

۱. پیش پردازنده ASSUME ثبات DS را با DATASG مرتبط می‌کند. توجه کنید که برنامه به ثبات ES نیاز ندارد، اگر چه برخی از برنامه نویسان آن را بعنوان یک تمرین استاندارد مقداردهی می‌کنند. ASSUME اطلاعاتی راجع به

اسمبلر فراهم می‌سازد، که هیچ کد ماشینی برای آن تولید نمی‌شود.

۲. اولین دستور MOV، DATASG را در ثبات AX ذخیره می‌کند. یک دستور نمی‌تواند یک سگمنت را در یک ثبات سگمنت ذخیره نماید - اسمبلر بسادگی تشخیص می‌دهد که آدرس DATASG بارگذاری کند. توجه کنید که کد ماشین در سمت چپ R ---- B8 است. چهار خط فاصله به این معنی است که اسمبلر در این جا آدرس DATASG را نتوانسته تشخیص بدهد، سیستم این آدرس را فقط وقتی برنامه مقصد لینک شده و برای اجرا بارگزاری می‌شود. تشخیص می‌دهد. از آنجائیکه بارگذار یک برنامه را در هر جایی از حافظه ممکن است قرار دهد، اسمبلر آدرس را باز می‌گذارد و این مکان را با یک R نشان می‌دهد، بارگذار آن را با مقدار واقعی آدرس کامل، جایگزین می‌کند.

۳. سومین دستور MOV محتویات ثبات AX را داخل ثبات DS کپی می‌کند. از آنجائیکه هیچ دستور معتبری برای انتقال مستقیم به ثبات DS وجود ندارد، باید دو دستور برای مقدار دهی DS کد نویسی نمائید.

اگر چه بارگذار SS و CS را وقتی یک برنامه را جهت اجرا بارگذاری می‌کند بطور اتوماتیک مقدار دهی می‌نماید، اگر لازم باشد مقدار دهی DS و ES بعهده خود شماست.

اگر تا کنون بخوبی متوجه این اعمال تعریف نشده‌اید، نیازی نیست که کاملاً آن را متوجه شوید. همه برنامه‌ها در این کتاب از یک تعریف استاندارد و مقدار دهی استفاده می‌کنند و شما براحتی می‌توانید در ایجاد برنامه‌های خودتان از آنها استفاده کنید. برای این منظور اسکلت یک برنامه را بر روی دیسک ذخیره کنید و برای هر برنامه جدید که می‌خواهید ایجاد کنید اسکلت برنامه را بر روی یک فایل با نام صحیح خودش کپی کنید و از ویرایشگر برای تکمیل دستورات اضافی استفاده نمائید.

اولین دستور پس از مقدار دهی ثبات DS ، MOV AX,FLDD است که در موقعیت افست 0005 آغاز می‌شود و کد ماشینی A1 0000 را تولید می‌کند. فاصله بین A1 (عملگر) و 0000 (عملوند) فقط برای خواناتر شدن برنامه است. دستور بعدی، دستور ADD AX,FLDE است که از موقعیت افست 0008 آغاز می‌شود و ۴ بایت کد ماشین تولید می‌کند. دستور MOV FLDF,AC مجموع AX را به FLDE در افست 0004 در سگمنت داده کپی می‌کند. در این مثال، دستورات ماشین ۱ یا ۳ یا ۴ بایت طول دارند.

جمله بعدی در برنامه END حاوی عملوند MAIN است که با نام PROC در افست 0000 مرتبط است، در این موقعیت در سگمنت کد برنامه بارگذار، کنترل را برای اجرا منتقل می‌کند. برنامه بعدی سگمنت‌ها و جدولهای گروهی و جدولهای سمبول را لیست می‌کند.

سگمنت‌ها و جدول گروه‌ها

اولین جدول در انتهای لیست اسمبل شده، همه سگمنت‌های تعریف شده و گروه‌ها را نشان می‌دهد. توجه کنید که سگمنت‌ها به همان ترتیبی که کد شده‌اند، لیست نشده‌اند، اسمبلر در این مثال آنها را به ترتیب حروف الفبای نام مرتب می‌کند. (این برنامه گروه ندارد، عنوانی است که بعداً خواهیم گفت). جدول طول هر سگمنت را به بایت، تنظیم می‌کند (همه بر روی یک پاراگراف) نوع ترکیب و کلاس را نیز فراهم می‌نماید. اسمبلر نام کلاسها را به حروف بزرگ تبدیل می‌کند.

جدول سمبولها

دومین جدول نام فیلدهای داده در سگمنت داده (FLDD ، FLDE و FLDF) و برچسب‌های دستورات در سگمنت کد را فراهم نموده است. برای MAIN عبارت FPROC را که بیانگر روالی از نوع دور است، تایپ کنید. (دور، زیرا نقطه ورود برای اجرای MAIN خارج از این برنامه شناخته شود). ستون مقدار افست، شروع سگمنت برای نامها، برچسب‌ها و روالها را مشخص می‌کنند. ستونی با عنوان Attr (برای صفت) سگمنت را طوریکه عنصر تعریف شده

است، فراهم می‌سازد. ضمیمه D همه انتخابهای این جدول را توضیح می‌دهد. برای حذف کردن جدول پس از فرمان MASM/TASM از /N استفاده کنید. برای سه ورودی آخر :

@ CPU در زمان اسمبل کردن، پردازشگر را تعیین می‌کند، بر اساس تنظیم شماره بیتها :

7 حق ویژه دستورات	8086	0
8 پردازشگر عددی 8087	80186	1
10 پردازشگر عددی 80287	80286	2
11 پردازشگر عددی 80387	80486	4
پنتیوم	80586	5

@ FILENAME نام برنامه را می‌دهد

@VERSION نسخه اسمبلر را به شکل n.nn نمایش می‌دهد.

```
 A05ASM1 (EXE)  Move and add operations
 1                      ; ---------------------------------------------
 2   0000               STACKSG SEGMENT PARA STACK    'Stack'
 3   0000  20*(0000)            DW     32 DUP(0)
 4   0040               STACKSG ENDS
 5                      ; ---------------------------------------------
 6   0000               DATASG  SEGMENT PARA 'Data'
 7   0000  00AF         FLDD    DW     175
 8   0002  0096         FLDE    DW     150
 9   0004  ????         FLDF    DW     ?
10   0006               DATASG  ENDS
11                      ; ---------------------------------------------
12   0000               CODESG  SEGMENT PARA 'Code'
13   0000               MAIN    PROC   FAR
14                              ASSUME SS:STACKSG,DS:DATASG,CS:CODESG
15   0000  B8 0000s             MOV    AX,DATASG       ;Set address of data
16   0003  8E D8                MOV    DS,AX           ;  segment in DS
17
18   0005  A1 0000r             MOV    AX,FLDD         ;Move 0175 to AX
19   0008  03 06 0002r          ADD    AX,FLDE         ;Add  0150 to AX
20   000C  A3 0004r             MOV    FLDF,AX         ;Move to FLDF
21   000F  B8 4C00              MOV    AX,4C00H        ;End processing
22   0012  CD 21                INT    21H
23   0014               MAIN    ENDP                   ;End of procedure
24   0014               CODESG  ENDS                   ;End of segment
25                              END    MAIN            ;End of program

Symbol Name             Type     Value

??DATE                  Text     "05/17/xx"
??FILENAME              Text     "a05asm1 "
??TIME                  Text     "16:06:45"
??VERSION               Number   040A
@CPU                    Text     0101H
@CURSEG                 Text     CODESG
@FILENAME               Text     A05ASM1
@WORDSIZE               Text     2
MAIN                    Far      CODESG:0000
FLDD                    Word     DATASG:0000
FLDE                    Word     DATASG:0002
FLDF                    Word     DATASG:0004

Groups & Segments       Bit Size  Align  Combine Class

CODESG                  16  0014  Para   none    CODE
DATASG                  16  0006  Para   none    DATA
STACKSG                 16  0040  Para   Stack   STACK
```

شکل ۳-۵ لیست توربواسمبلر

لیست‌گیری توربو اسمبلر

شکل ۳-۵ لیست همان برنامه را فراهم کرده، اما با TASM آنها را اجرا نموده است. تفاوتهای کد تولید در سمت چپ لیست به ترتیب شماره دستورات بشرح زیر است :

	منبع	کد مقصد	منظور
۳	DW 32 DUP(0)	20* (0000)	۲۰ کپی از کلمه صفر
۹	DW ?	????	یک مقدار تعریف نشده
۱۵	MOV AX,DATASG	B8 0000S	0000 = عملوند تکمیل نشده و S = آدرس‌سگمنت
۱۸	MOV AX,FLDD	A1 0000r	0000 = آدرس افست FLDD و r = مقداری که بعد گذاشته می‌شود

طبق لیست معمول برنامه‌های اسمبل شده MASM و TASM یک کد تولید می‌کند.

```
 A05ASM2 (EXE)  Move and add operations                    Page   1-1
 1                              page  60,132
 2                      TITLE   A05ASM2 (EXE)  Move and add operations
 3                      ;-----------------------------------------------
 4                              .MODEL  SMALL
 5                              .STACK  64              ;Define stack
 6                              .DATA                   ;Define data
 7 0000  00AF           FLDD    DW      175
 8 0002  0096           FLDE    DW      150
 9 0004  0000           FLDF    DW      ?
10                      ;-----------------------------------------------
11                              .CODE                   ;Define code segment
12 0000                 MAIN    PROC    FAR
13 0000  B8 ---- R              MOV     AX,@data        ;Set address of data
14 0003  8E D8                  MOV     DS,AX           ;   segment in DS
15
16 0005  A1 0000 R              MOV     AX,FLDD         ;Move 0175 to AX
17 0008  03 06 0002 R           ADD     AX,FLDE         ;Add  0150 to AX
18 000C  A3 0004 R              MOV     FLDF,AX         ;Store sum in FLDF
19
20 000F  B8 4C00                MOV     AX,4C00H        ;End processing
21 0012  CD 21                  INT     21H
22 0014                 MAIN    ENDP                    ;End of procedure
23                              END     MAIN            ;End of program

Segments and Groups:
                N a m e                 Length   Align   Combine   Class
DGROUP . . . . . . . . . . .            GROUP
  _DATA  . . . . . . . . . .            0006     WORD    PUBLIC    'DATA'
  STACK  . . . . . . . . . .            0040     PARA    STACK     'STACK'
_TEXT  . . . . . . . . . .              0014     WORD    PUBLIC    'CODE'

Symbols:
                N a m e                 Type     Value   Attr
MAIN . . . . . . . . . . . .            F PROC   0000    _TEXT  Length = 0014

FLDD . . . . . . . . . . . .            L WORD   0000    _DATA
FLDE . . . . . . . . . . . .            L WORD   0002    _DATA
FLDF . . . . . . . . . . . .            L WORD   0004    _DATA

@CODE  . . . . . . . . . . .            TEXT     _TEXT
@CODESIZE  . . . . . . . . .            TEXT     0
@CPU . . . . . . . . . . . .            TEXT     0101h
@DATASIZE  . . . . . . . . .            TEXT     0
@FILENAME  . . . . . . . . .            TEXT     a05asm2

     0 Warning Errors
     0 Severe  Errors
```

شکل ۴-۵ برنامه اسمبل شده با پیش پردازنده‌های ساده سگمنت

استفاده از پیش پردازنده‌های ساده سگمنت

شکل ۳-۴ نحوه کد کردن یک برنامه با استفاده از پیش پردازنده‌های ساده سگمنت را نشان می‌دهد.
شکل ۴-۵ لیست اسمبل شده همان برنامه را فراهم نموده است. برای پیش پردازنده‌های سگمنت ساده باید DS را چنین مقدار دهی کنید :

```
MOV   AX,@data
MOV   DS,AX
```

اولین بخش از جدول سمبول با عنوان "سگمنت‌ها و گروهها" سه سگمنت نامگذاری شده با اسمبلر و لیست الفبایی آن را نشان می‌دهد :

- DATA - با طول ۶ بایت.
- STACK با طول 40H (۶۴بایت)
- TEXT- برای سگمنت کد با طول 14H (۲۰ بایت)

با عنوان "سمبولها" نام‌های تعریف شده در برنامه یا نامهای پیش فرض لیست شده است. پیش پردازنده‌های ساده سگمنت مقداری معادل تعریف شده فراهم می‌سازند، که با سمبول @ آغاز می‌شوند و شما در برنامه برای ارجاع به آن آزادید، مانند data @، آنها چنین هستند :

@ CODE معادل با نام سگمنت کد، TEXT_
@ CODESIZE برای مدلهای Small و Medium صفر می‌شود
@ CPU مدل پردازشگر
@ DATASIZE برای مدلهای Small و Medium صفر می‌شود
@ FILENAME نام برنامه
@ VERSION نسخه اسمبلر (h . nn)

از code @ و data @ در ASSUME و دستورات اجرایی می‌توانید بدین شکل استفاده کنید MOV AX,@ data

اسمبلر دوگذره

بسیاری از اسمبلرها دو یا چند گذر در یک برنامه منبع دارند تا ارجاع جلوتر به آدرسهایی در برنامه که هنوز به آنها نرسیده است، را تجزیه و تحلیل نمایند. در طی گذر اول، اسمبلر برنامه منبع ورودی را می‌خواند و جدول سمبولی از نامها و برچسب‌های استفاده شده در برنامه را می‌سازد، که نامهای فیلدهای داده و برچسب‌های برنامه و موقعیت‌های مرتبط آنها در داخل سگمنت می‌باشد. شما چنین جدول سمبولی را بلافاصله بعد از برنامه اسمبل شده در شکل ۳-۵ می‌بینید، که به ترتیب افست FLDD ، FLDE و FLDF مقادیر 0000 ، 0002 و 0004 بایت می‌باشد. اگر چه برنامه هیچ برچسب دستوری ندارد این برچسب در صورت وجود در سگمنت کد با افست خودش ظاهر خواهد شد. گذر اول میزان کد تولید شده برای هر دستور را تعیین می‌کند. MASM تولید کد مقصد را در گذر اول آغاز می‌کند، در حالیکه TASM این کار را در گذر دوم انجام می‌دهد.

در گذر دوم، اسمبلر از جدول سمبولی که در گذر اول ساخته شده است استفاده می‌کند. در این حالت که اسمبلر طول و موقعیت مرتبط با هر فیلد داده و دستور را می‌داند، می‌تواند کد مقصد برای هر دستور را کامل کند. سپس در صورت نیاز، فایلهای مقصد (OBJ.)، و لیست (LST.) و ارجاع متقابل (REF.) را تولید می‌کند. یک مشکل ممکن در گذر اول ***ارجاع به بعد*** است. انواع خاصی از دستورات در سگمنت کد ممکن است به برچسبی رجوع کنند که اسمبلر هنوز به تعریف آن نرسیده است. MASM کد مقصدی می‌سازد که بر پایه حدسی است که از طول هر دستور زبان ماشین تولید شده است. اگر تفاوتی بین گذر اول و گذر دوم در طول دستورات باشد، MASM یک پیغام خطا بدین شکل را نشان می‌دهد "Phase errore between passes" این خطا به ندرت ایجاد می‌شود، اگر ظاهر شد، باید علت ایجاد آن را

پیگیری و تصحیح نمائید. در نگارش ششم، MASM، طول دستورات بادقت و کارآیی بیشتری دستیابی می‌شوند، در صورت لزوم چندین گذر در فایل تعبیه می‌شوند، TASM یک برنامه را با یک گذر می‌تواند اسمبل نماید، اما در صورت لزوم و در صورت عدم بروز اشکال ارجاع به بعد بیشتر از یک گذر نیز می‌توانید داشته باشید. (ضمیمه D را ببینید).

لینک کردن یک برنامه مقصد

وقتی برنامه شما هیچ پیغام خطایی ندارد، مرحله بعدی لینک کردن ماجول مقصد است. 'A05ASM1.OBJ' که توسط اسمبلر ایجاد شده است و حاوی فقط کد ماشین است. لینکر عملیات زیر را انجام می‌دهد :

- ترکیب، در صورت درخواست چندین ماجول اسمبل شده مجزا را به صورت یک برنامه اجرایی ترکیب می‌کند، مانند دو یا چند برنامه اسمبلی یا یک برنامه اسمبلی با یک برنامه C.
- تولید یک ماجول EXE. و مقداردهی آن با دستورات ویژه برای سهولت ترتیب بارگذاری جهت اجرا. وقتی شما یک یا چند ماجول OBJ. را به هم لینک کرده و در یک ماجول EXE. قرار می‌دهید، ممکن است ماجول EXE. را دفعات زیادی اجرا کنید اما هر زمان که بخواهید برنامه را تغییر دهید، باید برنامهٔ منبع را تصحیح کنید، آن را مجدداً به شکل obj. اسمبل کنید و این ماجول OBJ. را به یک ماجول EXE. تبدیل (لینک) نمایید. اگر در ابتدا این مراحل برای شما واضح نیست، فقط با کمی تجربه بطور اتوماتیک این کارها را انجام خواهید داد. ممکن است برخی از برنامه‌های EXE. را به برنامه‌های COM. تبدیل کنید. فصل ۷ را برای جزئیات بیشتر ببینید.

نسخه لینکر مایکروسافت LINK است، در حالیکه نسخه بورلند آن TINK می‌باشد. شما با تایپ کردن LINK یا TLINK در خط فرمان یا اعلان می‌توانید آن را اجرا نمائید. (MASM 6.0 فرمان ML را برای هم اسمبل کردن و هم لینک کردن فراهم نموده است). این بخش نحوه لینک کردن را با استفاده از خط فرمان نشان می‌دهد، ضمیمه D را برای استفاده از اعلان ببینید. فرمان لینک کردن چنین است.

```
LINK/TLINK objfile , exefile [,mapfile] [,libraryfile]
```

- Objfile فایل مقصد تولید شده توسط اسمبلر را مشخص می‌سازد. لینکر توسعه OBJ. را برای آن در نظر می‌گیرد، بنابراین نیازی نیست آن را واردکنید. درایو، زیرشاخه و نام‌فایل ممکن‌است با نام‌منبع یکی‌باشد یا با آن متفاوت‌باشد.
- exefile برای فایل EXE. تولید شده فراهم شده است. درایو، زیر شاخه و نام فایل ممکن است با نام منبع یکی باشد یا با آن متفاوت باشد.
- Mapfile برای تولید یک فایل با توسعه MAP. که بر ارتباط موقعیت و اندازه هر سگمنت و هر خطایی که در مرحله link رخ می‌دهد دلالت دارد. یک نوع خطا، اشتباه در تعریف یک سگمنت پشته است. وارد کردن CON (برای کنسول) به لینکر می‌گوید تا فایل map را بر روی صفحه نمایش دهد (علاوه بر آن روی دیسک می‌نویسد) بنابراین شما می‌توانید برای خطاها بلافاصله آن را ببینید.
- Libraryfile برای انتخاب کتابخانه فراهم شده است، که شما در این مرحله از برنامه نویسی اسمبلی بدان نیاز ندارید این مثال فایل مقصد A05ASM.OBJ که با اسمبل کردن تولید شده، لینک می‌کند. لینکر فایل EXE. را بر روی درایو D می‌نویسد، map را نمایش می‌دهد (CON) واز انتخاب کتابخانه صرفنظر می‌کند :

```
LINK D:A05ASM,D:,CON
```

اگر نام فایل با نام فایل منبع یکی باشد، نیازی به تکرار آن نیست ، ارجاع به درایو برای درخواست فایل کافی است. ضمیمه D بقیه انتخاب‌ها را تکمیل نموده است.

نقشه برای اولین برنامه

برای برنامه A05ASM1 ، LINK و TLINK هر دو این MAP را تولید می‌کنند :

START	STOP	LENGTH	NAME	CLASS
00000H	0003FH	0040H	STACKSG	STACK
00040H	00045H	0006H	DATASG	DATA
00050H	00063H	0014H	CODESG	CODE
program entry point at 0005:0000				

- پشته اولین سگمنت است و در افست 0 بایت از شروع برنامه آغاز می‌شود. زیرا ۳۲ کلمه تعریف می‌شود، ۶۴ بایت طول دارد همانطور که طول آن (40H) نشان می‌دهد.
- سگمنت داده در مرز پاراگراف بعدی از افست 40H آغاز می‌شود.
- سگمنت کد در مرز پاراگراف بعدی از افست 50H آغاز می‌شود. (برخی از اسمبلرها سگمنت‌ها را به ترتیب حروف الفبا مرتب می‌کنند).
- نقطه ورود به برنامه 0005:0000 که به شکل "افست : سگمنت مرتبط" به آدرس اولین دستور اجرایی اشاره می‌کند. در حقیقت، آدرس مرتبط شروع در سگمنت [0]5، افست 0 بایت می‌باشد، که با سگمنت کد از مرز 50H مطابقت دارد. بارگذاربرنامه وقتی برنامه را جهت‌اجرا در حافظه بارگذاری‌می‌کند،ازاین مقدار استفاده می‌کند دراین حالت تنها خطایی که ممکن است با آن مواجه شوید این است که نام فایل را اشتباه وارد کنید. راه حل شروع مجدد با فرمان Link است.

نقشه لینک برای برنامه دوم

START	STOP	LENGTH	NAME	CLASS
00000H	00013H	0014H	-TEXT	CODE
00014H	00019H	0006H	-DATA	DATA
00020H	0005FH	0040H	STACK	STACK
program entry point at 0000 : 0000				

نقشه لینک برای برنامه دوم (A05ASM2)، که از پیش پردازنده‌های سگمنت ساده شده استفاده می‌کند تنظیم اولیه متفاوتی با برنامه قبل دارد. اول، اسمبلر سگمنت‌ها را بطور فیزیکی بر اساس حروف الفبا مرتب نموده است، دوم، سگمنت‌های بعدی بر روی مرز **کلمه** مرتب شده‌اند، این نقشه لینک را ببینید.

- سگمنت کد هم اکنون اولین سگمنت است و در افست 0 بایت از شروع برنامه قرار دارد.
- سگمنت داده از مرز کلمه بعدی از افست 14H آغاز می‌شود.
- پشته از مرز کلمه بعدی از افست 20H آغاز می‌شود.
- نقطه ورود برنامه 0000:0000 کد به معنی موقعیت مرتبط با سگمنت کد با شروع از سگمنت 0، افست صفر می‌باشد.

اجرای یک برنامه

با داشتن یک برنامه اسمبل شده و لینک شده، می‌توان آن را اجرا کرد. اگر فایل EXE. در درایو پیش فرض باشد، باید بارگذار را برای خواندن آن بر روی حافظه جهت اجرا آماده کنید، با تایپ دستور زیر :

A05ASM1 (.EXE بدون توسعه) یا A05ASM1.EXE

اگر قسمت توسعه فایل را حذف کنید بارگذار آن را EXE. (یا COM.) فرض می‌کند. اما چون این برنامه هیچ خروجی قابل دیدنی تولید نمی‌کند ، به نظر می‌رسد بهتر باشد که با DEBUG آن را پیگیری و اجرا نمایید. فرمان زیر را با توسعه EXE وارد کنید :

DEBUG D :A05ASM1.EXE

DEBUG ماجول برنامه EXE. را بارگذاری کرده و اعلان خط فاصله را نشان می‌دهد .

برای دیدن سگمنت پشته، فرمان D SS:0 را وارد کنید، پشته حاوی صفر است چرا که این چنین مقدار دهی شده است.

برای دیدن سگمنت کد، فرمان D CS:0 را وارد کنید. کد ماشین‌های نمایش داده شده را با سگمنت کد در لیست اسمبل شده مقایسه کنید : DB ---- 8ED8A10000 . . .

این حالت، لیست اسمبل شده، کد ماشین را با دقت نشان نمی‌دهد، زیرا اسمبلر آدرس عملوند اولین دستور را نمی‌داند. می‌توانید این آدرس را با امتحان کردن کد نمایش داده شده تعیین کنید.

برای دیدن محتویات ثباتها، R و سپس (Enter) را بفشارید. SP (اشاره‌گر پشته) باید 40H که اندازه پشته است باشد (۳۲کلمه = ۶۴ بایت = 40H) IP (اشاره‌گر دستور) باید 0000H باشد. SS و CS باید به بروز صحیحی برای اجرا مقداردهی شوند، مقدار آنها به مکان حافظه‌ای کد برنامه در آن بارگذاری می‌شود بستگی دارد.

اولین دستور MOV AX,XXXX جهت اجرا آماده است. این دستور و MOV بعدی برای مقدار دهی ثبات DS هستند. برای اجرای اولین MOV ، T و سپس (Enter) را بفشارید. DS را چک کنید که با یک آدرس سگمنت مقدار دهی شده است.

سومین MOV محتویات FLDD را در AX بارگذاری می‌کند. بار دیگر T را بفشارید و توجه کنید که اکنون AX حاوی 00AF می‌باشد. حال T را بفشارید تا دستور ADD اجرا شود و بخاطر بسپارید که AX حاوی 0145 می‌باشد. T را بفشارید تا AX در افست 0004 از سگمنت داده ذخیره شود. برای تست محتویات سگمنت داده، فرمان D DS:0 را وارد کنید. عملیات سه عنصر داده را نمایش می‌دهد 0096004501 AF، که بایت‌ها، برای هر کلمه با ترتیب معکوس هستند. در این‌جا، با استفاده از فرمان L می‌توانید برنامه را مجدداً بارگذاری و اجرا کنید یا با فشردن Q در DEBUG خارج شوید.

لیست ارجاع متقابل

اسمبلر یک فایل CRF. یا XRF. انتخابی تولید می‌کند که شما در تولید یک لیست ارجاع متقابل از شناسه‌های برنامه، یا سمبول‌ها می‌توانید از آن استفاده کنید. اما شما هنوز باید فایل CRF. را به یک فایل ارجاع متقابل مرتب شده، درست تبدیل نمایید. یک برنامه در دیسک اسمبلر این عملیات را انجام می‌دهد. CREF برای مایکرو سافت یا TCREF برای بورلند. می‌توانید CREF یا TCREF را با یک فرمان خطی یا اعلان وارد کنید. این بخش از فرمان خطی استفاده می‌کند. ضمیمه D را برای استفاده از اعلان ببینید. فرمانی که فایل ارجاع متقابل را تبدیل می‌کند چنین است :

```
CREF/TCREF Xreffile,reffile
```

xreffile فایل ارجاع متقابل تولید شده توسط اسمبلر را مشخص می‌سازد. برنامه توسعه را خودش در نظر می‌گیرد ، بنابراین نیازی نیست آن را وارد نمائید. شما می‌توانید شماره دیسک درایو را مشخص نمائید.

reffile برای تولید فایل REF. از reffile استفاده می‌شود. مسیر و نام فایل هم می‌تواند با مسیر و فایل برنامه یکسان و هم متمایز باشد.

شکل ۵-۵ لیست ارجاع متقابل تولید شده با CREF برنامه شکل ۲-۵ را نشان می‌دهد. سمبول‌های اولین ستون به ترتیب الفبا هستند. اعداد دومین ستون که به شکل #n هستند شماره خط در فایل LST. که سمبول در آن خط تعریف شده است می‌باشد. اعداد سمت راست این ستون شماره‌خط‌هایی هستند که به هر سمبول در آن خطوط رجوع شده است. برای مثال CODESG در خط ۱۷ تعریف شده و در خط ۱۹ و ۲۹ به آن مراجعه شده است. FLDF در خط ۱۴ تعریف شده است و در خط +۲۵ به آن مراجعه شده است، '+' به معنی آن است که این مقدار در طی اجرای برنامه تغییر کرده است، (توسط MOV FLDF,AX) اسمبل کردن چندین برنامه فایلهای غیر ضروری تولید می‌کند. شما می‌توانید فایلهای LST. ، OBJ. و CRF. را حذف کنید، فایل ASM. منبع برنامه را، در صورتیکه تغییرات زیادی ایجاد کرده‌اید و فایلهای EXE. را جهت اجرا نگهدارید.

```
A05ASM1 (EXE)  Move and add operations

Symbol Cross-Reference  (# definition, + modification)

@CPU . . . . . .    1#
@VERSION . . . . .  1#

MAIN . . . . . .    18#   28    30

CODE . . . . . .    17
CODESG . . . . . .  17#   19    29

DATA . . . . . .    11
DATASG . . . . . .  11#   15    19    20

FLDD . . . . . .    12#   23
FLDE . . . . . .    13#   24
FLDF . . . . . .    14#   25+

STACK. . . . . .    4
STACKSG. . . . . .  4#    9     19


 12 Symbols
```

شکل ۵-۵ جدول ارجاع متقابل

```
 1                          page 60,132
 2                TITLE     A05ASM3 (EXE)  Illustrate assembly errors
 3                ; ---------------------------------------------------
 4 0000           STACKSG   SEGMENT PARA STACK 'Stack'
 5 0000 0020[               DW      32 DUP(0)
 6      0000
 7           ]
 8
 9 0040           STACKSG   ENDS
10                ; ---------------------------------------------------
11 0000           DATASG    SEGMENT PARA 'Data'
12 0000 00AF      FLDD      DW      175
13 0002 0096      FLDE      DW      150
14 0004           FLDF      DW
a05asm3.ASM(11): error A2027: Operand expected
15 0004           DATASG    ENDS
16                ; ---------------------------------------------------
17 0000           CODESG    SEGMENT PARA 'Code'
18 0000           BEGIN     PROC    FAR
19                          ASSUME  CS:CODESG,DS:DATASG
20 0000 A1 0000 U           MOV     AX,DATSEG          ;Address of data
a05asm3.ASM(17): error A2009: Symbol not defined: DATSEG
21 0003 8B D0               MOV     DX,AX              ;   segment in DS
22
23                          MOV     AS,FLDD            ;Move 0175 to AX
a05asm3.ASM(20): error A2009: Symbol not defined: AS
24 0005 03 06 0002 R        ADD     AX,FLDE            ;Add  0150 to AX
25 0009 A3 0000 U           MOV     FLDQ,AX            ;Store sum in FLDF
a05asm3.ASM(22): error A2009: Symbol not defined: FLDQ
26 000C B8 4C00             MOV     AX,4C00H           ;End processing
27 000F CD 21               INT     21H
28 0011           BEGIN     ENDP
a05asm3.ASM(25): error A2006: Phase error between passes
29 0011           CODESG    ENDS
30                          END     BEGIN

     0 Warning Errors
     5 Severe  Errors
```

شکل ۶-۵ تشخیص اسمبلر

تشخیص خطا

اسمبلر برای خطاهای برنامه نویسی که از قانون خود تخلف کرده‌اند، روشهای تشخیص فراهم نموده است برنامه شکل ۶-۵ همان برنامه شکل ۲-۵ است، بجز آنکه تعدادی خطای عمدی جهت توضیح در آن گنجانده شده است. برنامه تحت MASM یا TASM اجرا می‌شود که هر دو لیست خطاهای مشابهی تولید می‌کند.

خط توضیح

۱۴ تعریف FLDF یک عملوند نیاز دارد.

۱۹ پیش‌پردازنده ASSUME، SS رابه STACKSG مرتبط نکرده‌است، اگرچه اسمبلر این‌حذف راتشخیص‌نداده‌است.

۲۰ DATSEG باید DATASG می‌بود.

۲۱ DX باید DS کد شود، اگر چه اسمبلر نمی‌داند که این یک خطا است.

۲۳ AS باید AX کد شود.

۲۵ FLDQ باید FLDF کد شود.

۲۸ تصحیح خطاهای دیگر باعث حذف این تشخیص خواهد شد.

آخرین پیغام خطا "Phase errors between passes" وقتی رخ می‌دهد که آدرس تولید شده در گذر اول، اسمبلر دو گذره، با آنچه در گذر دو بدست آمده، متفاوت است. برای جداسازی خطاهای مبهم با MASM (قبل از نسخه 6.0) از انتخاب D/ برای لیست کردن هم فایل‌های گذر یک و گذر دو و مقایسه آدرس‌های افست استفاده کنید.

شماره‌گر موقعیت اسمبلر

اسمبلر یک ***شماره‌گر موقعیت*** دارد که برای حسابرسی هر عنصر تعریف شده در سگمنت داده استفاده می‌شود. شکل ۵-۵ و ۵-۳ تاثیر آن را توسط سه عنصر داده تعریف شده مشخص می‌کند.

```
0000  . . .  FLDD  DW
0002  . . .  FLDE  DW
0004  . . .  FLDF  DW
```

در ابتدا، شماره‌گر موقعیت با صفر تنظیم می‌شود، وقتی اسمبلر اولین عنصر داده FLDD را تشخیص می‌دهد از آنجائیکه FLDD یک کلمه تعریف شده است، اسمبلر شمارشگر موقعیت را ۲ تا می‌افزاید و مقدار آن 0002 خواهد شد، جایی که FLDE قرار دارد. از آنجائیکه FLDE نیز یک کلمه تعریف شده است، اسمبلر مجدداً ۲ تا افزایش می‌یابد، برای عنصر داده بعدی FLDE نیز یک کلمه است. شمارشگر موقعیت مجدداً ۲ تا افزایش می‌یابد و مقدار آن 0006 خواهد شد، اما عنصر داده دیگری وجود ندارد.

اسمبلر روشهایی برای تغییر مقدار جاری شمارشگر موقعیت دارد. برای مثال، با استفاده از پیش‌پردازنده EVEN یا ALIGN بسادگی می‌توان یک آدرس را از مرز عدد زوج قرار داد (فصل ۶)، یا با استفاده از پیش‌پردازنده ORG یک برنامه از یک افست خاص (فصل ۷) آغاز می‌شود یا عناصر داده را با نامهایی متفاوت مجدداً تعریف نمود (فصل ۲۶).

نکات کلیدی

- MASM و TASM هر دو یک فرمان خطی برای اسمبل کردن که شامل (در انتها) نام برنامه منبع می‌باشد، دارند. MASM برای ورود برخی انتخابها نیز اعلان فراهم نموده است.
- اسمبلر یک برنامه منبع را به یک فایل OBJ. تبدیل می‌کند و یک لیست‌انتخابی و فایل‌ارجاع متقابل نیز تولیدمی‌کند.
- سگمنت‌ها و جدول گروهها در انتهای لیست اسمبل شده بیانگر سگمنت‌ها و گروه‌های تعریف شده در برنامه است. جدول سمبول‌ها همه سمبول‌ها را نشان می‌دهد (نام داده‌ها و برچسب‌های دستورات).

- لینکر (TLINK یا LINK) یک فایل OBJ. را به فایل EXE. اجرایی تبدیل می‌کند.
- پیش پردازنده‌های سگمنت ساده نام DATA- را برای سگمنت داده، STACK را برای سگمنت پشته، و TEXT- را برای سگمنت کد تولید می‌کنند که همه از پیش تعریف شده‌اند.
- برنامه CREF یا(TCREF) یک لیست ارجاع متقابل مفید تولید می‌کند.

پرسش‌ها

۵-۱. فرمان خطی بنویسید که اسمبلر یک برنامه منبع با نام SQUEEZE.ASM با فایلهایی CRF. ، OBJ. و LST. اسمبل نماید. فرض کنید که برنامه منبع و اسمبلر همه در درایو E هستند.

۵-۲. از فرمان خطی LINK یا TLINK برای لینک کردن SQUEEZE.OBJ از پرسش ۵-۱ استفاده کنید.

۵-۳. فرمان خطی برای SQUEEZE.EXE از پرسش ۵-۲ را برای انجام کارهای زیر کد کنید : (الف) اجرای مستقیم از DOS، (ب) اجرا از طریق DEBUG.

۵-۴. هدف هر یک از فایلهای زیر را توضیح دهید : (الف) file.EXE ، (ب) file.OBJ ، (ج) file.MAP ، (د) file.ASM ، (ه) file.CRF ، (و) file.LST.

۵-۵. فرض کنید از تعریف سگمنت معمول استفاده شده و DATSEGM نام سگمنت داده است، دو دستور MOV برای مقداردهی ثبات DS را کد کنید.

۵-۶. یک برنامه اسمبلی با استفاده از تعریف معمول سگمنت برای منظورهای زیر بنویسید : (الف) انتقال مقدار 50 شانزدهی به ثبات AL، (ب) شیفت محتویات AL یک بیت به سمت چپ (SHL AL,1) (ج) انتقال مقدار 18H به CL ، (د) ضرب AL در CL (MUL CL). دستورات لازم برای خاتمه اجرای برنامه را بخاطر داشته باشید. برنامه نیازی به تعریف یا مقدار دهی سگمنت داده ندارد. یک اسکلت برنامه را کپی کنید و با استفاده از ویرایشگر آن را توسعه بدهید. آن را اسمبل کنید، لینک کنید و با استفاده از DEBUG سگمنت کد و ثباتها را پیگیری و تست نمایید.

۵-۷. برنامه پرسش ۵-۶ را با استفاده از پیش پردازنده‌های تعریف ساده سگمنت بنویسید. آن را اسمبل کنید، لینک کنید و کد مقصد، جدول‌های سمبول و نقشه لینک را با برنامه اصلی مقایسه کنید.

۵-۸. یک سگمنت داده با برنامه پرسش ۵-۶. برای منظورهای زیر بیفزائید :

- عنصر ۱ بایتی (DB) با نام FIELDX حاوی 50H و فیلد دیگری با نام FIELDY حاوی 18H تعریف کنید.
- عنصر ۲ بایتی (DW) با نام FIELDZ حاوی هیچ مقداری تعریف کنید.
- محتویات FIELDX را به ثبات AL منتقل کنید و آن را یک بیت به سمت چپ شیفت بدهید.
- AL را در FIELDY ضرب کنید (MUL FIELDY)
- حاصل که در AX است به FIELDZ منتقل کنید.

برنامه را اسمبل کنید، لینک کنید و با استفاده از DEBUG آن را تست نمایید.

۵-۹. برنامه پرسش ۵-۸ را با استفاده از تعریف ساده سگمنت بنویسید آن را اسمبل کنید و لینک کنید و کد مقصد، جداول سمبول و نقشه لینک آن را با برنامه اصلی مقایسه کنید. از DEBUG برای تست برنامه استفاده کنید.

۵-۱۰. برای هر یک از عناصر داده زیر ، محتویات شمارشگر موقعیت را نشان دهید :

```
0000    WORDA   DW   0
....    WORDB   DW   0
....    BYTEA   DB   0
                EVEN
....    WORDC   DW   0
....    BYTEB   DB   0
```

فصل ۶

دستورات سمبولیک و آدرس‌دهی

هدف : فراهم سازی مبانی مجموعه دستورات زبان اسمبلی و موارد ضروری برای آدرس دهی داده‌ها.

مقدمه

این فصل مقدمه‌ای از مجموعه دستورات پردازشگر خلاصه شده است و سپس مبنای شکل آدرسدهی مورد استفاده تا انتهای کتاب را توضیح می‌دهد. دستوراتی که این فصل در بر دارد بشرح زیر هستند :
MOV ، MOVSX ، MOVZX ، XCHNG ، LEA ، INC ، DEC و INT و نحوه استفاده از مقادیر ثابت در عملوند دستورات به شکل مقادیر بلافصل. در انتها، این فصل قرار گرفتن آدرسها و مقدمه نوشتن بر روی سگمنت‌ها را توضیح می‌دهد.

مجموعه دستورات سمبولیک

در زیر لیستی است از دستورات سمبولیک برای خانواده پردازشگرهای 80X86که بر اساس خلاصه آنها مرتب شده است. اگر چه این لیست سودمند به نظر می‌رسد ، خیلی از دستورات به ندرت لازم می‌شوند.

محاسباتی

ADC: جمع با نقلی
ADD: جمع اعداد دودویی
DEC:کاهش یک
DIV: تفریق بدون علامت
IDIV: تقسیم (صحیح) علامتدار
IMUL: ضرب (صحیح) علامتدار
INC: افزایش یک
MUL: ضرب بدون علامت
NEG: منفی کردن
SBB: تفریق با قرض
SUB: تفریق مقادیر دودویی
XADD: معاوضه و جمع

تبدیل BCD - ASCII

AAA: تنظیم ASCII بعد از جمع
AAD: تنظیم ASCII قبل از تقسیم
AAM: تنظیم ASCII بعد از ضرب
AAS: تنظیم ASCII بعد از تفریق
DAA: تنظیم دهدهی بعد از جمع
DAS: تنظیم دهدهی بعد از تفریق

شیفت بیت‌ها

RCL: چرخش به چپ با نقلی
RCR: چرخش به راست با نقلی
ROL: چرخش به چپ
SAR: شیفت جبری به راست
SHL: شیفت منطقی به چپ
SHR: شیفت منطقی به راست

ROR: چرخش به راست
SAL: شیفت جبری به چپ
SHLD: شیفت به چپ دو بار (+80386)
SHRD: شیفت به راست دو بار(+80386)

مقایسه

BSR/BSF: پیمایش بیت (+80386)
BT/BTC/BTR/BTS: تست بیت (+80386)
CMP: مقایسه
CMPSN: مقایسه رشته‌ها
CMPXCHG: مقایسه و معاوضه (+486)
CMPXCHG8B: مقایسه و معاوضه (+586)
TEST: تست بیت‌ها

انتقال داده

LDS: بارگذاری ثبات سگمنت داده
LEA: بارگذاری آدرس مؤثر
LES: بارگذاری ثبات سگمنت اضافی
LODS: بارگذاری رشته
LSS: بارگذاری ثبات سگمنت پشته
MOV: انتقال داده
MOVS: انتقال رشته
MOVSX: انتقال با توسعه علامت
MOVZX: انتقال با توسعه صفر
STOS: ذخیره رشته
XCHG: معاوضه
XLAT: ترجمه

عملیات پرچم

CLC: پاک کردن پرچم نقلی
CLD: پاک کردن پرچم جهت
CLI: پاک کردن پرچم وقفه
CMC: متمم پرچم نقلی
LAHF: ذخیره پرچم در AH
PUPF: برداشتن پرچم از پشته
PUSHF: گذاشتن پرچم‌ها بر روی پشته
SAHF: ذخیره AH در پرچم‌ها
STC: یک گذاشتن در پرچم نقلی
STD: یک گذاشتن در پرچم جهت
STI: یک گذاشتن در پرچم وقفه

ورودی / خروجی

IN: وارد کردن کلمه یا بایت
INSn: وارد کردن رشته (+80286)
OUT: خارج کردن کلمه یا بایت
OUTSn: خارج کردن رشته (+80286)

عملیات منطقی

AND: AND منطقی
OR: OR منطقی
NOT: NOT منطقی
XOR: OR انحصاری

حلقه

LOOP: تکرار حلقه تا تکمیل شدن
LOOPE: تکرار حلقه تا مساوی شدن
LOOPZ: تکرار حلقه تا صفر شدن
LOOPNE: تکرار حلقه تا مساوی نشدن
LOOPNZ: تکرار حلقه تا صفر نشدن
LOOPNEW: تکرار حلقه تا مساوی نشدن(+80386)
LOOPNZW: تکرار حلقه تا صفر نشدن(+80386)

کنترل پردازشگر

ESC: گریز
NOP: هیچ عمل
HLT: وارد کردن حالت Halt
WAIT: قرار دادن پردازشگر در حالت انتظار
LOCK: قفل گذرگاه

عملیات پشته

POP: برداشتن کلمه از پشته
PUSH: گذاشتن کلمه بر پشته
POPA: برداشتن همه ثبات‌های عمومی از پشته (+80286)
PUSHA: گذاشتن همه ثبات‌های عمومی بر پشته (+80286)

عملیات رشته‌ای

CMPS: مقایسه رشته
LODS: بارگذاری رشته
MOVS: انتقال رشته
REP: تکرار رشته
REPE: تکرار تا مساوی است
REPZ: تکرار تا صفر است
REPNE: تکرار تا مساوی نیست
REPNZ: تکرار تا صفر نیست
SCAS: پیمایش رشته
STOS: ذخیره رشته

پرش (شرطی)

INTO: وقفه در صورت سرریز
JA: پرش اگر بیشتر است
JAE: پرش اگر بیشتر یا مساوی است
JB: پرش اگر کمتر است JNGE:
JBE: پرش اگر کمتر یا مساوی است
JC: پرش اگر نقلی وجود دارد
JCXZ: پرش اگر CX صفر است
JE: پرش اگر مساوی است
JG: پرش اگر بزرگتر است
JGE: پرش اگر بزرگتر نیست
JL: پرش اگر کوچکتر است
JLE: پرش اگر کوچکتر یا مساوی است
JNA: پرش اگر بیشتر نیست
JNAE: پرش اگر بیشتر یا مساوی نیست
JNB: پرش اگر کمتر نیست
JNBE: پرش اگر کمتر یا مساوی نیست
JNC: پرش اگر نقلی وجود ندارد
JNE: پرش اگر مساوی نیست
JNG: پرش اگر بزرگتر نیست
پرش اگر بزرگتر یا مساوی نیست
JNL: پرش اگر کوچکتر نیست
JNLE: پرش اگر کوچکتر یا مساوی نیست
JNO: پرش اگر سرریز وجود ندارد
JNP: پرش اگر توازن وجود ندارد
JNS: پرش اگر علامت ندارد
JNZ: پرش اگر صفر نیست
JO: پرش اگر سرریز ندارد
JP: پرش اگر توازن فرد است
JPE: پرش اگر توازن زوج است
JPO: پرش اگر توازن فرد است
JS: پرش اگر علامت دارد
JZ: پرش اگر صفر است

پرش (بدون شرط)

CALL: فراخوانی یک زیر
INT: وقفه
IRET: بازگشت از وقفه
JMP: پرش بدون شرط
RET: بازگشت
RETN/RETF: بازگشت به نزدیک / بازگشت به دور

تبدیل نوع

CBW: تبدیل بایت به کلمه
CWD: تبدیل کلمه به دو کلمه
CWQ: تبدیل دوکلمه به چهار کلمه (+80386)
CWDE: تبدیل دو کلمه به دو کلمه توسعه یافته (+80386)

عملوندهای دستورات

یک عملوند منبع داده برای یک دستور است، برخی دستورات مانند CLC یا RET نیازی به عملوند ندارند، در حالیکه بقیه دستورات یک یا دو عملوند دارند. وقتی دو عملوند وجود دارد، عملوند دوم منبع داده‌ای است که باید حمل شود (بلافصل) یا آدرس (از یک ثبات یا در حافظه) داده می‌باشد. داده منبع توسط عملیات تغییر نمی‌یابد. عملوند اول مقصد داده‌ای یا حافظه‌ای است که باید پردازش شود. شکل دستور در اینجا ذکر شده است :

[lable:]	عملیات	عملوند ۲ و عملوند ۱

حال بیایید چگونگی تاثیر گذاری عملوند بر آدرسدهی داده را امتحان کنیم.

عملوندهای ثبات

برای این نوع، ثبات می‌تواند نام هر یک از ثبات‌های ۸، ۱۶ یا ۳۲ بیتی باشد. وابسته به دستور ثبات ممکن است در اولین عملوند، دومین عملوند، یا هر دو قرار گیرد، همانطور که مثالهای زیر مشخص می‌سازد.

```
WORDA   DW    ?          ; تعریف یک کلمه
        ...
        MOV   DX,WORDA   ; ثبات در اولین عملوند
        MOV   WORDA,CX   ; ثبات در دومین عملوند
        MOV   EDX,EBX    ; ثبات در دو عملوند
```

از آنجائیکه پردازش داده بین ثبات‌ها، هیچ ارجاعی به حافظه ندارد، سریع‌ترین نوع عملیات است.

عملوندهای بلافصل

در شکل بلافصل دومین عملوند حاوی مقدار ثابت یا یک عبارت است. (اولین عملوند نمی‌تواند یک مقدار بلافصل باشد.) فیلد مقصد در اولین عملوند طول داده را تعریف می‌کند و ممکن است یک ثبات یا یک موقعیت حافظه باشد. در اینجا چند مثال آورده شده است :

```
COUNT   DB    ?          ; تعریف یک کلمه
        ...
        ADD   BX,25      ; جمع با ۲۵ با BX
        MOV   COUNT,50   ; انتقال ۵۰ به COUNT
```

یکی از بخش‌های بعدی عملوند بلافصل را با جزئیات بیشتر توضیح می‌دهد.

عملوندهای مستقیم حافظه

در این شکل، یکی از عملوندها به یک موقعیت حافظه رجوع می‌کند و دیگری به یک ثبات رجوع می‌کند. (تنها دستوراتی که اجازه می‌دهند هر دو عملوند آدرس مستقیم حافظه باشید MOVS و CMPS می‌باشند).

ثبات DS ثبات سگمنت پیش فرض برای آدرسدهی داده‌ها در حافظه است در اینجا چند مثال آورده شده است.

```
WORDA   DW    0            ; تعریف یک کلمه
BYTEA   DB    0            ; تعریف یک بایت
        ...
        MOV   BX,WORDA     ; بارگذاری WORDA در AX
        ADD   BYTEA,DL     ; جمع DL با BYTEA
        MOV   CX,DS :[38B0H] ; انتقال کلمه حافظه در افست 38B0H
        INC   BYTE PTR [1B0H] ; افزایش بایتی که در افست 1B0H قرار دارد
```

در دو دستور آخر از کروشه بعنوان ***شاخص نشانه‌گذاری*** که مبنی بر ارجاع به حافظه است، استفاده شده است. (یک مقدار تفاوت مکان مثل 38B0H با آدرس DS ترکیب می‌شود). حذف کروشه، مثلاً به این شکل MOV CX,38B0H ، بر یک مقدار بلافصل دلالت دارد - به تفاوت عینی این دو توجه کنید. آخرین مثال بایت حافظه در تفاوت مکان 1B0H را افزایش می‌دهد (تفاوت مکان با آدرس DS ترکیب می‌شود). چون عملوند [1B0H] بر یک نقطه شروع از حافظه دلالت دارد، به BYTE PTR احتیاج دارید تا طول را نیز تعریف کنید.

در مثالهای زیر، یک عنصر داده با نام CODETBL مانند یک آدرس افست در یک عملوند دستور عمل می‌کند :

```
CODETBL DB    DUP (?)      ; تعریف یک جدول از بایت‌ها
        ...
        MOV   CL,CODETBL [3] ; گرفتن بایت از CODETBL
        MOV   CL,CODETBL +3  ; همان عملیات
```

اولین MOV از یک شاخص نشانه‌گذاری برای دستیابی به چهارمین بایت CODETBL استفاده می‌کند. (زیرا CODETBL[0] اولین بایت است، لذا CODETBL [3] چهارمین بایت خواهد بود). دومین دستور MOV از عملگر(+) برای ایجاد همان تاثیر استفاده می‌کند.

عملوندهای غیر مستقیم حافظه

آدرسدهی غیر مستقیم یک تکنیک هوشمندانه است که مزیت قابلیت‌های کامپیوتر برای آدرسدهی تفاوت مکان، سگمنت را ارائه می‌دهد. ثباتهایی که برای این منظور استفاده می‌شوند BP ، SI ، DI و BX هستند که در داخل کروشه بعنوان یک عملگر نشانگذاری استفاده می‌شوند. ثبات‌های BX ، DI و SI با ثبات DS برای پردازش داده‌های سگمنت داده به شکل DS:BX و DS:DI و DS:SI در ارتباط هستند. ثبات BP با ثبات SS به شکل SS:BP در ارتباط است، برای دستکاری داده‌های پشته که در فصل ۲۳ وقتی فراخوانی زیر برنامه‌ها و عبور پارامترها را توضیح می‌دهیم این کار را انجام می‌دهیم.

وقتی اولین عملوند حاوی یک آدرس غیر مستقیم باشد ، دومین ثبات حاوی یک مقدار بلافصل یا ثبات خواهد بود، وقتی دومین عملوند حاوی یک آدرس غیر مستقیم باشد ، اولین عملوند به یک ثبات رجوع می‌کند. یک آدرس غیر مستقیم مانند [DI] به اسمبلر می‌گوید که آدرس حافظه برای استفاده در ثبات DI است در مثال زیر، اولین BX , MOV را با آدرس تفاوت مکان DATAVAL مقدار دهی می‌کند. دومین MOV، با استفاده از آدرس BX در موقعیت حافظه که به آن اشاره می‌کند صفر ذخیره می‌سازد، در این حالت :

```
DATAVAL   DB    50                    ; تعریف یک بایت
          ...
          MOV   BX,OFFSET DATAVAL     ; BX با افست بارگذاری می‌شود
          MOV   [BX],25               ; انتقال ۲۵ به DATAVAL
```

تاثیر این دو دستور MOV مانند کد کردن MOV DATAVAL,25 است. اگر چه استفاده برای آدرسدهی نشانگذاری چنین جزئی نیست. دستور بعدی صفر را به موقعیت ۳ بایت بعد از DATAVAL منتقل می‌کند :

```
MOV [BX+2],0  ; DATAVAL +2 انتقال صفر به
```

شما می‌توانید ثبات‌ها را در یک آدرس غیر مستقیم ترکیب کنید ، برای مثال [BX+DI] به معنی آدرس داخل BX بعلاوه آدرس داخل DI می‌باشد.

توجه کنید که کروشه با ثبات BP ، BX ، DI یا SI بمعنی عملوند غیر مستقیم است و پردازشگر در هنگام اجرای برنامه با محتویات ثبات مانند یک آدرس تفاوت مکان رفتار می‌کند. در اینجا چند مثال دیگر از عملوندهای غیر مستقیم آورده شده است :

```
ADD   CL,[BX]          ;  DS:BX = دومین عملوند
MOV   BYTEPTR [DI],25  ;  DS :DI = اولین عملوند
ADD   [BP],CL          ;  SS : BP = اولین عملوند
```

جانشین سازی آدرس. این روش از یک آدرس جانشین سازی برای عملوند استفاده می‌کنند. دستور MOV زیر محتویات BL را به DATAB (یک جدول ۴۰ بایتی) منتقل می‌کند، دقیقاً در مکانی از DATAB که با محتویات DI مشخص شده است :

```
DATATAB  DB    40DUP(?)          ; تعریف جدول
         ...
         MOV   DATATAB [DI],BL   ; BL انتقال به جدول
```

نشان گذاری در 80386 و پردازشگرهای بعدی. این پردازشگرها آدرسی را که با هر ترکیبی از یک یا چندین ثبات عمومی، یک افست و یک ضریب مقدار(۱ و ۲ و ۴ یا ۸) مرتبط با محتویات یکی از ثبات‌ها تولید شده باشد، مجاز می‌داند. برای مثال دستور

```
MOV EBX,[ECX*2+ESP+4]
```

مقداری را به EBX منتقل می‌کند که شامل محتویات (۲ ضرب ECX) بعلاوه محتویات (۴ بعلاوه ESP) باشد.

دستورات MOV

(دستورات MOV) داده ارجاع شده توسط آدرس دومین عملوند را به آدرس اولین عملوند منتقل (کپی) می‌کنند. دومین فیلد بدون تغییر می‌ماند. عملوندهایی که به حافظه یا ثبات رجوع می‌کند در اندازه باید یکی باشند. (هر دو می‌توانند بایت ، هر دو کلمه، هر دو دوکلمه‌ای باشند). قالب کلی دستور MOV چنین است.

[Label:]	MOV	register/memory,register/memory/immediate

در اینجا چهار مثال از عملیات خلاصه شده MOV معتبر آورده شده است عناصر داده نیز داده شده است :

```
BYTEFLD   DB   ?    ;   تعریف یک بایت
WORDFLD   DW   ?    ;   تعریف یک کلمه
```

۱. انتقال ثبات

```
MOV   EDX,ECX       ;   ثبات به ثبات
MOV   DS,BX         ;   ثبات به ثبات سگمنت
MOV   BYTEFLD,DH    ;   ثبات به حافظه - مستقیم
MOV   [DI],BX       ;   ثبات به  حافظه - غیر مستقیم
```

۲. انتقال بلافصل

```
MOV   CX,40         ;   بلافصل به ثبات
MOV   BYTEFLD,40    ;   بلافصل به حافظه - مستقیم
MOV   WORDFLD[BX],40 ;  بلافصل به‌حافظه - غیرمستقیم
```

۳. انتقال حافظه مستقیم

```
MOV   CH,BYTEFLD    ;   حافظه به ثبات - مستقیم
MOV   CX,WORDFLD[BX] ;  حافظه به ثبات - غیرمستقیم
```

۴. انتقال ثبات سگمنت

```
MOV   CX,DS         ;   ثبات سگمنت به ثبات
MOV   WORDFLD,DS    ;   ثبات سگمنت به حافظه
```

می‌توان به یک ثبات یک بایت (MOV CH,BYTEFLD) یک کلمه (MOV CX,WORDFLD) یک دو کلمه (MOV ECX,DWORDFLD) منتقل نمود. عملوند فقط بر قسمتی از ثبات که ذکر شده تاثیر می‌گذارد، مثلاً انتقال یک بایت به CH بر CL تاثیری ندارد.

از عملیات MOV غیر معتبر حافظه به حافظه (این را بخاطر بسپارید)، بلافصل به ثبات سگمنت، و ثبات سگمنت به ثبات سگمنت می‌باشد، انجام این عملیات بیشتر از یک دستور نیاز دارد.

دستورات انتقال و پرکردن

برای دستورات MOV باید مبدأ و مقصد اندازه یکسانی داشته باشند، مثل بایت به بایت و کلمه به کلمه در پردازشگرهای 80386 و بالاتر با دستورات MOVSX و MOVZX (انتقال و پرکردن) انتقال داده از یک بایت یا کلمه مبدأ به یک کلمه یا دو کلمه مقصد، براحتی انجام می‌پذیرد. قالب کلی برای MOVSX و MOVZX چنین است :

[label:]	MOVSX/MOVZX	register/memery,register/memory/immediate

MOVSX برای مقادیر محاسباتی علامتدار استفاده می‌شود، یک بایت یا کلمه را به یک کلمه یا دو کلمه منتقل می‌کند و با **بیت علامت** (سمت چپ‌ترین بیت مبدأ) سمت چپ بیت‌های مقصد را پر می‌کند.

MOVZX برای مقادیر عددی بدون علامت استفاده می‌شود، یک بایت یا یک کلمه را به یک کلمه یا دو کلمهٔ مقصد منتقل می‌کند، و با **بیت صفر** سمت چپ مقصد را پر می‌کند. بعنوان مثال ، یک بایت حاوی 10110000 را به یک کلمه منتقل می‌کند، نتیجه در مقصد به نوع دستور بستگی دارد.

```
MOVSX    CX,10110000 B  ; CX = 1111111110110000
MOVZX    CX,10110000 B  ; CX = 0000000010110000
```

در اینجا چند مثال دیگر از استفاده MOVSX و MOVZX آورده شده است :

```
BYTE1   DB      25          ; بایت
WORD1   DW      40          ; کلمه
DWORD1  DD      160         ; دو کلمه
        . . .
        MOVSX   CX,BYTE1    ; بایت به کلمه
        MOVZX   WORD1,BH    ; بایت به کلمه
        MOVSX   EBX,WORDI   ; کلمه به دوکلمه
        MOVZX   DWORD1,CX   ; کلمه به دوکلمه
```

فصل ۸ و ۱۳ داده‌های علامتدار و بدون علامت را با جزئیات دربر دارد.

عملوندهای بلافصل

در مثال زیر یک عملوند بلافصل آورده شده است :

```
MOV AX,0245H
```

در این دستور مقدار 0245H به AX منتقل می‌شود. سه بایت کد مقصد برای این دستور B84502 است،که B8 به معنی، انتقال یک مقدار بلافصل به ثبات AX است و دو بایت بعدی حاوی مقدار است (4502H، به ترتیب بایت معکوس است). خیلی از دستورات دو عملوند دارند. اولین عملوند ثبات به موقعیت حافظه و دومین عملوند یک ثابت بلافصل است.

استفاده از یک عملوند بلافصل پردازش سریع‌تری به نسبت تعریف یک ثابت عددی در سگمنت داده و ارجاع آن به عنوان یک عملوند MOV، خواهد داشت بعنوان مثال :

```
سگمنت داده :  AMTC  DW    0245H     ; تعریف AMTC بعنوان کلمه
              . . .
سگمنت کد :   MOV   AX,AMTC  ; انتقال AMTC به AX
```

قالب‌های بلافصل

در اینجا چند مثال از ثابت بلافصل معتبر آورده شده است :

```
شانزدهی :  0148H
دهدهی :    328 (که اسمبلر آن را به 0148H تبدیل می‌کند)
دودویی :   101001000 B (که به 0148H تبدیل خواهد شد)
```

طول عملوندهای بلافصل

طول یک ثابت بلافصل نمی‌تواند از طول تعریف شده با اولین عملوند تجاوز نماید. در مثال‌های نامعتبر زیر، عملوند بلافصل ۲ بایت است، در حالیکه ثبات AL فقط یک بایت است :

```
MOV  CL,0245H ; طول بلافصل نامعتبر است
```

اما اگر عملوند بلافصل کوتاه‌تر از عملوند دیگر باشد، به این شکل :

طول معتبر ; ADD CX,48H

اسمبلر عملوند بلافصل را به ۲ بایت توسعه می‌دهد، و در کد مقصد به شکل 4800H ذخیره می‌سازد. پردازشگرهای 80386 و بعد، عملوندهای بلافصل ۴ بایتی (دو کلمه‌ای) را نیز انجام می‌دهند مثلاً :

انتقال دو کلمه ; MOV ECX,12345678H

```
        page  60,132
TITLE   A06IMMED (EXE)  Example of immediate operands
        .MODEL  SMALL
        .STACK  64              ;Define stack
        .DATA                   ;Define data segment
FLDX    DB      150
FLDY    DW      300
.386
        .CODE
MAIN    PROC    FAR
        MOV     AX,@data        ;Set address of data
        MOV     DS,AX           ;   segment in DS
        MOV     CX,325          ;Move immediate
        ADD     CX,150          ;Add immediate
        MOV     EDX,0           ;Move immediate (80386+)
        ADD     DX,20H          ;Add immediate (hex)
        SUB     FLDX,250        ;Subtract immediate
        MOV     FLDY,40H        ;Move immediate
        MOV     AX,4C00H        ;End processing
        INT     21H
MAIN    ENDP
        END     MAIN
```

شکل ۱-۶ استفاده از عملوندهای بلافصل

شکل ۱-۶ مثالهایی از MOV ، ADD و SUB می‌باشند که این سه از دستوراتی هستند که عملوند بلافصل را مجاز می‌دانند. پیش پردازنده 386. مجوز تشخیص ارجاع به ثبات EDX را به اسمبلر می‌دهد. شما به یک پردازشگر 80386 یا بعد از آن برای اسمبل کردن این جمله نیاز ندارید اما به یکی برای اجرای آن نیاز دارید ؟ پردازش عناصر داده‌ای که از ظرفیت یک ثبات تجاوز می‌نمایند، کد نویسی نیاز دارد که فصل بعد حاوی این مطالب است.

دستور XCHG

دستور XCHG نوع دیگری از انتقال داده را انجام می‌دهد، اما چیزی بیشتر از کپی کردن یک داده از یک موقعیت به موقعیت دیگر است، XCHG دو عنصر داده را ***جابجا*** می‌کند. قالب کلی دستور XCHG چنین است :

[label:]	XCHG	register/memory,register/memory

عملیات XCHG معتبر ، شامل تعویض داده در ثبات و بین یک ثبات و حافظه است. در اینجا دو مثال آورده شده است

```
WORDQ   DW      ?               ; تعریف کلمه
        ...
        XCHG    CL,BH           ; تعویض محتویات دو ثبات
        XCHG    CX,WORDQ        ; تعویض محتویات ثبات و حافظه
```

دستور LEA

دستور LEA برای مقدار دهی ثبات با یک آدرس افست مفید است. قالب کلی دستور LEA چنین است :

[label:]	LEA	register,memory

یک مورد استفاده عمومی برای LEA جهت مقدار دهی یک افست در ثبات BX و DI و SI برای نشانگذاری آدرس

در حافظه است، که در این کتاب انجام خواهد شد. در اینجا یک مثال آورده شده است.

```
DATATBL  DB    25DUP(?)        ; تعریف یک جدول
BYTEFLD  DB    ?
         ...
         LEA   BX,DATATBL      ; بارگذاری آدرس افست
         MOV   BYTEFLD,[BX]    ; انتقال اولین بایت از DATABL
```

یک عملیات مشابه LEA، دستور MOV با عملگر OFFSET می‌باشد ، که کد ماشین کوتاه‌تری تولید می‌کند و چنین استفاده می‌شود : `MOV BX,OFFSET DATATBL ; بارگذاری آدرس افست`

دستورات INC و DEC

INC و DEC، دستوراتی مرسوم برای افزایش و کاهش محتویات یک واحد از ثبات و موقعیت حافظه می‌باشند. قالب کلی INC و DEC چنین است.

[label:]	INC/DEC	register/memory

توجه کنید که INC و DEC فقط یک عملوند دارند. بسته به نتیجه، عملیات پرچم‌های OF ، SF و ZF را صفر یا یک می‌نماید. که در دستورات شرطی برای منفی، صفر ، یا مثبت تست می‌شوند.

```
        page  60,132
TITLE   A06MOVE (EXE)  Extended move operations
;-----------------------------------------------------------
        .MODEL  SMALL
        .STACK  64
;-----------------------------------------------------------
        .DATA
HEADG1  DB      'InterTech'
HEADG2  DB      'LaserCorp', '$'
;-----------------------------------------------------------
        .CODE
A10MAIN PROC    FAR
        MOV     AX,@data        ;Initialize segment
        MOV     DS,AX           ;  registers
        MOV     ES,AX

        MOV     CX,09           ;Initialize to move 9 chars
        LEA     SI,HEADG1       ;Initialize address of HEADG1
        LEA     DI,HEADG2       ;    and HEADG2
A20:
        MOV     AL,[SI]         ;Get character from HEADG1,
        MOV     [DI],AL         ;  move it to HEADG2
        INC     SI              ;Incr next char in HEADG1
        INC     DI              ;Incr next pos'n in HEADG2
        DEC     CX              ;Decrement count for loop
        JNZ     A20             ;Count not zero? Yes, loop
                                ;Finished
        MOV     AH,09H          ;Request display
        LEA     DX,HEADG2       ;  of HEADG2
        INT     21H

        MOV     AX,4C00H        ;End processing
        INT     21H
A10MAIN ENDP
        END     A10MAIN
```

شکل ۲-۶ عملیات انتقال توسعه یافته

عملیات انتقال توسعه یافته

تا اینجا، برنامه‌ها داده‌های بلافصل را به یک ثبات منتقل می‌کنند داده‌ها را از حافظه تعریف شده به یک ثبات منتقل می‌کنند، محتویات ثبات را به حافظه منتقل می‌کنند و محتویات یک ثبات را به دیگری منتقل می‌کنند. در همه حالات

طول داده به یک یا دو بایت محدود می‌شود و هیچ عملیاتی داده‌ها را از یک ناحیه حافظه مستقیماً به ناحیه دیگری از حافظه منتقل نمی‌کنند. این بخش چگونگی انتقال داده‌هایی را که از ۲ بایت تجاوز می‌کنند توضیح می‌دهد. روش دیگر استفاده از دستورات رشته‌ای است که فصل ۱۲ آنها را در بر دارد.

در برنامه شکل ۲-۶، سگمنت داده حاوی دو فیلد ۹ بایتی تعریف شده با عنوان HEADG1 و HEADG2 است. هدف برنامه انتقال محتویات HEADG1 به HEADG2 است.

```
HEADG : 1   I  n  t  e  r  t  e  c  h
                ↓  ↓  ↓  ↓  ↓  ↓  ↓  ↓  ↓
HEADG : 2   I  a  s  e  r  c  r  o  p
```

از آنجائیکه این فیلدها ۹ بایت طول دارند، بیشتر از یک دستور MOV ساده مورد نیاز است. این برنامه حاوی چند نکته جدید است :

به منظور انتقال مرحله به مرحله HEADG1 به HEADG2، برنامه ثبات CX را با ۹ (طول هر دو فیلد) مقدار دهی می‌کند و از ثبات‌های SI و DI برای نشان گذاری استفاده می‌نماید. دو دستور LEA آدرس افست HEADG1 به HEADG2 را در SI و DI به این شکل بارگذاری می‌کنند :

```
بارگذاری آدرس افست‌های ;  LEA  SI,HEADG1
HEADG1 و HEADG2 ;          LEA  DI,HEADG2
```

برنامه با استفاده از آدرس ثبات‌ها، SI و DI اولین بایت از HEADG1 را به اولین بایت از HEADG2 منتقل می‌کند. کروشه اطراف SI و DI در عملوندهای دستور MOV به معنی این است که دستورات با استفاده از آدرس افست در ثبات داده شده موقعیت حافظه را دستیابی می‌نمایند. بنابراین :

```
MOV AL,[SI]
```

به معنی استفاده از آدرس افست در SI (HEADG1 +0) برای انتقال بایت ارجاع شده به ثبات AL است و دستور

```
MOV [DI],AL
```

به معنی انتقال محتویات ثبات AL به آدرس افست ارجاع شده توسط DI (HEADG2 +0) است. برنامه این دو دستور MOV را ۹ مرتبه تکرار می‌کند، یک مرتبه برای هر کاراکتر به ترتیبی که در فیلد قرار دارد. در انتها، از یک دستور پرش شرطی استفاده می‌کند که هنوز آن را توضیح نداده‌ایم. JNZ (پرش در صورتی که صفر نیست).

دو دستور INC ، SI و DI را یکی افزایش داده و DEC ، CX را یکی کاهش می‌دهد. همچنین DE پرچم صفر را تنظیم یا پاک می‌کند که به حاصل CX بستگی دارد، اگر حاصل صفر نیست. هنوز برای انتقال کاراکتر وجود دارد، و JNZ به برچسب A20 جهت تکرار دستورات MOV برمی‌گردد. و چون SI و DI یکی افزوده شده‌اند، MOV بعدی به HEADG1+1 و HEADG2+1 رجوع می‌کند. حلقه به همین روش ادامه پیدا می‌کند تا همهٔ ۹ کاراکتر تا HEADG2+8 منتقل شوند. برنامه دستوراتی را برای نمایش محتویات HEADG2 در انتهای پردازش نیاز دارد. این دستورات لازم چنین هستند :

۱) مقدار دهی تابع 09H به ثبات AH برای تقاضای نمایش.
۲) مقدار دهی آدرس HEADG2 در DX.
۳) اجرای دستور INT 21H.

عملیات INT همه کاراکترهایی را که با اولین بایت از HEADG2 آغاز می‌شود تا علامت پایان $، نمایش می‌دهد، که همه بعد از HEADG2 تعریف می‌شوند. فصل ۹ این عملیات را با جزئیات کامل در بر دارد. ممکن است خواسته باشیداین‌برنامه راوارد و اسمبل و لینک نمایید و سپس‌از DEBUG برای پیگیری آن استفاده‌کنید. به‌تاثیرات روی پشته، ثبات‌ها و IP توجه‌کنید. (بخصوص بعد از اجرای JNZ) از D DS :0 برای دیدن تغییرات HEADG2 استفاده کنید.

دستورات INT

در زمان اجرا، یک دستور INT پردازش را متوقف و ***جدول برداری وقفه*** را در حافظه پایینی برای تعیین آدرس روتین درخواستی دستیابی می‌کند. سپس عملیات به DOS یا BIOS منتقل می‌شود تا عمل خاص انجام گیرد و سپس به برنامه جهت آغاز مجدد پردازش بر می‌گردد. اغلب، وقفه مراحل کامل عملیات ورودی یا خروجی را انجام می‌دهد. وقفه به یک سری عملیات جهت سهولت خروج از برنامه و در تکمیل موفقیت‌آمیز، بازگشت به آن نیاز دارد. بدین منظور INT چنین عمل می‌کند :

- محتویات پرچم را بر روی پشته می‌گذارد. (ابتدا اشاره‌گر پشته ۲ تا کم می‌شود)
- پرچم وقفه و پرچم وقفه سخت‌افزاری را پاک می‌کند.
- ثبات CS را بر روی پشته می‌گذارد.
- اشاره‌گر دستور را در روی پشته می‌گذارد (حاوی آدرس بعدی است).
- سبب می‌شود تا عملیات لازم انجام شود.

برای بازگشت از یک وقفه، روتین یک IRET (بازگشت از وقفه) دارد، که ثبات‌ها را از روی پشته برمی‌دارد و سبب می‌شود تا به دستور بلافاصله بعد از INT برود.

از آنجائیکه پردازش‌های ذکر شده بطور اتوماتیک انجام می‌گیرد. توجه شما باید به تعریف یک پشتهٔ بقدر کافی بزرگ برای عملیات گذاشتن و برداشتن لازم از روی پشته باشد که برای عملیات INT صحیح استفاده می‌شود.

قرار دادن آدرس‌های داده

از آنجائیکه 8086 و 80286 یک گذرگاه داده ۱۶ بیتی (کلمه) دارند، اگر آدرس‌ها از یک عدد زوج آغاز شوند به مراتب سریع‌تر عمل می‌کنند. حالتی را در نظر بگیرید که افست 0012H و 0013H حاوی کلمه 63A7H باشد. پردازشگر کلمه کامل را که در افست 0012H است مستقیماً در حافظه دستیابی می‌کند. اما اگر کلمه در یک آدرس با عدد فرد مثل 0013H آغاز شود :

محتویات حافظه :

XX	63	A7	XX

افست :

0012 0013 0014 0015

در این حالت، پردازشگر باید دو دستیابی انجام دهد ابتدا، بایت‌هایی که در 0012H و 0013H قرار دارد دستیابی نموده و بایت 0013H (63) را به ثبات AL منتقل می‌کند. سپس بایت‌های 0014H و 0015H را دستیابی می‌کند و بایتی که در 0014H (A7) قرار دارد به ثبات AH منتقل می‌کند. حال AX حاوی A763H است.

نیازی نیست که برای موقعیت‌های زوج یا فرد برنامه‌نویسی خاصی انجام‌دهید و یابدانید که آدرس زوج است یا فرد عملیات دستیابی بطور اتوماتیک یک کلمه را از حافظه به ثبات معکوس می‌کند بنابراین به ترتیب صحیح قرار می‌گیرد.

پردازشگرهای 80386 و بعد یک گذرگاه داده ۳۲ بیتی دارند و بر این اساس، قرارگرفتن عناصر ارجاع شده را بر روی آدرسی که بر چهار قابل تقسیم باشد (یک آدرس دو کلمه‌ای) توصیه می‌کنند. (بطور تکنیکی پردازشگر 486 و بعد، قرار گرفتن آدرس بر روی مرز پاراگراف را توصیه می‌کنند.)

شما می‌توانید از پیش پردازنده‌های ALIGN یا EVEN برای قرارگرفتن عناصر برروی مرز استفاده‌کنید. برای مثال، هم ALIGN 2 یا EVEN بر روی یک مرز کلمه قرار می‌دهد و ALIGN 4 بر مرز یک دو کلمه‌ای قرار می‌دهد. وقتی اسمبلر آدرس یک عنصر را بر طبق یک مرز تنظیم می‌کند، همچنین شمارشگر موقعیت را نیز بر طبق آن تنظیم می‌کند.

از آنجائیکه سگمنت داده وقتی PARA تعریف می‌شود بر یک مرز پاراگراف آغاز می‌شود، شما می‌توانید داده‌ها را ابتدا با مقدار دو کلمه‌ای، سپس با مقادیر کلمه و در نهایت با مقادیر بایتی سازماندهی کنید. اگر چه، پردازشگرهای مدرن آنچنان سریع اجرا می‌کنند که شما احتمالاً هرگز متوجه تاثیر این قرار دادن نخواهید شد.

آدرسهای نزدیک و دور

یک آدرس در برنامه ممکن است نزدیک یا دور باشد. یک ***آدرس نزدیک*** شامل فقط، قسمت افست یک آدرس است. دستوری که به یک آدرس نزدیک ارجاع می‌کند سگمنت جاری را فرض می‌کند. نام‌گذاری شده، با DS برای سگمنت داده و CS برای سگمنت کد.

یک ***آدرس دور*** حاوی هم سگمنت و هم قسمت افست می‌باشد، به شکل افست سگمنت یک دستور ممکن است به یک آدرس دور از داخل سگمنت جاری یا سگمنت دیگری ارجاع کند.

اغلب همه برنامه‌نویس‌های اسمبلی از آدرس‌های نزدیک استفاده می‌کنند. برنامه‌های بزرگتر که حاوی چندین سگمنت هستند به آدرسهای دور نیاز دارند.

سگمنت مقدمه را کنار می‌گذارد

برای اغلب اهداف، ارجاع به ناحیه داده در یک برنامه به موقعیتی است که سگمنت داده بر طبق ثبات DS به آن را دستیابی می‌کند. این یک مورد است اما - مخصوصاً برای برنامه‌های بزرگ - داده‌ها موضوعی در ثبات سگمنت دیگر مانند ES یا در 386 و بعد، در FS یا GS است. یک مثال می‌تواند یک جدول بزرگ از داده‌های بارگذاری شده از دیسک بر روی سگمنت مجزا از حافظه باشد. شما می‌توانید از هر دستوری برای پردازش داده در دیگر سگمنت استفاده کنید، اما باید ثبات سگمنت‌صحیح را مشخص‌کنید. فرض‌کنید آدرس سگمنت دیگر درثبات ES است و BX حاوی یک‌آدرس افست داخل آن سگمنت می‌باشد. حدس بزنید لازم است ۲ بایت (یک کلمه) از این موقعیت به ثبات DX منتقل شود.

```
MOV  DX,ES:[BX]  ; انتقال از ES:[BX] به DX
```

کد کردن 'ES:' بر عملگر کنار گذاشتن دلالت دارد که به معنی، جایگذاری استفاده معمول ثبات سگمنت DS با 'ES' می‌باشد. مثال بعدی مقدار یک بایت را از CL به سگمنت دیگر منتقل می‌کند افستی که با مقدار SI بعلاوه ۳۶ شکل گرفته است :

```
MOV ES:[SI+36],CL ; انتقال CL به ES:[SI+36]
```

اسمبلر کد زبان ماشین را با درج یک پیشوند یک بایتی (26H) قبل از دستورالعمل تولید می‌کند درست مانند اینکه دو خط زیر را نوشته باشید.

```
ES: MOV DX,[BX]     ; انتقال ES:[BX] به DX
ES: MOV [SI+36],CL  ; انتقال CL به ES:[SI+36]
```

نکات کلیدی

- یک عملوند،منبع داده‌هارابرای یک دستورفراهم‌می‌سازد. یک دستور ممکن‌است یک، دویاهیچ عملونداشته‌باشد.
- وقتی دو عملوند وجود دارد، دومین عملوند مبدأ است، که یک داده بلافصل یا آدرس از یک ثبات یا از حافظه) داده را ارجاع می‌دهد. اولین عملوند مقصد است، که داده‌ای ارجاع شده است برای پردازش در یک ثبات یا در حافظه.
- در قالب بلافصل، دومین عملوند حاوی یک مقدار ثبات یا یک عبارت است. عملوندهای بلافصل با اندازه مقصد باید هم خوانی داشته باشند، که هر بایت، کلمه، یا دو کلمه باشند.
- در قالب حافظه غیر مستقیم، هر یک از عملوندها به یک موقعیت حافظه رجوع می‌کنند و عملوند دیگر به یک ثبات را ارجاع می‌دهد.
- آدرس دهی غیر مستقیم از قابلیت پردازشگر برای آدرس دهی افست سگمنت استفاده می‌کند.

ثبات‌های مورد استفاده BP ، BX ، DI و SI هستند. که داخل کروشه بعنوان یک عملگر نشانگذاری کد می‌شوند. BP با SS مرتبط است به شکل SS:BP، که برای دستیابی داده‌ها در پشته می‌باشد. BX ، DI و SI با DS به شکل DS:BX ، DS:DI و DS:SI که به ترتیب برای پردازش داده‌ها در سگمنت داده مرتبط است.

- شما ممکن است در یک آدرس غیرمستقیم ثبات‌ها را ترکیب کنید به شکل [BX+DI]، که به معنی آدرس در BX بعلاوه آدرس در DI می‌باشد.

- دستور MOV داده ارجاع شده با آدرس در دومین عملوند را به آدرس اولین عملوند منتقل می‌کند.
- دستور LEA برای مقدار دهی یک ثبات با یک آدرس افست مفید است.
- INC و DEC دستورات مرسوم برای افزایش یا کاهش یک واحد از محتویات یک ثبات یا موقعیت حافظه هستند.
- دستور INT پردازش برنامه شما را متوقف کرده، برای عمل خاص به DOS یا BIOS منتقل می‌شود و IRET به برنامه شما برای آغاز مجدد پردازش باز می‌گردد.

پرسش‌ها

۶-۱. برای یک دستور با دو عملوند، کدام عملوند مبدأ و کدام یک مقصد می‌باشند ؟

۶-۲. (الف) چرا در روش محاسبات دستورات زیر در اجرا متفاوتند ؟

```
ADD CX,2548H
ADD CX,[2548H]
```

(ب) برای دومین ADD، یک عملوند در کروشه قرار دارد. نام این جزء چیست ؟

۶-۳. (الف) چرا در طریق علمی دو دستور ADD زیر در اجرا متفاوتند ؟

```
ADD BX,25
ADD [BX],25
```

(ب) برای دومین ADD، چه ترتیبی از آدرسدهی شامل اولین عملوند خواهد شد ؟

۶-۴. عملیات این دستور را توضیح دهید. ADD DX,[BX+SI+8]

۶-۵. جمله زیر را در نظر بگیرید چه خطایی دارد : ADD [BX],[DI]

(الف) خطا را تشخیص دهید. (ب) نحوه تصحیح آن را توضیح دهید.

۶-۶. تعاریف داده زیر داده شده است، خطای جملات را بیابید و دستور لازم برای تصحیح آن را کد نویسی کنید :

```
BYTEX   DB  56          ADD   BYTEX,BYTEY          (الف)
BYTEY   DB  27          ADD   AL,WORDZ ;عملوند ۲ صحیح است   (ب)
WORDZ   DW  148         SUB   BL,047BH ; عملوند ۲ صحیح است  (ج)
```

۶-۷. موارد زیر را با دستورات با عملوندهای بلافصل کد کنید : (الف) جمع 48H با CX، (ب) تفریق 48H از DX، (ج) شیفت DH یک بیت به راست، (د) شیفت BYTEX یک بیت به چپ، (ه) ذخیره ۲۴۸ در CX، (و) مقایسه BYTEX با صفر.

۶-۸. یک دستور کد کنید که محتویات یک کلمه با نام WORDZ را با BX تعویض نماید.

۶-۹. دستوری کد کنید که BX را با آدرس یک عنصر به نام CODETBL تنظیم نماید.

۶-۱۰. با عبارات عمومی هدف از دستور INT را توضیح دهید

۶-۱۱. توضیح دهید چگونه (الف) دستور INT بر پشته تاثیر می‌گذارد و (ب) دستور IRET بر پشته تاثیر می‌گذارد.

۶-۱۲. کد نمایید، اسمبل کنید، لینک کنید و با استفاده از DEBUG برنامه زیر را تست نمایید.

- عنصر داده با نام BYTEX و BYTEY (حاوی هر مقداری) را تعریف کنید و یک عنصر کلمه با نام WORDZ (حاوی صفر) تعریف کنید.
- محتویات BYTEX را به AL منتقل کنید.
- محتویات BYTEY را با AL جمع کنید.
- مقدار بلافصل 34H را به CL منتقل کنید.
- محتویات AL و CL را تعویض نمائید.
- محتویات AL را در CL ضرب کنید (MUL CL)
- حاصل را از AX به WORDZ منتقل نمائید.

فصل ۷

نوشتن برنامه‌های COM.

هدف: توضیح و استفاده از برنامه‌های COM. و چگونگی تهیه یک برنامه زبان اسمبلی با قالب COM.

مقدمه

تا این فصل، ما فقط برنامه‌های EXE. را می‌نوشتیم، اسمبل می‌کردیم و اجرا می‌کردیم. برای یک برنامه EXE. لینکر بطور اتوماتیک یک قالب ویژه تولید می‌کند، وقتی آن را بر روی دیسک ذخیره می‌سازد، در ابتدای آن یک بلوک عنوان قرار می‌دهد که ۵۱۲ بایت یا بیشترطول دارد. (فصل ۲۴ جزئیاتی راجع به بلوک سرفصل فراهم ساخته است)

می‌توانید برای اجرا برنامه‌های COM. نیز، بنویسید. یک مثال از یک برنامه‌های COM. مورد استفاده COMMAND.COM. می‌باشد. مزیت استفاده از برنامه‌های COM. این است که آنها به نسبت برنامه‌های EXE. قابل مقایسه، کوچکتر هستند و برای فعالیت‌هایی مانند برنامه‌های مقیم راحت‌تر تطبیق می‌یابند. قالب COM. ریشه‌های خودش را درمیکروکامپیوترهای اخیر دارد وقتی اندازه برنامه به 64K محدود می‌شود.

تفاوت‌های بین یک برنامه EXE. و یک برنامه COM.

تفاوت علمی بین برنامه‌های اجرایی EXE. و COM. در اندازهٔ برنامه ، سگمنت بندی و مقدار دهی است.

اندازهٔ برنامه

یک برنامه EXE. ممکن است هر اندازه مجازی داشته باشد در حالیکه یک برنامه COM. حداکثر یک سگمنت و حداکثر اندازه ۶۴K دارند که شامل PSP نیز می‌باشد. PSP یک بلوک ۲۵۶ بایتی (100H) است که بارگذار برنامه بلافاصله در ابتدای برنامه‌های COM. و EXE. قرار می‌دهد در زمانیکه آن را از دیسک بر روی حافظه بارگذاری می‌کند. محدوده 64K برای یک برنامه COM. یک قانون عمومی است، شما می‌توانید یک جمله SEGMENT AT اضافی کد نمائید نکته‌ای که خارج از این فصل توضیح داده خواهد شد. یک برنامه COM. همیشه از برنامه EXE. مثل خودش کوچکتر است، یک دلیل آن این است که یک رکورد عنوان ۵۱۲ بایتی که مقدم بر برنامه EXE. بر روی دیسک است در مقدمه برنامه COM. نیست. (رکورد عنوان با PSP در فصل ۲۴ توضیح داده شده است). یک برنامه COM. یک تصویر مطلق از برنامه اجرایی بدون اطلاعات آدرسی قابل جایگزین است.

سگمنت‌ها

استفاده از سگمنت‌ها برای برنامه‌های COM. تفاوت‌های علمی دارد و به نسبت آنچه برای برنامه‌های EXE. وجود دارد، راحت است.

سگمنت پشته. شما برای برنامه‌های EXE. یک سگمنت پشته تعریف می‌کنید، در حالیکه برای برنامه‌های COM. اسمبلر بطور اتوماتیک پشته تعریف می‌کند. بنابراین اگر یک برنامه اسمبلی می‌نویسید که به قالب COM. تبدیل خواهد شد، باید تعریف پشته را حذف نمایید. اگر اندازهٔ 64K برنامه کافی نباشد، اسمبلر بعد از برنامه در حافظهٔ بالایی یک پشته برقرار می‌سازد.

سگمنت داده. برای یک برنامه EXE.، شما اغلب یک سگمنت داده تعریف می‌کنید و ثبات DS را با آدرس سگمنت داده مقدار دهی می‌کنید. برای یک برنامه COM. شما نباید یک سگمنت داده تعریف کنید و داده‌ها را در داخل سگمنت کد تعریف کنید. همانطوریکه دیدید این روش ساده‌تری برای دستیابی این حالت می‌باشد.

سگمنت کد. یک برنامه COM. ، PSP، پشته ، سگمنت داده و سگمنت کد را در یک سگمنت کد در حداکثر 64K ترکیب می‌کند.

مقدار دهی

وقتی بارگذار برنامه یک برنامه COM. را جهت اجرا بارگذاری می‌کند، بطور اتوماتیک چهار ثبات سگمنت را با آدرس PSP مقدار دهی می‌نماید. از آنجائیکه ثبات CS و DS حاوی آدرس سگمنت با مقدار صحیح است، برنامه شما نباید آن‌ها را مقدار دهی نماید.

از آنجائیکه آدرس دهی در یک افست از 100H بایت از شروع PSP است، شما باید یک پیش پردازنده ORG 100H، بلافاصله بعد از جمله‌های SEGMENT از سگمنت کد CODE. کد نمائید. پیش‌پردازنده GRG به اسمبلر می‌گوید که شمارشگر موقعیت را با 100H تنظیم کند. وقتی کد کردن برنامه COM. آغاز می‌شود، اسمبلر تولید کد مقصد را از یک افست 100H بایت بعد از شروع PSP آغاز می‌کند.

تبدیل به قالب COM.

اگر برنامه منبع در قالب EXE. نوشته شده است. با استفاده از یک ویرایشگر دستورات را به قالب COM. تبدیل کنید. قالب کد نویسی MASM و TASM برای برنامه‌های COM. یکسان است، اگر چه روش تبدیل تفاوت دارد.

تبدیل مایکروسافت

برای هر دو برنامه‌های EXE. و COM. تحت MASM مایکروسافت، برنامه را اسمبل کرده و یک فایل OBJ تولید می‌کنید و سپس فایلی OBJ را برای تولید یک برنامه EXE. لینک می‌کنید. اگر برنامه‌ای که جهت اجرا نوشته‌اید به شکل یک برنامه EXE. باشد، اکنون می‌توانید آن را اجرا کنید. اگر برنامه‌ای که برای اجرا نوشته‌اید برنامه COM. باشد، لینکر پیغام می‌دهد.

"Warning :No Stack Segment" می‌توانید این پیغام را ندیده بگیرید زیرا پیغامی مبنی بر تعریف نکردن پشته است. می‌توانید از برنامه‌ای با نام EXE2BIN جهت تبدیل برنامه‌های مایکروسافت EXE. به یک برنامه COM. استفاده کنید (در حقیقت این برنامه، برنامه‌های EXE.را به فایل BIN. (دودویی) تبدیل می‌کند، نام برنامه به معنی "تبدیل EXE به BIN" است اما باید نام برنامه شما با توسعهٔ فایل COM. باشد) فرض کنید که EXE2BIN در درایو پیش فرض است و یک فایل لینک شده با نام SENTINEL.EXE در درایو D می‌باشد، تایپ کنید.

```
EXE2BIN D:SENTINEL,D:SENTINEL.COM <Enter>
```

اولین عملوند از فرمان همیشه نام یک فایل EXE. است، بنابراین نیازی نیست تا توسعه EXE. را تایپ کنید. دومین عملوند ممکن است هر نام فایل معتبر با توسعه COM. باشد. اگر توسعه COM. را حذف کنید، EXE2BIN توسعه را

BIN. فرض می‌کند، که اگر می‌خواهید برنامه را اجرا کنید باید آن مجدداً با توسعه COM. نام‌گذاری کنید (برخی، بعضی وقتها، ممکن است فکر کنند که این ایده خوبی است)

تبدیل بورلند

اگر برنامه منبع شما توسط TASM بر طبق موارد ضروری برنامه‌های COM. نوشته شده است، می‌توانید برنامه مقصد را مستقیماً به برنامه COM. تبدیل کنید از انتخاب T/ در TLINK استفاده کنید

```
TLINK /T D:SENTINEL
```

وقتی تبدیل به قالب COM. کامل می‌شود، می‌توانید فایل‌های OBJ. و EXE. را حذف کنید.

مثالی از یک برنامه COM.

برنامه شکل ۱-۷ با نام A07COM1 مشابه برنامه شکل ۲-۵ است، اما حال با توجه به موارد ضروری برنامه‌های COM. نوشته است. توجه کنید تغییرات زیر به نسبت برنامه شکل ۲-۵ بوجود آمده است.

- هیچ سگمنت پشته و داده‌ای تعریف نشده است.
- جمله ASSUME به اسمبلر می‌گوید وقتی برنامه جهت اجرا بارگذاری شد، همه ثبات‌های سگمنت حاوی آدرس شروع سگمنت کد خواهند بود. (جایی که PSP آغاز می‌شود)
- پیش پردازنده ORG 100H به اسمبلر می‌گوید که شمارشگر موقعیت 100H بایت پس از شروع PSP باشد. بارگذار برنامه 100H را در ثبات اشاره‌گر دستور (IP) در زمان اجرای برنامه، ذخیره می‌سازد.
- یک دستور JMP کنترل اجرا را به بعد از تعریف داده‌ها منتقل می‌کند. برخی برنامه نویسان عناصر داده را پس از دستورات کد نویسی می‌کنند، بنابراین نیازی به قرار دادن دستور JMP نیست. کد نویسی عناصر داده‌ها، ابتدا ممکن است سرعت پردازش اسمبلی را بالا ببرد اما هیچ مزیت دیگری ندارد. مثال‌های این کتاب داده‌ها را فقط بر طبق برنامه نویسی مرسوم در ابتدا، تعریف نموده‌اند.
- نام‌های BEGIN و A10MAIN صرفاً توضیح هستند و هیچ معنی دیگری برای اسمبلر ندارند. می‌توانید هر نام معتبر دیگری بجای این برچسب‌ها استفاده نمایید.

```
        page 60,132
TITLE   A07COM1 COM program to move and add
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS:CODESG,SS:CODESG,ES:CODESG,
        ORG     100H            ;Start at end of PSP
BEGIN:  JMP     A10MAIN         ;Jump past data
; -----------------------------------------------------
FLDD    DW      175             ;Data definitions
FLDE    DW      150
FLDF    DW      ?
; -----------------------------------------------------
A10MAIN PROC    NEAR
        MOV     AX,FLDD         ;Move 0175 to AX
        ADD     AX,FLDE         ;Add  0150 to AX
        MOV     FLDF,AX         ;Store sum in FLDF
        MOV     AX,4C00H        ;End processing
        INT     21H
A10MAIN ENDP
CODESG  ENDS
        END     BEGIN
```

شکل ۱-۷ برنامه منبع COM. با سگمنت مرسوم

- INT 21H تابع 4CH پردازش مرسوم را خاتمه می‌دهد و خارج می‌شود و به سیستم عامل می‌رود. شما می‌توانید از دستور RET برای خارج شدن از برنامه COM. استفاده کنید. (این باعث می‌شود تا به یک INT 20H یک خروج قدیمی در بایت 01 از PSP منتقل شود)

در اینجا مراحل MASM و TASM جهت تبدیل برنامه به قالب COM. آورده شده است.

MASM	TASM
MASM D:A07COM1,D:	TASM D:A07COM1,D:
LINK D:A07COM1,D:	TLINK /T D:A07COM1,D:
EXE2BIN D:A07COM1,D: A07COM1.COM	

برنامه‌های EXE. و COM. به ترتیب ۷۹۲ و ۲۴ بایت طول دارند. تفاوت زیاد بین این دو به سبب ذخیره شدن ۵۱۲ بایت بلوک عنوان در شروع ماجول EXE. است. فرمان DEBUG D:A07COM1.COM را تایپ کنید و اجرای برنامه COM. را تا آخرین دستور پیگیری نمایید. همچنین می‌توانید وقتی برنامه COM. را کد نویسی می‌کنید از پیش پردازنده‌های سگمنت ساده شده، استفاده کنید مانند آنچه که در شکل ۲-۷ نشان داده شده است، فقط یک سگمنت کد تعریف کنید و هیچ سگمنت داده یا پشته تعریف نکنید.

پشته COM.

برای یک برنامه COM. بطور اتوماتیک پشته تعریف می‌کند و آدرس همان سگمنت را در همه چهار ثبات سگمنت تنظیم می‌کند. اگر 64K سگمنت برای برنامه بقدر کافی است، بارگذار برنامه پشته را در انتهای سگمنت تنظیم می‌کند و ثبات SP با آدرس بالای پشته تنظیم می‌شود.

اگر 64K سگمنت حاوی فضای کافی برای پشته نباشد بارگذار، پشته را در انتهای حافظه تنظیم می‌کند. در حالت دیگر، بارگذار یک کلمه صفر بر روی پشته می‌گذارد که اگر از RET برای انتهای اجرای یک برنامه COM. استفاده کردید، مانند یک افست IP عمل می‌کند.

اگر برنامهٔ شما بزرگ باشد ، یا حافظه محدود باشد ، درنحوه گذاشتن کلمات بر روی پشته باید دقت کنید. فرمان DIR بر طول یک فایل دلالت دارد و برای فضای در دسترس پشته ایده لازم را ارائه می‌دهد. خیلی از برنامه‌های کوچکتر در این کتاب در قالب COM. هستند و خیلی راحت‌تر در قالب EXE. درک می‌شوند .

```
        page 60,132
TITLE   A07COM2  COM program to move and add data
        .MODEL  SMALL
        .CODE
        ORG     100H            ;Start at end of PSP
BEGIN:  JMP     A10MAIN         ;Jump past data
; ------------------------------------------------
FLDD    DW      175             ;Data definitions
FLDE    DW      150
FLDF    DW      ?
; ------------------------------------------------
A10MAIN PROC    NEAR
        MOV     AX,FLDD         ;Move 0175 to AX
        ADD     AX,FLDE         ;Add  0150 to AX
        MOV     FLDF,AX         ;Store sum in FLDF
        MOV     AX,4C00H        ;End processing
        INT     21H
A10MAIN ENDP
        END     BEGIN
```

شکل ۲-۷ برنامه منبع *COM*. با پیش پردازنده‌های سگمنت ساده

نکات اشکال زدایی

حذف فقط یکی از موارد ضروری COM. سبب از کار افتادن برنامه خواهد شد. اگر EXE2BIN یک خطا بیابد براحتی شما را متوجه خواهد کرد که قادر به تبدیل فایل نیست، اما دلیلی ارائه نمی‌دهد. جملات ASSUME ، SEGMENT و END را چک کنید. اگر ORG 100H را حذف نمایید برنامه اجرایی داده‌های PSP را به طرز نادرستی ارجاع می‌دهد و نتایج غیر قابل پیش بینی به بار می‌آورد اگر یک برنامه COM.را تحت DEBUG اجرا می‌کنید از D CS:100 برای دیدن داده‌ها و دستورات استفاده کنید. در صورت خاتمهٔ برنامه، دیگر آن را پیگیری نکنید و با فرمان Q از DEBUG خارج شوید. کوشش برای اجرای ماجول EXE. که با قوانین COM. نوشته شده به شکست منجر می‌شود، حتماً آن را حذف نمایید.

نکات کلیدی

- یک برنامه COM. به یک سگمنت 64K محدود می‌شود و کوچکتر از برنامه EXE. مشابه است
- یک برنامه COM. پشته یا سگمنت داده تعریف نمی‌کند، همچنین ثبات DS را نیز مقدار دهی نمی‌کند.
- برنامه‌ای که به شکل COM. نوشته شده بلافاصله پس از جمله SEGMENT، سگمنت کد به ORG 100H نیاز خواهد داشت. جملات آدرس افست شروع اجرا را بلافاصله پس از PSP قرار می‌دهند.
- در MASM مایکروسافت، برنامه EXE2BIN فایل EXE را به قالب COM. تبدیل می‌کند. TLINK بورلند مستقیماً یک برنامه مقصد را به قالب COM. تبدیل می‌کند.
- سیستم در انتهای برنامه COM. پشته را نصب می‌نماید.

پرسش‌ها

۷-۱. حداکثر اندازه یک برنامه COM. چیست ؟

۷-۲. برای یک برنامه منبع که به قالب COM. تبدیل شده، چه سگمنتی را می‌توانید تعریف نمایید ؟

۷-۳. توضیح دهید که چرا ORG 100H را در شروع برنامه‌ای که به قالب COM. تبدیل می‌شود کد می‌نمایند .

۷-۴. چرا برای یک برنامه COM. نیازی به تعریف پشته نیست ؟

۷-۵. یک برنامه با نام PRESSURE.ASM وجود دارد. فرمان‌های لازم جهت تبدیل آن به قالب COM. با استفاده از (الف) MASM، (ب) TASM بنویسید.

۷-۶. برنامه پرسش ۱۲-۶ را با قالب COM. بازنویسی کنید. آن را اسمبل نمایید و به COM. تبدیل و سپس با DEBUG اجرا کنید.

فصل ۸

موارد ضروری برنامه نویسی برای منطق و کنترل

هدف : موارد ضروری برای کنترل برنامه (حلقه و پرش) ، مقایسه‌ای منطقی، عملیات بیت منطقی و سازماندهی برنامه.

مقدمه

تا این فصل، اغلب برنامه‌ها به یک روش عمودی اجرا می‌شدند یک دسته بلافاصله پس از دیگری، به ندرت یک مسأله برنامه‌نویسی به این سادگی است. اغلب برنامه‌ها حاوی تست‌های متعدد جهت تشخیص یکی از چندین فعالیت برای اجراست و چندین حلقه بطریقی که یک سری مراحل تا رسیدن به یک ضرورت خاص تکرار شود برای مثال یک تمرین عمومی تست رسیدن به انتهای اجراست.

موارد ضروری مانند این شامل انتقال کنترل به آدرس یک دستور است که بلافاصله پس از دستور جاری که اجرا شده است، نیست. انتقال کنترل ممکن است، به جلو باشد، اجرای یک سری جدید از مراحل، یا به عقب برای اجرای مجدد همان مراحل باشد. دستوراتی که می توانند کنترل را به خارج از ترتیب طبیعی جریان منتقل کنند، بنابراین باید یک مقدار افست را با IP جمع کنند. در زیر دستوراتی بطور خلاصه آورده شده که در این فصل مقدمه‌ای بر آن گفته خواهد شد.

عملیات مقایسه	عملیات انتقال	عملیات منطقی	شیفت و پرش
CMP	CALL	AND	SAR/SHR
TEST	JMP	NOT	SAL/SHL
	Jnnn*	OR	RCR/ROR
	LOOP	XOP	RCL/ROL
Jnnn به معنی همه دستورات پرش شرطی است مانند JNE و JL			

آدرسهای کوتاه، نزدیک و دور

اسمبلر سه نوع آدرس را پشتیبانی می کند که با فاصله از آدرس اجرای جاری برقرار می گردد:

۱) آدرس ***کوتاه*** محدود به فاصله ۱۲۸- تا ۱۲۷ بایت.

۲) یک آدرس ***نزدیک*** محدوده به فاصله ۷۶۸ تا ۳۲- تا ۷۶۷ تا ۳۲ بایت داخل همان سگمنت.

۳) یک آدرس ***دور*** که ممکن است فاصله‌ای بیشتر از ۳۲K داشته باشد یا در سگمنت دیگر باشد یک عملیات پرش یک آدرس کوتاه را با یک افست ۱ بایتی و آدرس نزدیک را با یک افست ۱ تا ۲ کلمه‌ای برقرار می سازد. یک آدرس دور را با یک آدرس سگمنت و یک افست برقرار می‌سازد. CALL که دستور معمول برای این منظور است زیرا لینک کردن به آدرس مورد تقاضا و بازگشت بعد از آن را به سهولت فراهم می سازد.

جدول زیر قانون فاصله از عملیات JMP و LOOP و CALL را لیست نموده است. کمتر نیاز به حفظ این قوانین می باشد. زیرا استفاده، معمول از این دستورات بندرت شکل ایجاد می کند.

دستورات	کوتاه ۱۲۸- تا ۱۲۸ همان سگمنت	نزدیک ۳۲۷۶۸- تا ۳۲۷۶۷+ همان سگمنت	دور بیشتر از ۳۲K یا در سگمنت دیگر
JMP	بله	بله	بله
Jnnn	بله	بله (80386+)	نه
LOOP	بله	نه	نه
CALL	N/A	بله	بله

برچسب‌های دستورات

عملوند دستورات JMP و Jnnn (پرش شرطی) و LOOP به برچسب یک دستور دیگر اشاره می‌کند. مثال زیر به P50 پرش دارد، آنجائیکه برچسب یک دستور INC می باشد:

```
        JMP   P50
        ...
P50:    INC   CX
        ...
```

برچسب دیگر یک دستور مانند P50 با یک علامت (:) خاتمه می‌یابد، که به آن یک صفت نزدیک می‌دهد که برچسب داخل زیر روال در همان سگمنت کد می‌باشد. مواظب باشید: حذف دو نقطه (:) یک خطای عمومی است، که اسمبلر اخطار می‌دهد. توجه کنید که برچسب دستور در یک عملوند دستور مانند (JMP P50) نیازی به (:) ندارد.

همچنین می‌توانید برچسب را در یک خط جدا کنید مثل

```
P50:
        INC   CX
```

در هر دو حالت، آدرس P50 اولین بایت دستور INC را ارجاع می‌دهد.

دستور JMP

یک دستور عمومی برای انتقال کنترل دستور JMP (پرش) می‌باشد، یک پرش بدون شرط، زیرا عملیات کنترل را تحت هر شرایطی منتقل می‌کند. همچنین JMP صف دستورات از قبل واکشی شده پردازشگر را از بین می برد، بنابراین برنامه‌ای که عملیات پرش زیادی دارد و سرعت پردازش بالا را از دست خواهد داد. قالب کلی JMP چنین است :

[Label:]	JMP	Short, near, or far adress

پرش کوتاه و نزدیک

یک عملیات JMP داخل همان سگمنت ممکن است کوتاه یا نزدیک باشد (یا حتی دور باشد اگر مقصد یک زیر روال با صفت FAR باشد). در اولین گذر در یک برنامه منبع، اسمبلر طول هر دستور را تعیین می‌کند، اما یک دستور JMP ممکن است دو ، سه یا چهار بایت طول داشته باشد. یک عملیات JMP به یک برچسب داخل ۱۲۸- تا ۱۲۷+ بایت یک **پرش کوتاه** است. اسمبلر یک بایت برای عملیات (EB) و یک بایت برای عملوند تولید می کند. عملوند مانند یک مقدار افست است که پردازشگر زمان اجرا آن را با ثبات IP جمع می‌کند. محدودهٔ 00H تا FFH، یا ۱۲۸- تا ۱۲۷+ می‌باشد. یک JMP که بیشتر از ۱۲۸- تا ۱۲۷+ بایت است یک **پرش نزدیک** (داخل ۳۲K) بطوریکه اسمبلر کد ماشین متفاوتی (E9) تولید می‌کند و یک عملوند ۲ بایتی (8086/80286) یا عملوند ۴ بایتی (80386 و بعد) دارد. هم اکنون از **پرش دور** صرفنظر می‌کنیم.

پرش های به سمت جلو و سمت عقب

یک پرش ممکن است به سمت عقب یا به سمت جلو باشد. اسمبلر ممکن است تا حال عملوند مورد نظر (پرش به سمت عقب) در طی ۱۲۸- بایت به آن مواجه شده باشد مثل :

```
P50:
 . . .
JMP    P50
```

در این حالت، اسمبلر یک دستور ماشین ۲ بایتی تولید می‌کند. در یک پرش به سمت جلو، اسمبلر هنوز با عملوند مورد نظر مواجه نشده است.

```
P50:    P90
 . . .
P90:
```

از آنجائیکه اسمبلر تا اینجا نمی‌داند که پرش به سمت جلو کوتاه یا نزدیک است. برخی نسخه‌ها آن را نزدیک فرض می کنند و یک دستور ۳ بایتی تولید می کنند. اما اگر پرش واقعاً نزدیک باشد، می‌توانید از عملگر SHORT برای ایجاد یک پرش کوتاه استفاده کنید و یک دستور ۲ بایتی با کد کردن به شکل زیر ایجاد کنید.

```
JMP   SHORT   P90
 . . .
P90:
```

برنامه : استفاده از دستور JMP

برنامه COM در شکل ۱-۸ استفاده از دستور JMP را مشخص می سازد. برنامه، ثبات‌های AX ، BX و CX را با مقدار ۱ ارزش دهی می کند و یک حلقه که اعمال زیر را انجام می‌دهد:

```
                        page 60,132
                TITLE   A08JUMP (COM)  Using JMP for looping
                        .MODEL SMALL
                        .CODE
0100                    ORG     100H
0100            A10MAIN PROC    NEAR
0100  B8 0001           MOV     AX,01       ;Initialize AX,
0103  BB 0001           MOV     BX,01       ;  BX, and
0106  B9 0001           MOV     CX,01       ;  CX to 01
0109            A20:
0109  05 0001           ADD     AX,01       ;Add 01 to AX
010C  03 D8             ADD     BX,AX       ;Add AX to BX
010E  D1 E1             SHL     CX,1        ;Double CX
0110  EB F7             JMP     A20         ;Jump to A20 label
0112            A10MAIN ENDP
                        END     A10MAIN
```

شکل ۱-۸ استفاده از دستور *JMP*

- جمع ۱ با AX.
- جمع AX با BX.
- دو برابر کردن مقدار CX.

در انتهای حلقه (بعد از SHL) دستور JMP A20 کنترل را به دستوری با برچسب A20 منتقل می کند. تاثیر تکرار حلقه سبب افزایش AX به شکل ۱، ۲، ۳، ۴،... خواهد شد، BX بر طبق مجموع ارقام اضافه خواهد شد به شکل ۱، ۳، ۶، ۱۰،... و CX دو برابر خواهد شد به شکل ۱، ۲، ۴، ۸،... از آنجائیکه حلقه خروج ندارد، پردازش بدون انتهاست و همیشه یک تمرین مطلوب نیست.

در برنامه A20، ۹- بایت با JMP فاصله دارد و شما بر این فاصله با انتخاب کد مقصد JMP EBF7 می‌توانید تاکید

کنید. EB کد ماشین برای JMP نزدیک است و F7H مقدار مکمل ۲ برابری ۹- است. در این نقطه IP، حاوی افست (0112H) دستور بعدی جهت اجراست. از آنجائیکه این یک پرش به سمت عقب است، عملوند F7 یک مقدار منفی است عملیات JMP و F7 (بطور تکنیکی، FFF7، زیرا اندازه IP یک کلمه است) را با IP جمع می‌کند، که حاوی افست 0112H دستور بلافاصله بعد از JMP می‌باشد.

	دهدهی	شانزدهی
ثبات دستور:	۲۷۴	0112
عملوند JMP:	-۹	FFF7 (مکمل دو)
آدرس پرش :	۶۵	(1)0109

اسمبلر آدرس پرش را 0109H محاسبه می‌کند، در حالیکه رقم نقلی خروجی در نظر گرفته نمی‌شود (همانطور که لیست برنامه برای آدرس افست A20 نمایش داده شده است)، عملیات مقدار افست IP را تغییر می‌دهد و صف دستورات را پاک می‌کند. بعنوان یک تمرین سودمند، برنامه را وارد کنید، آن را لینک نمایید و آن را به قالب COM. تبدیل کنید. از آنجائیکه عملوندهای بلافصل همه داده‌های لازم را تولید می‌کنند، هیچ داده‌ای تعریف نشده است. از DEBUG برای پیگیری ماجول COM. برای تعدادی از تکرارها و مشاهده تاثیر اجرا بر AX ، BX ، CX و IP را استفاده کنید.

یک بار AX حاوی 08 است BX و CX به ترتیب تا 24H (۳۶ دهدهی) و 80H (۱۲۸ دهدهی) افزایش می‌یابد. Q را بفشارید تا خارج شوید.

دستور LOOP

همانطور که در شکل ۱-۸ استفاده شده، دستور JMP یک حلقه بدون انتها را ایجاد می کند. اما یک تمرین استاندارد، کد نویسی روتینی است که حلقه به مقدار مشخص شده از زمان یا تحت رسیدن به شرایط خاصی پایان یابد. دستور LOOP، که این منظور را کفایت می کند، یک مقدار ابتدایی در ثبات CX لازم دارد برای هر تکرار، LOOP بطور اتوماتیک یک را از CX کم می‌کند. وقتی مقدار CX به صفر می‌رسد کنترل به دستور بعدی می رود، اگر مقدار CX غیر صفر باشد. کنترل به آدرس عملوند پرش خواهد داشت فاصله‌ای که که در عملوند قرار می‌گیرد باید یک پرش کوتاه داخل ۱۲۸- تا ۱۲۷+ بایت باید باشد. برای یک عملیاتی که خارج از این محدوده است، اسمبلر یک پیغام می دهد به این شکل 'relative jump out of range' قالب عمومی برای LOOP چنین است :

[Label:]	LOOP	Short-address

برنامه شکل ۲-۸ استفاده از LOOP را مشخص می سازد. همان عملیات برنامه شکل ۱-۸ را انجام می‌دهد بجز آنکه از یک دستور MOV برای مقدار دهی CX با 10 و خاتمه حلقه پس از 10 بار استفاده می‌کند. از آنجائیکه LOOP به استفاده از CX نیاز دارد، این برنامه از DX بجای CX برای دو برابر کردن مقدار ابتدایی ۱، استفاده می‌کند. دستور LOOP جایگزین JMP A20 و برای پردازش سریع‌تر، INC AX (افزایش AX با ۱) با ADD AX,01 جایگزین می‌شوند.

مانند دستور JMP، عملوند کد ماشین برای LOOP حاوی فاصله انتهای دستور تا آدرس A20 است که عملیات هنگام اجرا آن را با IP جمع می‌کند.

یک تجربه سودمند، تغییر کپی شکل ۱-۸ برای این تغییرات است آن را اسمبل کنید، لینک کنید و به COM. تبدیل کنید. از DEBUG برای پیگیری تا 10 بار حلقه استفاده کنید و تاثیر اجرای برنامه را بر روی AX ، BX ، CX ، DX و IP مشاهده نمایید. وقتی CX تا صفر کاهش می‌یابد، محتویات AX ، BX ، DX به ترتیب 0000BH ، 0042H و 0400H خواهد بود. Q را برای خروج از DEBUG بفشارید.

```
                        page 60,132
                TITLE   A08LOOP (COM)   Illustration of LOOP
                        .MODEL SMALL
                        .CODE
0100                    ORG     100H
0100            A10MAIN PROC    NEAR
0100  B8 0001           MOV     AX,01       ;Initialize AX,
0103  BB 0001           MOV     BX,01       ;  BX, and
0106  BA 0001           MOV     DX,01       ;  DX to 01
0109  B9 000A           MOV     CX,10       ;Initialize for
010C            A20:                        ;  ten loops
010C  40                INC     AX          ;Add 01 to AX
010D  03 D8             ADD     BX,AX       ;Add AX to BX
010F  D1 E2             SHL     DX,1        ;Double DX
0111  E2 F9             LOOP    A20         ;Decrement CX,
                                            ;  loop if nonzero
0113  B8 4C00           MOV     AX,4C00H    ;End processing
0116  CD 21             INT     21H
0118            A10MAIN ENDP
                        END     A10MAIN
```

شکل ۲-۱۸ استفاده از دستور *LOOP*

دو نوع مختلف LOOP داریم که هر دو نیز CX را یکی کاهش می‌دهند. LOOPZ/LOOPE (حلقه تا زمانی که مساوی باشد یا حلقه تا زمانی که صفر باشد) تا زمانی که مقدار CX صفر است یا شرط صفر تنظیم است به حلقه ادامه می‌دهد. LOOPNE/LOOPNZ (حلقه تا زمانی که مساوی نیست یا حلقه تا زمانی که صفر نیست) تا زمانی که CX صفر نیست یا شرط صفر تنظیم نیست به حلقه ادامه می‌دهد.

نه LOOP نه LOOPXXهای مختلف آن تنظیمات هر یک از پرچم‌ها در ثبات پرچم را تغییر نمی‌دهند. اما، از آنجائیکه دستورات دیگر داخل روتین حلقه، پرچم‌ها را تغییر می‌دهند، برنامه شکل ۲-۸ از LOOP استفاده می‌کند و نه از LOOPXXهای مختلف آن.

ثبات پرچم

مطالب باقیمانده در این فصل به دانستن جزئیات بیشتری راجع به ثبات پرچم نیاز دارد. این ثبات ۱۶ بیت دارد، که دستورات مختلفی تنظیم شده‌اند که بر وضعیت یک عملیات دلالت دارند. در همه حالت‌ها، یک پرچم تا زمانی که دستور دیگری آن را تغییر نداده تنظیم می‌ماند. ثبات پرچم در حالت حقیقی حاوی بیت‌های عمومی زیر است، که در اینجا از راست به چپ توضیح داده شده است:

شماره بیت :	۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰
پرچم :					O	D	I	T	S	Z		A		P		C

CF (پرچم نقلی). حاوی یک رقم نقلی (۰ یا ۱) از بالا رتبه‌ترین (سمت چپ‌ترین) بیت پس از عملیات محاسباتی و عملیات شیفت و چرخش می‌باشد. JC و JNC این پرچم را تست می‌کند.

PF (پرچم توازن). حاوی یک تست از ۸ بیت پایین رتبه داده عملیات می‌باشد. این پرچم توازن را با بیت توازن که در فصل ۱ توضیح داده شد اشتباه نکنید و بیت توازن بندرت در برنامه‌نویسی معمول مورد توجه قرار می‌گیرد. تعداد فرد بیت‌های ۱ پرچم را با صفر پاک می‌کند و تعداد زوج بیت‌های یک پرچم با یک تنظیم می‌کند. JP و JPO این پرچم را تست می‌نماید.

AF (پرچم نقلی معین). به محاسبات فیلدهای فشرده ASCII ، BCO مربوط است. این پرچم زمانی که یک عملیات محاسباتی یک بایتی سبب ایجاد رقم نقلی خروجی از بیت ۳ (چهارمین بیت از سمت راست) یک ثبات شود، تنظیم می‌گردد.

ZF (پرچم صفر). بر اساس نتیجه یک عملیات محاسباتی یا عملیات مقایسه یک یا صفر می‌شود. یک حاصل غیر صفر پرچم را صفر می‌کند و یک حاصل صفر آن را با یک تنظیم می‌کند. هر چند، صحیح به نظر نمی‌رسد به طور منطقی صحیح است. 0 به معنی نه (حاصل مساوی یا صفر نیست) و یک یعنی بله (حاصل صفر است.) JE و JZ (در بین دیگر دستورات) این پرچم را تست می‌کنند.

SF (پرچم علامت). بر طبق علامت (بیت بالا رتبه یا سمت چپ) تولید شده با عملیات محاسباتی تنظیم می‌گردد. یک مقدار مثبت پرچم را صفر می‌کند و یک حاصل صفر آن را با یک تنظیم می‌نمایید.JC و JL (در بین دیگر دستورات) این پرچم را تست می‌کنند.

TF (پرچم تله). وقتی تنظیم باشد، سبب می‌شود پردازشگر در حالت تمک مرحله‌ای اجرا کند، که یک دستور در هر زمان تحت کنترل کاربر می‌باشد. وقتی فرمان T را در DEBUG تایپ کنید DEBUG از این پرچم استفاده می‌کند.

IF (پرچم وقفه). وقفه‌ها را وقتی صفر است غیر فعال، وقتی یک است فعال می‌سازد. این پرچم در برنامه نویسی معمول بندرت استفاده می‌شود.

DF (پرچم جهت). در عملیات رشته‌ای برای تعیین جهت انتقال داده استفاده می شود. وقتی پرچکم صفر است، عملیات رشته‌ای انتقال داده را از چپ به راست انجام می‌دهد، وقتی پرچم یک است، عملیات رشته‌ای انتقال داده را از راست به چپ انجام می‌دهد.

OF (پرچم سرریز). بر یک رقم نقلی وارد شده و خارج شده از بیت علامت بالا رتبه (سمت چپ) پس از یک عملیات محاسبات علامتدار دلالت دارد. JO و JNO این پرچم را تست می‌کند.

دستور CMP

دستور CMP جهت مقایسه دو فیلد داده استفاده می‌شود، یک یا هر دو فیلد در ثبات هستند. قالب کلی آن چنین است :

[label:]	JMP	register/memory,register/memory/immedaite

بطور تکنیکی، امکان استفاده از CMP برای مقایسه رشته‌ها وجود دارد، اما CMPS (فصل ۱۲ آن را در بر دارد) برای این منظور دستور مناسبی است. نتیجه عملیات CMP بر روی پرچم‌های AF ، CF ، OF ، PF ، SF و ZF تاثیر می‌گذارد، ولی نباید این پرچم‌ها را بطور مجزا تست کنید. کد زیر ثبات DX را برای یک مقدار صفر تست می‌کند:

```
        CMP   DX,0          ; مقایسه DX با صفر
        JE    P50           ; اگر مساوی است، پرش کن به P50
              .             (اگر غیر صفر است عملیات را انجام بده)
              .
P50     ...                 ; نقطه پرش در صورتیکه DX صفر است
```

اگر DX حاوی صفر باشد، CMP، پرچم ZF را با یک تنظیم می‌کند و امکان دارد یا ممکن نیست که بر تنظیم دیگر پرچم‌ها تاثیر بگذارد. دستور JE فقط پرچم ZF را تست می‌کند. از آنجائیکه ZF حاوی یک می‌باشد (حاوی یک شرایط صفر)، JE کنترل را به آدرس عملوند P50 منتقل می‌کند (پرش می‌کند). در حقیقت عملیات CMP اولین و دومین عملوند را با هم مقایسه می‌کند: برای مثال، مقدار اولین عملوند بیشتر از دیگری، مساوی با یا کمتر از دومین عملوند است؟ (CMP مانند SUB است بدون ذخیره کردن مقدار). بخش بعدی راه‌های مختلف انتقال کنترل بر پایه شرایط تست شده را فراهم می‌سازد.

دستورات پرش شرطی

پردازشگر دستورات پرش شرطی متفاوتی را پشتیبانی می‌کند، که کنترل را بسته به تنظیم ثبات پرچم منتقل می‌کند.

برای مثال، دو فیلد را مقایسه می‌کنید و سپس بر طبق مقادیر پرچم که مقایسه تنظیم نموده است پرش را انجام می‌دهد.

قالب کلی دستور پرش شرطی چنین است:

[Label:]	Jnnn	Short-address

همانطور که توضیح داده شد، دستور LOOP ثبات CX را کاهش می‌دهد، اگر غیر صفر باشد کنترل به آدرس عملوند منتقل می‌شود. شما باید جمله LOOP A20 را در شکل ۲-۸ با دو عبارت جایگزین نمایید. کاهش CX و دیگر انجام پرش شرطی :

```
DEC   CX   ; مساوی حلقه
JNZ   A20  ;
...
```

DEC ، CX را یکی کاهش می‌دهد و ثبات پرچم را صفر یا یک می‌کند. JNZ پرچم صفر را تست می‌کند، اگر CX غیر صفر باشد، کنترل به A20 پرش دارد و اگر CX صفر باشد کنترل به دستور بعدی می‌رود. (یک عملیات پرش که شاخه دارد صف دستورات از پیش واکشی شده را از بین می‌برد) اگر چه محدودیت استفاده دارد در این مثال این دستور سریع‌تر اجرا می‌شود و به نسبت استفاده از دستورات DEC و JNZ بایت‌های کمتری استفاده می‌کند.

مانند دستورات JMP و LOOP، عملوند کد ماشین برای JNZ حاوی فاصله‌ای است از انتهای دستور تا آدرس A20، که عملیات آن را با ثبات IP جمع می‌کند. برای 286 / 8086 فاصله پرش شرطی باید کوتاه باشد، یعنی در فاصله ۱۲۸- تا ۱۲۷+ بایت. اگر عملیات بیشتر از این حد باشد اسمبلر پیغام می‌دهد "relative jump out of range" پردازشگرهای 80386 و بعد برای افست ۳۲ بیت (نزدیک) که آدرس داخل ۳۲K باشد را مجاز می‌دانند.

داده‌های علامتدار و بدون علامت

تفاوت گذاشتن بین هدف پرش‌های شرطی می‌تواند استفاده هر یک را واضح سازد. نوع داده‌ای که بر روی آن مقایسه یا محاسبه انجام می‌شود می‌تواند استفاده از هر کدام رامشخص سازد. با یک عنصر عددی **بدون علامت** (داده‌های منطقی) با همه بیت‌ها مانند بیت‌های داده رفتار می‌شود. انواع مثال از مقادیر عددی، شماره مشتری یا شماره تلفن می‌باشد. با یک عنصر عددی ***علامتدار*** (داده‌های محاسباتی) با سمت چپ‌ترین بیت بعنوان علامت، در حالیکه صفر مثبت است و یک منفی است، رفتار می‌کند، مثلاً کمیت، سود و فاصله است که مثبت یا منفی می‌توانند باشند. در مثال بعد، فرض کنید CX حاوی 11000110 و DX حاوی 00010110 است دستور :

```
CMP CX,DX
```

محتویات CX و DX را مقایسه می‌کند اگر با این داده‌ها به شکل بدون علامت رفتار نمایید مقدار CX بزرگتر است ، ولی اگر داده‌ها علامتدار باشند، مقدار CX کوچکتر است زیرا علامت منفی دارد. استفاده اینجا از CMP معتبر است و باید دستور پرش شرطی مناسب را انتخاب نمایید، مانند JB (پرش اگر کمتر است) را برای داده‌های بدون علامت یا JL (پرش اگر کوچکتر است) را برای داده‌های علامت دار استفاده کنید.

پرش‌های مربوط به داده‌های بدون علامت (منطقی)

برای داده‌های بدون علامت از دستورات پرش شرطی زیر استفاده کنید:

سمبول	توضیح	پرچم‌های تست شده
JE/JZ	پرش مساوی یا پرش صفر	ZF
JNE/JNZ	پرش غیر مساوی یا پرش غیر صفر	ZF
JG/JNBE	پرش بیشتر یا پرش کمتر نبودن یا مساوی	CF,ZF
JAE/JNB	پرش بیشتر نبودن یا مساوی یا پرش کمتر نبودن	CF
JB/JNAE	پرش کمتر یا پرش بیشتر نبودن یا مساوی	CF
JBE/JNA	پرش کمتر یا مساوی یا پرش بیشتر نبودن	AF,CF

می‌توانید هر یک از این پرش‌های شرطی را با یک یا دو عملیات سمبولیک خلاصه کنید، هر کدام که واضح‌تر و آشکارتر انتخاب کنید.

پرش‌های مربوط به داده‌های علامتدار (محاسباتی)

از جدول دستورات پرش شرطی زیر برای داده‌های علامتدار استفاده کنید:

سمبول	توضیح	پرچم‌های تست شده
JE/JNZ	پرش مساوی یا پرش صفر	ZF
JNE/JNZ	پرش غیر مساوی یا پرش غیر صفر	ZF
JA/JNLE	پرش بزرگتر یا پرش غیر کوچکتر یا مساوی	OF, SF, ZF
JGE/JNL	پرش غیر بزرگتر یا پرش غیر کوچکتر	OF, SF
JL/JNGE	پرش کوچکتر یا پرش غیر بزرگتر یا مساوی	OF, SF
JLE/JNG	پرش کوچکتر یا مساوی یا پرش غیر بزرگتر	OF, SF, ZF

پرش برای تست کردن مساوی یا صفر (JE/JZ) و برای تست کردن مساوی نبودن یا صفر نبودن (JNE/JNZ) در هر دو لیست داده‌های علامتدار و بدون علامت وجود دارد، زیرا یک شرط مساوی یا صفر، بدون توجه به حضور یا عدم حضور علامت انجام می‌گیرد.

تست‌های محاسباتی خاص

دستورات پرش شرطی زیر مورد استفاده خاص دارند:

سمبول	توضیح	پرچم‌های تست شده
JCXZ	پرش اگر CX صفر است	هیچ یک
JC	پرش اگر رقم نقلی وجود دارد	CF
JNC	پرش اگر رقم نقلی وجود ندارد	CF
JO	پرش اگر سرریز وجود دارد	OF
JNO	پرش اگر سرریز وجود ندارد	OF
JP/JPE	پرش اگر توازن وجود دارد یا توازن زوج است	PF
JNP/JPO	پرش اگر توازن وجود ندارد یا توازن فرد است	PF
JS	پرش اگر علامت دارد (منفی)	SF
JNS	پرش اگر علامت ندارد (مثبت)	SF

JCXZ محتویات ثبات CX را برای صفر بودن تست می‌کند. این دستور نیازی نیست تا بلافاصله بعد از یک عملیات محاسباتی یا مقایسه قرارگیرد. یک مورد استفاده JCXZ می‌تواند در شروع حلقه باشد، جهت اطمینان از انشعاب روتین اگر مقدار CX صفر است ، JC و JNC اغلب برای تست موفقیت در عملیات مورد استفاده قرار می‌گیرد.

اکنون ، انتظار نداشته باشید که همه دستورات را به خاطر بسپارید. توجه کنید که پرش برای داده‌های **بدون علامت** مساوی، بیشتر یا کمتر است، در حالیکه پرش برای داده‌های علامت‌دار مساوی، بزرگتر یا کوچکتر است. اسمبلر سمبول‌ها را با توجه به دستور استفاده شده به کد مقصد ترجمه می‌کند، بعنوان مثال JAE و JGE اگر چه مشابه به نظر می‌رسند، یک پرچم را تست نمی‌کنند (زیرا JAE داده‌ها را بدون علامت فرض می‌کند و JGE داده‌ها را علامتدار فرض می‌کند). پردازشگرهای 80386 و بعد پرش شرطی دور را مجاز می دانند. می‌توانید یک پرش دور یا کوتاه داشته باشید. برای مثال به شکل زیر:

```
JNB  SHORT address
JBE  FAR address
```

فراخوانی روال

سگمنت کد در مثال ما شامل فقط یک روال است، بدین شکل کد شده است:

```
proc-name   PROC   FAR
            .
            .
            .
proc-name   ENDP
```

در این حالت عملوند FAR به اسمبلر و لینکر آگاهی می‌دهد که نام روال نقطه شروع اجرای برنامه است، در حالیکه پیش پردازنده ENDP روال را تعریف می‌کند. یک سگمنت کد می‌تواند حاوی چندین روال باشد، که هر کدام با پیش‌پردازنده PROC و ENDP مربوطه مشخص می‌شود. یک روال فراخوانی شده (یا سابروتین) بخشی از کد است که یک وظیفه تعریف شده واضح را انجام می‌دهد (مانند تنظیم مکان‌نما یا گرفتن ورودی صفحه کلید) سازماندهی یک برنامه در روال فواید زیر را خواهد داشت :

- کاهش میزان کد ، زیرا یک روال عمومی می‌تواند از هر جایی که سگمنت کد فراخوانی شود.
- سازمان دهی بهتر برنامه.
- سهولت اشکال‌زدایی یک برنامه، زیرا معایب در صورت مجزا سازی واضح‌تر خواهد شد.
- کمک در نگهداری و تقویت برنامه، زیرا روال‌ها برای تغییر مشخص‌تر هستند.

عملیات CALL و RET

منظور از دستور CALL انتقال کنترل به روال فراخوانی شده است. (فقط مثالهای این کتاب هستند که پرش به روال شروع یک برنامه COM. دارند). دستور RET، در حقیقت المثنی CALL است، بازگشت از روال فراخوانی شده به روال فراخوانی اصلی. RET باید آخرین دستور در روال فراخوانی شده باشد.

قالب عمومی برای CALL و RET چنین است :

[label:]	CALL	proc-name
[label:]	RET	[POP-VALUE]

MASM 5.0 برای بازگشت نزدیک RETN و برای بازگشت دور RETF ابداع نموده است. کد مقصد ویژه که CALL و RET تولید می‌کنند به نوع عملیات یک زیر روال دور یا نزدیک بستگی دارد.

فراخوانی‌نزدیک و بازگشت. یک CALL برای زیرروال داخل‌همان سگمنت نزدیک‌است و مواردزیررا انجام‌می‌دهد.

- توسط PUSH ، SP ۲ تا کاهش می‌یابد و IP (حاوی افست دستور بعد از CALL) بروی پشته انتقال می‌یابد.
- آدرس افست زیر روال فراخوانی شده در IP قرار می‌گیرد. (این عملیات ، صف دستور از پیش واکشی شده پردازشگر را پاک می‌کند).
- یک RET (یا RETN) که از یک زیر روال نزدیک ، باز می‌گردد، اصولاً معکوس مراحل CALL را با عملیات POP دارد.
- انتقال مقادیر IP قبلی از پشته به IP (که صف دستور از پیش واکشی شده پردازشگر را پاک می‌کند)
- افزایش SP با ۲.

حال وقتی که اجرا از سر گرفته می‌شود CS:IP به دستور بعد از CALL اصلی در روال فراخوان اشاره می‌کند.

فراخوانی دور و بازگشت. یک CALL دور یک روال با برچسب FAR را فراخوانی می‌کند، که ممکن است در سگمنت دیگر، باشد. یک CALL دور ، هم CS و هم IP را بر روی پشته می‌گذارد و RET (یاRETF) هر دو را از پشته برمی‌دارد. فراخوانی دور و بازگشت از آن موضوعات فصل ۲۳ هستند.

مثالی از فراخوانی نزدیک و بازگشت

یک نوع ارگانیمز برای فراخوانی نزدیک و بازگشت در شکل ۳-۸ نشان داده شده‌اند. به نکات زیر توجه کنید:

- برنامه به یک روال دور A10MAIN و دو روال نزدیک ، B10، C10 تقسیم شده است. هر روال یک نام منحصر

بفرد دارد و حاوی ENDP خودش برای انتهای تعریف است.

- پیش پردازنده PROC برای A10MAIN صفت FAR دارند زیرا نقطه ورود و از خارج از برنامه می‌باشد.
- پیش پردازنده PROC برای B10، C10 صفت NEAR دارد مبنی بر اینکه این روال‌ها داخل سگمنت کد جاری می‌باشند. از آنجائیکه حذف این صفت سبب می‌شود تا اسمبلر پیش فرض NEAR را بپذیرد، خیلی از مثال‌های بعدی آن را حذف می‌کنند.
- در روال A10MAIN، دستور CALL کنترل برنامه را به روال B10 منتقل می‌کند و اجرای آن را آغاز می‌نماید.
- در روال B10 دستور CALL کنترل را به روال C10 منتقل می‌کند و اجرای آن را آغاز می‌نماید.
- در روال C10 دستور RET سبب بازگشت کنترل به دستور بلافاصله بعد از CALL C10 می‌شود.
- در روال B10 دستور RET سبب بازگشت کنترل به دستور بلافاصله بعد از CALL B10 می‌شود.
- روال A10MAIN پردازش را از این نقطه از سر می‌گیرد.
- RET همیشه به روتین فراخوان برمی‌گردد. اگر B10 با یک دستور RET خاتمه نیابد، پردازش در B10 ادامه می‌یابد و مستقیماً به C10 کشیده می‌شود. در حقیقت، اگر C10 حاوی یک RET نباشد، برنامه بعد از انتهای C10 تا هر جایی دستور باشد اجرا خواهد شد با نتایجی غیر قابل پیش‌بینی. توجه کنید که هر روال در یک مرز پاراگراف آغاز می‌شود: A10MAIN بر افست 0000 و B10 بر 0008 و C10 بر 000C (12).

همانطور که توضیح داده شد، می‌توانید کنترل را به یک روال نزدیک با کد خطی معمول منتقل کنید و همچنین می‌توانید با یک دستور پرش وارد یک روال نزدیک شوید. اگر چه برای وضوح و استحکام بیشتر از CALL برای انتقال به یک روال استفاده می‌کنید و برای خاتمه اجرای یک روال از RET/RETN استفاده می‌کنید.

```
                        page 60,132
                TITLE   A08CALLP (EXE)  Calling procedures
                        .MODEL SMALL
                        .STACK 64
                        .DATA
                ;-----------------------------------------------
                        .CODE
0000            A10MAIN PROC    FAR
0000  E8 0008 R        CALL    B10             ;Call B10
                ;       ...
0003  B8 4C00           MOV     AX,4C00H        ;End processing
0006  CD 21             INT     21H
0008            A10MAIN ENDP
                ;-----------------------------------------------
0008            B10     PROC    NEAR
0008  E8 000C R        CALL    C10             ;Call C10
                ;       ...
000B  C3                RET                     ;Return to
000C            B10     ENDP                    ;  caller
                ;-----------------------------------------------
000C            C10     PROC    NEAR
                ;       ...
000C  C3                RET                     ;Return to
000D            C10     ENDP                    ;  caller
                ;-----------------------------------------------
                        END     A10MAIN
```

شکل ۳-۸ روال‌های فراخوان

تاثیر اجرای برنامه بر روی پشته

تا اینجا، برنامه‌های ما نیاز اندکی برای گذاشتن داده‌ها بر روی پشته داشتند و در نتیجه باید یک پشته کوچک تعریف می‌شد. همانطور که در شکل ۳-۸ مشخص شد، یک روال فراخوانی شده می‌تواند روال دیگری را فراخوانی نماید، که آن نیز به نوبه خود هنوز می‌تواند یک روال دیگری را فراخوانی کند، بنابراین پشته باید بقدر کافی بزرگ باشد تا حاوی

همه آدرس‌های گذاشته شده باشد. همه این موارد ساده‌تر از آنچه که به نظر می‌رسد، معلوم شد و یک پشته تعریف شده با ۳۲ کلمه برای اغلب اهداف کافی است.

CALL و PUSH هر دو یک آدرس یا مقدار یک کلمه‌ای را بر روی پشته می‌گذارند. RET و POP از روی پشته کلمه‌ای را که قبلاً گذاشته شده بود دستیابی کرده و برمی‌دارند. همه این عملیات آدرس افست در ثبات SP را برای کلمه بعدی افزایش یا کاهش می‌دهند. زیرا باید عملیات RET و POP با عملیات CALL و PUSH اصلی جور باشند.

در بارگذاری یک برنامه EXE. جهت اجرا، برنامه بارگذار مقادیر ثبات‌های زیر را مقدار دهی می‌کند:

- DS و ES : آدرس PSP یک ناحیه ۲۵۶ بایتی است که در ابتدای ماجول برنامه اجرایی در حافظه قرار می‌گیرد.
- CS : آدرس سگمنت کد است - نقطه ورود به برنامه شما.
- IP : صفر، اگر اولین دستور اجرایی در شروع سگمنت کد باشد.
- SS : آدرس سگمنت پشته.
- SP : افست بالای پشته برای مثال، برای یک پشته تعریف شده به شکل STACK 64 (۶۴ بایت یا ۳۲ کلمه). SP حاوی ۶۴ یا 40H خواهد بود.

بیایید برنامه سادهٔ شکل ۳-۸ را در طی اجرای آن پیگیری کنیم. عملاً روال فراخوانی شده حاوی هر تعداد از دستورات می‌تواند باشد.

موقعیت در دسترس جاری برای گذاشتن یا برداشتن ، بالای پشته است. برای این مثال، بارگذار برنامه باید SP را با اندازه پشته تنظیم نماید، یعنی ۶۴ بایت (40H). کلمات حافظه حاوی بایت‌های به ترتیب معکوس هستند، برای مثال 0003 خواهد شد 0300. برنامه این عملیات را انجام می‌دهد:

CALL B10، SP را ۲ تا کاهش می‌دهد، از 40H به 3EH، سپس IP (حاوی 0003، موقعیت دستور بعدی) را بر روی بالای پشته در افست 3EH قرار می‌دهد. پردازشگر از آدرسی که توسط CS:IP شکل گرفته برای انتقال کنترل به B10 استفاده می‌کند.

CALL B10(PUSH 000B) : افست پشته

SP=3E00H	0036	0038	003A	003C	003E
	xxxx	xxxx	xxxx	xxxx	0300

- در روال B10 و CALL C10 و SP را تا ۲ کاهش می‌دهد و 3CH خواهد شد. سپس IP (حاوی 000B) را بر بالای پشته در افست 3CH قرار می‌دهد. پردازشگر از آدرس CS:IP برای انتقال کنترل به C10 استفاده می‌کند.

CALL C10(PUSH 000B) : افست پشته

SP=3C00H	0036	0038	003A	003C	003E
	xxxx	xxxx	xxxx	0B00	0300

- برای بازگشت از C10 دستور RET افست (000B) را ازبالای پشته در 3CH برمی‌دارد، آن را در IP قرارمی‌دهد و SP را ۲ تا می‌افزاید که 3EH خواهد شد. IP سبب یک بازگشت اتوماتیک به افست 000BH در روال B10 خواهدشد.

RET (POP 000B) : افست پشته

SP=3E00H	0036	0038	003A	003C	003E
	xxxx	xxxx	xxxx	0B00	0300

- RET در انتهای روال B10 آدرس (0003) را از بالای پشته در 3EH برمی‌دارد و در IP می‌گذارد و SP را ۲ تا می‌افزاید که خواهد شد 40H. افست IP سبب یک بازگشت اتوماتیک به افست 0003H خواهد شد، در جائیکه برنامه اجرای فرد را خاتمه می‌دهد.

RET (POP 0003) : افست پشته

SP=4000H	0036	0038	003A	003C	003E
	xxxx	xxxx	xxxx	0B00	0300

- اگر شما از DEBUG برای دیدن پشته استفاده کنید، ممکن است داده‌های بی‌ضرری در سمت چپ بیایید که توسط اجرای برنامه قبلی ایجاد شده‌اند.

عملیات بولی

منطق بولی در طراحی مدارات بسیار مهم است و با منطق برنامه‌نویسی برابری می‌کند. دستورات منطق بولی AND، OR ، XOR ، TEST و NOT هستند که برای صفر و یک کردن بیت‌ها و دستکاری داده‌های ASCII در اهداف محاسباتی (فصل ۱۳) استفاده می‌شوند قالب کلی عملیات بولی چنین است.

[label:]	operation	register/memory, register/memory/immediate

اولین عملوند یک بایت، کلمه یا دو کلمه (در 80386 و بعد) یک ثبات یا حافظه را ارجاع می‌دهد و تنها مقداری است که تغییر می‌کند. دومین عملوند یک ثبات یا مقدار بلافصل را ارجاع می‌دهد. عملیات بیت‌های دو عملوند ارجاع شده را جور می‌کند و پرچم‌های CF ، OF ، PF ، SF و ZF را بر طبق آن تنظیم می‌کند (AF تعریف نشده است).

- AND : اگر بیت‌ها جور شده هر دو ۱ باشند، حاصل ۱ خواهد بود. در بقیه شرایط حاصل صفر است.
- OR : اگر هر یک از (یا هر دو) از بیت‌های جور شده ۱ باشند، حاصل ۱ خواهد بود. اگر هر دو بیت ۰ باشند حاصل ۰ است.
- XOR : اگر یکی از بیت‌های جور شده، ۰ و دیگری ۱ باشد، حاصل ۱ خواهد بود. اگر هر دو بیت یکسان (هر دو ۰ یا هر دو ۱) باشند، حاصل ۰ خواهد بود.
- TEST : پرچم‌ها را مانند عملیات AND تنظیم می‌کند، اما بیت‌های ارجاع شده در اولین عملوند را تغییر نمی‌دهد.

عملیات AND ، OR و XOP زیر تاثیر استفاده از بیت‌های داده را مشخص می‌سازد:

	AND	OR	XOR
عملوند ۱ :	0101	0101	0101
عملوند ۲ :	0011	0011	0011
نتیجه دو عملوند ۱ قرار می‌گیرد :	0001	0111	0110

یک قانون مفید را بخاطر بسپارید : AND کردن بیت با 0 آن را صفر می‌کند، در حالیکه OR بیت با ۱ آن را یک می‌کند.

مثالهایی از عملیات بولی

برای مثالهای غیر مرتبط زیر، فرض کنید BL حاوی 1010 CH,0011 حاوی 10100011 می‌باشد :

```
۱.  AND   BL,F0H    ; BL را با 1010 0000 تنظیم می‌کند
۲.  AND   BL,00H    ; BL را با 0000 0000 تنظیم می‌کند
۳.  AND   BL,CH     ; BL را با 0010 0010 تنظیم می‌کند
۴.  OR    CH,BL     ; CH را با 1011 1011 تنظیم می‌کند
۵.  OR    CH,CH     ; SF و ZF را تنظیم می‌کند
۶.  XOR   BL,OFFH   ; BL را با 1100 0101 تنظیم می‌کند
۷.  XOR   BL,BL     ; BL را با 0000 0000 تنظیم می‌کند
```

مثال ۱ همهٔ بیت سمت چپ AL را صفر می‌کند، مثال ۲ و ۷ روشهایی از پاک کردن یک ثبات با صفر می‌باشد اگر چه استفاده از CMP واضح‌تر است، می‌توانید از OR برای اهداف زیر استفاده کنید:

```
۱.  OR    DX,DX     ; تست DX برای صفر
    JZ    . . .     ; پرش اگر صفر است
۲.  OR    DX,DX     ; تست DX برای صفر
    JS    . . .     ; پرش اگر علامت منفی است
```

TEST مانند AND عمل می‌کند، ولی فقط بر پرچم‌ها اثر می‌گذارد، در اینجا چند مثال آورده شده است .

```
۱.  TEST  CX,0FFH      ; آیا CX حاوی
    JZ    ...          ; یک مقدار صفر است؟
۲.  TEST  BL,00000001B ; آیا BL حاوی
    JNZ   ...          ; یک مقدار فرد است؟
۳.  TEST  CL,11110000B ; آیا ۴ بیت سمت چپ
    JNZ   ...          ; در CL غیر صفر است؟
```

دستور NOT

دستور NOT بیت‌های یک بایت، کلمه یا دو کلمه‌ای (80386 و بعد) در یک ثبات یا حافظه را معکوس می‌کنند، بنابراین 0ها 1 خواهد شد و 1ها 0 خواهد شد. قالب کلی چنین خواهد بود :

[label:]	NOT	register/memory

برای مثال اگر BL حاوی 1010 0011 باشد، دستور NOT BL با BL را به 010 1100 تغییر خواهد داد. (تاثیر آن دقیقاً مانند 0FFH ، XORBL در مثال ۶ اخیر می‌باشد.)

پرچم‌ها بی‌تاثیر باقی می‌مانند. NOT با NEG تفاوت دارد، که یک مقدار دودویی را از مثبت به منفی یا برعکس ، توسط معکوس ساختن بیت‌ها و جمع آن با یک انجام می‌دهد.

برنامه : تغییر حروف بزرگ به حروف کوچک

دلایل متعددی برای تبدیل، بین حروف بزرگ به حروف کوچک وجود دارد. مثلاً ممکن است فایل داده‌ای را دریافت کرده باشید که همه داده‌ها با حروف بزرگ باشد. یا برنامه‌ای که اجازه وارد کردن مقادیر را هم به صورت حروف بزرگ و هم به صورت حروف کوچک (مثل YES یا yes) به کاربر می‌دهد و برای تست آن را به حروف بزرگ تبدیل می‌کند. حروف بزرگ را از A تا Z معادل مقادیر ASCII در مبنای شانزده 41H تا 54H و حروف کوچک a تا z معادل 61H تا 6AH می‌باشند. تنها تفاوت این است که بیت پنجم برای حروف بزرگ 0 و برای حروف کوچک 1 می‌باشد.

برنامه COM. در شکل ۴-۸ محتویات عنصر داده CONAME را از حروف بزرگ به حروف کوچک با شروع از CONAME+1 تبدیل می‌کند. برنامه ثبات BX را با آدرس CONAME+1 مقداردهی می‌کند و از این آدرس برای انتقال هر کاراکتر با آدرس شروع CONAME+1 در AH استفاده می‌کند. اگر مقدار بین 41H و 5AH باشد، یک دستور XOR بیت 5 را با صفر پاک می‌کند.

حروف بزرگ		حروف کوچک	
حرف A :	01000001	حرف a :	01100001
حرف Z :	01011010	حرف z :	01111010
بیت :	۷۶۵۴۳۲۱۰	بیت :	۷۶۵۴۳۲۱۰

```
XOR  AH,00100000B
```

همه کاراکترها بجز A تا Z بدون تغییر باقی می‌مانند. آنگاه روال، کاراکتر تغییر یافته را به محل CONAME باز می‌گرداند و ثبات BX را برای کاراکتر بعدی افزایش می‌دهد. حلقهٔ برنامه ۱۶ بار تکرار می‌شود، یک بار برای هر کاراکتر با شروع از CONAME+1 با استفاده از این روش، ثبات BX مانند یک ثبات نشانگذاری برای آدرس دهی موقعیت‌های حافظه عمل می‌کند. شما همچنین از SI و DI برای این منظور می‌توانید استفاده کنید.

در انتها، برنامه محتویات تغییر یافته CONAME را نمایش می‌دهد. یک بار که برنامه اسمبل شود می‌توانید آن را به شکل استاندارد یا داخل DEBUG اجرا نمایید.

شیفت بیت‌ها

دستورات شیفت، که قسمتی از قابلیت منطقی کامپوتر هستند، می‌توانند عملیات زیر را انجام دهند.

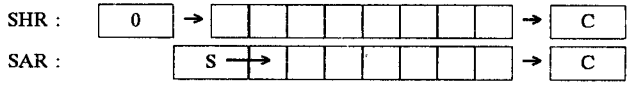
- ارجاع یک ثبات یا آدرس حافظه.
- شیفت بیت‌ها به چپ یا راست.
- شیفت تا ۸ بیت در یک بایت، ۱۶ بیت در یک کلمه و ۳۲ بیت در یک دو کلمه (80386 و بعد).
- شیفت منطقی (بدون علامت) یا محاسباتی (علامتدار).

دومین عملوند حاوی مقدار شیفت است که یک مقدار ثابت (یک مقدار بلافصل) یا یک ارجاع به ثبات CL می‌باشد. برای پردازشگرهای 8088/8086 مقدار شیفت فقط یک است، مقادیر بزرگتر از ۱ باید در ثبات CL گذاشته شوند، در حالیکه پردازشگرهای بعد از آن‌ها مقادیر شیفت تا ۳۱ را نیز مجاز می‌دانند. قالب کلی دستور شیفت چنین است :

[label:]	shift	register/memory, CL/immediate

شیفت به راست بیت‌ها

عملیات SHR و SAR بیت‌ها در ثبات تعیین شده را به راست شیفت می‌دهند. هر بیت خروجی شیفت داده شده وارد پرچم نقلی می‌شود. SHR (شیفت منطقی به راست) برای داده‌های منطقی (بدون علامت) و SAR (شیفت محاسباتی به راست) برای داده‌های محاسباتی علامتدار فراهم شده است :



دستورات مرتبط زیر استفاده از SHR را برای شیفت داده‌های بدون علامت استفاده می‌کنند:

دستور	توضیح:	دودویی	دهدهی	CF
MOV BH,10110111B	; مقدار توضیح BH	10110111	183	-
SHR BH,01	; شیفت یکی به راست	01011011	91	1
MOV CL,02	; تنظیم مقدار شیفت			
SHR BH,CL	; شیفت‌به‌راست ۲ مرتبه	00010110	22	1
SHR BH,02(80286+)	; شیفت‌به‌راست ۲ مرتبه	00000101	5	1

اولین SHR محتویات BH را یک بیت به راست شیفت می‌دهد. بیت شیفت داده شده در پرچم نقلی قرار می‌گیرد و یک صفر سمت چپ BH را پر می‌کند. دومین SHR ، BH را دو بیت بیشتر شیفت می‌دهد. پرچم نقلی به ترتیب حاوی ۱ و ۱ خواهد شد و ۲ بیت صفر سمت چپ BH را پر می‌کنند. سومین SHR ، BH را دو بیت بیشتر شیفت خواهد داد.

SAR با SHR در یک روش مهم تفاوت دارد : SAR از بیت علامت برای پر کردن بیت خالی استفاده می‌کند با این روش، مقادیر مثبت و منفی علامت را نگه می‌دارند. دستورات مرتبط زیر استفاده از SAR را برای شیفت دادن داده‌های بدون علامت بطریقی که علامت بیت یک باشد، مشخص می‌سازد.

دستور	توضیح:	دودویی	دهدهی	CF
MOV BH,10110111B	; مقدار توضیح BH	10110111	-37	-
SHR BH,01	; شیفت یکی به راست	01011011	-37	1
MOV CL,02	; تنظیم مقدار شیفت			
SAR BH,CL	; شیفت به راست ۲ مرتبه	11110110	-10	1
SHR BH,02 (80286+)	; شیفت به راست ۲ مرتبه	11111101	-3	1

شیفت به راست مخصوصاً برای **تقسیم بر دو** مقادیر و اجرای سریع‌تر عمل تقسیم مورد استفاده قرار می‌گیرد. در مثالهای SHR و SAR اولین شیفت به راست در حقیقت تقسیم بر دو می‌کند و دومین و سومین شیفت به راست هر بار بر ۴ تقسیم می‌کند.

تقسیم‌بردواعداد فردمانند ۵ و ۷، اعداد ۲ و ۳ تولیدمی‌کند و درنتیجه پرچم‌نقلی ۱ خواهدشد. بعداز عملیات‌شیفت، شمامی‌توانید از دستور JC (پرش اگر رقم نقلی وجود دارد) برای تست بیت شیفت داده‌شده و پرچم‌نقلی استفاده‌کنید.

```
TITLE     A08CASE (COM)  Change uppercase to lowercase
          .MODEL SMALL
          .CODE
          ORG    100H
BEGIN:    JMP    A10MAIN
; ------------------------------------------------
CONAME    DB     'INTERTECH SYSTEMS', '$'
; ------------------------------------------------
A10MAIN   PROC   NEAR
          LEA    BX,CONAME+1      ;1st char to change
          MOV    CX,16            ;No. of chars to change
A20:
          MOV    AH,[BX]          ;Character from CONAME
          CMP    AH,41H           ;Is it
          JB     A30              ;  upper
          CMP    AH,5AH           ;  case
          JA     A30              ;  letter?
          XOR    AH,00100000B     ;Yes, convert
          MOV    [BX],AH          ;Restore in CONAME
A30:
          INC    BX               ;Set for next char
          LOOP   A20              ;Loop 16 times
                                  ;Done,
          MOV    AH,09H           ;  display
          LEA    DX,CONAME        ;  CONAME
          INT    21H
          MOV    AX,4C00H         ;End processing
          INT    21H
A10MAIN   ENDP
          END    BEGIN
```

شکل ۴-۸ تغییر حروف بزرگ به حروف کوچک

شیفت ثبات‌های ۳۲ بیتی

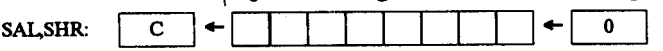
مثال زیر BX:AX را به ECX منتقل می‌کند، در حالیکه یک عملیات شیفت مقدار را به ۲ تقسیم می‌کنند :

```
MOV CX,BX    ; BX به پایین رتبه ECX
SHL ECX,16   ; شیفت به بالا رتبه ECX
MOV CX,AX    ; AX به پائین رتبه ECX
SHR ECX,01   ; تقسیم بر دو ECX
```

در 80386 و بعد، SHRD نیز برای شیفت مقادیر ۱۶ و ۳۲ بیتی استفاده می‌شود.

شیفت به چپ بیت‌ها

عملیات SHR (شیفت به چپ منطقی) و SAL (شیفت به چپ محاسباتی) بیت‌های ثبات مشخص شده را به چپ شیفت می‌دهند. هر بیت شیفت دادهٔ خروجی به پرچم نقلی وارد می‌شود. هر دو دستور در عملیات یکسان هستند و هر دو برای داده‌های منطقی (بدون علامت) و محاسباتی (علامتدار) فراهم شده‌اند.



دستورات مرتبط زیر استفاده از SHL به منظور شیفت داده‌های بدون علامت را مشخص می‌سازد.

دستور	توضیح:	دودویی	دهدهی	CF
MOV BH,10110111B	; مقدار توضیح BH	00000101	5	-
SHL BH,01	; شیفت یکی به چپ	00001010	10	0
MOV CL,02	; تنظیم مقدار شیفت			
SHL BH,CL	; شیفت‌به چپ ۲ مرتبه	00101000		
SHL (80286+)BH,02	; شیفت‌به چپ ۲ مرتبه	10100000	160	0

اولین SHL محتویات BH را یک بیت به چپ شیفت می‌دهد. ۱ بیت شیفت داده شده در پرچم نقلی قرار می‌گیرد و یک بیت 0 سمت راست BH را پر می‌کند. دومین SHL ، BH را دو بیت بیشتر شیفت می‌دهد. پرچم نقلی به ترتیب 0 و 0 خواهد شد و دو بیت 0 سمت راست BH را پر می‌کند. سومین SHL و BH را دو بیت بیشتر شیفت می‌دهد.

شیفت به چپ همیشه در سمت راست 0 قرار می‌دهد. همانطور که در نتیجه است ، SHL و SAL با هم یکسانند، بنابراین SAL می‌تواند در مثالهای قبلی با همان تاثیر استفاده شود. شیفت به چپ مخصوصاً برای دو برابر کردن مقادیر و اجرای سریعتر علمی به نسبت یک عملیات ضرب استفاده می‌شود. در مثالهای عملیات شیفت به چپ، اولین شیفت به چپ تاثیر ضرب در دو را دارد و دومین و سومین شیفت به چپ هر کدام در ۴ ضرب می‌شود.

بعد از عملیات شیفت، شما می‌توانید از دستور JC (پرش اگر رقم نقلی وجود دارد) برای تست بیت شیفت داده شده به داخل پرچم نقلی استفاده کنید.

شیفت دادن ثبات‌های ۳۲ بیتی

مثال زیر DX:AX را به ECX منتقل می‌کند و یک عملیات شیفتی مقدار آن را دو برابر می‌سازد :

```
MOV CX,DX      ; انتقال DX به پایین رتبه ECX
SHL ECX,16     ; شیفت به بالا رتبه ECX
MOV CX,AX      ; انتقال AX به پایین رتبه ECX
SHL ECX,01     ; ضرب ECX در ۲
```

برای 80386 و بعد SHLD برای شیفت مقادیر ۱۶ و ۳۲ بیتی می‌تواند استفاده شود.

چرخش بیت‌ها

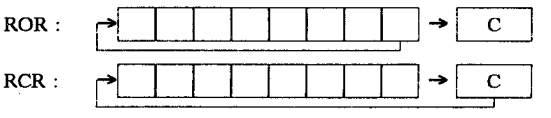
دستورات چرخش، که بخشی از قابلیت منطق کامپیوتر است می‌تواند فعالیت‌های زیر را انجام دهد:

- ارجاع یک ثبات یا حافظه.
- چرخش به راست یا چپ. بیتی که به خارج شیفت داده شده است موقعیت بیت خالی را در حافظه یا ثبات پر می‌کند و همچنین در پرچم نقلی کپی می‌شود.
- چرخش تا ۸ بیت در یک بایت، ۱۶ بیت در یک کلمه و ۳۲ بیت در یک دو کلمه‌ای (80836 و بعد).
- چرخش منطقی (بدون علامت) یا محاسباتی (علامتدار).
- دومین عملوند حاوی مقدار چرخش است، که یک مقدار ثابت است (یک مقدار بالافصل) یا ارجاع به ثبات CL می‌باشد. برای پردازشگر 8088/8086 مقدار چرخش باید فقط ۱ باشد، مقادیر بزرگتر از ۱ باید در ثبات CL باشد در حالیکه پردازشگرهای بعدی مقدار چرخش تا ۳۱ را مجاز می دانند. قالب کلی برای چرخش چنین است :

[label:]	register/memory,CL/immediate

چرخش بیت به راست

عملیات ROR و RCR بیت‌های یک ثبات مشخص شده را به راست چرخش می‌دهند، هر بیت که با چرخش خارج می‌شود، وارد پرچم نقلی می‌شود. ROR (چرخش منطقی به راست) برای داده‌های منطقی (بدون علامت) و RCR (چرخش با رقم نقلی به راست) برای داده‌های محاسباتی (علامتدار) فراهم شده است.



دستورات مرتبط زیر ROR را مشخص می‌سازد.

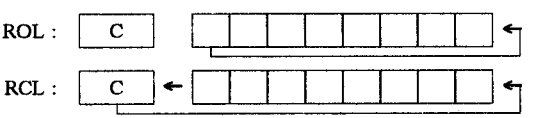
دستور	توضیح	دودویی	CF
MOV BL,10110111B	; مقداردهی BH	10110111	-
ROR BL,01	; چرخش یکی به راست	01011011	1
MOV CL,03	; تنظیم مقدار چرخش		
ROR BL,CL	; چرخش به راست ۳ مرتبه	01111011	0
ROR BL,02(80286+)	; چرخش به راست ۳ مرتبه	01101111	0

اولین ROR سمت راست‌ترین بیت از BL را به موقعیت خالی سمت چپ چرخش می‌دهد و به CF نیز می‌فرستد. دومین و سومین عملیات ROR سه بیت سمت راست را به موقعیت‌های خالی سمت چپ چرخش می‌دهد و در CF نیز می‌فرستد.

RCR با ROR در این روش تفاوت دارد. هر بیت که به خارج و به راست چرخش می‌یابد ابتدا به داخل CF منتقل می‌شود و بیت CF به موقعیت بیت خالی در سمت چپ نیز منتقل می‌شود.

چرخش بیت‌ها به چپ

ROL و RCL بیت‌های یک ثبات مشخص را به چپ چرخش می‌دهند. هر بیت که با چرخش خارج می‌شود وارد پرچم نقلی می‌شود. ROL (چرخش منطقی به چپ) برای داده‌های منطقی (بدون علامت) و RCL (چرخش با رقم نقلی به راست) برای داده‌های محاسباتی (علامتدار) فراهم شده است :



دستورات مرتبط زیر ROL را مشخص می‌سازند:

دستور	توضیح	دودویی	CF
MOV BL,10110111B	; مقداردهی BL	10110111	-
ROL BL,01	; چرخش یکی به چپ	01101111	1
MOV CL,03	; تنظیم مقدار چرخش		
ROL BL,CL	; چرخش به چپ ۳ مرتبه	01111011	0
ROL BL,02	; چرخش به چپ ۳ مرتبه	11011011	0

اولین ROL سمت‌چپ‌ترین بیت از BL را به موقعیت خالی سمت‌راست چرخش‌می‌دهد و به CF نیزمی‌فرستد. دومین و سومین ROL سه بیت سمت چپ را به موقعیت‌های خالی سمت‌راست چرخش می‌دهد و در CF نیز می‌فرستد.

RCL با ROL در این روش تفاوت دارد. هر بیت که به خارج و به راست چرخش می‌یابد ابتدا به داخل CF منتقل می‌شود و بیت CF به موقعیت بیت خالی در سمت چپ نیز منتقل می‌شود.

بعد از یک عملیات چرخش، می‌توانید از دستور JC (پرش اگر رقم نقلی وجود دارد) برای تست بیت چرخش شده در CF استفاده کنید.

چرخش و شیفت دو کلمه‌ای

می‌توانید از دستورات شیفت و چرخش جهت ضرب و تقسیم مقادیر دو کلمه‌ای توسط ضرب در ۲ استفاده کنید فرض کنید یک مقدار ۳۲ بیتی دارید که مقدار بیت‌های سمت چپ در DX و بیت‌های سمت راست در AX قرار گرفته

است، به شکل DX:AX. دستوراتی که ضرب در ۲ را انجام می‌دهند می توانند چنین باشند.

```
SHL AX,1        ; استفاده از شیفت چپ برای ضرب
RCL DX,1        ; زوج DX:AX در ۲
```

عملیات SHL همه بیت‌های AX را به چپ شیفت‌می‌دهد، و بیت سمت‌چپ به پرچم‌نقلی منتقل‌می‌شود. RCL، DX را به چپ چرخش داده و بیت CF را در بیت خالی سمت راست قرار می‌دهد. برای ضرب در ۴، بعد از RCL-SHL یک جفت RCL-SHL همانند قرار دهید.

برای تقسیم، مجدداً یک مقدار ۳۲ بیت در DX:AX در نظر بگیرید. دستورات برای تقسیم بر ۲ چنین هستند :

```
SAR DX,1        ; استفاده از شیفت چپ برای تقسیم
RCR AX,1        ; زوج DX:AX بر ۲
```

برای تقسیم بر ۴ ، بعد از زوج RCR-SAR یک زوج ROR-SAR همانند قرار دهید. SHLD و SHRD برای پردازشگرهای 80586 و بعد شیفت با دقت مضاعف را فراهم نموده است.

جداول پرش

یک برنامه ممکن است روتینی برای تست نمودن تعدادی از شرایط مرتبط داشته باشد، هر یک پرش به روتین دیگر لازم دارد. فرض کنید، برای مثال، سیستمی برای یک کمپانی وجود دارد، که کدهای ویژه برای مشتریان بر حسب درجه اعتبار و میزان خرید آنها در نظر گرفته است. کدهایی (با نام DISCODE) بر میزان سود پیشنهادی و بقیه پردازش‌هایی که ممکن است برای مشتریان نیاز باشد، دلالت دارد. کدهای مشتریان ۰، ۱، ۲، ۳، ۴ می‌باشد.

یک روش مرسوم برای دستیابی به کد به ترتیب مقایسه کردن برای هر کد مشتری است :

```
CMP   DISCODE , 0   ; Code = 0 ?
JE    D00DSCT
CMP   DISCODE , 1   ; Code = 1 ?
JE    D10DSCT
CMP   DISCODE , 2   ; Code = 2 ?
JE    D20DSCT
CMP   DISCODE , 3   ; Code = 3 ?
JE    D30DSCT
CMP   DISCODE , 4   ; Code = 4 ?
JE    D40DSCT
...
```

بدینوسیله، فرصت خطاها زیاد است زیرا لازم است تا کدهای صحیح برطبق مقادیر آنها با هم جور باشند و پرش به روتین صحیح انجام گیرد. یک راه حل ظریف شامل جدولی از آدرس‌های پرش است. همانطور که در بخشی از برنامه در شکل ۵-۸ نشان داده شده است CUSTTAB پنج آدرس پیاپی در یک کلمه (۲ بایت برای هر یک) تعریف شده است.

روال B10JUMP یک کاراکتر از صفحه کلید می‌گیرد و در AL می‌گذارد و آن را به BX منتقل می‌کند. مقدار دو برابر می‌شود، بنابراین ۰، صفر می‌ماند، ۱ دو خواهد شد، ۲ چهار خواهد شد و الی آخر. مقدار دو برابر شده یک افستْ در جدول فراهم شده است : CUSTTAB+0 اولین آدرس است، CUSTTAB+2 دومین آدرس است. CUSTTAB+4 سومین آدرس است و الی آخر. عملوند دستور JMP [CUSTTAB+BX] است، که با آدرس شروع جدول بعلاوه یک افست در جدول شکل می‌گیرد سپس عملیات به روتین صحیح مستقیماً پرش خواهد داشت.

یک ضرورت مهم در برنامه این است که فقط می‌تواند از مقادیر شانزدهی 00-40 استفاده کند. هر مقدار دیگری ممکن است نتایج شومی در بر داشته باشد و برنامه با این احتمال باید تست شود. اگر برنامه را اجرا نمودید مقادیر معتبر (0-4) برای بررسی تاثیر این منطق وارد نمایید. دستور MOVZX برای انتقال یک بایت کاراکتر صفحه کلید به BX استفاده می‌شود که فقط برای پردازشگرهای 80386 و بعد عمل می‌کند. برای پردازشگرهای قبلی، می‌توانید آن را با دستورات زیر جایگزین کنید.

```
XOP BH,BH       ; پاک کردن بالای BX
MOV BL,AL       ; انتقال کد بهره
```

```
                                TITLE   A08JMPTB (EXE)  Using a jump table
                                        .MODEL SMALL
                                        .STACK 64
                                        .DATA
0000  001E R                    CUSTTAB DW      B10CDE0   ;Table of addresses
0002  0025 R                            DW      B11CDE1
0004  002C R                            DW      B12CDE2
0006  0033 R                            DW      B13CDE3
0008  003A R                            DW      B14CDE4
000A  43 6F 64 65 20 30         MESSG0  DB      'Code 0 processing', '$'
001C  43 6F 64 65 20 31         MESSG1  DB      'Code 1 processing', '$'
002E  43 6F 64 65 20 32         MESSG2  DB      'Code 2 processing', '$'
0040  43 6F 64 65 20 33         MESSG3  DB      'Code 3 processing', '$'
       73 73 69 6E 67 24
0052  43 6F 64 65 20 34         MESSG4  DB      'Code 4 processing', '$'
                                ; ---------------------------------------
                                        .CODE
                                .386
0000                            A10MAIN PROC    FAR
0000  B8 ---- R                         MOV     AX,@data        ;Initialize
0003  8E D8                             MOV     DS,AX           ;  segment
0005  8E C0                             MOV     ES,AX           ;  registers
0007  E8 000F R                         CALL    B10JUMP
000A  B8 4C00                           MOV     AX,4C00H        ;End processing
000D  CD 21                             INT     21H
000F                            A10MAIN ENDP
                                ;
000F                            B10JUMP PROC    NEAR
000F  B4 10                             MOV     AH,10H          ;Get KB char
0011  CD 16                             INT     16H             ;  into AL
0013  24 07                             AND     AL,00000111B    ;Clear left 5 bits
0015  0F B6 D8                          MOVZX   BX,AL           ;Move AL to BX
0018  D1 E3                             SHL     BX,01           ;Double value
001A  FF A7 0000 R                      JMP     [CUSTTAB+BX]    ;Jump to cust rtne
001E  8D 16 000A R              B10CDE0: LEA    DX,MESSG0       ;Code 0 routine
0022  EB 1D 90                          JMP     B90
0025  8D 16 001C R              B11CDE1: LEA    DX,MESSG1       ;Code 1 routine
0029  EB 16 90                           JMP     B90
002C  8D 16 002E R              B12CDE2: LEA    DX,MESSG2       ;Code 2 routine
0030  EB 0F 90                           JMP     B90
0033  8D 16 0040 R              B13CDE3: LEA    DX,MESSG3       ;Code 3 routine
0037  EB 08 90                           JMP     B90
003A  8D 16 0052 R              B14CDE4: LEA    DX,MESSG4       ;Code 4 routine
003E  EB 01 90                           JMP     B90
0041                            B90:
0041  B4 09                              MOV     AH,09H          ;Display
0043  CD 21                              INT     21H
0045  C3                                RET
0046                            B10JUMP ENDP
                                        END     A10MAIN
```

شکل ۵-۸ استفاده از جدول پرش

سازمان یک برنامه

موارد زیر نمونه مراحل نوشتن یک برنامه اسمبلی می‌باشند:

۱- ایده روشنی از مسئله‌ای که برنامه باید آن را حل کند، داشته باشید.

۲- ایده‌های خود را واژه‌های کلی، به طور خلاصه شرح‌دهد و منطق کلی را طراحی نمایید. برای مثال اگر مسئله عملیات انتقال ضرب باشد، با تعریف فیلدهایی که باید منتقل شود شروع نمایید. سپس استراتژی دستورالعمل‌ها یعنی روالهایی برای مقداردهی اولیه، برای استفاده یک پرش شرطی و برای استفاده یک حلقه طراحی کنید. موارد زیر، که نشان دهنده منطق اصلی است شبه کدی است که اکثر برنامه نویسان برای طراحی یک برنامه استفاده می‌کنند.

- مقدار دهی پشته و ثبات‌های سگمنت.
- فراخوانی روال پرش.

- فراخوانی روال حلقه.
- خاتمه پردازش.

روال پرش می‌تواند به صورت زیر طراحی شود:

مقدار دهی ثبات‌ها برای شمارش، آدرس نامها

پرش ۱ :

- انتقال یک کاراکتر از اسم.
- افزودن برای کاراکترهای بعدی نام.
- کاهش شمارش : اگر غیر صفر است ، پرش ۱ اگر صفر است، بازگشت.

روال حلقه، بطور خلاصه می تواند با روش مشابه توصیف گردد.

۳- برنامه را به واحدهای منطقی سازماندهی نمایید به طوری که روالهای مرتبط متعاقب یکدیگر قرار گیرند. رویه‌ای که در مورد ۲۵ خط (اندازه صفحه تصویر) باشد برای اشکال زدایی آسانتر است.

۴- برنامه‌های مثال را به عنوان یک راهنما استفاده نمایید، سعی کنید تمام موارد تکنیکی را بخاطر بسپارید.

۵- توضیحات را برای مشخص کردن وظیفه یک روال، وظایف اعمال حسابی و مقایسه‌ای و وظیفه دستورالعملی که به ندرت کاربرد دارد، استفاده کنید. (یک مثال LOOPNE است : حلقه را تا زمانیکه مساوی نیست تکرار کن).

۶- برای وارد کردن برنامه‌های این متن به طور قابل توجهی از دستور العملهای JMP و LOOP پرش‌های شرطی، CALL و روال‌های فراخوانده شده، استفاده می‌کنند. با دانستن اصول بنیانی اسمبلر، اکنون در موقعیت شروع برنامه‌نویسی پیشرفته قرار دارید.

نکات کلیدی

- یک آدرس کوتاه که با یک افست بیان می‌شود به فاصله ۱۲۸- تا ۱۲۷ بایت محدود می‌شود. یک آدرس نزدیک با یک افست محدود به فاصله ۳۲۷۶۸- تا ۳۲۷۶۷ بایت در داخل همان سگمنت می‌شود. یک آدرس دور در سگمنت دیگر با یک آدرس سگمنت و افست بیان می‌شود.
- یک برچسب دستور مانند : P50 یک علامت : مبنی بر آنکه یک برچسب نزدیک می‌باشد، دارد.
- برچسب‌های پرش‌شرطی و دستورات LOOP بایدکوتاه باشد. عملوندیک بایت کدمقصد تولیدمی‌کند: 01H تا 7FH محدوده ۱+ تا ۱۲۷+ و FFH تا 80H محدوده ۱- تا ۱۲۸- را دربر دارد. از آنجائیکه دستورات ماشین از ۱ تا ۴ بایت طول متفاوت دارند، محدوده مشخص نیست، اما دقت بر روی دو صفحه کامل کد منبع یک راهنمای واقعی است.
- زمان استفاده از LOOP و CX را با یک مقدار مثبت مقدار دهی کنید. زیرا LOOP و CX راکاهش می‌دهد و آن را برای صفر بودن بررسی می‌کند.
- وقتی یک‌دستور یک پرچم را تنظیم می‌کندپرچم به همان حالت تنظیم باقی می‌ماند تا دستور دیگری آن راتغییردهد.
- با توجه به اینکه کدام عملیات پردازش بر روی داده‌های علامت دار یا بدون علامت انجام شده است دستور پرش شرطی مناسب را انتخاب نمایید.
- از CALL برای دستیابی به یک روال استفاده کنید، و باید شامل RET/RETN درانتهای روال جهت بازگشت باشد. یک روال فراخوانده شده ممکن است روال دیگری را فراخوانی کند و اگر ترتیب مرسوم در آن رعایت شود، RET سبب خواهد شد تا آدرس صحیح از پشته برداشته شود.
- از شیفت به چپ برای دو برابر کردن مقدار و شیفت به راست برای تقسیم بر ۲ استفاده کنید. در انتخاب مناسب دستور شیفت برای داده‌های بدون علامت و علامتدار دقت کنید.

پرسش

۸-۱. این انواع آدرسدهی را شرح دهید: (الف) کوتاه، (ب) نزدیک، (ج) دور.

۸-۲. (الف) حداکثر تعداد بایت که یک JMP نزدیک، یک LOOP و یک دستور پرش شرطی می‌تواند پرش کند چقدر است؟ ، (ب) مشخصه عملوند کد ماشین که سبب این محدودیت می‌شود چیست؟

۸-۳. یک دستور JMP از موقعیت افست 05C8H آغاز می‌شود. آدرس افست انتقال را بسته به کد مقصدهای زیر برای عملوند JMP تعین کنید : (الف) 14H، (ب) 7DH، (ج) A3H.

۸-۴. یک برنام بنویسید که سری فیبرناچی را محاسبه کند: ۱۳، ۸، ۵، ۳، ۲، ۱ و ۱ (بجز دو عدد اول سری، هر عدد مجموع دو عدد ماقبل خود می‌باشد). از LOOP استفاده کنید و آن را طوری تنظیم نمایید که ۱۲ بار تکرار شود. آن را اسمبل نمایید، لینک نمایید و از DEBUG برای پیگیری روتین استفاده کنید.

۸-۵. فرض کنید، AX و BX حاوی داده‌های بدون علامت هستند و CX و DX حاوی داده‌های علامتدار هستند. دستورات پرش شرطی و CMP (در صورت لزوم) برای موارد زیر را تعیین کنید: (الف) آیا AX مساوی یا کوچکتر از BX است؟ (ب) آیا CX مساوی یا کوچکتر از DX است؟ (ج) آیا مقدار بیشتر از DX است؟ (د) آیا مقدار AX بیشتر از BX است؟ (ه) آیا DX حاوی صفر است؟ (و) آیا سرریز وجود دارد؟

۸-۶. در موارد زیر، چه پرچمهایی متاثر می‌شوند و محتوی آنها چه خواهد بود؟ (الف) پردازش در حالت تک مرحله‌ای است، (ب) انتقال رشته از راست به چپ انجام می‌گیرد، (ج) نتیجه منفی است، (د) نتیجه صفر است، (ه) یک سرریز اتفاق افتاد.

۸-۷. برنامه شکل ۳-۸ را ببینید و توضیح دهید که اگر روال B10 یک RET نداشته باشد بر اجرای برنامه چه تاثیری خواهد گذاشت.

۸-۸. تفاوت بین تعریف عملوند یک PROC با NEAR و یا FAR را شرح دهید.

۸-۹. تعیین کنید سه روشی را که یک برنامه می‌تواند در زمان اجرا وارد یک روال شود.

۸-۱۰. در یک برنامه EXE. F10 ، G10 را فراخوانی می‌کند، G10 ، H10 را فراخوانی می‌کند و H10 نیز J10 را فراخوانی می‌کند. در نتیجه این فراخوانی‌ها، پشته حاوی چه تعداد آدرس خواهد شد؟

۸-۱۱. فرض کنید که CH حاوی 01111001 و یک عنصر با نام TESTVAL حاوی 11100011 می‌باشد. تاثیر عملیات غیر مرتبط زیر بر CX را تعیین کنید. (الف) OR CH,TESTVAL، (ب) AND CH,TESTVAL، (ج) XOR CH,TESTVAL ، (د) AND CH,00000000B ، (ه) XOR CH,11111111B

۸-۱۲. برنامه شکل ۴-۸ را با موارد زیر بازنویسی کنید : محتویات CONAME را با حروف کوچک تعریف کنید و دستورا را بطوری کد نمایید که حروف کوچک را به حروف بزرگ تبدیل کند.

۸-۱۳. فرض‌کنید BX حاوی 1011100110111001 است و CL حاوی 03 می‌باشد.محتویات BX را درمبنای شانزده پس از اجرای دستورات نامرتبط زیر تعیین کنید: (الف) SHL BL,1، (ب) SHL BX,CL، (ج) SHR BX,CL (د) SHR BX,1، (ه) SAL BH,1، (و) ROR BX,CL، (ز) ROR DL,CL

۸-۱۴. از دستورات شیفت، انتقال و جمع برای مقداردهی CX با 40H و ضرب آن در 10 استفاده کنید.

۸-۱۵. روال انتهای بخش "چرخش بیت" محتویات DX:AX را در ۲ ضرب می‌کند. روتین را طوری اصلاح کنید که اعمال زیر را انجام دهد. (الف) محتویات DX:AX را در ۴ ضرب کند، (ب) محتویات DX:AX را بر ۴ تقسیم کند، (ج) ۴۸ بیت DX:AX:BX را در ۲ ضرب کند.

۸-۱۶. محتویات CX:BX را به EAX منتقل کنید و با دستور شیفت EAX را در ۴ ضرب کنید.

فصل ۹

مقدمه‌ای بر پردازش صفحه کلید و صفحه نمایش

هدف: مقدمه‌ای بر موارد ضروری نمایش اطلاعات بر روی صفحه نمایش و پذیرش ورودی از صفحه کلید.

مقدمه

تا این فصل، برنامه‌های ما عناصر داده را یا در ناحیه داده یا به شکل داده بلافصل در داخل عملوند دستور تعریف می‌کردند. اغلب برنامه‌ها نیاز به ورودی از صفحه کلید، دیسک، موس، یا مودم دارند و خروجی را در قالب مناسب بر روی صفحه نمایش، چاپگر، یا دیسک فراهم می‌سازند. این فصل موارد ضروری بنیادی جهت نمایش اطلاعات بر روی صفحه نمایش و پذیرش ورودی از صفحه کلید را در بردارد.

دستور (وقفه) INT، ورودی و خروجی را در اغلب موارد دستیابی می‌کند. دو وقفه‌ای که این فصل در بردارد توابع INT 10H برای دستیابی صفحه نمایش و توابع INT 21H برای نمایش بر روی صفحه و پذیرش ورودی صفحه کلید است. این ***توابع*** (یا سرویس‌ها) عمل ویژه‌ای را تقاضا می‌کنند. باید مقدار تابع را در ثبات AH، جهت تعیین نوع سرویسی که وقفه باید انجام دهد، قرار دهید.

عملیات BIOS سطح پایین مانند INT 10H کنترل را مستقیماً به BIOS می‌فرستد. اما برای سهولت برخی عملیات پیچیده‌تر، INT 21H سرویس وقفه‌ای دارد که ابتدا کنترل را به DOS منتقل می‌کند برای مثال: ورودی صفحه کلید ممکن است شامل تعدادی کاراکتر وارد شده و کنترل حداکثر تعداد باشد.

عملیات INT 21H بسیاری از این پردازش‌های سطح بالای اضافی را انجام داده و برای پردازش داده‌های سطح پایین کنترل را بطور اتوماتیک به BIOS منتقل می‌کند. طبق معمول، این کتاب به مقدار 0DH مانند کاراکتر Enter برای صفحه کلید و برای صفحه نمایش و چاپگر مانند Carriage Return اشاره دارد.

عملیاتی که در این فصل معرفی شده‌اند بشرح زیر می‌باشد :

توابع 21H	وقفه	توابع 10H	وقفه
نمایش کاراکتر بر روی صفحه	02H	تنظیم مکان نما	02H
نمایش رشته بر روی صفحه	09H	چرخش طوماری صفحه	06H
ورودی صفحه کلید	0AH		
ورودی صفحه کلید	3FH		
نمایش بر روی صفحه	40H		

فصل ۱۰ و ۱۱ جزئیات پیشرفته‌تری از دستیابی صفحه نمایش و صفحه کلید را در بر دارد.

صفحه نمایش

یک نمایشگر ۲۵ سطر (از ۰ تا ۲۴) و ۸۰ ستون (از ۰ تا ۷۹) دارد. سطرها و ستونها شبکه‌ای از موقعیت‌های قابل آدرس‌دهی فراهم می‌سازند که در هر کدام مکان‌نما می‌تواند تنظیم شود. در اینجا چند مثال از موقعیت‌های مکان‌نما ذکر شده است :

	قالب دهدهی		قابل‌شانزدهی	
موقعیت در صفحه‌نمایش	سطر	ستون	سطر	ستون
گوشه چپ بالا	00	00	00H	00H
گوشه راست بالا	00	79	00H	4FH
مرکز صفحه	12	39/40	0CH	27H/28H
گوشه چپ پایین	24	00	18H	00H
گوشه راست پایین	24	79	18H	4FH

بخشی از حافظه به نمایش محتویات صفحه نمایش اختصاص داده شده است. ناحیه نمایش تک رنگ در BIOS از موقعیت B000[0]H آغاز می‌شود و 4K بایت از حافظه را پشتیبانی می‌کند، که 2K آن برای کاراکترها و 2K دیگر برای صفت هر کاراکتر می‌باشد. از جمله صفات کاراکترها عبارتند از: رنگ متون عبارات، چشمک زن، دارای شدت نور زیاد و کلمات زیر خط دار. ناحیه نمایش رنگی 16K بایت را پشتیبانی می‌کند، که از موقعیت B800[0]H در BIOS آغاز می‌شود. برای نمایش کاراکترهای معمولی می‌توانید هم در ***حالت متنی*** یا ***حالت گرافیکی*** پردازش نمایید. برای حالت متنی، ناحیه نمایش چهار صفحه با شماره ۰ تا ۳ وجود دارد و هر صفحه دارای ۸۰ ستون است هر کاراکتر با دو بایت نشان داده می‌شود یک بایت برای کاراکتر و یک بایت برای صفت آن (مانند رنگ) فراهم شده است. صفحه‌ها و صفات مربوطه با جزئیات بیشتر در فصل ۱۰ بررسی می‌گردد، در این فصل فقط با صفحه 0 سر و کار خواهیم داشت.

وقفه‌هایی که نمایش صفحه را دستیابی می‌نمایند، بسته به نوع کارت گرافیکی نصب شده مانند VGA یا SVGA داده را مستقیماً به صفحه نمایش ارسال می‌دارند. بطور تکنیکی امکان ارسال داده مستقیماً به صفحه نمایش وجود دارد، اما تضمینی وجود ندارد که آدرسهای حافظه در همه مدلهای کامپیوتر یکسان باشند، بنابراین نوشتن مستقیم داده بر روی ناحیه نمایش ممکن است سریعتر باشد، اما مخاطره‌آمیز است. روش مطمئن‌تر استفاده از عملیات INT 10H و INT 21H است که موقعیت ناحیه نمایش ویدئویی را می‌داند.

تنظیم مکان‌نما

تنظیم مکان نما یک ضرورت عمومی برای متنی است، زیرا موقعیت آن مشخص کننده جایی است که کاراکتر بعدی باید نمایش داده شود. (حالت گرافیکی مکان‌نما را پشتیبانی نمی‌کند). INT 10H، عملیات BIOS برای دستیابی صفحه نمایش می‌باشد و تابع 02H در AH به عملیات برای تنظیم مکان‌نما پیغام می‌دهد. شماره صفحه لازم را (که معمولاً صفر است) در ثبات BH، سطر را در DH و ستون را در DL وارد نمایید. محتویات بقیه ثبات‌ها اهمیتی ندارد.

مثال زیر مکان نما را در سطر ۸ و ستون ۱۵ تنظیم می‌کند:

```
MOV AH,02H  ; درخواست تنظیم مکان نما
MOV BH,00   ; شماره صفحه صفر
MOV DH,08   ; ستون ۸
MOV DL,15   ; ستون ۱۵
INT 10H     ; فراخوانی سرویس وقفه
```

برای تنظیم سطر و ستون در DX، می‌توانید از یک دستور MOV با یک مقدار شانزدهی بلافصل استفاده کنید :

```
MOV DX,080F H ; سطر A و ستون ۱۵
```

پاک کردن صفحه نمایش

INT 10H تابع 06H از وقفه 10H عمل، پاک کردن صفحه نمایش حرکت طوماری را انجام می‌دهد. شما می‌توانید همه یا بخشی از صفحه نمایش را با شروع از هر موقعیت صفحه و خاتمه در هر موقعیت شماره‌گذاری شده بالاتر، پاک کنید. برای استفاده از این تابع ثبات‌های زیر را مقدار دهی کنید:

- AH = تابع 06H.
- AL = شماره خطهایی که حرکت طوماری دارند، با صفر برای همه صفحه.
- BH = مقدار صفت (رنگ، نمایش معکوس رنگ، چشمک زن).
- CX = ستون: سطر شروع.
- DX = ستون: سطر خاتمه.

CX و DX ناحیه نمایش (یا پنجره‌ای) که باید حرکت طوماری داشته باشد تعریف می‌کنند و AL تعداد خطوطی که باید حرکت طوماری به سمت بالا داشته باشد را مشخص می‌سازد. برای پاک کردن یک صفحه کامل ستون: سطر آغاز را در CX به شکل 00:00H و ستون سطر خاتمه را در Dx به شکل 18:4FH مشخص کنید. صفت 71H در مثال زیر یک صفحه کامل را با زمینه سفید (صفت 7) و پیش زمینه آبی (صفت ۱) تنظیم می‌کند.

برای مثال، به منظور حرکت طوماری پنجره‌ای از صفحه، از ستون ۰۵، سطر ۰۰، تا ستون ۱۲، سطر ۷۹ باید مقدار 0500H را در CX و 0C4FH را در DX قرار دهید.

```
MOV   AX,0600H   AH=06 ;(حرکت طوماری) AL=00 (صفحه کامل)
MOV   BH,71H     ; پس زمینه سفید (7)، پیش زمینه آبی (۱)
MOV   CX,0000H   ; ستون : سطر بالای چپ
MOV   DX,184FH   ; ستون : سطر پایین راست
INT   10H        ; فراخوانی سرویس وقفه
```

دقت کنید که اشتباهاً موقعیت راست پایین بیشتر از 184FH تنظیم نشود. فصل بعد حرکت طوماری را با جزئیات بیشتر توصیف می‌کند.

یک برنامه اغلب باید پیغامی نمایش بدهد تا از کاربر تقاضای ورود داده یا یک فعالیت را داشته باشد. ابتدا روش نسخه DOS اصلی را بررسی می‌کنیم که شامل دستیابی به فایل می‌باشد. عملیات اصلی تحت هر نسخه کار می‌کند و از بسیاری جنبه‌ها برای استفاده ساده‌تر و راحت‌تر می‌باشد، اگر چه استفاده از عملیات جدیدتر برای توسعه نرم‌افزاری، پیشنهاد می‌شود.

INT 21H تابع 09H برای نمایش صفحه

سادگی INT 21H تابع 09H اصلی برای نمایش، هنوز آن را در معرض استفاده عمومی نگه می‌دارد. بدین منظور لازم است تا رشته نمایش را در ناحیه داده تعریف کنیم و در انتها یک علامت محدود کننده دلار ($ یا 24H) قرار دهیم، که عملیات برای خاتمه نمایش از آن استفاده می‌کند. مثال بعد از این مطلب را مشخص می‌سازد:

```
CUSTMSG DB 'customer name ?','$' ; رشته نمایش
```

می‌توانید علامت دلار را بلافاصله پس از رشته نمایش به شکلی که نشان داده شده، داخل رشته به شکل 'Customer name ? $' یا خط بعد به شکل $ DB کد نویسی نمایید مشکل این روش این است که از این تابع برای نمایش کاراکتر $ نمی‌توانید استفاده کنید.

تابع 09H را در ثبات AH تنظیم کنید آدرس رشته نمایش را در DX بارگذاری کنید و دستور INT21H را قرار دهید.

```
MOV   AH,09H       ; درخواست نمایش
LEA   DX,CUSTMSG   ; بارگذاری آدرس اعلان
INT   21H          ; فراخوانی سرویس وقفه
```

عملیات INT کاراکتر را از چپ به راست نمایش می‌دهد و انتهای داده را وقتی به علامت دلار ($) می‌رسد تشخیص می‌دهد. عملیات محتویات ثبات‌ها را تغییر نمی‌دهد. یک رشته نمایش که از ستون سمت راست تجاوز می‌کند بطور اتوماتیک در سطر بعدی ادامه می‌یابد و در صورت لزوم حرکت طوماری خواهد داشت. اگر علامت دلار انتهای رشته را حذف کنید، عملیات نمایش کاراکترها را به ترتیب موقعیت حافظه‌ای ادامه می‌دهد تا به یک علامت دلار برسد اگر وجود داشته باشد.

استفاده از توابع 09H برای نمایش کاراکترهای ASCII

اغلب ۲۵۶ کاراکتر ASCII با سمبولی که قابل نمایش در صفحه نمایش ویدئویی است بیان می‌شوند. برخی مقادیر، مانند 00H و FFH سمبول قابل نمایشی ندارند و به شکل جای خالی ظاهر می‌شوند، اگر چه کاراکتر جای خالی ASCII ، 20H می‌باشد.

برنامه COM. در شکل ۱-۹ محدوده کامل کاراکترهای ASCII را نمایش می‌دهد. روال A10MAIN سه روال را فراخوانی می‌کند.

- B10SCRN از INT 10H تابع 06H برای پاک کردن صفحه نمایش استفاده می‌کند.
- C10CURS از INT 10H تابع 02H برای مقدار دهی مکان‌نما با 00:00H استفاده می‌کند.
- D10DISP از INT 21H تابع 09H برای نمایش محتویات ASCHAR استفاده می‌کند، که با 00H مقداردهی شده و مرتباً افزایش می‌یابد و هر کاراکتر نمایش داده می‌شود تابه FFH برسد. اولین خط نمایش داده شده با یک جای خالی (00H) آغاز می‌شود و سپس، دو صورت شاد، (02H,01H) و سپس یک دل (03H)، خشت (04H) و خاج (05H). کاراکتر 06H به شکل یک پیک ظاهر می‌شود، اما با کاراکتر کنترلی بعدی پاک می‌شود. کاراکتر 07H سبب ایجاد یک صدا از بلندگو خواهد شد، 08H سبب baskspaces خواهد شد. 09H یک tab ایجاد می‌کند، 0AH باعث قرار گرفتن مکان‌نما در خط بعد خواهد شد و 0DH (Enter) با شروع خط بعدی خواهد رفت و البته با تابع 09H سمبول دلار ،24H، نمایش داده نخواهد شد. (همانطور که خواهید دید می‌توانید از سرویس‌های BIOS برای نمایش سمبول صحیح این کاراکتر خاص استفاده کنید). نت موزیکال 0EH است و 7FH تا FFH کاراکترهای توسعه یافته ASCII می‌باشند. می‌توانید برنامه را مجدداً با چشم‌پوشی از نمایش کاراکترهای کنترلی بازنویسی کنید. دستورات زیر همه کاراکترهای بین 08H و 0DH را در نظر نمی‌گیرد. ممکن است خواسته باشید با چشم‌پوشی از 08H و 0DH این برنامه را تجربه کنید.

```
D20:
        CMP     ASCHAR,08H      ; کمتر از 08H؟
        JB      D30             ; بله، بپذیر
        CMP     ASCHAR,0DH      ; کمتر یا مساوی 0DH؟
        JBE     D40             ; بله، چشم پوشی کن؟
D30:
        MOV     AH,09H          ; نمایش <08H> و 0DH
        INT     21H
D40:                            ; فراخوانی سرویس وقفه
        INC     ASCHAR
        LOOP    D20
```

این تمرین از نمایش کلیه کاراکترهای کنترلی چشم‌پوشی می‌کند. توجه کنید که نمایش آنها روش معمول این عملیات است.

تکلیف: برنامهٔ قبل را مجدداً ایجاد نموده، اسمبل و لینک کرده و به قالب فایل COM. تبدیل و اجرا نمایید.

```
                page 60,132
TITLE           A09DISAS (COM)  Display ASCII character set
                .MODEL SMALL
                .CODE
                ORG     100H
BEGIN:          JMP     SHORT A10MAIN
ASCHAR          DB      00,'$'              ;Display character

;                       Main procedure:
;                       ---------------
A10MAIN         PROC    NEAR
                CALL    B10SCRN             ;Clear screen
                CALL    C10CURS             ;Set cursor
                CALL    D10DISP             ;Display characters
                MOV     AX,4C00H            ;End
                INT     21H                 ;  processing
A10MAIN         ENDP
;                       Clear screen:
;                       -------------
B10SCRN         PROC    NEAR
                MOV     AX,0600H            ;Scroll full screen
                MOV     BH,07               ;Attribute: white on black
                MOV     CX,0000             ;Upper left location
                MOV     DX,184FH            ;Lower right location
                INT     10H                 ;Call BIOS
                RET                         ;Return to caller
B10SCRN         ENDP
;                       Set cursor to 00,00:
;                       --------------------
C10CURS         PROC    NEAR
                MOV     AH,02H              ;Request set cursor
                MOV     BH,00               ;Page number 0
                MOV     DX,0000             ;Row 0, column 0
                INT     10H                 ;Call BIOS
                RET                         ;Return to caller
C10CURS         ENDP
;                       Display ASCII characters:
;                       -------------------------
D10DISP         PROC
                MOV     CX,256              ;Initialize 256 iterations
                LEA     DX,ASCHAR           ;Initialize address of ASCHAR
D20:
                MOV     AH,09H              ;Display ASCII ASCHAR
                INT     21H
                INC     ASCHAR              ;Increment for next character
                LOOP    D20                 ;Decrement CX, loop nonzero
                RET                         ;Return to caller
D10DISP         ENDP
                END     BEGIN
```

شکل ۱-۹ استفاده از وقفهٔ شمارهٔ *21H* برای نمایش مجموعه کاراکترهای *ASCII*

INT 21H تابع 0AH برای ورودی صفحه کلید

INT 21H تابع 0AH برای پذیرش داده‌ها از صفحه کلید بطور ویژه قدرتمند می‌باشد. ناحیه ورودی کاراکترهای وارد شده به یک ***لیست پارامتر*** حاوی فیلدهای ویژه که عملیات INT را بر روی آن پردازش انجام دهد، نیاز دارد (اگر با یک زبان سطح بالا کار کنید، ممکن است از عبارت ***رکورد*** یا ***ساختار*** استفاده نمایید). ابتدا عملیات باید از حداکثر طول داده ورودی مطلع باشد. هدف این است که کاربر از وارد کردن تعداد زیادی کاراکتر جلوگیری کند، عملیات بلندگو را به صدا در آورده و کاراکترهای اضافی دیگری نمی‌پذیرد. دوم، عملیات تعداد بایت‌هایی را که واقعاً وارد شده در لیست پارامتر وارد می‌نماید. لیست پارامتر شامل عناصر زیر است :

۱) اولین ورودی نام لیست پارامتر را به شکل LABEL BYTE فراهم می‌سازد. LABEL یک پیش پردازنده است با

صفت نوع BYTE، که سبب قرار گرفتن بر مرز یک بایت خواهد شد. از آنجائیکه این یک قرارگیری معمول می‌باشد، اسمبلر شمارشگر موقعیت را نمی‌افزاید. با استفاده از LABEL می‌توانید یک نام برای لیست پارامترها تعیین کنید.

۲) اولین بایت لیست پارامتر حاوی حداکثر تعداد کاراکتر ورودی است. حداقل مقدار این بایت ۰ است و از آنجا که یک فیلد ۱ بایتی است، حداکثر آن FFH یا ۲۵۵ می‌باشد. برای حداکثر، بر اساس نوع داده که از کاربر پذیرفته می‌شود تصمیم بگیرید.

۳) بایت دوم برای عملیات جهت ذخیره ساختن تعداد حقیقی کاراکترهای تایپ شده به شکل مقادیر دودویی است.

۴) بایت سوم فیلدی را آغاز می‌کند که حاوی کاراکترهای تایپ شده از چپ به راست باشد.

مثال بعد یک لیست پارامتر از یک ناحیه ورودی تعریف می‌کند :

```
PARALST  LABEL   BYTE          شروع لیست پارامتر ;
MAXLEN   DB      20            حداکثر تعداد کاراکتر ورودی ;
ACTLEN   DB      ?             تعداد واقعی کاراکتر ورودی ;
KBDATA   DB      20DUP(' ')    کاراکترهای وارد شده از صفحه کلید ;
```

در لیست پارامتر، پیش پردازنده LABEL به اسمبلر پیغام می‌دهد که در مرز بایت قرار گیرد و نام این موقعیت را PARALST بگذارد. از آنجائیکه LABEL هیچ فضایی اشغال نمی‌کند PARALST و MAXLEN به یک موقعیت حافظه اشاره می‌کند. MAXLEN حداکثر تعداد کاراکتر قابل پذیرش از صفحه کلید را تعریف می‌کند، ACTLEN فضایی برای عملیات فراهم می‌سازد تا تعداد واقعی کاراکتر وارد شده در آن جایگزین شود و KBDATA، ۲۰ فضا برای کاراکترها رزرو می‌کند. می‌توانید از هر نام معتبر برای این فیلد استفاده نمایید. برای درخواست ورودی، تابع 0AH را در AH تنظیم کنید، آدرس لیست پارامتر (در مثال PARALST) را در DX قرار دهید و INT 21H را فراخوانی کنید:

```
MOV   AH,0AH       درخواست ورودی صفحه کلید ;
LEA   DX,PARALST   بارگذاری آدرس لیست پارامتر ;
INT   21H          فراخوانی سرویس وقفه ;
```

عملیات INT منتظر می‌شود تا کاربر کاراکترها را تایپ کند و بررسی کند از حداکثر ۲۰ تجاوز ننماید. عملیات هر کاراکتر تایپ شده را بر روی صفحه منعکس می‌کند و مکان‌نما را پیش می‌برد. و کاربر با فشردن <Enter> پایان ورودی را علامت می‌دهد. عملیات کاراکتر <Enter> (0DH) را به فیلد ورودی KBDATA منتقل می‌کند، اما جزء طول واقعی ورودی شمرده نمی‌شود. اگر نامی مانند porter + <Enter> وارد کنید، لیست پارامتر چنین خواهد بود.

ASCII :	20	6	p	o	r	t	e	r	#				. . .
Hex :	14	06	50	6F	72	74	65	72	0D	20	20	20	. . .

عملیات طول نام ورودی، 06H، را در دومین بایت از لیست پارامتر، با نام ACTLEN در مثال ارائه می‌دهد. کاراکتر Enter (0DH) در KBDATA +6 خواهد بود. (سمبول # در اینجا بر این کاراکتر دلالت دارد، زیرا 0DH هیچ سمبول قابل چاپی ندارد). حداکثر طول با 0DH، ۲۰ داده شده، کاربر می‌تواند فقط ۱۹ کاراکتر تایپ کند.

این عملیات کاراکتر Bakespace را می‌پذیرد و آن را اعمال می‌کند اما در شمارش محسوب نمی‌شود. به استثنای کاراکتر bakespace کاراکترهای ورودی بیشتر از حداکثر تعداد کاراکترها نمی‌تواند باشد. در مثال قبلی، اگر یک کاربر ۲۰ کاراکتر وارد کند و <Enter> را نفشارد، بلندگو بوق می‌زند و در اینجا فقط کاراکتر Enter پذیرفته خواهد شد. عملیات، کلیدهای توسعه یافته تابعی مانند F1 ، Home ، Pgup و مکان نما را نمی‌پذیرد. اگر شما بر پذیرش هر یک در این کلیدها تاکید دارید، از INT 16H یا INT 21H تابع 01H استفاده کنید این دو وقفه در فصل ۱۱ توضیح داده خواهند شد.

برنامه: پذیرش و نمایش نام‌ها

برنامه شکل ۲-۹ از کاربر می‌خواهد تا یک نام را وارد کند و سپس نام را در مرکز صفحه نمایش خواهد داد و بلندگو به صدا در می‌آید. برای مثال، اگر کاربر نام Dana Porter را وارد کند، برنامه موارد زیر را انجام می‌دهد :

۱) تقسیم طول ۱۱ بر ۲: ۵=۲/۱۱، که از باقیمانده صرفنظر خواهد شد.

۲) این مقدار از ۴۰ کم می‌شود: ۳۵ = ۵ - ۴۰.

```
                page  60,132
TITLE     A09CTRNM (EXE)  Accept names, center on screen
;-----------------------------------------------------
          .MODEL SMALL
          .STACK 64
;-----------------------------------------------------
          .DATA
PARLIST   LABEL  BYTE             ;Name parameter list:
MAXNLEN   DB     20               ;  maximum length of name
ACTULEN   DB     ?                ;  no. of characters entered
KBNAME    DB     21 DUP(' ')      ;  entered name
PROMPT    DB     'Name? ', '$'
;-----------------------------------------------------
          .CODE
          .386
A10MAIN   PROC   FAR
          MOV    AX,@data         ;Initialize segment
          MOV    DS,AX            ;  registers
          MOV    ES,AX
          CALL   Q10CLR           ;Clear screen
A20LOOP:
          MOV    DX,0000          ;Set cursor to 00,00
          CALL   Q20CURS
          CALL   B10PRMPT         ;Display prompt
          CALL   C10INPT          ;Provide for input of name
          CALL   Q10CLR           ;Clear screen
          CMP    ACTULEN,00       ;Name entered?
          JE     A30              ;  no, exit
          CALL   D10CODE          ;Set bell and '$'
          CALL   E10CENT          ;Center, display name
          JMP    A20LOOP
A30:
          MOV    AX,4C00H         ;End processing
          INT    21H
A10MAIN   ENDP
;                 Display prompt:
;                 --------------
B10PRMPT  PROC   NEAR
          MOV    AH,09H           ;Request display
          LEA    DX,PROMPT
          INT    21H
          RET
B10PRMPT  ENDP
;                 Accept input of name:
;                 --------------------
C10INPT   PROC   NEAR
          MOV    AH,0AH           ;Request keyboard
          LEA    DX,PARLIST       ;  input
          INT    21H
          RET
C10INPT   ENDP
```

شکل ۲-۹ الف پذیرش و نمایش نام‌ها

در روال E10CENT، دستور SHR طول ۱۱ را یک بیت به راست شیفت می‌دهد، که تاثیر آن تقسیم طول بر ۲ است. بیت‌های 00001011 خواهد شد 00000101 یا همان ۵. دستور NEG علامت را معکوس می‌کند، یعنی S+ به S- تبدیل خواهد شد. ADD مقدار 40 را با S- جمع می‌کند، که موقعیت شروع ستون ۳۵ در ثبات DL داده خواهد شد. با تنظیم مکان‌نما در سطر ۱۲، ستون ۳۵ نام بر روی صفحه چنین نمایش داده خواهد شد :

```
                                سطر ۱۲ :  Dana Porter
                                           |    |
                                ستون :     35   40

;                       Set bell and '$' delimiter:
;                       ---------------------------
D10CODE     PROC    NEAR
            MOVZX   BX,ACTULEN          ;Replace 0DH with 07H
            MOV     KBNAME[BX],07
            MOV     KBNAME[BX+1],'$'    ;Set display delimiter
            RET
D10CODE     ENDP
;                       Center and display name:
;                       ------------------------
E10CENT     PROC    NEAR
            MOV     DL,ACTULEN          ;Locate center column:
            SHR     DL,1                ;  divide length by 2,
            NEG     DL                  ;  reverse sign,
            ADD     DL,40               ;  add 40
            MOV     DH,12               ;Center row
            CALL    Q20CURS             ;Set cursor
            MOV     AH,09H
            LEA     DX,KBNAME           ;Display name
            INT     21H
            RET
E10CENT     ENDP
;                       Clear screen:
;                       -------------
Q10CLR      PROC    NEAR
            MOV     AX,0600H            ;Request scroll screen
            MOV     BH,30               ;Color attribute
            MOV     CX,0000             ;From 00,00
            MOV     DX,184FH            ;To 24,79
            INT     10H
            RET
Q10CLR      ENDP
;                       Set cursor row/column:
;                       ----------------------
Q20CURS     PROC    NEAR                ;DX set on entry
            MOV     AH,02H              ;Request set cursor
            MOV     BH,00               ;Page #0
            INT     10H
            RET
Q20CURS     ENDP
            END     A10MAIN
```

شکل ۲-۹ ب پذیرش و نمایش نامها

توجه کنید که دستورات D10CODE کاراکتر (07H) را در ناحیه ورودی بلافاصله بعد از نام جایگزین می‌کنند :

```
MOVZX   BX,ACTULEN      ; BX قرار دادن 0DH با 07H در
MOV     KBNAME[BX],07H
```

MOVZX، BX را با تعداد کاراکتر تنظیم می‌کند. در دستور MOV، شاخص نشانگذاری در کروشه به معنی آن است که BX برای فعالیت مانند یک ثبات نشان گذاری خاص می‌باشد که برای سهولت آدرس دهی توسعه یافته است. MOV طول BX را با آدرس KBNAME ترکیب می‌کند و 07H را به آدرس محاسبه شده منتقل می‌کند. برای یک طول ۱۱، دستور 07H را در KBNAME+11 پس از نام جایگزین می‌کند (با جایگزینی کاراکتر Enter). دستور بعدی در D10CODE یک جدا کننده $ بعد از 07H جایگزین می‌کند مانند INT 21H تابع 09H می‌تواند نام را نمایش دهد و بلندگو را به صدا در آورد.

پاسخ دادن فقط با کلید Enter

تا وقتیکه کاربر فقط کلید <Enter> را برای جواب به اعلان بفشارد برنامه پذیرش و نمایش نامها را ادامه می‌دهد

INT 21H تابع 09H آن را می‌پذیرد و مانند زیر یک طول 00H را در لیست پارامتر قرار می‌دهد :

14	00	0D	. . .

لیست پارامتر (hex) :

اگر طول صفر باشد، برنامه ورودی را خاتمه یافته معین می‌کند، همانطور که در دستور CMP ACTLEN,00 در A20LOOP نشان داده شد.

پاک کردن کاراکتر Enter

می‌توانید از کاراکترهای ورودی برای اهداف مختلف مانند چاپ کردن یک گزارش، ذخیره کد یک جدول یا نوشتن روی دیسک استفاده کنید. برای این منظور باید کاراکتر Enter (0DH) را در هر جای KBNAME با یک فاصله (20H) جایگزین کنید. فیلدی که حاوی طول داده ورودی است، ACTLEN موقعیت رابطه‌ای کاراکتر Enter را فراهم می‌سازد. برای مثال، اگر ACTLEN حاوی ۱۱ باشد، پس به کاراکتر Enter در موقعیت KBNAME+11 قرار دارد. شما می‌توانید این طول را به ثبات BX برای نشانگذاری آدرس KBNAME بشکل زیر منتقل نمایید :

```
MOVZX   BX,ACTULEN      ; BX را با مقدار 0B (۱۱) تنظیم می‌کند
MOV     KBNANE[BX],20H  ; پاک کردن کاراکتر Enter
```

دستور MOVZX مقدار طول ۱۱ را در BX قرار می‌دهد. MOV یک جای خالی (20H) را به آدرس مشخص شده یا اولین عملوند منتقل می‌کند. آدرس KBNAME بعلاوه محتویات BX در حقیقت همان KBNAME+11 می‌باشد.

پاک کردن ناحیه ورودی

هر کاراکتر که وارد می‌شود در ناحیه ورودی جایگزین مقادیر قبلی می‌شود و تا هنگامی که کاراکترهای دیگری جایگزین نشوند باقی می‌ماند. ورودی‌های متوالی زیر را در نظر بگیرید :

ورودی	PARLIST (hex)											
۱. Paine	14	05	50	61	69	6E	65	0D	20	20	...	20
۲. Franklin	14	08	46	72	61	6E	6B	6C	69	6E	0D	...	20
۳. Adams	14	05	41	64	61	6D	73	0D	69	6e	0D	...	20

اولین نام، Paine، فقط ۵ بایت نیاز دارد. نام دوم، Franklin کاملاً جایگزین نام کوتاه‌تر Paine خواهد شد. اما چون نام‌سوم، Adurns، کوتاه‌تراز Franklin، می‌باشد،فقط جایگزین Frank خواهدشد و کاراکتر Enter جایگزین L می‌شود. دو حرف باقیمانده (in) هنوز پس از Adams قرار دارد. می‌توانید KBNAME قبلی را برای اعلان یک نام پاک کنید:

```
        MOV   CX,20             ; مقداردهی برای ۲۰ بار حلقه
        MOV   SI,0000           ; نقطه شروع نام
B30:    MOV   KBNAME[SI],20H    ; انتقال یک فاصله با نام
        INC   SI                ; افزایش برای کاراکتر بعدی
        LOOP  B30               ; تکرار ۲۰ مرتبه
```

به جای ثبات SI، می‌توانید از DI یا BX استفاده کنید. همچنین اگر روتین یک کلمه با ۲ فاصله را انتقال داد، فقط ده بار تکرار حلقه لازم است. اما، از آنجائیکه KBNAME به شکل DB (بایت) تعریف شده، شما باید طول آن را با یک عملوند WORD و PTR (اشاره‌گر) همان طور که در زیر می‌بینید بازنویسی کنید.

```
        MOV   CX,10             ; مقدار دهی برای ۱۰ حلقه
        LEA   SI,KBNAME         ; مقدار دهی شروع نام
B30:
        MOV   WORD PTR[SI],2020H ; انتقال دو فاصله به نام
        INC   SI                ; افزایش دو موقعیت در نام
        INC   SI                ;
        LOOP  B30               ; تکرار ده مرتبه
```

تفسیر MOV در B30، انتقال یک کلمه فاصله، به موقعیت حافظه‌ای که ثبات SI به آدرس آن اشاره می‌کند، این مثال از LEA برای مقدار دهی پاک کردن KBNAME استفاده می‌کند و از یک متد متفاوت واضح جهت MOV در B30 استفاده می‌کند زیرا نمی‌توانید چنین دستوری بنویسید.

```
MOV  WORD PTR[RBNAME],2020H ; اولین عملوند نامعتبر است
```

پاک کردن ناحیه ورودی مشکل نام‌های کوتاه را که در انتهای آنها داده‌های قبلی وجود دارد، حل می‌کنند. جهت پردازش سریعتر می‌توانید فقط موقعیت سمت راست آخرین نام وارد شده را پاک کنید.

استفاده از کاراکترهای کنترلی در نمایش صفحه

یک روش برای تاثیر بیشتر نمایش استفاده از کاراکترهای کنترلی Carriage Return ، Line Feed و Tab می‌باشد. می‌توانید آنها را به شکل ASCII یا مقادیر شانزدهی مانند این کدنویسی نمایید.

کاراکتر کنترلی	ASCII	شانزدهی	تاثیر بر مکان نما
Carraige return	13	0DH	تنظیم موقعیت چپ صفحه
Line Feed	10	0AH	رفتن به خط بعدی
Tab	09	09H	رفتن به ایست tab بعدی

از این کاراکترهای کنترلی برای دستکاری مکان وقتی می‌خواهید خروجی را نمایش دهید یا ورودی را بپذیرید، می‌توانید استفاده کنید. در اینجا چند مثال ذکر شده که محتویات رشته کاراکتری با نام REPTITL نشان می‌دهد و سپس Carriage Return (13) و Line Feed (10) کاراکتر را در خط بعدی تنظیم می‌کند.

```
REPTITL   DB 09          ; 'Intertech Crop Annual Report', 13, 10,'$'
          . . .
MOV       AH,09H                    درخواست نمایش ;
LEA       DX,REPTITL                قرار دادن آدرس تیتر ;
INT       21H                       فراخوانی روتین وقفه ;
```

استفاده از EQU برای تعریف مجدد کاراکترهای کنترلی برنامه را خواناتر می‌سازد :

```
CR        EQU   13     (0DHیا)
LF        EQU   10     (0AHیا)
TAB       EQU   09     (09Hیا)
REPTITL   DB    TAB,   'Intertech Crop Annual Report',CR,LF,'$'
```

استفاده از تابع 02H از وقفه 21H برای نمایش کاراکترها

ممکن است تابع 02H را برای نمایش یک کاراکتر مفید احساس کنید. کاراکتر مورد نظر را در DL قرار دهید تا در موقعیت جاری مکان نما توسط تابع نمایش داده شود و سپس INT 21H را فراخوانی نمایید. کاراکترهای Tab ، Carriage Return و Line Feed عملکردهای خود را دارند. دستورات چنین هستند :

```
MOV   AH,02H    ; درخواست نمایش کاراکتر
MOV   DL,Char   ; کاراکتر جهت نمایش
INT   21H       ; فراخوانی سرویس وقفه
```

مثال زیر نشان می‌دهد که چگونه می‌توان از این سرویس برای نمایش یک رشته کاراکتری استفاده کرد. رشته‌ای که باید نمایش داده شود در COTITLE قرار دارد. برنامه آدرس COTITLE را در ثبات DI و طول آن را در ثبات CX قرار می‌دهد. حلقه شامل افزایش DI (با دستور INC) برای هر کاراکتر متوالی و کاهش CX (با دستور LOOP) برای تعداد کاراکترهای نمایش داده شده است. در اینجا دستورات آورده شده است.

```
COTITLE  DB    'Intertech Crop' ,13 ,10
         MOV   AH,02H          ; درخواست نمایش کاراکتر
         MOV   CX,17           ; طول رشته کاراکتری
         LEA   DI,COTITLE      ; آدرس رشته کاراکتری
C50:     MOV   DL,[DI]         ; کاراکتر برای نمایش
         INT   21H             ; فراخوانی سرویس وقفه
         INC   DI              ; افزایش برای کاراکتر بعدی
         LOOP  C50             ; تکرار ۱۷ مرتبه
         ...                   ; پایان
```

دستگیره فایل

در این مبحث نحوه استفاده از دستگیره فایل را برای عملیات صفحه نمایش و صفحه کلید مشاهده خواهید کرد این نوع عملیات بیشتر در سیستم‌های UNIX و OS/2 متداول است. یک ***دستگیره فایل*** عددی است که به ابزار خاصی اشاره دارد. از آنجائیکه دستگیره فایل استاندارد قبلاً تنظیم می‌شود، نباید آن را تعریف کنید:

دستگیره	ابزار
۰۰	ورودی، معمولاً صفحه کلید (کنسول)، اما ممکن است نشان گذاری شده باشد.
۰۱	خروجی، معمولاً صفحه نمایش (کنسول)،اما ممکن است نشان گذاری شده باشد.
۰۲	خروجی خطا، صفحه نمایش (کنسول)، اما ممکن است نشان گذاری نشده باشد.
۰۳	Auxiliary device (AUX)
۰۴	چاپگر (LPTI یا PRN)

همانطور که نشان داده شده، دستگیره فایل معمول برای ورودی صفحه کلید ۰۰ و ۰۱ برای صفحه نمایش است.

دستگیره‌های فایل برای ابزارهای دیسکی (فصل ۱۷) باید در برنامه شما تنظیم شوند. می‌توانید از این سرویس‌ها برای نشانگذاری ابزارهای دیگر ورودی و خروجی استفاده نمایید، اگر چه این مطلب اینجا بررسی نمی‌شود.

استفاده از تابع 40H از وقفه 21H برای نمایش صفحه

این تابع برای انجام عملیات نمایش از دستگیره وقفه استفاده می‌کند. برای درخواست این سرویس، ثبات‌های زیر را بارگذاری کنید:

تعداد کاراکترهای نمایش = CX	تابع AH = 40H
آدرس ناحیه نمایش = DX	دستگیره فایل BX = 01

در یک عملیات INT موفق تعداد بایت‌های نوشته شده در AX قرار می‌گیرد و پرچم نقلی پاک می‌شود (می‌توانید این موضوع را بررسی کنید.) یک عملیات INT ناموفق پرچم نقلی را یک می‌کند و یک کد خطا در AX قرار می‌دهد. 05H یعنی دستیابی رد شد (برای یک ابزار نامعتبر یا متصل نشده) یا 06H یعنی دستیابی نامعتبر است. از آنجائیکه BX می‌تواند هم حاوی طول یا کد خطا باشد، تنها راه تشخیص یک کد خطا بررسی پرچم نقلی است ، اگرچه نمایش خطا بندرت اتفاق می‌افتد.

```
JC  error-routine ; بررسی بروز خطا
```

این عملیات بر روی کاراکترهای کنترلی 07H ، 08H ، 0AH و 0DH مانند INT 21H تابع 09H عمل می‌کند. دستورات زیر تابع 40H را مشخص می‌سازند :

```
COTITLE  DB    Intertech Crop',0DH,0AH'  ; ناحیه‌نمایش
         MOV   AH,40H                    ; درخواست نمایش
         MOV   BX,01                     ; دستگیره فایل برای صفحه
         MOV   CX,17                     ; نمایش ۱۷ کاراکتر
         LEA   DX,COTITLE                ; ناحیه نمایش
         INT   21H                       ; فراخوانی سرویس وقفه
```

توجه کنید که طول COTITLE شامل 0DE و 0AH نیز می‌باشد.

تمرین: نمایش بر روی صفحه

بیایید از DEBUG برای بررسی تاثیر داخلی استفاده از دستگیره فایل برای نمایش نام شما، استفاده کنیم. DEBUG را اجرا کنید و وقتی اعلان آن ظاهر شد، تایپ کنید A 100، جملات برنامه اسمبلی زیر را (نه اعداد سمت چپ) از موقعیت افست 100H وارد کنید (بخاطر بسپارید DEBUG اعداد وارد شده، در مبنای شانزده فرض می‌کند).

دستورات، AH را برای درخواست نمایش و افست 10EH در DX را - موقعیت حاوی نام شما- تنظیم می‌کند.

وقتی دستورات را وارد کردید، <Enter> را مجدداً بفشارید. برای unassemble کردن برنامه فرمان U (U 100,100) را وارد کنید و با فرامین R و T اجرا را پیگیری کنید. وقتی به (INT21) رسیدید فرمان P را وارد کنید وقفه تا دستور NOP اجرا می‌شود و نام را بر روی صفحه نشان خواهد داد. از فرمان Q برای خروج از DEBUG استفاده کنید.

```
100     MOV   AH,90
102     MOV   BX,01
105     MOV   CX,XX
108     MOV   DX,10E     ;  (طول نام را در اینجا قرار دهید)
10B     INT   21
10D     NOP
10E     DB    'x . . . x'  ;  (نام را در اینجا قرار دهید)
```

INT 21H تابع 3FH برای ورودی صفحه کلید

INT 21H تابع 3FH از دستیابی فایل برای درخواست ورودی صفحه کلید استفاده می‌کند، اگر چه تا اندازه‌ای عملیات کم مهارتی است، ثبات‌های زیر را بارگذاری کنید:

CX = حداکثر تعداد کاراکترهای پذیرفته شده AH = 3FH تابع

DX = آدرس ناحیه‌ای برای کاراکترهای وارد شده BX = 00 دستگیرهٔ فایل

یک عملیات INT موفق پرچم نقلی را صفر می‌کند (که ممکن است آن را تست کنید) و AX را با تعداد کاراکترهای وارد شده تنظیم می‌کند.

یک عملیات INT ناموفق که بدلیل یک دستیابی نامعتبر رخ می‌دهد، پرچم نقلی را یک می‌کند و یک کد خطا در AX می‌گذارد : 05H یعنی دستیابی رد شد (برای یک ابزار نامعتبر یا متصل نشده) یا 06H یعنی دستیابی نامعتبر. از آنجائیکه AX هم حاوی طول و هم حاوی یک کد خطا می‌تواند باشد تنها روش تعیین خطا بررسی پرچم نقلی است، اگر چه خطای صفحه کلید بندرت رخ می‌دهد.

مشابه INT 21H تابع 0AH تابع 3FH نیز بر روی کلید Backspce فعال است، اما کلیدهای تابعی توسعه یافته مانند F1 ، Home و PageUp را در نظر نمی‌گیرد.

دستورات زیر نحوه استفاده از تابع 3FH را مشخص می‌سازد:

```
KBINPUT  DB     20DUP (' ')     ;  ناحیه ورود
         . . .
         MOV    AH,3FH          ;  درخواست ورودی صفحه کلید
         MOV    BX,00           ;  دستیابی فایل برای صفحه کلید
         MOV    CX,20           ;  حداکثر ۲۰ کاراکتر
         LEA    DX,KBINPUT      ;  ناحیه ورود
         INT    21H             ;  فراخوانی سرویس وقفه
```

عملیات INT منتظر می‌ماند تا شما یک کاراکتر را وارد کنید، اما متاسفانه تعداد کاراکترها را بررسی نمی‌کند که آیا از حداکثر تعداد در ثبات CX تجاوز می‌کند (در مثال ۲۰ می‌باشد) فشردن <Enter>(0DH) انتهای ورودی را علامت می‌دهد. برای مثال، تایپ کردن

کاراکترهای 'Intertech Crop' این رشته را به KBINPUT به شکل زیر ارائه می‌دهد.

| Intertech Crop | 0DH | 0AH |

بلافاصله پس از کاراکترهای تایپ شده Enter (0DH) است که تایپ کردید و 0AH که تایپ نکردید. بهمین دلیل، حداکثر تعداد و طول ناحیه ورودی که تعریف می‌کنید باید برای ۲ کاراکتر اضافی نیز در نظر بگیرید. اگر شما کاراکترهای کمتری از حداکثر وارد کنید موقعیت‌های حافظه پس از کاراکترهای تایپ شده هنوز حاوی محتویات قبلی است.

یک عملیات INT موفق پرچم نقلی را پاک می‌کند و AX را با تعداد کاراکترهای تحویل داده شده تنظیم می‌کند در مثال قبل، این عدد ۱۴ بعلاوه ۲ برای کاراکترهای 0DH و 0AH است، یا ۱۶. بر طبق این، یک برنامه از مقدار AX برای تعیین تعداد واقعی کاراکترهای تایپ شده می‌تواند استفاده کند. اگر چه این مطالب برای انواع پاسخ Yes و NO می‌باشد، برای پاسخ‌هایی با طول متغییر مانند نام‌ها مفید می‌باشد.

اگر تعداد کاراکترها از حداکثر ثبات CX تجاوز نماید، عملیات در واقع هم کاراکترها را می‌پذیرد. فرض کنید در یک حالت حداکثر 08,CX است و کاربر کاراکترهای 'PC Exchange' را تایپ نموده است. عملیات اولین ۸ کاراکتر را در ناحیه ورودی تنظیم می‌کند. 'PC Excha' که هیچ کاراکتر 0DH یا 0AH پس از آن نیست و AX را با طول 08 تنظیم می‌کند. حال می‌بینید، عملیات INTبعدی که اجرا شود مستقیماً نامی را از صفحه کلید نمی‌پذیرد، زیرا هنوز آخر رشته قبلی را در بافر خودش دارد. پس 'nge' و پس آن 0DH و 0AH را به ناحیه ورودی می‌دهد و AX را با 05 مقدار می‌دهد هر دو عملیات معمولی هستند و پرچم نقلی را پاک می‌کنند :

```
PC Excha        AX = 08   : اولین INT
nge, 0DH, 0AH   AX = 05   : دومین INT
```

یک برنامه می‌تواند بگوید که کاربر یک تعداد معتبر کاراکتر را وارد کرده اگر (الف) تعداد بازگشت شده در AX کمتر از تعداد CX باشد یا (ب) تعداد بازگشت شده در AX مساوی CX باشد و آخرین ۲ کاراکتر ناحیه ورودی 0DH و 0AH باشد. اگر هیچکدام از شرایط فوق صحیح نباشد، باید چندین INT اضافی برای پذیرش کاراکترهای باقیمانده فراخوانی کنید. بعد از این همه، ممکن است هنوز نگران این باشید که نقطه مشخص کردن حداکثر تعداد در CX چیست.

تمرین: وارد کردن داده

در اینجا یک تمرین DEBUG ذکر شده که با آن می‌توانید تاثیر استفاده از INT 21H تابع 3FH را برای وارد کردن داده ببینید. برنامه به شما اجازه می‌دهد تا ۱۲ کاراکتر را وارد کنید البته با کاراکترهای Enter و Linefeed اجرا کنید و وقتی اعلان ظاهر شد، تایپ کنید 100 A تا دستورات زیر از موقعیت 100H وارد کنید:

```
100   MOV   AH,3F
102   MOV   BX,00
105   MOV   CX,0C
108   MOV   DX,10F
10B   INT   21
10D   JMP   100
      10F   DB   20 20 20 20 20 20 20 20 20 20 20 20
```

وقتی دستورات را وارد کردید مجدداً <Enter> را بفشارید. برنامه AH و BX را برای درخواست ورودی از صفحه کلید و حداکثر طول را در CX تنظیم می‌کند. همچنین افست 10FH را در DX موقعیت DB جایی که کاراکترهای وارد شده آغاز می‌شوند تنظیم می‌کند.

از فرمان U (U 100,10E) برای Unassembel کردن برنامه استفاده کنید. از فرامین R و T برای پیگیری اجرای

چهار دستور MOV استفاده کنید و در موقعیت 10BH فرمان P[1] را جهت اجرای وقفه وارد کنید، عملیات منتظر می‌شود تا کاراکترها و سپس کلید <Enter> را بفشارید. محتویات ثبات AX و پرچم نقلی را بررسی کنید، فرمان D DS:10F را جهت نمایش کاراکترهای داده شده در حافظه استفاده کنید. این حلقه را بطور نامحدود می‌توانید تکرار کنید و Q را برای خروج بفشارید.

نکات کلیدی

- نمایش رنگی پایه 16K بایت را پشتیبانی می‌کند و می‌تواند در حالت رنگی یا تک رنگ عمل نماید. می‌توانید برای کاراکترهای معمولی هم در حالت متنی و یا در حالت گرافیکی پردازش نمایید.
- نمایش تک رنگ 4K بایت حافظه را پشتیبانی می‌کند، 2K برای کاراکترها و 2K برای صفت هر کاراکتر.
- دستورات INT 10H کنترل را به BIOS، برای عملیات نمایش منتقل می‌کنند. دو عملیات معمول تابع 02H (تنظیم مکان‌نما) و 06H (چرخش طوماری صفحه) می‌باشد.
- INT 21H توابع خاصی برای دستیابی به پیچیدگی‌های ورودی خروجی فراهم می‌سازد.
- وقتی از INT 21H تابع 09H برای نمایش استفاده می‌کنید، یک جداکنندهٔ $ بلافاصله بعد از ناحیه نمایش تعریف کنید. یک جداکنندهٔ اشتباه، سبب تاثیرات نامشخصی بر روی صفحه خواهد شد.
- INT 21H تابع 0AH برای ورودی صفحه کلید اولین بایت حاوی حداکثر مقدار خواهد بود و بطور اتوماتیک مقدار واقعی را در دومین، بایت می‌گذارد.
- یک دستگیره فایل عددی است که به ابزار خاص اشاره دارد. اعدادی برای دستگیره فایل از 00 تا 04 قبلاً تنظیم شده‌اند، در حالیکه بقیه می‌توانند با برنامه شما تنظیم شوند.
- برای INT 21H تابع 40H جهت نمایش، از 01 در BX استفاده کنید.
- برای INT 21H تابع 3FH جهت ورودی صفحه کلید از 00 در BX استفاده کنید. عملیات Enter و LineFeed را پس از کاراکترهای تایپ شده در ناحیه ورودی قرار می‌دهد، اما کاراکترها را بررسی نمی‌کند که آیا از حداکثر مشخص شده تجاوز می‌کند.

پرسش‌ها

۹-۱. در یک صفحه ۸۰ ستونی، مقدار شانزدهی برای (الف) موقعیت سمت راست بالا و (ب) موقعیت سمت چپ پایین چیست ؟

۹-۲. دستوراتی را بنویسید که مکان نما را در سطر ۱۶ و ستون ۲۰ تنظیم نماید.

۹-۳. دستوراتی را بنویسید که صفحه را پاک کند، در سطر ۶، ستون 0 با هر صفت رنگ آغاز شود.

۹-۴. عناصر داده‌ای را تعریف کنید و با استفاده از INT 21H تابع 09H پیغام 'what is the date (mm/dd/yy) ?' را نشان دهد و پس از آن یک صدای زنگ شنیده شود.

۹-۵. عناصر داده‌ای را تعریف کنید و با استفاده از INT 21H تابع 0AH ورودی بر طبق قالب پرسش ۹-۴ از صفحه کلید بپذیرد.

۹-۶. در بخشی با عنوان پاک کردن ناحیه ورودی نشان دادیم که چگونه می‌توان ناحیه ورودی صفحه کلید با نام KBNAME را پاک کرد. مثال را بطریقی تغییر دهید که فقط کاراکترهای بلافاصله در سمت راست آخرین نام وارد شده را پاک کند.

1- proceed

۹-۷. برنامه ۲-۹ را با تغییرات زیر وارد کنید:(الف) بجای سطر ۱۲ در مرکز سطر ۰۹ تنظیم نمایید، (ب) بجای پاک کردن کامل صفحه، فقط سطر ۰ تا ۸ را پاک کنید، از صفت 17H در عملیات استفاده کنید. برنامه را اسمبل، لینک و تست نمایید.

۹-۸. دستگیرهٔ فایل استاندارد برای موارد زیر را معین کنید: (الف) چاپگر، (ب) ورودی صفحه کلید، (ج) نمایش صفحه معمولی.

۹-۹. عناصر داده‌ای را تعریف کنید و با استفاده از INT 21H تابع 40H پیغام 'What is the date (mm/dd/yy)?' را نشان دهد و پس از آن یک صدای زنگ شنیده شود.

۹-۱۰. عناصر داده‌ای را تعریف کنید و با استفاده از INT 21H تابع 3FH ورودی بر طبق قالب پرسش ۹-۹ از صفحه کلید بپذیرد.

۹-۱۱. پرش ۹-۷ را با استفاده از INT 21H تابع 3FH و 40H برای ورودی و نمایش بازنویسی کنید. این برنامه را اسمبل، لینک و تست نمایید.

فصل ۱۰

موارد پیشرفتهٔ پردازش صفحه نمایش

هدف: موارد پیشرفتهٔ دستیابی صفحه نمایش شامل حرکت طوماری، معکوس کردن رنگ ویدئو، چشمک‌زن و استفاده از گرافیک رنگی.

مقدمه

فصل ۹ مقدمه‌ای بر موارد پایه‌ای بود که بر دستکاری صفحه و ورودی صفحه کلید متمرکز بود. این فصل موارد پیشرفته‌ای مرتبط با تطبیق دهنده‌های ویدئوئی، تنظیم حالت (متنی یا گرافیک) و دیگر موارد دستیابی صفحه را فراهم می‌سازد.

اولین بخش تطبیق دهنده‌های ویدئویی عمومی را شرح می‌دهد و ارتباط آن‌ها با ناحیه نمایش ویدئوئی را بیان می‌دارد. این بخش در حالت متنی نحوه‌استفاده از بایت صفت برای رنگ، چشمک‌زن و شدت نوربالا، همچنین دستوراتی که اندازه و موقعیت مکان‌نما، برای حرکت طوماری به‌بالا یا پایین صفحه و نمایش‌کاراکترها راتوضیح می‌دهد. بخش‌های بعدی نحوه استفاده از حالت گرافیکی، با توابع مختلف مورد استفاده برای نمایش گرافیکی را توضیح می‌دهد. این فصل سرویس‌های توصیه شده زیر را توسط BIOS INT 10H در بر دارد.

00H تنظیم حالت ویدئو
01H تنظیم اندازه مکان‌نما
02H تنظیم موقعیت مکان‌نما
03H خواندن موقعیت مکان‌نما
04H خواندن قلم نوری
05H انتخاب صفحه فعال
06H حرکت طوماری صفحه به بالا
07H حرکت طوماری به پایین
08H خواندن صفت / کاراکتر
09H نمایش صفت / کاراکتر
0AH نمایش کاراکتر
0BH تنظیم خصوصیات رنگ
0CH نوشتن نقطه پیکسل
0DH خواندن نقطه پیکسل
0EH نوشتن تله تایپ
0FH گرفتن حالت ویدئویی جاری
11H تولید کننده کاراکتر
12H انتخاب روتین
13H نمایش رشته کاراکتری
1BH بازگشت اطلاعات وضعیت
1CH ذخیره/بازیابی‌وضعیت‌ویدئو

تطبیق دهنده ویدئو

عمومی‌ترین تطبیق دهنده‌های ویدئو شامل موارد زیر می‌باشد :

MDA تطبیق دهنده نمایش تک رنگ
CGA تطبیق دهنده گرافیک رنگی
EGA تطبیق دهنده گرافیک گرانتر
MCGA آرایه گرافیک چند رنگ (PS/2 مدل‌های 25 و 30)
VGA آرایه گرافیک ویدئو

VGA و SVGA بعد از آن جایگزین تطبیق دهنده ویدئویی CGA و EGA شدند. نرم افزارهایی که برای CGA و EGA نوشته شده‌اند با یک سیستم VGA نیز اجرا می‌شوند، اگر چه نرم افزارهایی که بطور خاص برای VGA نوشته شده‌اند با یک VGA یا یک EGA اجرا می‌شود.

یک نمایش ویدئویی شامل سه جزء اصلی زیر می‌باشد. ویدئو کنترلر، ویدئو BIOS، ناحیه نمایش ویدئو.

۱) **ویدئو کنترلر**، واحد کاری، سیگنال‌های پیمایش مونیتور را برای متن منتخب یا حالت گرافیکی تولید می‌کند. پردازشگر کامپیوتر دستورات را به ثبات‌های کنترلر می‌فرستد و وضعیت را از آنها می‌خواند.

۲) **ویدئو BIOS**، که مانند یک رابط با تطبیق دهنده، ویدئو عمل می‌کند، حاوی روتین‌هایی برای تنظیم مکان‌نما و نمایش کاراکترهاست.

۳) ناحیه نمایش ویدئو در حافظه حاوی اطلاعاتی است که مونیتور باید نمایش دهد. وقفه‌هایی که نمایش‌های صفحه را دستیابی می‌کنند، داده‌ها را مستقیماً به این ناحیه می‌فرستند. حالت‌های ویدئویی مختلف در نواحی مختلف از ناحیه نمایش ویدئو قرار می‌گیرند. در زیر آدرس‌های سگمنت شروع برای تطبیق دهنده‌های ویدئوئی اصلی ذکر شده است :

- A000:[0] برای توصیف فونت وقتی درحالت متنی است و برای درجه بالای گرافیکی VGA، EGA و MCGA.
- B000:[0] حالت متنی تک رنگ برای VGA ، EGA و MDA .
- B800:[0] حالت‌های متنی و گرافیکی برای VGA ، EGA ، MCGA و CGA .

تنظیم حالت ویدئو

حالت ویدئو با چندین عامل تعیین می‌گردد مانند متن یا گرافیک، رنگی یا تک رنگ، درجه تنظیم صفحه و تعداد رنگ‌ها، می‌توانید از BIOS INT 10H تابع 00H برای مقدار دهی حالت برنامه اجرایی جاری یا تغییر وضعیت بین حالت‌های متن و گرافیک استفاده کنید. تنظیم حالت صفحه را نیز پاک می‌کند.

بعنوان مثال، حالت 03 حالت متنی ۲۵ سطر ×۸۰ ستون، رنگی و درجه تنظیم ۴۰۰×۷۲۰ برای یک مونیتور VGA را فراهم می‌سازد.

برای تنظیم حالت INT 10H با تابع 00H در ثبات AH و حالت مورد نظر در AL را فراخوانی کنید. مثال زیر حالت ویدئو را برای متن رنگی استاندارد هر نوع از مونیتور رنگی تنظیم می‌کند (این یک روش سریع برای پاک کردن صفحه است).

```
                درخواست تنظیم حالت ;
MOV   AH,00H
                متن رنگی استاندارد ۲۵×۸۰ ;
MOV   AL,03H
                فراخوانی سرویس وقفه ;
INT   10H
```

اگر نرم‌افزاری برای یک مونیتور ویدئویی ناشناخته می‌نویسید، می‌توانید از INT 10H تابع 0FH (بعداً گفته خواهد شد) که حالت ویدئو جاری را در AL برمی گرداند، استفاده کنید. راه‌حل دیگر استفاده از BIOS INT 11H برای تعیین ابزارهای متصل به سیستم می‌باشد، اگر چه اطلاعات ارائه شده بیشتر اولیه هستند. عملیات یک مقدار را در AX برمی گرداند که بیت‌های ۴ و ۵ بر حالت ویدئو دلالت دارند :

- ۰۱: ۴۰ ستون × ۲۵ سطر، یک تطبیق دهنده رنگی.
- ۱۰: ۸۰ ستون × ۲۵ سطر، یک تطبیق دهنده رنگی.
- ۱۱: ۸۰ ستون × ۲۵ سطر، یک تطبیق دهنده تک رنگ.

می‌توانید AX را برای نوع مونیتور بررسی کنید و سپس حالت را بر طبق آن تنظیم نمایید.

Mode	Size	Type	Adapter	Resolution	Colors
00	(25 rows, 40 cols)	Mono	CGA	320 x 200	
			EGA	320 x 350	
			MCGA	320 x 400	
			VGA	360 x 400	
01	(25 rows, 40 cols)	Color	CGA	320 x 200	16
			EGA	320 x 350	16 of 64
			MCGA	320 x 400	16 of 262,144
			VGA	360 x 400	16 of 262,144
02	(25 rows, 80 cols)	Mono	CGA	640 x 200	
			EGA	640 x 350	
			MCGA	640 x 400	
			VGA	720 x 400	
03	(25 rows, 80 cols)	Color	CGA	640 x 200	16
			EGA	640 x 350	16 of 64
			MCGA	640 x 400	16 of 262,144
			VGA	720 x 400	16 of 262,144
07	(25 rows, 80 cols)	Mono	MDA	720 x 350	
			EGA	720 x 350	
			VGA	720 x 400	

Note: MDA: Monochrome display adapter
CGA: Color graphics adapter
MCGA: Multicolor graphics array
VGA: Video graphics array

شکل ۱-۱۰ حالت‌های متنی برای نمایش ویدئو

استفاده از حالت متنی

حالت متنی برای نمایش معمول مجموعه ۲۵۶ کاراکتر ASCII توسعه یافته بر بروی صفحه استفاده می‌شود. پردازش در هر دو حالت رنگی و تک رنگ مشابه است، بجز آنکه در حالت رنگی صفت زیر خط‌دار وجود ندارد.

- حالت‌های متنی 00 (تک رنگ و 01 (رنگی). قالب ۴۰ ستونی را فراهم می‌سازد، اگر چه اصولاً برای CGA طراحی شده بر روی سیستم‌های VGA و EGA نیز کار می‌کند.
- حال متنی 02 (تگ رنگ) و 03 (رنگی). قالب ۸۰ ستونی را فراهم می‌سازد. اگر چه اصولاً برای CGA طراحی شده بر روی سیستم‌های VGA و EGA نیز کار می‌کند.
- حالت متنی 07 (تک رنگ). حالت تک رنگ استاندارد برای VGA ، EGA و MDA با درجه تنظیم صفحه قابل توجه می‌باشد.

بایت صفت

یک **بایت صفت** در حالت متن (نه گرافیکی) مشخصات کاراکتر نشان داده شده را تعیین می‌کند. وقتی یک برنامه یک صفت را تنظیم می‌کند، این تنظیم باقی می‌ماند تا همه کاراکترهای بعدی نیز با همان خصوصیات نشان داده شوند تا زمانیکه عملیات دیگری آن را تغییر دهد. با استفاده از توابع INT 10H می‌توان صفات یک صفحه را تولید کرد و عملیاتی مانند: حرکت طوماری به بالا، حرکت طوماری به پایین، خواندن صفت یا کاراکتر و نمایش صفت یا کاراکتر را انجام داد. اگر شما از DEBUG برای دیدن ناحیه نمایش ویدئویی سیستم استفاده کنید، می‌توانید بترتیب یک بایت کاراکتر و پس از آن یک بایت صفت را ببینید. بایت طبق برای موقعیت بیت‌ها قالب زیر را دارد :

حروف R ، G و B بر موقعیت بیت برای قرمز، سبز و ابی، برای هر یک از رنگ اضافی زیر دلالت دارد.

	پیش زمینه		پس زمینه	
صفت :	B G R I		B G R	BL
شماره بیت:	۰ ۱ ۲ ۳		۴ ۵ ۶	۷

- بیت ۷ (BL) حالت چشمک زن را تنظیم می‌کند.
- بیت‌های ۴ تا ۶ **پس‌زمینه** صفحه را تعیین‌می‌کند.
- بیت ۳ (I) **شدت بالا** را تنظیم می‌کند.
- بیت‌های ۰ تا ۲ پیش زمینه صفحه را تعیین می‌کند (برای کاراکترهایی که نمایش داده خواهد شد).

مونیتورهای گرافیکی رنگی RGB معمولاً سیگنال‌های ورودی می‌پذیرند که به سه دستگاه پرتاپ الکترون مجزا - قرمز، سبز و آبی - برای هر یک از رنگ‌های اضافی فرستاده می‌شود. بیت‌های RGB یک رنگ را تعریف می‌کنند، هم بر روی تک رنگ یا رنگی، 000 سیاه است و 111 سفید است. برای مثال، یک صفت که مقدار 00000111 را دارد یعنی پیش زمینه سفید بر روی زمینه سیاه.

نمایش رنگ

برای اغلب مونیتورها، پس زمینه می‌تواند ۸ رنگ و پیش زمینه می‌تواند ۱۶ رنگ را نمایش دهد. چشمک زن و شدت فقط برای پیش زمینه استفاده می‌شود. درضمن برای حاشیه می‌تواند ۱۶ رنگ را انتخاب نمایید. مونیتورهای رنگی حالت زیر خط دار ندارند، در عوض تنظیم بیت 0 رنگ آبی را برای پیش زمینه انتخاب می‌کند.

رنگ	I	R	G	B	رنگ	I	R	G	B
سیاه	0	0	0	0	خاکستری	1	0	0	0
آبی	0	0	0	1	آبی روشن	1	0	0	1
سبز	0	0	1	0	سبز روشن	1	0	1	0
فیروزه‌ای	0	0	1	1	فیروزه‌ای روشن	1	0	1	1
قرمز	0	1	0	0	قرمز روشن	1	1	0	0
صورتی	0	1	0	1	صورتی روشن	1	1	0	1
قهوه‌ای	0	1	1	0	زرد	1	1	1	0
سفید	0	1	1	1	سفید با شدت بالا	1	1	1	1

سه رنگ اصلی قرمز (R)، سبز (G) و آبی (B) می‌باشد. می‌توانید این سه رنگ را در بایت صفت طوری ترکیب کنید که ۸ رنگ (شامل سیاه و سفید) و با تنظیم شدت بالا ۱۶ رنگ در بایت صفت داشته باشد.

اگر رنگ‌های پیش زمینه و پس زمینه یکی باشد، کاراکترهای نمایش داده شده قابل دیدن نخواهد بود. با استفاده از بایت صفت می‌توانید کاراکتر پیش زمینه را چشمک‌زن نمایید. در اینجا برخی از انواع صفت وقتی که BL به منظور چشمک‌زن باشد، آورده شده است.

پیش زمینه	پس زمینه	پس زمینه				پیش زمینه				
		BL	R	G	B	I	R	G	B	شانزدهی
سیاه	سیاه	0	0	0	0	0	0	0	0	00
سیاه	آبی	0	0	0	0	0	0	0	1	01
آبی	قرمز	0	0	0	1	0	1	0	0	14
سبز	فیروزه‌ای	0	0	1	0	0	0	1	1	23
سفید	صورتی‌روشن	0	1	1	1	1	1	0	1	7D
سبز	سبز(چشمک‌زن)	1	0	1	0	1	0	0	0	A8

نمایش تک رنگ

برای یک مونیتور تک رنگ، بایت صفت مانند روش نشان داده شده برای مونیتور رنگی استفاده می‌شود بجز آنکه بیت 0 برای تنظیم صفت زیر خط دار می‌باشد. برای مشخص کردن صفات از ترکیب بیت‌های زیر استفاده کنید.

شانزدهی	پس‌زمینه I	R	G	B	پیش‌زمینه BL	R	G	B	مورد	پیش‌زمینه	پس‌زمینه
00H	0	0	0	0	0	0	0	0	دیده‌نمی‌شود	سیاه	سیاه
07H	0	1	1	1	0	0	0	0	معمولی	سفید	سیاه
87H	0	1	1	1	1	0	0	0	چشمک‌زن	سفید	سیاه
0FH	1	1	1	1	0	0	0	0	شدت بالا	سفید	سیاه
70H	0	0	0	0	0	1	1	1	معکوس	سیاه	سفید
F0H	0	0	0	0	1	1	1	1	معکوس‌چشمک‌زن	سیاه	سفید
01H	0	0	0	1	0	0	0	0	زیر خط دار		

تنظیم صفت

می‌توانید با استفاده از INT 11H نوع مونیتور نصب شده را تعیین کنید. سپس برای حالت رنگی از ترکیب رنگ توصیف شده یا برای تک رنگ از 07H برای صفت معمولی (پس زمینه سیاه، پیش زمینه سفید) استفاده کنید. صفت تنظیم شده تا هنگامی که عملیات دیگری آن را تغییر ندهد باقی می‌ماند. حالت متنی صفحات ۰-۳ نمایش را پشتیبانی می‌کند، که صفحه ۰ همان صفحه معمولی است. یک مثال، عملیات INT 10H زیر (قبلاً توضیح داده شد) از تابع 09H برای نمایش ۱۲ ستاره قهوه‌ای چشمک زن (1110) بر روی یک پس زمینه سیاه (0001) استفاده می‌کند:

```
MOV   AH,09H      ; درخواست نمایش
MOV   AL,'*'      ; ستاره
MOV   BH,00H      ; شماره صفحه صفر
MOV   BL,1EH      ; صفت رنگ (00011110)
MOV   CX,12       ; ۱۲ کارکتر متوالی
INT   10H         ; فراخوانی سرویس وقفه
```

می‌توانید از DEBUG برای بررسی این مثال و ترکیب دیگر رنگ‌ها نیز استفاده کنید.

صفحات نمایش

حالت‌های متنی مختلف، ذخیره سازی داده‌ها در حافظه ویدئویی در صفحات را مجاز می‌دانند. در یک صفحه، داده‌های یک صفحه نمایش کامل با شماره ۰ تا ۳ برای حالت ۸۰ ستونی معمول (۰ تا ۷ برای صفحه نمایش ۴۰ ستونی) را ذخیره می‌سازد. در حالت ۸۰ ستونی، شماره صفحه صفر پیش فرض است و در ناحیه نمایش ویدئوئی B800[0] آغاز می‌شود، صفحه ۱ از B900[0]، صفحه ۲ در BA00[0] و صفحه ۳ در BB00[0] آغاز می‌شود. اگر چه در هر لحظه فقط یک صفحه نمایش داده می‌شود ولی می‌توانید هر یک از صفحات در حافظه را قالب‌بندی کنید. هر کاراکتر که نمایش داده می‌شود به ۲ بایت حافظه نیاز دارد - یک بایت برای کاراکتر و بایت دوم برای صفت آن. با این روش یک صفحه کامل کاراکترها که ۸۰ ستون و ۲۵ سطر دارد به ۴۰۰۰=۲×۲۵×۸۰ بایت نیاز دارد. میزان حافظه که برای هر صفحه واقعاً در نظر گرفته می‌شود 4K یا 4,096 بایت می‌باشد، که ۹۶ بایت بلافاصله بعد از آن بلااستفاده است.

استفاده از INT 10H برای حالت متنی

اخیراً، از INT 10H تابع 00H برای تنظیم حالت نمایش استفاده کردیم. INT 10H سرویس‌های دیگری برای سهولت و دستکاری صفحه نمایش فراهم نموده است. عملیات INT محتویات ثبات‌های BP، BX، CX، DX، DI و SI را حفظ می‌کند ولی محتویات AX را حفظ نمی‌کند این نکته‌ای است که به هنگام استفاده از INT 10H در یک حلقه باید بخاطر بسپارید. بخش‌های بعدی هریک از تابع را توصیف می‌کند.

INT 10H تابع 00H: تنظیم حالت ویدئو

همانطور که قبلاً توضیح داده شد، از این عملیات برای تنظیم حالت در AL، مانند 03 برای رنگی یا 07 برای تک رنگ می‌توانید استفاده نمایید. (شکل ۱-۱۰ را ببینید).

INT 10H تابع 01H، تنظیم اندازه مکان نما

مکان‌نما بخشی از مجموعه کاراکتر ASCII نیست. جهت کنترل آن کامپیوتر دارای سخت‌افزار مخصوصی می‌باشد و برای استفاده از آن اعمال INT ویژه‌ای وجود دارد. نماد مکان‌نمای معمولی با خط زیر یا کاراکتر شکست مشابه می‌باشد. برای تنظیم اندازه آن از حیث عمودی از تابع 01H استفاده کنید. این ثبات‌ها را تنظیم کنید:

- بیت‌های ۰-۴ از CH = بالای مکان‌نما (خط شروع پیمایش)
- بیت‌های ۰-۴ از CL = پایین مکان‌نما (خط پایان پیمایش)

این اندازه بین خط بالای پیمایش و خط پایین می‌تواند باشد - ۰:۱۴ برای ۰:۱۳VGA برای تک رنگ EGA و ۰:۷ برای CGA، کدهای زیر مکان‌نما را در VGA به حداکثر اندازه بزرگ می‌کنند.

```
MOV   AH,01H  ; درخواست تنظیم اندازه مکان‌نما
MOV   CH,00   ; خط شروع پیمایش
MOV   CL,14   ; خط پایان پیمایش
INT   10H     ; فراخوانی سرویس وقفه
```

حال مکان‌نما به صورت یک مربع مستطیل توپر چشمک می‌زند. اندازه مکان‌نما را می‌توانید در هر جایی بین این محدوده‌ها، مثل 04/08 یا 03/10 و غیره تنظیم نمایید. تا زمانیکه عمل دیگری اندازه مکان‌نما را تغییر نداده است، مکان‌نما در همان نوع قبلی باقی خواهد ماند. استفاده از 0:14 (VGA)، 12:13 (تک رنگ یا EGA) یا (CGA) 6:7 مکان‌نما را به حالت عادی برمی گرداند. اگر از محدوده مکان‌نما در مونیتور مطمئن نیستید از تابع 03H در DEBUG بدین منظور استفاده کنید.

INT10H تابع 02H: تنظیم موقعیت مکان نما

این عمل بر طبق مختصات سطر و ستون، مکان‌نما را در هر جای صفحه تصویر قرار می‌دهد. این ثبات‌ها را تنظیم کنید:

BH = شماره صفحه، می‌تواند ۰، ۱، ۲ و یا ۳ برای حالت متنی ۸۰ ستونی باشد.

DH = سطر.

DL = ستون.

موقعیت مکان‌نما در یک صفحه مستقل از موقعیت آن در صفحه دیگری است. این مثال سطر ۱۲، ستون ۳۰ برای صفحه صفر را تنظیم می‌نماید.

```
MOV   AH,02H  ; درخواست تنظیم مکان‌نما
MOV   BH,00H  ; صفحه شماره صفر
MOV   DH,12   ; سطر ۱۲
MOV   DL,30   ; ستون ۳۰
INT   10H     ; فراخوانی سرویس وقفه
```

INT 10H تابع 03H: خواندن موقعیت مکان‌نما

یک برنامه می‌تواند سطر، ستون و اندازه جاری مکان‌نما را تعیین کند، بویژه در حالتی که از یک صفحه موقتاً استفاده می‌کند و باید صفحه اصلی را ذخیره و بازیابی نماید. شماره صفحه را در BH مانند تابع 02H قرار دهید.

```
MOV   AH,03H ; درخواست موقعیت مکان‌نما
MOV   BH,00  ; صفحه شماره صفر
INT   10H    ; فراخوانی سرویس وقفه
```

این عملیات مقادیر زیر را برمی‌گرداند:

AX = بدون تغییر CH = خط شروع پیمایش DH = سطر

BX = بدون تغییر CL = خط انتهای پیمایش DL = ستون

مثال زیر از تابع 03H برای خواندن مکان‌نما و تعیین موقعیت و اندازه آن استفاده می‌کند، سپس با استفاده از تابع 02H مکان‌نما را به ستون بعدی می‌برد.

```
MOV   AH,03H ; درخواست تنظیم مکان‌نما
MOV   BH,00  ; صفحه صفر
INT   10H    ; بازگشت ستون در DL
MOV   AH,02H ; درخواست تنظیم مکان‌نما
INC   DL     ; در ستون بعدی
INT   10H    ; فراخوانی سرویس وقفه
```

INT 10H تابع 05H: انتخاب صفحه فعال

تابع 05H برای حالت‌های متنی ۰-۳ و ۱۳-۱۶ به شما اجازه می‌دهد تا صفحه‌ای که باید نمایش داده شود را انتخاب نمایید. همچنین می‌توانید صفحات مختلف ایجاد نمایید و درخواست نمایش متناوب آن‌ها را در صفحات ۵-۳ (در حالت ۸۰ ستونی) داشته باشید. این عملیات ساده مقداری را برنمی‌گرداند.

```
MOV   AH,05H     ; درخواست صفحه فعال
MOV   AL,Page#   ; شمارهٔ صفحه فعال
INT   10H        ; فراخوانی سرویس وقفه
```

INT 10H تابع 06H: حرکت طوماری صفحه به بالا

وقتی یک برنامه قسمت پایین صفحه نمایش و بعد از انتهای صفحه را نمایش می‌دهد خط بعدی شروع به بالا آمدن می‌کند. اما، حتی اگر عمل وقفه ستون 0 را مشخص کند، خطوط جدید با فرورفتگی چند مکان به سمت داخل صفحه تصویر نمایش داده می‌شوند و ممکن است خطوط بعدی در موقع نمایش مورب به نظر برسند. راه‌حل این مشکل حرکت طوماری صفحه تصویر می‌باشد، بنابراین خطوط به بالا صفحه حرکت می‌کنند و خطوط خالی در انتهای صفحه ظاهر خواهند شد.

قبلاً از تابع 06 برای پاک کردن صفحه تصویر استفاده کردید، که یک مقدار صفر در AL سبب چرخش صفحه به بالا و پاک کردن آن خواهد شد. تنظیم یک مقدار غیر صفر در AL سبب خواهد شد تا تعدادی از خطوط در صفحه به سمت بالا بروند. ثبات‌های زیر را مقدار دهی کنید:

ستون: سطر آغاز = CX تعداد سطرها (۰۰ برای صفحه کامل) = AL

ستون: سطر انتها = DX صفت = BH

مثال زیر یک صفت رنگ را تنظیم و صفحه را یک خط به صورت طوماری به بالا حرکت می‌دهد.

```
MOV   AX,060H  ; درخواست حرکت طوماری به بالا، یک خط
MOV   BH,07    ; پس زمینه قهوه‌ای، پیش زمینه آبی
MOV   CX,0000  ; از 00:00 تا
MOV   DX,184FH ; صفحه کامل 24:79
INT   10H      ; فراخوانی سرویس وقفه
```

یک روش استاندارد برای حرکت طوماری صفحه تصویر در زیر آورده شده است:

۱) یک عنصر داده به نام ROW با ارزش اولیه صفر تعریف کنید. ROW را برای برقراری مکان سطر استفاده نمایید.
۲) یک خط را نمایش دهید و مکان‌نما را به خط بعدی منتقل کنید.
۳) بررسی کنید که آیا ROW نزدیک به پایین صفحه تصویر است یا خیر. (CMP ROW,22)
۴) اگر جواب بلی است ، یکی به ROW اضافه کنید (INC ROW) و خارج شوید.
۵) اگر جواب خیر است، یک خط را به صورت طوماری حرکت دهید و ROW را جهت برقرار کردن مجدد مکان‌نما استفاده نمایید.

ثبات‌های CX و Dx امکان حرکت طوماری هر بخش از صفحه را برقرار می‌کنند. دقت کنید که AL با فاصله DX و CX مقدار بگیرد، مخصوصاً وقتی بخشی از صفحه را مشخص می‌کنید. دستورات زیر یک پنجره ایجاد می‌کند (با صفت خودش) با ۷ سطر و ۳۰ ستون که بالا ی سمت چپ آن در ۱۲:۲۵ و بالای سمت راست آن در ۱۲:۵۴ می‌باشد و پایین سمت چپ در ۸:۲۵ و پایین سمت راست آن در ۱۸:۵۴ قرار دارد.

```
MOV   AX,0607H      ; درخواست حرکت طوماری ۷ خط
MOV   BH,30H        ; پیش زمینه فیروزه‌ای ،پس زمینه سیاه
MOV   CX,0C19H      ; از سطر ۱۲ و ستون ۲۵ تا
MOV   DX,1236H      ; سطر ۱۸ و ستون ۵۴
INT   10 H          ; فراخوانی روتین وقفه
```

این مثال ۷ خط را حرکت طوماری می‌دهد، که چون همان فاصله بین سطر ۱۲ و ۱۸ است این ناحیه پاک می‌شد. این یک تجربه عمومی برای ایجاد یک برنامه جهت حرکت طوماری (و پاک کردن) همه سطرهاست و بترتیب یک سطر در هر لحظه انجام می‌شود. ازآنجائیکه صفت یک پنجره تا زمانی که عملیات دیگری آن را تغییر نداده است باقی می‌ماند، می‌توانید پنجره‌های مختلف با صفات مختلف در یک زمان تعریف نمایید.

INT 10H تابع 07H: حرکت طوماری صفحه به پایین

برای حالت متنی، حرکت طوماری به پایین سبب می‌شود تا خطوط پایین از صفحه خارج شوند و خطوط خالی در بالای صفحه ظاهر گردند. بجز آنکه این تابع حرکت طوماری به پایین دارد، کاملاً مانند تابع 06 که به سمت بالا حرکت طوماری دارد، عمل می‌کند. ثبات‌های زیر را مقدار دهی کنید:

AL = تعداد سطرها (۰۰ برای همه صفحه)　　CX = ستون : سطر آغاز
BH = صفت　　DX = ستون : سطر انتها

INT 10H تابع 08H: خواندن صفت /کاراکتر در موقعیت جاری مکان‌نما

تابع 08H می‌تواند هم در حالت متنی و هم در حالت گرافیکی کاراکتر یا صفت آن را از ناحیه نمایش ویدئویی بخواند. شماره صفحه را صفر در BH قرار دهید و مانند مثال زیر :

```
MOV   AH,08H      ; درخواست خواندن صفت /کاراکتر
MOV   BH,00       ; تنظیم صفحه صفر
INT   10H         ; فراخوانی سرویس وقفه
```

عملیات، کاراکتر را در AL و صفت آن را به AH باز می‌گرداند. در حالت گرافیک، عملیات 00H را برای یک کاراکتر غیر ASCII باز می‌گرداند. چون این عمل در هر زمان یک کاراکتر را می‌خواند، برای خواندن کاراکترهای متوالی باید یک حلقه، کد نمایید.

INT 10H تابع 09H : نمایش صفت/کاراکتر در موقعیت جاری مکان نما

یک عملیات مهم در اینجا آورده شده است که تعداد مشخص شده کاراکتر را در حالت متنی یا گرافیکی بر طبق صفت داده شده نمایش می‌دهد. این ثبات‌ها را تنظیم نمایید:

AL = کاراکتری ASCII جهت نمایش BL = صفت

BH = شماره صفحه CX = شمارش

تعداد شمارش CX، دفعاتی که عملیات، نمایش کاراکتر AL را تکرار می‌کند مشخص می‌سازد. مثال زیر یک صفت رنگ را برقرار می‌سازد و ۶۰ "صورت شاد" را نمایش می‌دهد (01H)

```
MOV   AH,09H      درخواست نمایش ;
MOV   AL,01H      "صورت شاد" برای نمایش ;
MOV   BH,0        شماره صفحه صفر (معمول) ;
MOV   BL,16H      پس زمینه آبی، پیش زمینه قهوه‌ای ;
MOV   CX,60       ۶۰ بار تکرار کاراکتر ;
INT   10H         فراخوانی سرویس وقفه ;
```

این عملیات در صورت برخورد با کاراکترهای زنگ، Carriage Retrrn ، Line Feed یا Tab مکان‌نما را تغییر نمی‌دهد، بلکه آنها را بصورت کاراکترهای ASCII نمایش می‌دهد. مثال زیر سه دل چشمک‌زن را در حالت معکوس نمایش، نشان می‌دهد:

```
MOV   AH,09H      درخواست نمایش ;
MOV   AL,03H      کاراکتر دل ;
MOV   BH,00       صفحه صفر ;
MOV   BL,0F0H     نمایش معکوس و چشمک زن ;
MOV   CX,10       ده مرتبه ;
INT   10H         فراخوانی سرویس وقفه ;
```

نمایش کاراکترهای مختلف به یک حلقه نیاز دارد. در حالت متنی و نه گرافیکی، کاراکترها اگر از سمت راست تجاوز نمایند، تابع 09H بطور اتوماتیک نمایش را بر روی سطر بعدی در ستون 00 ادامه می‌دهد. برای نمایش یک اعلان یا پیغام، یک روتین راکد نمایید که CX را با 01 تنظیم نماید و به منظور انتقال کاراکتر در هر لحظه از حافظه به AL تکرار شود. (چون CX اشغال شده است، نمی‌توانید از دستور LOOP استفاده نمایید) همچنین، بعد از نمایش هر کاراکتر، از INT 10H تابع 02H برای تغییر مکان‌نما ستون بعدی استفاده نمایید.

INT 10H تابع 0AH: نمایش کاراکتر در موقعیت جاری مکان نما

این عملیات یک کاراکتر را در حالت متن یا گرافیک نمایش می‌دهد. تنها تفاوت بین تابع 0AH و 09H در این است که در حالت متنی تابع 0AH از صفت جاری استفاده می‌کند در حالیکه تابع 09H صفت را نیز تنظیم می‌نماید. در اینجا این کد برای تابع 0AH نوشته شده است.

```
MOV   AH,0AH          درخواست نمایش ;
MOV   AL,Char         کاراکتر نمایش ;
MOV   BH,Page#        شماره صفحه ;
MOV   CX,repetition   تعداد تکرار کاراکتر ;
INT   10H             فراخوانی سرویس وقفه ;
```

عملیات INT 21H که می‌تواند رشته‌ای از کاراکترها را نمایش دهد و به کاراکترهای کنترلی صفحه پاسخ دهد، بیشتر از عملیات INT 10H مرسوم می‌باشد.

INT 10H تابع 0EH: نوشتن تله تایپ

این عملیات امکان استفاده از یک مونیتور را به عنوان یک ترمینال ساده، به شما می‌دهد. ثبات AH را با 0EH، ثبات AL را با کاراکتری که باید نمایش داده شود، ثبات BL را با رنگ پیش زمینه (حالت گرافیکی) و ثبات BH را با شماره صفحه مقدار دهید :

```
MOV   AH,0EH        درخواست نمایش ;
MOV   AL,Char       کاراکتر نمایش ;
MOV   BH,Page#      شماره صفحه (برخی سیستم‌ها) ;
MOV   BL,Color      رنگ پیش زمینه (حالت گرافیکی) ;
INT   10H           فراخوانی سرویس وقفه ;
```

کاراکترهای کنترلی به عنوان فرامینی برای قالب بندی صفحه تصویر عمل می‌کنند. عملیات بطور اتوماتیک مکان‌نما را تغییر می‌دهد، کاراکترهای اضافی بر عرض صفحه تصویر را روی خط بعدی نشان می‌دهد، صفحه تصویر را به صورت طوماری حرکت می‌دهد و صفات جاری صفحه تصویر را حفظ می‌نماید.

INT 10H تابع 0FH: بدست آوردن حالت جاری ویدئو

از این تابع برای تعیین حالت ویدئو جاری می‌توان استفاده کرد. (تابع 00H را ببینید) در اینجا یک مثال ذکر شده است.

```
MOV   AH,0FH ; درخواست حالت ویدئو
INT   10H    ; فراخوانی سرویس وقفه
CMP   AL,03  ; اگر حالت 03 است
JE    ...    ; پرش
```

این عملیات مقادیر زیر را باز می‌گرداند :

AL = حالت جاری ویدئو AH = کاراکترهای هر خط ۲۰، ۴۰، یا ۸۰ BH = شماره صفحه جاری.

INT 10H تابع 11H: تولید کننده کاراکتر

این تابع پیچیده برای مقدار دهی و تنظیم حالت محیط ویدئویی سیستم‌های VGA ، EGA و MCGA می‌باشد. بحث راجع به آن خارج از حد این کتاب است.

INT 10H تابع 12H: انتخاب روتین صفحه نمایش

این تابع مونیتورهای VGA و EGA را پشتیبانی می‌کند. برای حصول اطلاعاتی راجع به مونیتور 10H را در BL قرار دهید و از این تابع استفاده کنید. عملیات مقادیر زیر را باز می‌گرداند.

- BH = 00H برای رنگی و 01H برای تک رنگ.
- BL = 00H برای 64K ، 01H برای 128K ، 02H برای 192K و 03H برای 256k .
- CH = بیت‌های تطبیق دهنده.
- CL = تنظیم سوئیچ‌ها.

این عملیات تعدادی از توابع پرکار، مانند 30H (انتخاب خطوط پیمایش)، 31H (بارگذاری صفت پیش فرض) و 34H (مکان‌نما) را پشتیبانی می‌کند.

INT 10H تابع 13H: نمایش رشته کاراکتر

در مونیتورهای VGA و EGA، این عملیات رشته را با هر طول و با هر انتخاب تنظیم صفت و انتقال مکان‌نما، نشان می‌دهد. ثبات‌های BP: ES باید حاوی افست: سگمنت برای هر رشته نمایش داده شده باشند. این عملیات بر

روی کاراکترهای کنترلی فعال است، در اینجا یک مثال آورده شده است.

```
MOV   AH,13h          ; درخواست نمایش رشته
MOV   AL,Subfunction  ; 00، 01، 02، یا03
MOV   BH,Page#        ; شماره صفحه
MOV   BL,attributes   ; صفت صفحه
LEA   BP,address      ; آدرس رشته در ES:BP
MOV   CX,Length       ; طول رشته
MOV   DX,Screen       ; موقعیت شروع مرتبط در صفحه
INT   10H             ; فراخوانی سرویس وقفه
```

چهار زیر تابع که در AL تنظیم می‌شود عبارتند از:

- 00 نمایش صفت و رشته، عدم تغییر موقعیت مکان‌نما
- 01 نمایش صفت و رشته، تغییر موقعیت مکان‌نما
- 02 نمایش کاراکتر و سپس صفت، عدم تغییر موقعیت مکان‌نما
- 03 نمایش کاراکتر و سپس صفت، تغییر موقعیت مکان‌نما

برنامه: نمایش مجموعه کاراکترهای ASCII

برنامه شکل ۱-۹ با استفاده از INT 21H مجموعه کاراکترهای ASCII را نشان می‌دهد، اما این عملیات بر روی کاراکترهای کنترلی ***فعال‌تر*** است، تا ***نمایش*** آنها. برای حل این مشکل برنامه شکل ۲-۱۰ با استفاده از INT 10H با توابع زیر بازنویسی شده است:

```
TITLE     A10BIOAS (COM)  INT 10H to display ASCII character set
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
CTR       DB      00              ;Counter for ASCII characters
COL       DB      24              ;Column of screen
ROW       DB      04              ;Row of screen
MODE      DB      ?               ;Video mode
;                 Main procedure:
;                 --------------
A10MAIN   PROC    NEAR
          CALL    B10MODE         ;Get/set video mode
          CALL    C10CLR          ;Clear screen
A20:
          CALL    D10SET          ;Set cursor
          CALL    E10DISP         ;Display characters
          CMP     CTR,0FFH        ;Last character displayed?
          JE      A30             ;  yes, exit
          INC     CTR             ;Increment char. counter
          ADD     COL,02          ;Increment column
          CMP     COL,56          ;At end of column?
          JNE     A20             ;  no, bypass
          INC     ROW             ;  yes, increment row
          MOV     COL,24          ;  and reset column
          JMP     A20
A30:
          CALL    F10READ         ;Get keyboard character
          CALL    G10MODE         ;Restore video mode
          MOV     AX,4C00H        ;End of processing
          INT     21H
A10MAIN   ENDP
;                 Get and set video mode:
```

شکل الف ۲-۱۰ INT 10H برای نمایش مجموعه کاراکترهای ASCII

```
;                       ----------------------
B10MODE     PROC    NEAR
            MOV     AH,0FH          ;Request get mode
            INT     10H
            MOV     MODE,AL         ;Save mode
            MOV     AH,00H          ;Request set new mode
            MOV     AL,03           ;Standard color
            INT     10H
            RET
B10MODE     ENDP
;                   Clear screen and create window:
;                   ------------------------------
C10CLR      PROC    NEAR
            MOV     AH,08H          ;Request get current
            INT     10H             ;  attribute in AH
            MOV     BH,AH           ;Move it to BH
            MOV     AX,0600H        ;Scroll whole screen
            MOV     CX,0000         ;Upper left location
            MOV     DX,184FH        ;Lower right location
            INT     10H
            MOV     AX,0610H        ;Create 16-line window
            MOV     BH,16H          ;Blue back, brown foregrd.
            MOV     CX,0418H        ;Upper left corner  04:24
            MOV     DX,1336H        ;Lower right corner 19:54
            INT     10H
            RET
C10CLR      ENDP
;                   Set cursor to row and column:
;                   -----------------------------
D10SET      PROC    NEAR
            MOV     AH,02H          ;Request set cursor
            MOV     BH,00           ;Page 0 (normal)
            MOV     DH,ROW          ;New row
            MOV     DL,COL          ;New column
            INT     10H
            RET
D10SET      ENDP
;                   Display ASCII characters:
;                   -------------------------
E10DISP     PROC    NEAR
            MOV     AH,0AH          ;Request display
            MOV     AL,CTR          ;ASCII char
            MOV     BH,00           ;Page 0
            MOV     CX,01           ;One character
            INT     10H
            RET
E10DISP     ENDP
;                   Force pause, get keyboard character:
;                   ------------------------------------
F10READ     PROC    NEAR
            MOV     AH,10H          ;Request get character
            INT     16H
            RET
F10READ     ENDP
;                   Restore original video mode:
;                   ----------------------------
G10MODE     PROC    NEAR
            MOV     AH,00H          ;Request set mode
            MOV     AL,MODE         ;Original value
            INT     10H
            RET
G10MODE     ENDP
            END     BEGIN
```

شکل ب ۲-۱۰ *INT 10H* برای نمایش مجموعه کاراکترهای *ASCII*

0FH بدست آوردن حالت جاری ویدئو و ذخیره آن.

00H تنظیم حالت ویدئو با 03 در این برنامه، و بازیابی حالت اصلی هنگام خروج.

08H خواندن صفت موقعیت جاری مکان‌نما برای استفاده در تابع 06H.

06H حرکت طوماری صفحه به بالا به منظور پاک کردن کامل صفحه، یا استفاده از صفاتی که خوانده شده است. همچنین، ایجاد یک پنجره ۱۶ خطی با پیش‌زمینه قهوه‌ای و پس زمینه آبی برای کاراکترهای نمایش داده شده.

02H تنظیم اولیه مکان‌نما، و تغییر مکان آن برای هر کاراکتر نمایش داده شده.

0AH نمایش هر کاراکتر، شامل کاراکترهای کنترلی در موقعیت جاری مکان‌نما.

کاراکترها در ۱۶ سطر و ۱۶ ستون نمایش داده می‌شوند. این برنامه مانند دیگر برنامه‌های این کتاب، سعی شده واضح‌تر باشد تا اینکه سریع باشد، شما می‌توانید این برنامه را برای سریع‌تر اجرا شدن بازنویسی کنید، برای مثال، برای سطر، ستون و تولید کاراکترهای ASCII از ثبات استفاده کنید. همچنین، از آنجائیکه INT 10H فقط محتویات ثبات AX را از بین می‌برد، مقادیر دیگر ثباتها نباید دوباره بارگذاری شوند. اما برنامه به مقدار قابل توجهی سریع‌تر نخواهد شد و وضوح خود را نیز از دست می‌دهد.

کاراکترهای ASCII برای کادرها و منوها

در میان کاراکترهای ASCII توسعه یافته ۱۲۸-۲۵۵ (FFH - 80H) تعدادی کاراکتر خاص وجود دارند که برای نمایش اعلان‌ها، منوها، کادرها مفید هستند، همانطور که در ضمیمه B و شکل ۳-۱۰ نشان داده شده است.

مثال زیر از INT 10H تابع 09H برای نمایش یک خط افقی با طول ۲۵ موقعیت استفاده می‌کند :

```
MOV   AH,09H          ; درخواست نمایش ;
MOV   AL,0C4H         ; یک خط افقی ;
MOV   BH,00           ; صفحه صفر ;
MOV   BL,0FH          ; زمینه سفید، پیش زمینه سفید، با شدت زیاد ;
MOV   CX,25           ; ۲۵ موقعیت ;
INT   10H             ; فراخوانی سرویس وقفه ;
```

بخاطر بسپارید که اگر چه تابع 09H رشته‌ای از کاراکترها را نمایش می‌دهد، اما مکان‌نما را تغییر نمی‌دهد. روش ساده‌تر برای نمایش یک کادر تعریف آن در سگمنت داده و آنگاه نمایش کامل ناحیه است، مثال بعد یک منو تعریف می‌کند که به شکل یک کادر با خط تکی راست می‌باشد.

```
CR    EQU  0DH                ;Carriage return
LF    EQU  0AH                ;Line feed
MENU  DB   0DAH,  17 DUP(0C4H), 0BFH, CR, LF
      DB   0B3H,  ' Add records     ', 0B3H, CR, LF
      DB   0B3H,  ' Delete records  ', 0B3H, CR, LF
      DB   0B3H,  ' Enter orders    ', 0B3H, CR, LF
      DB   0B3H,  ' Print report    ', 0B3H, CR, LF
      DB   0B3H,  ' Update accounts ', 0B3H, CR, LF
      DB   0B3H,  ' View records    ', 0B3H, CR, LF
      DB   0C0H,  17 DUP(0C4H), 0D9H, CR, LF
      ...
      MOV  AH,40H             ;Request display
      MOV  BX,01              ;File handle for screen
      MOV  CX,168             ;Number of characters: 8 rows
      LEA  DX,MENU            ; x 21 columns
      INT  21H                ;Call interrupt service
```

در فصل بعدی، شکل ۱-۱۱ و ۲-۱۱ یک منوی مشابه با خط دوتایی با کاراکترهای 'dotson' برای نمایش سایه در سمت راست و پایین کادر را مشخص می‌سازد.

در لیست زیر کاراکترهای مختلف، ذکر شده است.

B0H یک چهارم نقطه	DFH سایه توپر، نیمه بالا
B1H یک دوم نقطه	DDH سایه توپر، نیمه چپ
B2H سه چهارم نقطه	DEH سایه توپر، نیمه راست
DBH سایه توپر	DCH سایه توپر، نیمه پایین

برنامه: چشمک زن، نمایش معکوس و حرکت طوماری صفحه نمایش

برنامه شکل ۴-۱۰ نام‌ها را از صفحه کلید می‌پذیرد و آنها را روی صفحه نمایش می‌دهد ، برای جلب توجه بیشتر اعلان را با نمایش معکوس نشان می‌دهد، (آبی روی سفید) و نام‌ها را به طور عادی می‌پذیرد (سفید روی آبی) و این نام‌ها را در ستون ۴۰ روی همان سطر با حالت چشمک زن و نمایش معکوس نشان می‌دهد:

```
Name ? Benjamin Franklin          Benjamin Franklin [Blinking]
       |                                  |
   ستون صفر                           ستون ۴۰
```

هنگام نمایش اسم و اعلان آن، برای کنترل محل مکان‌نما، برنامه، ROW را برای افزایش سطر، COL را برای حرکت مکان نما به صورت افقی تعریف می‌کند. (INT 10H تابعی 09H به طور خودکار موقعیت مکان را عوض نمی‌کند). برنامه شامل روال‌های زیر می‌باشد.

```
                  SINGLE      DOUBLE
CHARACTER          LINE        LINE          MIXED LINES
Straight Lines:
  Horizontal       C4H  -      CDH  =
  Vertical         B3H  |      BAH  ‖
Corners:
  Top left         DAH  ┌      C9H  ╔      D6H  ╓      D5H  ╒
  Top right        BFH  ┐      BBH  ╗      B7H  ╖      B8H  ╕
  Bottom left      C0H  └      C8H  ╚      D3H  ╙      D4H  ╘
  Bottom right     D9H  ┘      BCH  ╝      BDH  ╜      BEH  ╛
Middle:
  Left             C3H  ├      CCH  ╠      C7H  ╟      C6H  ╞
  Right            B4H  ┤      B9H  ╣      B6H  ╢      B5H  ╡
  Top              C2H  ┬      CBH  ╦      D2H  ╥      D1H  ╤
  Bottom           C1H  ┴      CAH  ╩      D0H  ╨      CFH  ╧
Center Cross       C5H  ┼      CEH  ╬      D7H  ╫      D8H  ╪

Blocks:
  One-quarter dots on      B0H  ▒   Solid shadow, upper half   DFH  ▀
  One-half dots on         B1H  ▒   Solid shadow, left half    DDH  ▌
  Three-quarter dots on    B2H  ▓   Solid shadow, right half   DEH  ▐
  Solid shadow             DBH  ▓   Solid shadow, lower half   DCH  ▄
```

شکل ۳-۱۰ کاراکترهای *ASCII* برای کادر و منو

- A10MAIN منطق اصلی پذیرش هر تعداد از ورودی صفحه کلید را فراهم می‌سازد.
- B10PROM یک اعلان برای کاربر جهت وارد کردن نام نمایش می‌دهد.
- C10INPT از INT 21H تابع 0AH برای ورودی صفحه کلید استفاده می‌کند.
- D10NAME پایین صفحه را نمایش می‌دهد تا به سطر ۲۰ برسد و آنگاه برای هر سطر اعلان اضافی یک بار حرکت

طوماری به سمت بالا انجام می‌دهد.

- E10DISP از INT 10H تابع 09H برای نمایش هر نام مانند کاراکترهای منحصر بفرد استفاده می‌کند.
- Q10SCRN حرکت طوماری صفحه را دستکاری می‌کند و تعداد AX را ورودی در نظر می‌گیرد.
- Q20CURS مکان‌نما را بر طبق مقدار جاری در ROW و COL تنظیم می‌کند.

```
TITLE     A10NMSCR (EXE) Reverse video, blinking, scrolling
          .MODEL  SMALL
          .STACK  64
; ---------------------------------------------------------
          .DATA
PARLIST   LABEL   BYTE              ;Name parameter list:
MAXLEN    DB      20                ;  maximum length of name
ACTLEN    DB      ?                 ;  no. of chars entered
KBNAME    DB      20 DUP(' ')       ;  name

COL       DB      00
COUNT     DB      ?
PROMPT    DB      'Name? '
ROW       DB      00
; ---------------------------------------------------------
          .CODE
A10MAIN   PROC    FAR
          MOV     AX,@data          ;Initialize segment
          MOV     DS,AX             ;  registers
          MOV     ES,AX
          MOV     AX,0600H
          CALL    Q10SCRN           ;Clear screen
A20LOOP:
          MOV     COL,00            ;Set column to 0
          CALL    Q20CURS
          CALL    B10PROM           ;Display prompt
          CALL    C10INPT           ;Provide for input of name
          CMP     ACTLEN,00         ;No name?
          JNE     A30               ;  if not, bypass
          MOV     AX,0600H          ;If so,
          CALL    Q10SCRN           ;  clear screen,
          MOV     AX,4C00H          ;End of processing
          INT     21H
A30:
          CALL    D10NAME           ;Display name
          JMP     A20LOOP
A10MAIN   ENDP
;                 Display prompt:
;                 ---------------
B10PROM   PROC    NEAR
          LEA     SI,PROMPT         ;Set address of prompt
          MOV     COUNT,05
B20:
          MOV     BL,71H            ;Reverse video
          CALL    E10DISP           ;Display routine
          INC     SI                ;Next character in name
          INC     COL               ;Next column
          CALL    Q20CURS           ;Set cursor
          DEC     COUNT             ;Countdown
          JNZ     B20               ;Loop n times
          RET
B10PROM   ENDP
;                 Accept input of name:
;                 ---------------------
C10INPT   PROC    NEAR
          MOV     AH,0AH            ;Request keyboard
          LEA     DX,PARLIST        ;  input
          INT     21H
          RET
C10INPT   ENDP
```

شکل الف ۴-۱۰ چشمک زن، نمایش معکوس ، حرکت طوماری

```
;                   Display name with blinking reverse video:
;                   -----------------------------------------
D10NAME     PROC    NEAR
            LEA     SI,KBNAME           ;Initialize name
            MOV     COL,40              ;Set screen column
D20:
            CALL    Q20CURS             ;Set cursor
            MOV     BL,0F1H             ;Blink reverse video
            CALL    E10DISP             ;Display routine
            INC     SI                  ;Next character in name
            INC     COL                 ;Next screen column
            DEC     ACTLEN              ;Countdown name length
            JNZ     D20                 ;Loop n times

            CMP     ROW,20              ;Near bottom of screen?
            JAE     D30
            INC     ROW                 ;  no, increment row
            RET
D30:
            MOV     AX,0601H            ;  yes,
            CALL    Q10SCRN             ;  scroll screen
            RET
D10NAME     ENDP
;                   Display character:
;                   ------------------
E10DISP     PROC    NEAR                ;BL (attribute) set on entry
            MOV     AH,09H              ;Request display
            MOV     AL,[SI]             ;Get name character
            MOV     BH,00               ;Page number
            MOV     CX,01               ;One character
            INT     10H
            RET
E10DISP     ENDP
;                   Scroll screen:
;                   --------------
Q10SCRN     PROC    NEAR                ;AX set on entry
            MOV     BH,17H              ;White on blue
            MOV     CX,0000
            MOV     DX,184FH            ;Full screen
            INT     10H
            RET
Q10SCRN     ENDP
;                   Set cursor row/col:
;                   -------------------
Q20CURS     PROC    NEAR
            MOV     AH,02H
            MOV     BH,00               ;Page
            MOV     DH,ROW              ;Row
            MOV     DL,COL              ;Column
            INT     10H
            RET
Q20CURS     ENDP
            END     A10MAIN
```

شکل ب ۴-۱۰ چشمک زن، نمایش معکوس ، حرکت طوماری

نمایش ویدئویی مستقیم

برای برخی کاربردها، نمایش ویدئو به سیستم عامل و BIOS فرستاده می‌شود که بطور قابل توجهی کند می‌باشد. روش سریع‌تر برای نمایش کاراکترها (متن یا گرافیک) انتقال مستقیم آن‌ها به ناحیه نمایش ویدئویی مناسب می‌باشد. برای مثال آدرس صفحه 0در ناحیه ویدئو برای انتقال 03 (رنگی، متن) است. B800[0]H است. هر کاراکتر صفحه دو بایت حافظه نیاز دارد - یکی برای کاراکتر و دیگری که بلافاصله پس از آن قرار دارد و برای صفت آن در نظر گرفته می‌شود. با صفحه‌ای به اندازه ۸۰ ستون و ۲۵ سطر، یک صفحه ناحیهٔ ویدئو به ۴۰۰۰ = ۲ × ۲۵ × ۸۰ بایت نیاز دارد.

اولین بایت ناحیه نمایش ویدئو، یک موقعیت در صفحه یعنی سطر 0 و ستون 0 را بیان می‌سازند و بایتی که در

موقعیت F9EH و F9FH قرار دارد موقعیت سطر ۲۴ و ستون ۷۹ را بیان می‌کند.

انتقال ساده یک کارکتر: صفت به ناحیه ویدئو از صفحه فعال سبب می‌شود تا کاراکتر بلافاصله بر صفحه نشان داده شود. می‌توانید با فرامین DEBUG این مطلب را بررسی کنید. ابتدا، ناحیه ویدئو در B800[0] را با فرمان زیر نشان دهید:

```
D B800 :00
```

```
TITLE     P10GRAFX (COM)  Graphics display
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN     PROC    NEAR
          MOV     AH,0FH          ;Preserve
          INT     10H             ;  original
          PUSH    AX              ;  video mode
          CALL    B10MODE         ;Set graphics mode
          CALL    C10DISP         ;Display color graphics
          CALL    D10KEY          ;Get keyboard response
          POP     AX              ;Restore
          MOV     AH,00H          ;  original mode
          INT     10H             ;  (in AL)
          MOV     AX,4C00H        ;Exit to DOS
          INT     21H
BEGIN     ENDP

B10MODE   PROC    NEAR
          MOV     AH,00H          ;Set EGA/VGA graphics mode
          MOV     AL,10H          ;640 cols x 350 rows
          INT     10H
          MOV     AH,0BH          ;Set background palette
          MOV     BH,00           ;Background
          MOV     BL,07H          ;Gray
          INT     10H
          RET
B10MODE   ENDP

C10DISP   PROC    NEAR
          MOV     BX,00           ;Set initial page,
          MOV     CX,64           ;  color, column,
          MOV     DX,70           ;  and row
C20:
          MOV     AH,0CH          ;Write pixel dot
          MOV     AL,BL           ;Set color
          INT     10H             ;BX, CX, & DX are preserved
          INC     CX              ;Increment column
          CMP     CX,576          ;Column at 576?
          JNE     C20             ;  no, loop
          MOV     CX,64           ;  yes, reset column
          INC     BL              ;Change color
          INC     DX              ;Increment row
          CMP     DX,280          ;Row at 280?
          JNE     C20             ;  no, loop
          RET                     ;  yes, terminate
C10DISP   ENDP

D10KEY    PROC    NEAR
          MOV     AH,10H          ;Request keyboard
          INT     16H             ;  input
          RET
D10KEY    ENDP
          END     BEGIN
```

شکل ۵-۱۰ استفاده از نمایش ویدئویی مستقیم

آنچه که بر روی صفحه است وقتی شما فرمان را تایپ می‌کنید نمایش خواهد داد که اغلب مجموعه‌ای از بایت‌ها حاوی 2007H می‌باشد (برای کاراکتر جای خالی، پس زمینه سیاه و پیش زمینه سفید) توجه کنید که DEBUG و شما برای یک ناحیه نمایش و صفحه رقابت دارید. سعی کنید که صفحه را با فرامین زیر که "صورت شاد" نمایش می‌دهند

تغییر دهید (۰۱ و ۰۲ و ۰۳) با صفات متفاوت (۲۵، ۳۶ و ۴۷) که در سطر بالا و پایین صفحه.

```
E B800 : 000   01  25  02  36  03  47
E B800 : F90   01  25  02  36  03  47
```

برنامه شکل ۵-۱۰ مثالی در انتقال مستقیم داده به ناحیه نمایش در B900[0] که صفحه یک است ارائه می‌دهد. این برنامه از SEGMENT AT برای تعریف ناحیه نمایش ویدئویی BIOS استفاده می‌کند که مانند یک سگمنت فرضی می‌باشد. DSPAREA موقعیت صفحه یک در شروع سگمنت را مشخص می‌سازد. این برنامه کاراکترها را در سطر ۵ تا ۲۰ و ستون ۱۰ تا ۶۹ نمایش می‌دهد، اولین سطر رشته‌ای از کاراکتر A (41H) با صفت 01H نمایش می‌دهد. دومین سطر رشته‌ای از کاراکتر B(42H) با صفت 02H نمایش می‌دهد و به همین ترتیب با افزایش هر سطر کاراکتر و صفت آن نیز افزایش می‌یابد. برنامه‌ای که موقعیت شروع صفحه در ناحیه نمایش ویدئو را برقرار می‌سازد برپایه این حقیقت بنا شده است که ۱۶۰=۲×۸۰ ستون در یک سطر قرار دارد. برای سطر ۵، ستون ۱۰ موقعیت شروع ۸۲۰=(۲×ستون ۱۰) + (سطر ۵×۱۶۰) می‌باشد. پس از نمایش یک سطر برنامه ۴۰ موقعیت در ناحیه نمایش برای شروع در سطر بعدی پیش می‌رود و تا به حروف Q (51H) برسد خاتمه می‌یابد.

سگمنت نمایش ویدئو برای صفحه یک بعنوان DSPSEG و صفحه بعنوان DSPAREA تعریف شده است. در شروع، برنامه حالت جاری و صفحه را ذخیره می‌کند و سپس حالت 03 را برای صفحه یک تنظیم می‌نماید.

در روال B10PROC، کاراکتر شروع و صفت در AX و افست شروع ناحیه ویدئو در DI مقدار دهی می‌شود.

دستور MOV WORD PTR [DSPAREA+DI],AX محتویات AL (کاراکتر) را به اولین بایت ناحیه نمایش و AH (صفت) را به دومین بایت منتقل می‌کند. روتین LOOP این دستور را ۶۰ بار اجرا می‌کند، کاراکتر: صفت را در اطراف صفحه نمایش می‌دهد. سپس صفت: کاراکتر را می‌افزاید و ۴۰ را با DI جمع می‌کند - ۲۰ برای انتهای سطر جاری و ۲۰ برای مشخص کردن شروع سطر بعدی (روی صفحه برای هر یک ۱۰ ستون سپس روتین نمایش سطر بعدی از کاراکترها را بیان می‌کند. در خاتمه نمایش، روال CIOINPT منتظر می‌شود تا کاربر یک کلید را بفشارد و سپس برنامه حالت اصلی را و صفحه را قبل از خاتمه بازیابی می‌کند.

استفاده از حالت گرافیکی

تطبیق دهنده‌های گرافیکی دوحالت اصلی عملیاتی دارند: متن (پیش فرض) و گرافیک. با استفاده از BIOS INT 10H تابع 00H می‌توان حالت متن یا گرافیک را تنظیم نمود، همانطور که دو مثال زیر نشان می‌دهد:

۱. تنظیم حالت گرافیکی برای VGA

```
درخواست تنظیم حالت ;   MOV   AH,00H
گرافیک رنگی ;          MOV   AL,0CH
فراخوانی سرویس وقفه;   INT   10H
```

۲. تنظیم حالت متن :

```
درخواست تنظیم حالت ;   MOV   AH,00H
متن رنگی ;             MOV   AL,03H
فراخوانی سرویس وقفه ;  INT   10H
```

درجه تنظیم و حالت‌های تطبیق دهنده‌های گرافیکی که در شکل ۶-۱۰ نشان داده شده است، مانند زیر می‌باشد.

- ***حالت گرافیکی 04H ، 05H و 06H***. این حالت‌های اصلی CGA، به منظور بالا بردن تطبیق در VGA و EGA نیز استفاده می‌شود، بنابراین بسیاری از برنامه‌هایی که برای CGA نوشته می‌شوند در VGA یا EGA نیز اجرا می‌شوند. آدرس ناحیه نمایش ویدئو در این حالت B800[0] می‌باشد.

Mode	Type	Adapter	Resolution	Colors
04H	Color	CGA, EGA, MCGA, VGA	320 x 200	4
05H	Mono	CGA, EGA, MCGA, VGA	320 x 200	
06H	Mono	CGA, EGA, MCGA, VGA	640 x 200	
0DH	Color	EGA, VGA	320 x 200	16
0EH	Color	EGA, VGA	640 x 200	16
0FH	Mono	EGA, VGA	640 x 350	
10H	Color	EGA, VGA	640 x 350	16
11H	Color	MCGA, VGA	640 x 480	2 of 262,144
12H	Color	VGA	640 x 480	16 of 262,144
13H	Color	MCGA, VGA	320 x 200	256 of 262,144

شکل ٦-١٠ حالت‌های گرافیکی برای نمایش ویدئو

- ***حالت‌های گرافیکی 0DH ، 0EH ، 0FH و 10H.*** این حالت‌های EGA اصلی برای بالا بردن تطبیق در VGA نیز استفاده می‌شوند، بنابراین بسیاری از برنامه‌هایی که برای EGA نوشته می‌شود و VGA نیز اجرا می‌شود. این حالت‌ها ۸، ۴، ۲ و ۲ صفحه از ناحیه نمایش ویدئو را پشتیبانی می‌کنند که صفحهٔ صفر، پیش فرض می‌باشد. آدرس ناحیه نمایش ویدئو برای این حالت A000[0] می‌باشد
- ***حالت‌های گرافیکی 11H ، 12H و 13H.*** این حالت‌ها برای VGA (و اکنون به ندرت (MCGA) بصورت خاص طراحی شده است و با دیگر تطبیق دهنده‌های ویدئویی قابل استفاده نمی‌باشد. آدرس ناحیه نمایش ویدئو برای این حالت A000[0] می‌باشد.

در حالت گرافیکی، ROM حاوی الگوی نقطه‌ای برای فقط ۱۲۸ کاراکتر اول می‌باشد. INT IFH دستیابی به IK ناحیه حافظه را فراهم می‌کند که ۱۲۸ کاراکتر بالایی را تعریف می‌کند، ۸ بایت برای هر کاراکتر در نظر گرفته می‌شود.

پیکسل‌ها

حالت‌های گرافیکی، از ***پیکسل‌ها*** (اجزای تصویر یا Pels) برای تولید الگوی رنگ استفاده می‌کنند. برای مثال، حالت 04H برای گرافیک رنگی استاندارد ۲۰۰ سطر از ۳۲۰ پیکسل را فراهم می‌سازد. هر بایت ۴ پیکسل را بیان می‌کند (۲ بیت برای هر پیکسل) از ۰ تا ۳ که به شکل زیر است:

بایت :	C1	C0	C1	C0	C1	C0	C1	C0
پیکسل :	0		1		2		3	

در هر فرمان، چهار رنگ در دسترس می‌باشد، که از ۰ تا ۳ شماره‌گذاری می‌شود. محدوده چهار رنگ بدلیل آن است که ۲ بیت، ۴ ترکیب بیتی را فراهم می‌سازد 00, 01, 10, 11 می‌توانید پیکسل 00 را برای هر یک از ۱۶ رنگ پس زمینه انتخاب نمایید.

رنگ		رنگ	
خاکستری	1 0 0 0	سیاه	0 0 0 0
آبی روشن	1 0 0 1	آبی	0 0 0 1
سبز روشن	1 0 1 0	سبز	0 0 1 0
فیروزه‌ای روشن	1 0 1 1	فیروزه‌ای	0 0 1 1
قرمز روشن	1 1 0 0	قرمز	0 1 0 0
صورتی روشن	1 1 0 1	صورتی	0 1 0 1
زرد	1 1 1 0	قهوه‌ای	0 1 1 0
سفید	1 1 1 1	خاکستری‌روشن	0 1 1 1

و می‌توانید پیکسل‌های 01، 10 و 11 را برای هر یک از دو جعبه رنگ انتخاب نمایید.

C1	C0	PALETTE 0	PALETTE 1
0	0	پس زمینه	پس زمینه
0	1	سبز	فیروزه‌ای
1	0	قرمز	صورتی
1	1	قهوه‌ای	سفید

از INT 10H تابع 0BH برای انتخاب یک جعبه رنگ و پس زمینه استفاده نمایید. اگر رنگ پس زمینه زرد و جعبه رنگ 0 را انتخاب نمایید، رنگ‌های در دسترس زرد، سبز، قرمز و قهوه‌ای می‌باشند. یک بایت شامل مقدار پیکسل 10101010 همه را قرمز نشان می‌دهد. اگر شما رنگ پس زمینه آبی و جعبه رنگ را انتخاب نمایید، رنگ‌های در دسترس آبی، فیروزه‌ای، صورتی، و سفید است.

INT 10H برای گرافیک

همانطور که قبلاً توضیح داده شد BIOS INT 10H دستکاری تمام صفحه را برای حالت گرافیک و متن‌ها تسهیل می‌کند. این عملیات محتویات ثبات‌های BX ، CX ، DX ، DI ، SI و BP را حفظ می‌کند ولی AX را حفظ نمی‌کند. بخش بعد هر یک از توابع INT 10H که با گرافیک کار می‌کنند، توضیح می‌دهد.

INT 10H تابع 00H: تنظیم حالت ویدئو

تابع 00H در AH و حالت 12H در AL حالت گرافیک رنگی استاندارد VGA را تنظیم می‌نماید:

```
درخواست تنظیم حالت برای ;     MOV   AH,00H
درجه تنظیم VGA ۴۸۰ × ۶۴۰ ;     MOV   AL,12H
فراخوانی سرویس وقفه ;          INT   10H
```

تنظیم حالت گرافیکی سبب می‌شود مکان‌نما ناپدید شود.

INT 10H تابع 04H : خواندن موقعیت قلم نوری

از این تابع در حالت گرافیک برای تعیین وضعیت یک قلم نوری استفاده نمایید. عملیات اطلاعات را ارائه می‌دهد.

- AH اگر موقعیت عوض شده باشد صفر و اگر عوض نشده باشد یک است.
- DX سطر در DH و ستون در DL .
- CH:BX موقعیت پیکسل با خط افقی در BH و ستون یا نقطه در BX.

INT 10H تابع 08H : خواندن صفت یا کاراکتر در موقعیت جاری

این تابع می‌تواند هم کاراکترها و هم صفت آنها را از ناحیه نمایش چه در حالت متن و یا گرافیک بخواند. بخش قبلی را ببینید، استفاده از INT 10H برای حالت متن.

INT 10H تابع 09H: نمایش صفت یا کاراکتر در موقعیت جاری

برای حالت گرافیکی، از BL برای تعریف رنگ پیش زمینه استفاده نمایید. اگر بیت ۷ صفر است، رنگ تعریف شده جایگزین رنگ پیکسل قبلی می‌شود، اگر بیت ۷ یک باشد، رنگ تعریف شده با آن ترکیب (XOR) می‌شود. بخش قبلی با عنوان استفاده از INT 10H برای حالت متن را ببینید.

INT 10H تابع 0AH: نمایش یک کاراکتر در موقعیت مکان نما

بخش قبلی با عنوان استفاده از INT 10H برای حالت متن را ببینید.

INT 10H تابع 0BH: تنظیم جعبه رنگ

از این تابع برای تنظیم جعبه رنگ و نمایش یک کاراکتر گرافیکی استفاده نمایید. مقدار BH (00 یا 01) منظور از ثبات BL را تعیین می‌نماید:

- BH = 00. انتخاب رنگ پس زمینه، در حالیکه BL حاوی مقدار رنگ در بیت‌های ۰-۳ می‌باشد. (۱۶ رنگ)

```
MOV   AH,0BH        درخواست ;
MOV   BH,00H        پس زمینه ;
MOV   BL,04         رنگ قرمز ;
INT   10H           فراخوانی سرویس وقفه ;
```

- BH = 01. انتخاب جعبه رنگ گرافیکی، در حالیکه BL حاوی جعبه رنگ خواهد بود (0 یا 1).

```
MOV   AH,0BH        درخواست رنگ ;
MOV   BH,01         انتخاب جعبه رنگ ;
MOV   BL,00         شماره صفر (سبز،قرمز،قهوه‌ای) ;
INT   10H           فراخوانی سرویس وقفه ;
```

یک بار که یک جعبه رنگ را تنظیم نمایید ، باقی می‌ماند، اما اگر این جعبه رنگ را عوض کنید صفحه کامل به آن ترکیب رنگ تغییر می‌یابد. اگر از تابع 0BH در حالت متن استفاده کنید، مقدار برای رنگ 0 از جعبه رنگ، رنگ حاشیه را تعیین می‌نماید.

INT 10H تابع 0CH: نوشتن نقطه پیکسل

از تابع 0CH برای نمایش یک رنگ انتخابی استفاده کنید (پس زمینه و جعبه رنگ). این ثبات را تنظیم می‌کند.

ستون = CX رنگ پیکسل = AL سطر = DX شماره صفحه (VGA/EGA)=BH

مقدار حداقل برای ستون یا سطر صفر است و حداکثر مقدار به حالت ویدئو بستگی دارد. مثال زیر یک پیکسل در سطر 50، ستون 200 را تنظیم می‌نماید:

```
MOV   AH,0CH        درخواست، نوشتن نقطه ;
MOV   AL,03H        رنگ پیکسل ;
MOV   BH,0          شماره صفحه صفر ;
MOV   CX,200        موقعیت افقی (ستون) ;
MOV   DX,50         موقعیت عمودی (سطر) ;
INT   10H           فراخوانی سرویس وقفه ;
```

حالت‌های (VGA/EGA) 0DH ، 0EH ، 0FH و 10H به ترتیب ۸، ۴، ۲ و ۲ صفحه ناحیه نمایش ویدئو فراهم می‌سازد. صفحه پیش فرض صفر است.

INT 10H تابع 0DH : خواندن نقطه پیکسل

این عملیات، مخالف تابع 0CH است، یک نقطه را برای تعیین مقدار رنگ آن می‌خواند. BH را با شماره صفحه (VGA/EGA) را با ستون، DX را با سطر تنظیم نمایید حداقل مقدار برای سطر و ستون صفر است و حداکثر مقدار به حالت ویدئو بستگی دارد. عملیات رنگ پیکسل را در AL باز می‌گرداند.

INT 10H تابع 0EH : نوشتن تله تایپ

بخش قبلی با عنوان استفاده از INT 10H برای حالت متن را ببینید.

INT 10H تابع 10H : تنظیم ثبات‌های جعبه رنگ

این تابع سیستم‌های VGA /EGA را دستکاری می‌کند. یک کد زیر تابع در AL عملیات را تعیین می‌کند.

- 00 تنظیم یک ثبات جعبه رنگ، در حالیکه BH حاوی مقدار تنظیم و BL حاوی ثبات تنظیم می‌باشد.
- 01 تنظیم ثبات تحت پیمایش[1] در حالیکه BH حاوی مقدار تنظیم می‌باشد.
- 02 تنظیم همه ثبات‌های جعبه رنگ و تحت پیمایش. ES:DX به یک جدول ۱۷ بایتی اشاره می‌کند، در حالیکه بایت‌های ۰-۱۵ مقدار جعبه رنگ و بایت ۱۶ مقدار تحت پیمایش می‌باشد.
- 03 تغییر بیت چشمک زن / شدت، در حالیکه 00 در BL شدت را توانا می‌سازد و 01 چشمک زن را توانا می‌سازد.

دیگر کدهای زیر تابع AL برای VGA تحت تابع 10H بشرح زیر است : 07H (خواندن ثبات جعبه رنگ خاص) 08H (خواندن ثبات تحت پیمایش) ،09H (خواندن همه ثبات‌های جعبه رنگ و تحت پیمایش) 10H (تنظیم ثبات رنگ خاص) 12H (تنظیم بلوک ثبات‌های رنگ)، 13H (انتخاب رنگ صفر)، 15H (خواندن ثبات رنگ خاص)، 17H (خواندن بلوک ثبات‌های رنگ) و 1AH (خواندن وضعیت رنگ صفحه).

INT 10H تابع 1AH: خواندن / نوشتن کد ترکیبی نمایش

این عملیات کدهایی که نوع نمایش مورد استفاده را تعیین می‌کند، باز می‌گرداند.

INT 10H تابع 1BH: بازگشت اطلاعات وضعیت / تابعی

این عملیات پیچیده اطلاعاتی را در یک بافر ۶۴ بایتی باز می‌گرداند که حالت ویدئو، اندازه مکان‌نما صفحه پشتیبانی شده و غیره را مشخص می‌سازد.

INT 10H تابع 1CH: ذخیره و بازیابی وضعیت ویدئو

این تابع وضعیت ویدئو را ذخیره و بازیابی می‌نماید، که شامل وضعیت ثبات‌های رنگ، ناحیه داده BIOS و سخت‌افزار ویدئو می‌باشد.

برنامه: تنظیم و نمایش حالت گرافیکی

برنامه شکل ۷-۱۰ شامل توابع INT 10H زیر برای نمایش گرافیکی می‌باشد:

0FH = نگهداری حالت اصلی.

00H = تنظیم حالت گرافیکی 12H .

0BH = انتخاب رنگ پس زمینه سبز.

0CH = نوشتن نقطه پیکسل برای ۶۴۰ ستون و ۴۸۰ سطر.

پنجره فعال ۲۱۰ سطر و ۵۱۲ ستون را نمایش می‌دهد (ستون ۶۴ تا ۵۷۶). توجه کنید که سطرها و ستون‌ها با عبارت نقطه‌ها و نه کاراکترها می‌باشند.

1- overscan

```
TITLE     A10GRAFX (COM)  Graphics display
          .MODEL SMALL
          .CODE
          ORG     100H
A10MAIN   PROC    NEAR
          MOV     AH,0FH          ;Preserve
          INT     10H             ;  original
          PUSH    AX              ;  video mode
          CALL    B10MODE         ;Set graphics mode
          CALL    C10DISP         ;Display color graphics
          CALL    D10KEYB         ;Get keyboard response
          POP     AX              ;Restore
          MOV     AH,00H          ;  original mode
          INT     10H             ;  (in AL)
          MOV     AX,4C00H        ;End of processing
          INT     21H
A10MAIN   ENDP

;                 Set graphics mode:
B10MODE   PROC    NEAR
          MOV     AH,00H          ;Request VGA graphics mode
          MOV     AL,12H          ;640 cols x 480 rows
          INT     10H
          MOV     AH,0BH          ;Request color palette
          MOV     BH,00           ;Background
          MOV     BL,07H          ;Gray
          INT     10H
          RET
B10MODE   ENDP

;                 Display rows of graphics dots:
C10DISP   PROC    NEAR
          MOV     BX,00           ;Set initial page,
          MOV     CX,64           ;  column,
          MOV     DX,70           ;  and row
C20:
          MOV     AH,0CH          ;Request pixel dot
          MOV     AL,BL           ;Color
          INT     10H             ;BX, CX, & DX are preserved
          INC     CX              ;Increment column
          CMP     CX,576          ;Column at 576?
          JNE     C20             ;  no, loop
          MOV     CX,64           ;  yes, reset column
          INC     BL              ;Change color
          INC     DX              ;Increment row
          CMP     DX,280          ;Row at 280?
          JNE     C20             ;  no, loop
          RET                     ;  yes, ended
C10DISP   ENDP

;                 Delay, wait for keyboard entry:
D10KEYB   PROC    NEAR
          MOV     AH,10H          ;Request keyboard
          INT     16H             ;  input
          RET
D10KEYB   ENDP
          END     A10MAIN
```

شکل ۷-۱۰ نمایش گرافیک رنگ

این برنامه برای هر سطر رنگ را افزایش می‌دهد (بنابراین بیت‌های 0000 خواهد شد 0001) و چون فقط ۴ بیت سمت راست استفاده می‌شود، رنگ‌ها پس از هر ۱۶ سطر تکرار خواهد شد. نمایش از ۶۴ ستون پس از سمت چپ آغاز و ۶۱ ستون به سمت راست مانده خاتمه می‌یابد.

در خاتمه، برنامه منتظر می‌شود تا کاربر یک کلید را بفشارد و سپس صفحه نمایش را با حالت اصلی دوباره تنظیم نماید می‌توانید این برنامه را برای حالت‌های گرافیکی مختلف تغییر دهید.

تعیین نوع تطبیق دهنده ویدئو

از آنجائیکه تطبیق دهنده‌های گرافیکی ویدئو سرویس‌های متفاوتی را پشتیبانی می‌سازند، زمانی ممکن است بخواهید نوع تطبیق دهنده نصب شده بر روی سیستم را بدانید. یک روش توصیه شده این است که ابتدا برای VGA، سپس برای EGA و سپس در انتها برای CGA و MDA بررسی کنید. در اینجا مراحل آورده شده است

۱) برای تعیین اینکه VGA نصب شده :

```
MOV   AH,1AH          ; درخواست تابع VGA
MOV   AL,0            ; زیر تابع 0
INT   10H             ; فراخوانی سرویس وقفه
CMP   AL,1AH          ; اگر AL حاوی 1AH در زمان بازگشت باشد
JE    VGAFOUND        ; سیستم حاوی VGA می‌باشد
```

۲) برای تعیین اینکه یک EGA نصب شده است :

```
MOV   AH,12H          ; درخواست تابع EGA
MOV   BL,10H          ; در حافظه EGA
INT   10H             ; فراخوانی سرویس وقفه
CMP   BL,10H          ; اگر BL حاوی 10H نیست
JNE   EGAFOUND        ; سیستم حاوی یک EGA می‌باشد
```

چون یک EGA ممکن است در MDA یا CGA نصب شده باشد، ممکن است تعیین کنید که آیا EGA فعال است. ناحیه داده BIOS در 40:0087 حاوی یک بایت دستور EGA است. بیت ۳ را بررسی کنید، 0 به معنی آن است که EGA فعال است و 1 به معنی غیر فعال بودن آن می‌باشد.

۳) برای تعیین اینکه آیا یک CGA یا MDA نصب شده است، کلمه موقعیت 40:0063 را بررسی کنید اگر حاوی آدرس پایه کنترل کننده حافظه است.

نکات کلیدی

- بایت صفت در حالت متن برای چشمک زن، نمایش معکوس و شدت بالا است. برای متن رنگی بیت‌های RGB توانایی انتخاب رنگ را دارند ولی حالت زیر خط دار را فراهم نمی‌سازند.
- BIOS INT 10H توابعی جهت پردازش کل صفحه نمایش را فراهم می‌سازد، مانند تنظیم حالت ویدئو، تنظیم موقعیت مکان‌نما، حرکت طوماری صفحه، خواندن از صفحه کلید و نوشتن کاراکترها.
- اگر برنامه شما خطوط را به سمت پایین صفحه نمایش می‌دهد، از INT 10H تابع 06H جهت حرکت طوماری به سمت بالا قبل از رسیدن نمایش به پایین صفحه استفاده نمایید.
- سرویس‌های INT 10H که یک کاراکتر را نمایش می‌دهند بطور خودکار مکان‌نما را تغییر نمی‌دهند.
- حافظه ۱۶K برای نمایش رنگی، ذخیره صفحات اضافی را ممکن می‌سازد. چهار صفحه ۸۰ ستونی وجود دارد.
- روش سریع‌تر برای نمایش کاراکترهای صفحه (متن یا گرافیک)، انتقال مستقیم آنها به ناحیه نمایش ویدئویی مناسب می‌باشد.
- یک پیکسل (عنصر تصویر) شامل یک تعداد مشخص بیت‌هاست که به تطبیق دهنده گرافیکی و درجه تنظیم آن (پایین، متوسط و یا بالا) بستگی دارد.
- برای حالت‌های گرافیکی 04 و 05 می‌توانید ۴ رنگ، یکی از ۱۶ رنگ در دسترس و سه رنگ دیگر از یک جعبه رنگ انتخاب نمایید.

پرسش‌ها

۱-۱۰. بایت صفات به فرم دودویی را برای مونیتور رنگی در موارد زیر بیابید: (الف) زرد روی قهوه‌ای ، (ب) فیروزه‌ای روشن روی صورتی ، (ج) سبز روی سیاه چشمک زن.

۲-۱۰. بایت صفحه به فرم دودویی را برای مونیتور تک رنگ در موارد زیر بیابید: (الف) فقط زیر خط‌دار ، (ب) سفید روی سیاه، شدت معمولی ، (ج) نمایش معکوس، با شدت.

۳-۱۰. روتین‌های زیر را کد نمایید : (الف) حالت را برای یک صفحه تک رنگ ۸۰ ستونی تنظیم نمایید، (ب)اندازه مکان‌نما را طوری تنظیم کنید که خط شروع ۴ و خط انتها ۱۰ باشد، (ج) صفحه ۱۴ خط به سمت بالا، حرکت طوماری داشته باشد، (د) ۲۰ نقطه چشمک زن با کاراکتر یک دوم نقطه روشن (B1H) نمایش دهد.

۴-۱۰ با حالت متن ۰۳، تعداد رنگ برای پس زمینه و پیش زمینه چه تعداد می‌باشد؟

۵-۱۰. دستوراتی را کد نمایید که ۶ کاراکتر (05H) را در حالت متنی با رنگ زرد بر روی آبی نمایش دهد.

۶-۱۰. کدام حالت اجازه استفاده از صفحات نمایش را می‌دهد؟

۷-۱۰. برنامه‌ای بنویسید که از INT 10H تابع 0AH برای پذیرش داده از صفحه کلید و تابع 09H جهت نمایش کاراکترها استفاده نماید. این برنامه صفحه را پاک خواهد کرد، رنگ‌های صفحه را تنظیم می‌نماید، مکان‌نما را در سطر ۸ ستون ۱۰ تنظیم می‌نماید و مجموعه‌ای از داده‌ها را از صفحه کلید در موقعیت جاری مکان‌نما می‌پذیرد. مجموعه داده باید ۴ تا ۵ خط باشد (هر خط حداکثر ۲۵ کاراکتر طول دارد)، که در انتهای هر خط <Enter> زده می‌شود و در فیلدهای سگمنت داده با نام‌های LINE1 ، LINE2 و... ذخیره خواهد شد شما می‌توانید از رنگ‌های مختلف ، نمایش معکوس، یا رنگ زدن استفاده نمایید. سپس مکان‌نما را در سطر ۱۸ ستون ۱۰ تنظیم نموده و داده‌های وارد شده را بطوریکه در یک خط قرار بگیرید نمایش دهید. این برنامه هر تعداد داده را می‌پذیرد تا هنگامی که کاربر بدون ورود داده Enter را بفشارد. برنامه را طوری بنویسید که یک روتین اصلی کوتاه با یک سری از زیر برنامه‌های فراخوانی شده داشته باشد.

۸-۱۰. برنامه پرسش ۷-۱۰ را بطوری بازنویسی نمایید که برای ورودی صفحه کلید از INT 10H و برای نمایش از INT 10H تابع 09H استفاده نمایید.

۹-۱۰. توضیح دهید که بایت صفت عمومی چگونه تعداد رنگ‌های در دسترس را محدود می‌نماید.

۱۰-۱۰. دستوراتی را بنویسید که حالت گرافیکی برای درجه تنظیم موارد زیر را برقرار نماید: (الف) ۲۰۰×۳۲۰، (ب) ۲۰۰×۶۴۰، (ج) ۴۸۰×۶۴۰.

۱۱-۱۰. دستوراتی بنویسید که رنگ سبز را برای پس زمینه در حالت گرافیکی انتخاب نماید.

۱۲-۱۰. دستوراتی بنویسید که در حالت گرافیکی نقطه‌ای را از سطر ۴۴، ستون ۱۲۰ بخواند.

۱۳-۱۰. برنامه شکل ۷-۱۰ را طوری بازنویسی کنید که موارد زیر را فراهم سازد: (الف) پس زمینه رنگ آبی، (ب) سطر شروع ۵۰ و خاتمه ۴۰۰ باشد، (ج) ستون شروع ۷۲ و خاتمه ۵۶۸ باشد.

فصل ۱۱

نکات پیشرفتهٔ پردازش صفحه کلید

هدف: همه عملیات صفحه کلید و نکات پیشرفته در روی صفحه کلید را در بردارد، شامل حالت شیفت، بافر صفحه کلید، و کدهای پیمایش.

مقدمه

این فصل بسیاری از عملیات مختلف دستکاری ورودی صفحه کلید را توصیف می‌کند که برخی موارد استفاده خاص دارند. از این عملیات INT 21H تابع 0AH (فصل ۹) و INT 16H (این فصل) می‌باشد که اغلب همه عملیات مورد نیاز صفحه کلید را فراهم می‌سازند.

از عناوین دیگری که این فصل شامل می‌شود بایت وضعیت شیفت صفحه کلید، کدهای پیمایش و ناحیه بافر صفحه کلید می‌باشد. بایت ***وضعیت شیفت*** که در ناحیه داده BIOS قرار دارد توانایی تشخیص را به یک برنامه می‌دهد، برای مثال، آیا کلیدهای <Ctrl> ، <Shift> یا <ALt> فشرده شده است. ***کد پیمایش***، یک عدد منحصر بفرد تعیین شده برای هر کلید از صفحه کلید می‌باشد که سیستم را توانا می‌سازد تا منبع کلید فشرده شده را تعیین کند و برنامه را توانا می‌سازد تا بررسی کند که آیا کاربر یک کلید تابعی گسترش یافته مانند <Home>، <Pageup>، یا <Arrow> را فشرده است و ***ناحیه بافر صفحه کلید*** فضایی در حافظه فراهم می‌سازد تا قبل از آن که برنامه واقعاً درخواست ورودی بنماید بتوانید تایپ نمایید.

عملیاتی که در این فصل معرفی می‌شوند موارد زیر می‌باشد:

توابع INT 21h		توابع INT 16H	
01H	ورودی صفحه کلید با انعکاس	00H	خواندن یک کاراکتر
06H	I/O کنسول مستقیم	01H	تعیین وجود یک کاراکتر
07H	ورودی صفحه کلیدمستقیم بدون‌انعکاس	02H	بازگشت وضعیت شیفت جاری
08H	ورودی صفحه کلید بدون انعکاس	05H	نوشتن صفحه کلید
0AH	بافر کردن ورودی صفحه کلید	10H	خواندن کاراکتر از صفحه کلید
0BH	چک کردن وضعیت صفحه کلید	11H	تعیین وجود یک کاراکتر
0CH	پاک کردن بافر و برداشتن تابع	12H	بازگشت وضعیت شیفت جاری

صفحه کلید

صفحه کلید سه نوع اصلی کلید دارد :

۱) ***کاراکترهای استاندارد***، که شامل حروف A تا Z، شماره‌های ۰ تا ۹ و برخی کاراکترها مانند %، $ و # .

۲) ***کلیدهای تابعی گسترش یافته*** که شامل:

- کلیدهای تابعی برنامه، مانند <F1> و <F1> + <Shift> .
- کلیدهای عددی با حالت NUM LOCK خاموش: <Home>، <End>، <Arrows>، <Del>، <Ins>، <Pageup> و <Pg Dn> و کلیدهایی که در صفحه کلیدهای توسعه یافته مانند همین کلیدها عمل می‌نمایند.
- <Alt> + حرف الفبا و کلیدهای تابعی برنامه + <Alt> .

۳) ***کلیدهای کنترلی*** <*Alt*> ، <*Ctrl*> و <*Shift*> که در ارتباط با بقیه کلیدها عمل می‌کنند. BIOS آنها را مانند کاراکترهای ASCII به برنامه شما تحویل نمی‌دهد. BIOS با آنها متفاوت با بقیه کلیدها رفتار می‌کند بدینوسیله که وضعیت جاری آنها را در بایت وضعیت شیفت در ناحیه داده BIOS با آنها متفاوت با بقیه کلیدها رفتار می‌کند بدینوسیله که وضعیت جاری آنها را در بایت وضعیت شیفت در ناحیه داده BIOS تغییر می‌دهد.

- PCهای اصلی با ۸۳ کلیدش زودتر آسیب می‌بیند چون برخی کلیدها مثلاً کلیدهای عددی دو عمل را انجام دهند. پس کلیدهای عددی که با این کلیدها مشترکند مانند <Home>، <End>، <Arrows>، <Del>، <Ins>، <PgUp> و <PgDn> با تعویض حالت <Numlock> عمل می‌نمایند. برای رفع این مشکل که کلیدها بر روی هم قرار می‌گیرند، طراحان یک صفحه کلید توسعه یافته با ۱۰۱ کلید و پس از آن ۱۰۴ کلید برای Windows تولید نمودند. از ۱۸ کلید جدید، فقط <F11> و <F12> تابع جدیدی فراهم نموده‌اند، مابقی کلیدها همان کار کلیدهای صفحه کلید اصلی را انجام می‌دهند. اگر برنامه شما امکان استفاده از کلیدهای <F11> و <F12> یا هر ترکیبی از کلیدهای جدید را فراهم می‌سازد، کاربر باید یک صفحه کلید توسعه یافته و یک کامپیوتر با BIOS که بتواند آن را پردازش نماید داشته باشد. برای اغلب عملیات دیگر صفحه کلیدها، نیازی به نوع خاصی از صفحه کلید ندارد.

وضعیت شیفت صفحه کلید

ناحیه داده BIOS در سگمنت 40[0]H در حافظه پایینی حاوی تعدادی عنصر داده مفید می‌باشد. آنها شامل اولین بایت از وضعیت شیفت جاری صفحه کلید در 40: 07H می‌باشد، که وقتی بیت‌ها یک باشد بر موارد زیر دلالت دارد.

BIT	ACTION	BIT	ACTION
7	Insert active	3	<Alt> pressed
6	CapsLock state active	2	<Ctrl> pressed
5	NumLock state active	1	<Left Shift> pressed
4	Scroll Lock state active	0	<Right Shift> pressed

با استفاده از INT 10H تابع 02H (بعداً گفته می‌شود). می‌توانید این مقدار را بررسی کنید. توجه کنید که فعال، یعنی کاربر هم اکنون کلید را فشرده است، رها ساختن کلید سبب می‌شود BIOS مقادیر بیت را پاک نماید. صفحه کلید ۸۳ کلیدی فقط از این بایت وضعیت شیفت استفاده می‌نمایند.

صفحه کلیدهای توسعه یافته دو کلید (راست و چپ) <Ctrl> و <Alt> دارند، بنابراین اطلاعات اضافی برای بررسی آنها نیاز است. بایت دوم از وضعیت صفحه کلید برای صفحه کلیدهای توسعه یافته در 40:18H قرار دارد، که بیت یک در آن به معنی موارد زیر است:

بیت‌های ۰، ۱ و ۲ با صفحه کلید توسعه یافته در ارتباط است. که می‌توانید بررسی کنید کدام یک از کلیدهای <Ctrl> یا <Alt> یا هر دو فشرده شده است.

یک بایت وضعیت صفحه کلید دیگر در 40:96H قرار دارد. عناصری که برای ما در آن اهمیت دارد بیت ۴ است که وقتی یک باشد یعنی یک صفحه کلید توسعه یافته نصب شده است.

BIT	ACTION	BIT	ACTION
7	Insert pressed	3	Ctrl/NumLock (pause) active
6	CapsLock pressed	2	SysReq pressed
5	NumLock pressed	1	Left Alt pressed
4	Scroll Lock pressed	0	Left Ctrl pressed

تمرین: بررسی وضعیت شیفت

برای دیدن تاثیر بر بایت وضعیت شیفت با فشردن کلیدهای <Ctrl>، <Alt>، <Shift>، DEBUG را اجرا نمائید. فرمان D 40:17 را برای دیدن محتویات بایت وضعیت واردِ کنید. کلیدهای <Caps Lock>، <Num Lock> و <Scroll lock> را بفشارید و فرمان 40:17 را مجدداً برای دیدن نتیجه بر روی بایت وضعیت وارد کنید. بایت 40:17H باید 70H و بایت 40:18H احتمالاً 00H خواهد بود. بایت 40:96H باید وجود (یا عدم وجود) یک صفحه کلید توسعه یافته را نشان دهد. سعی کنید که محتویات بایت وضعیت 40:17H را تغییر دهید - تایپ کنید: E 40:1700. اگر کلیدهای قفل کننده صفحه کلید شما روشن باشد، خاموش خواهد شد. حال با وارد کردن دستور E 40:17 70 آنها را روشن نمایید.

می‌توانید ترکیب‌های مختلفی ایجاد کنید، اگر چه تایپ کردن یک فرمان DEBUG معتبر، مشکل است آنهم وقتی کلیدهای <Ctrl> و <Alt> فشرده شده است. کلید Q را برای خروج از DEBUG بفشارید.

بافر صفحه کلید

یک عنصر سودمند در ناحیه داده BIOS در 40:1EH به نام ***بافر صفحه کلید*** است. قبل از آنکه یک برنامه درخواست ورودی از صفحه کلید را داشته باشد، این عنصر اجازه نوشتن تا ۱۵ کاراکتر را می‌دهد. وقتی شما یک کلید را می‌فشارید، پردازشگر صفحه کلید یک کد پیمایش کلید تولید می‌کند (یک عدد منحصر بفرد نسبت داده شده) و به طور خودکار BIOS INT 09H را درخواست می‌نماید.

به عبارت ساده، روتین INT 09H کد پیمایش را از صفحه کلید دریافت می‌کند، آن را به کاراکتر ASCII تبدیل و به ناحیه بافر صفحه کلید تحویل می‌دهد. سپس INT 16H (عملیات سطح پایین‌تر صفحه کلید) کاراکتر را از بافر می‌خواند و آن را به برنامه شما تحویل می‌دهد. برنامه شما هیچگاه نیازی به درخواست INT 09H ندارد، زیرا وقتی یک کلید را بفشارید، BIOS بطور خودکار آن را انجام می‌دهد. بخش بعدی INT 09H و بافر صفحه کلید را با جزئیاتش در بر دارد.

استفاده از INT 21H برای ورودی صفحه کلید

این بخش سرویس‌های INT 21H را در بر دارد که ورودی صفحه کلید را دستکاری می‌کند. برای این نوع از ورودی صفحه کلید، کد تابع مورد نیاز را در AH قرار دهید و INT 21H را فراخوانی نمایید. همه این عملیات بجز تابع AH فقط یک کاراکتر می‌پذیرد. برای دستکاری یک رشته کاراکتر، باید حلقه‌ای کد نمایید که در هر بار یک کاراکتر را بپذیرد، کلیدهای Enter و Baekspace را بررسی کنید، در صورت لزوم بر روی صفحه آن را منعکس سازد، و در نهایت مکان نما را پیش برد). این عملیات با تابع 3FH کنار گذاشته می‌شود (فصل ۹)، اما در اینجا برای تکمیل شدن آورده شده است. در بحث عملیاتی که گفته خواهد شد، عبارت پاسخ به درخواست <Break> و <Ctrl> یعنی اگر کاربر کلیدهای <Break> + <Ctrl> یا <C> + <Ctrl> را با هم بفشارد سیستم برنامه را تمام خواهد کرد.

INT 21H تابع 01H: ورودی صفحه کلید با انعکاس

این عملیات یک کاراکتر را از بافر صفحه کلید می‌پذیرد، اگر وجود نداشت، برای ورودی از صفحه کلید منتظر می‌ماند. این عملیات یکی از دو کد وضعیت، زیر را باز می‌گرداند.

- AL = یک مقدار غیر صفر یعنی یک کاراکتر ASCII وجود دارد (مانند یک حرف یا عدد). که عملیات آن را بر روی صفحه منعکس خواهد نمود.
- AL = صفر یعنی کاربر یک کلید تابعی توسعه یافته مانند <Home>، <F1>، <PgUP> را فشرده و AH هنوز حاوی تابع اصلی است. عملیاتی که توابع توسعه یافته را بدون تکنیک دستکاری می‌کنند سعی در انعکاس آنها بر روی صفحه دارند و می‌توان کد پیمایش کلید تابعی را از AL دریافت نمود و بلافاصله عملیات INT 21H را تکرار کرد. عملیات به درخواست <Break> و <Ctrl> پاسخ می‌دهد.

مثال زیر این تابع را مشخص می‌سازد :

```
MOV  AH,01H        ; درخواست ورودی صفحه کلید
INT  21H           ; فراخوانی سرویس وقفه
CMP  AL,00         ; آیا کلید تابعی توسعه یافته فشرده شده ؟
JNZ  ...           ; نه، کاراکتر ASCII
INT  21H           ; بله تکرار عملیات
...                ; برای کد پیمایش
```

INT 21H تابع 06H : I/Oکنسول مستقیم

این عملیات ناآشنا می‌تواند هر کاراکتر یا کد کنترلی را با هیچ نوع دخالتی از DOS منتقل نماید. دو نسخه وجود دارد، برای ورودی و برای خروجی برای ورودی 0FFH را در DL قرار دهید. اگر هیچ کاراکتر در بافر صفحه کلید وجود نداشته باشد، عملیات پرچم صفر را یک می‌کند و برای ورودی منتظر نمی‌شود. اگر یک کاراکتر در بافر منتظر باشد، عملیات کاراکتر را در AL ذخیره و پرچم صفر را پاک می‌کند. این عملیات کاراکتر را بر روی صفحه منعکس نمی‌کند و کلیدهای <Break> + <Ctrl> یا Prtsc + <Ctrl> را بررسی نمی‌کند. یک مقدار غیر صفر در AL به معنی کاراکتر ASCII استاندارد مانند حروف و اعداد است. صفر در AL به معنی آن است که کاربر یک کلید تابعی توسعه یافته مانند <Home>، <F1> یا <PgUp> را فشرده است. برای دریافت کد پیمایش بلافاصله عملیات INT 21H را تکرار نمایید. برای خروجی صفحه، کاراکتر ASCII را در DL (نه 0FFH) قرار دهید.

INT 21H تابع 07H: ورودی مستقیم صفحه کلید بدون انعکاس

این عملیات مشابه تابع 01H عمل می‌کند، بجز آنکه کاراکتر بر روی صفحه منعکس نخواهد شد و به <Ctrl> <Break> + نیز پاسخ نمی‌دهد. می‌توانید از این عملیات زمانی استفاده کنید که می‌خواهید یک کلمه رمز را که نباید دیده شود وارد کنید یا نمی‌خواهید صفحه نمایش را خراب کنید.

INT 21H تابع 08H: ورودی صفحه کلید بدون انعکاس

این عملیات مشابه تابع 01H عمل می‌کند، بجز آنکه کاراکتر بر روی صفحه نمایش منعکس نخواهد شد.

INT 21H تابع 0AH: ورودی صفحه کلید بافر شده

این عملیات صفحه کلید مفید، در فصل ۹ با جزئیات گفته شد.

INT 21H تابع 0BH : بررسی وضعیت صفحه کلید

این عملیات، اگر کاراکتر ورودی در بافر صفحه کلید در دسترس باشد در AL ، FFH باز می‌گرداند و اگر هیچ کاراکتری در دسترس نباشد 00H باز می‌گرداند. توجه کنید که این عملیات منتظر فشردن کلید از طرف کاربر نخواهد شد، بافر را بررسی خواهد کرد. این تابع با بقیه آنهایی که منتظر ورودی صفحه کلید نخواهند شد در ارتباط هستند.

INT 21H تابع 0CH : پاک کردن بافر صفحه کلید و برداشتن تابع

ممکن است از این عملیات در ارتباط با توابع 01H ، 006H ، 08H یا 0AH استفاده نمایید.

تابع مورد نیاز را در AL قرار دهید:

```
MOV   AH,0CH        ; درخواست ورودی صفحه کلید
MOV   AL,Function   ; تابع مورد نیاز
MOV   DX,KBAREA     ; ناحیه ورودی صفحه کلید
INT   21H           ; فراخوانی سرویس وقفه
```

این عملیات بافر صفحه کلید را پاک می‌کند، تابع AL را اجرا کرده و یک کاراکتر می‌پذیرد (منتظر آن می‌شود) بر طبق تابعی که در AL درخواست شده است. از این عملیات برای برنامه‌هایی می‌توانید استفاده کنید که به کاربر اجازه نمی‌دهند تا زودتر تایپ نمایند.

استفاده از INT 16H برای ورودی صفحه کلید

INT 16H، عملیات صفحه کلید BIOS پایه است که بطور وسیع و زیاد توسط توسعه دهندگان نرم‌افزار استفاده می‌شود، بر طبق کد تابعی در AH، سرویس‌های زیر را فراهم می‌سازد.

INT 16H تابع 00H: خواندن یک کاراکتر

این عملیات، فقط کلیدهای صفحه کلید ۸۳ تایی را دستکاری می‌نماید، و ورودی از کلیدهای اضافی بر روی صفحه کلیدهای گسترش یافته مانند <F11> و <F12> نمی‌پذیرد. تابع 10H را ببینید، که مشابه این یکی است و صفحه کلیدهای گسترش یافته را نیز دستکاری می‌نماید.

INT 16H تابع 01H: تعیین حضور یک کاراکتر

این عملیات کلیدهای صفحه کلید ۸۳ تایی را دستکاری می‌نماید، اما ورودی از کلیدهای اضافی در صفحه کلیدهای گسترش یافته را تشخیص نمی‌دهد. تابع 11H را ببینید، که مشابه این یکی است و همچنین صفحه کلیدهای گسترش یافته را نیز دستکاری می‌نماید.

INT 16H تابع 02H : بازگرداندن وضعیت شیفت جاری

این عملیات، وضعیت شیفت صفحه کلید را در ناحیه داده BIOS در موقعیت 417H (40:17H) در AL باز می‌گرداند. (بخش اخیر "وضعیت شیفت صفحه کلید" را ببینید). مثال زیر بررسی می‌کند آیا کلید <Left Shift> (بیت ۱) یا <Righ Shift> (بیت ۰) فشرده شده است :

```
MOV   AH,02H          ; درخواست ورودی صفحه کلید
INT   16H             ; فراخوانی سرویس وقفه
OR    AL,0000 0011B   ; شیفت راست یا شیفت چپ فشرده شده
JE    XX              ; بله . . .
```

تابع 12H را ببینید که وضعیت شیفت را در 418H برای صفحه کلیدهای گسترش یافته بررسی می‌کند.

INT 16H تابع 05H : نوشتن بر صفحه کلید

این عملیات به برنامه شما اجازه می‌دهد تا کاراکترها را در بافر صفحه کلید جایگزین نمایید مانند آنکه یک کاربر یک کلید را فشرده باشد. کاراکتر ASCII را در CH و کد پیمایش آن را در CL وارد کنید. عملیات به شما اجازه می‌دهد تا هنگامیکه بافر پر شود کاراکتر را وارد کنید.

INT 16H تابع 10H : خواندن یک کاراکتر صفحه کلید

این عملیات مانند تابع 00H است، بجز آنکه کلیدهای صفحه کلید گسترش یافته را نیز می‌پذیرد. این عملیات برای یک کاراکتر داده شده بافر صفحه کلید را بررسی می‌کند. اگر کاراکتری نباشد، عملیات منتظر می‌شود تا کاربر یک کلید را بفشارد. اگر کاراکتر باشد، عملیات آن را در AL و کد پیمایش آن را در AH ارائه می‌دهد. اگر کلید فشرده شده یک کلید تابعی توسعه یافته باشد مانند <Home> یا <F1>، کاراکتر AL، صفر خواهد بود. در صفحه کلیدهای گسترش یافته <F11> و <F12> ، 00H را در AL باز می‌گرداند اما کلیدهای کنترلی جدید دیگر (که قبلاً وجود داشته) مانند <Home> و <Pg Up> در AL ، 0E0H باز می‌گردانند. سه حالت ممکن چنین است.

AL	AH	کلید فشرده شده
کاراکتر ASCII	کد پیمایش	کاراکتر ASCII حاوی
00H	کد پیمایش	کلید تابعی توسعه یافته
E0H	کد پیمایش	کلید کنترلی توسعه یافته موجود

برای تعیین اینکه آیا کاربر یک کلید تابعی توسعه یافته را فشرده، می‌توانید AL را با 00H یا E0H مقایسه کنید.

```
MOV   AH,10H      ; درخواست ورودی صفحه کلید BIOS
INT   21H         ; فراخوانی سرویس وقفه
CMP   AL,00H      ; کلید تابعی توسعه یافته؟
JE    K10         ; بله، خروج
CMP   AL,0E0H     ; کلید تابعی توسعه یافته؟
JE    K10         ; بله، خروج
```

از آنجائیکه هیچ کاراکتری بر روی صفحه نمایش منعکس نمی‌شود، می‌توانید یک عملیات نمایش صفحه را برای این منظور انجام دهید.

INT 16H تابع 11H: تعیین وجود یک کاراکتر

این عملیات مشابه تابع 01H است، بجز آنکه کلید تابعی توسعه یافته اضافی از صفحه کلید توسعه یافته را می‌پذیرد. اگر کاراکتر داده شده در بافر صفحه کلید وجود داشته باشد، عملیات پرچم صفر را پاک می‌کند و کاراکتر را در AL و کد پیمایش را در AH ارائه می‌دهد، کاراکتر وارد شده در بافر باقی می‌ماند. اگر کاراکتر در بافر صفحه کلید وجود داشته باشد، عملیات پرچم صفر را تنظیم می‌کند و منتظر نمی‌شود. توجه کنید که عملیات، مورد خواندن از بافر را نیز داراست، زیرا کاراکتر در بافر صفحه کلید باقی می‌ماند تا تابع 10H آن را بخواند.

INT 16H تابع 12H : باز گرداندن وضعیت شیفت صفحه کلید جاری

این عملیات مشابه تابع 02H است که وضعیت شیفت صفحه کلید را از ناحیه داده BIOS در موقعیت (40:17H) 417H به AL ارائه می‌دهد. این عملیات همچنین وضعیت شیفت توسعه یافته را به AH ارائه می‌دهد، یک بیت به معنی موارد زیر است :

BIT	KEY	BIT	KEY
7	SysReq pressed	3	Right Alt pressed
6	Caps Lock pressed	2	Right Ctrl pressed
5	Num Lock pressed	1	Left Alt pressed
4	Scroll Lock pressed	0	Left Ctrl pressed

کلیدهای تابعی توسعه یافته و کدهای پیمایش

یک کلید تابعی توسعه یافته مانند <F1> یا <Home> یک عملی را درخواست می‌نماید تا یک کاراکتر ارائه دهد. چنین نیست که در طراحی سیستم این کلیدها برای انجام عمل خاصی در نظر گرفته شده باشند، برنامه نویس است که تعیین می‌کند، برای مثال وقتی کلید <Home> فشرده شد مکان‌نما در گوشه سمت چپ بالای صفحه باشد یا با فشردن <End> مکان نما در انتهای متن صفحه قرار گیرد. شما می‌توانید بسادگی این کلیدها را برای انجام عملیات غیر مرتبطی برنامه ریزی کنید. هر کلید یک **کد پیمایش** تعیین شده دارد که با 01 برای <ESC> آغاز می‌شود. (ضمیمه F را ببینید، لیست کاملی از این کدها را در بر دارد). توسط کد پیمایش در یک برنامه ممکن است تا منبع هر کلید فشرده شده را تعیین نماید. برای مثال، یک برنامه می‌تواند INT 16H تابع 10H برای درخواست ورودی یک کاراکتر صادر کند عملیات با یکی از دو روش پاسخ می‌دهد، بسته به آن که شما یک کلید کاراکتر را بفشارید یا یک کلید تابعی توسعه یافته. برای یک کاراکتر، مانند حرف A، عملیات دو عنصر را ارائه می‌دهد :

۱) در ثبات AH ، کد پیمایش حرف A ، 1EH .

۲) در ثبات AL ، کاراکتر ASCII ، حرف (41H) .

صفحه کلید شامل دو کلید برای کاراکترهایی مانند - ، + و * است. فشردن کلید ستاره کد 2AH کاراکتر را در AL و یکی از دو کد پیمایش را در AH تنظیم می‌کند، به کلیدی که فشرده شده بستگی دارد: برای ستاره‌ای که بالای رقم ۸ است 09H، یا برای ستاره با اعداد کناری صفحه کلید 29H می‌باشد. مثال زیر کد پیمایش را بررسی می‌کند تا تعیین کند کدام یک فشرده شده است:

```
CMP   AL,2AH          ستاره؟ ;
JNE   H30             نه، خروج ;
CMP   AH,09H  ; کد پیمایش روی کلید ۸؟ ;
JE    H40             بله، خروج ;
```

اگر شما یک کلید تابعی توسعه یافته مانند <Ins> را بفشارید، عملیات این دو عنصر را ارائه می‌دهد:

۱) در ثبات AH: کد پیمایش <Ins> , 52H

۲) در ثبات AL: صفر، یا E0H برای کلید کنترلی جدید بر روی صفحه کلیدهای پیشرفته.

بعد از یک عملیات INT 16H (و برخی عملیات INT 21H)، می‌توانید AL را بررسی کنید. اگر حاوی 00H یا E0H باشد، درخواست برای یک تابع توسعه یافته است، در غیر اینصورت، عملیات یک کاراکتر را ارائه می‌دهد. مثال زیر برای یک کلید تابعی توسعه یافته بررسی می‌کند:

```
MOV   AH,10H  ; درخواست ورودی صفحه کلید ;
INT   16H             فراخوانی سرویس وقفه ;
CMP   AL,00H          تابع توسعه یافته؟ ;
JE    K20             بله، خروج ;
CMP   AL,0E0H         تابع توسعه یافته؟ ;
JE    K20             بله، خروج ;
```

در مثال بعد، اگر یک کاربر کلید <Home> (با کد پیمایش 47H) را بفشارد، مکان نما در سطر 0 و ستون 0 تنظیم می‌شود :

```
        MOV   AH,10H    ; درخواست ورودی صفحه کلید
        INT   16H       ; فراخوانی سرویس وقفه
        CMP   AL,00H    ; تابع توسعه یافته؟
        JE    K30       ; بله، بگذر
        CMP   AL,0E0H   ; تابع توسعه یافته؟
        JNE   K90       ; نه، خروج
K30:    CMP   AH,47H    ; کد پیمایش برای (Home)؟
        JNE   K90       ; نه ، خروج
        MOV   AH,02H    ; درخواست
        MOV   BH,00     ; تنظیم مکان نما
        MOV   DX,00     ; در 00:00
        INT   10H       ; فراخوانی سرویس وقفه
```

کلیدهای تابعی <F1> تا <F10> کد پیمایش از 3BH تا 44H را تولید می‌کنند و <F11> و <F12>، 85H و 86H را تولید می‌کنند. مثال زیر کلید تابعی <F10> را بررسی می‌کند :

```
        CMP   AH,44H    ; کلید تابعی <F10>؟
        JE    H50       ; بله خروج ؟
```

در H50، برنامه می‌تواند هر فعالیت مورد نیاز را انجام دهد.

تمرین صفحه کلید

تمرین DEBUG زیر، تاثیر کلید زنی با کاراکترهای مختلف را تجربه می‌کند. از فرمان A 100 برای وارد کردن دستورات زیر استفاده کنید :

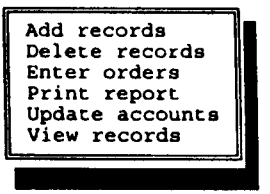
```
MOV   AH,10 (یا 00 برای یک صفحه کلید ۸۳ تایی)
INT   16
JMP   100
```

از فرمان P برای اجرای عملیات INT استفاده نمایید. کاراکترهای معمولی مختلف را با <Shift> و <Ctrl> بفشارید و نتیجه AH (کد پیمایش) و AL (کاراکتر) را با لیست ضمیمه F مقایسه نمایید

برنامه: انتخاب از یک منو

برنامه بعد یک منو با سایه پایین را همانطور که در فصل ۱۰ توضیح داده شد و در شکل ۱-۱۱ نشان داده شده است نمایش می‌دهد. این منو در سگمنت داده تعریف شده و با یک کادر دو خطی احاطه شده است.

یک کاربر با فشردن <Up Arrow> یا <Down Arrow> و <Enter> یک عنصر از منو را انتخاب می‌نماید.



شکل ۱-۱۱ منو با سایه پایین

برنامه در شکل ۲-۱۱ لیست شده است. روال‌ها و آنچه که انجام می‌دهند و در زیر توضیح داده شده است :

- A10MAIN، روال Q10CLEAR را برای پاک کردن صفحه فراخوانی می‌کند، B10MENU را برای نمایش منو فراخوانی می‌کند و اولین عنصر را با نمایش معکوس تنظیم نموده و C10INPT را پذیرش ورودی صفحه کلید فراخوانی می‌کند.
- B10MENU مجموعه کامل انتخاب منو را نمایش می‌دهد. ابتدا از INT 10H تابع 09H برای نمایش سایه استفاده می‌کند، ۸ خط از ۱۹ کاراکتر سایه (0DBH). سپس روال، منوی تعریف شده در سگمنت داده با نام MENU را در بالای سایه با تفاوت مکان ۱ سطر و ستون نمایش می‌دهد.

```
TITLE     A11SELMU (EXE) Select item from menu
; ------------------------------------------------------
          .MODEL SMALL
          .STACK 64
; ------------------------------------------------------
          .DATA
TOPROW    EQU    00              ;Top row of menu
BOTROW    EQU    07              ;Bottom row of menu
LEFCOL    EQU    16              ;Left column of menu
COL       DB     00              ;Screen column
ROW       DB     00              ;Screen row
COUNT     DB     ?               ;Characters per line
ATTRIB    DB     ?               ;Screen attribute
NINTEEN   DB     19              ;Width of menu
MENU      DB     0C9H, 17 DUP(0CDH), 0BBH
          DB     0BAH, ' Add records     ', 0BAH
          DB     0BAH, ' Delete records  ', 0BAH
          DB     0BAH, ' Enter orders    ', 0BAH
          DB     0BAH, ' Print report    ', 0BAH
          DB     0BAH, ' Update accounts ', 0BAH
          DB     0BAH, ' View records    ', 0BAH
          DB     0C8H, 17 DUP(0CDH), 0BCH
PROMPT    DB     09, 'To select an item, use <Up/Down Arrow>'
          DB     ' and press <Enter>.'
          DB     13, 10, 09, 'Press <Esc> to exit.'
; ------------------------------------------------------
          CODE
.386
A10MAIN   PROC   FAR
          MOV    AX,@data        ;Initialize segment
          MOV    DS,AX           ;  registers
          MOV    ES,AX
          CALL   Q10CLEAR        ;Clear screen
          MOV    ROW,BOTROW+2    ;Set row
          MOV    COL,00          ;  and column
          CALL   Q20CURS         ;  for cursor
          MOV    AH,40H          ;Request display
          MOV    BX,01           ;Handle for screen
          MOV    CX,81           ;Number of characters
          LEA    DX,PROMPT       ;Address of prompt
          INT    21H             ;Call interrupt service
A10LOOP:
          CALL   B10MENU         ;Display menu
          MOV    COL,LEFCOL+1
          CALL   Q20CURS         ;Set cursor
          MOV    ROW,TOPROW+1    ;Set row to top item
          MOV    ATTRIB,16H      ;Set reverse video
          CALL   D10DISP         ;Highlight current menu line
          CALL   C10INPUT        ;Provide for menu selection
          CMP    AL,0DH          ;Enter key pressed?
          JE     A10LOOP         ;  yes, continue
          MOV    AX,0600H        ;Esc pressed (indicates end)
          CALL   Q10CLEAR        ;Clear screen
          MOV    AX,4C00H        ;End of processing
          INT    21H
A10MAIN   ENDP
```

شکل الف ۲-۱۱-انتخاب یک عنصر از یک منو

```
;                   Display full menu:
;                   ------------------
B10MENU     PROC    NEAR
            MOV     ROW,TOPROW+1        ;Set top row of shadow
B20:
            MOV     COL,LEFCOL+1        ;Set left column of shadow
            CALL    Q20CURS             ;Set cursor next column
            MOV     AH,09H              ;Request display
            MOV     AL,0DBH             ;Shadow character
            MOV     BH,00               ;Page 0
            MOV     BL,60H              ;Black on brown
            MOV     CX,19               ;19 characters
            INT     10H
            INC     ROW                 ;Next row
            CMP     ROW,BOTROW+2        ;All rows displayed?
            JNE     B20                 ;No, repeat
            MOV     ROW,TOPROW          ;Set top row of menu
            LEA     SI,MENU             ;Address of menu
            MOV     ATTRIB,71H          ;Blue on white
B30:
            MOV     COL,LEFCOL          ;Set left column of menu
            MOV     COUNT,19            ;No. of cols to display
B40:
            CALL    Q20CURS             ;Set cursor next column
            MOV     AH,09H              ;Request display
            MOV     AL,[SI]             ;Get character from menu
            MOV     BH,00               ;Page 0
            MOV     BL,71H              ;Blue on white
            MOV     CX,01               ;One character
            INT     10H
            INC     COL                 ;Set for next column,
            INC     SI                  ;  next character
            DEC     COUNT               ;Last character?
            JNZ     B40                 ;No, repeat
            INC     ROW                 ;Next row
            CMP     ROW,BOTROW+1        ;All rows displayed?
            JNE     B30                 ;  No, repeat
            RET                         ;  Yes, return
B10MENU     ENDP
;                    Accept input for request:
;                    -------------------------
C10INPUT    PROC    NEAR
            MOV     AH,10H              ;Request keyboard
            INT     16H                 ;  input
            CMP     AH,50H              ;Down arrow?
            JE      C20
            CMP     AH,48H              ;Up arrow?
            JE      C30
            CMP     AL,0DH              ;Enter key?
            JE      C90
            CMP     AL,1BH              ;Escape key?
            JE      C90
            JMP     C10INPUT            ;None, retry
C20:        MOV     ATTRIB,71H          ;Blue on white
            CALL    D10DISP             ;Set old line to normal video
            INC     ROW                 ;Increment for next row
            CMP     ROW,BOTROW-1        ;Past bottom row?
            JBE     C40                 ;  no, ok
            MOV     ROW,TOPROW+1        ;  yes, reset
            JMP     C40
```

شکل ب ۲-۱۱ انتخاب یک عنصر از یک منو

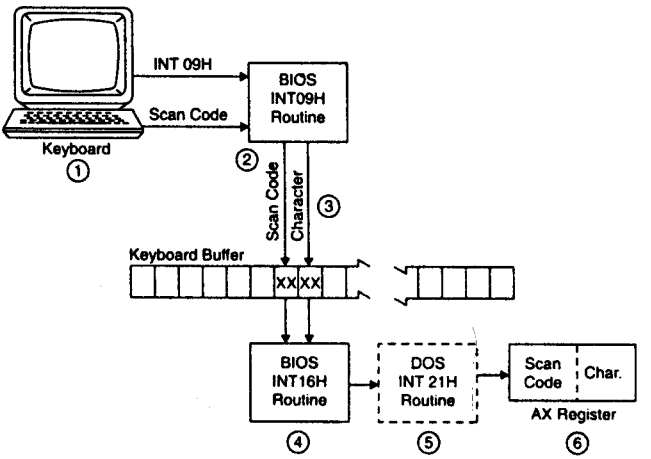
```
C30:        MOV     ATTRIB,71H          ;Blue on white
            CALL    D10DISP             ;Set old line to normal video
            DEC     ROW
            CMP     ROW,TOPROW+1        ;Below top row?
            JAE     C40                 ;  no, ok
            MOV     ROW,BOTROW-1        ;  yes, reset
C40:        CALL    Q20CURS             ;Set cursor
            MOV     ATTRIB,17H          ;White on blue
            CALL    D10DISP             ;Set new line to reverse video
            JMP     C10INPUT
C90:        RET
C10INPUT    ENDP
;                   Set menu line to normal/highlight:
;                   ----------------------------------
D10DISP     PROC    NEAR
            MOVZX   AX,ROW              ;Row tells which line to set
            MUL     NINTEEN             ;Multiply by length of line
            LEA     SI,MENU+1           ;  for selected menu line
            ADD     SI,AX
            MOV     COUNT,17            ;Characters to display
D20:
            CALL    Q20CURS             ;Set cursor next column
            MOV     AH,09H              ;Request display
            MOV     AL,[SI]             ;Get character from menu
            MOV     BH,00               ;Page 0
            MOV     BL,ATTRIB           ;New attribute
            MOV     CX,01               ;One character
            INT     10H
            INC     COL                 ;Next column
            INC     SI                  ;Set for next character
            DEC     COUNT               ;Last character?
            JNZ     D20                 ;  no, repeat
            MOV     COL,LEFCOL+1        ;Reset column to left
            CALL    Q20CURS             ;Set cursor
            RET
D10DISP     ENDP
;                   Clear screen:
;                   -------------
Q10CLEAR    PROC    NEAR
            MOV     AX,0600H
            MOV     BH,61H              ;Blue on brown
            MOV     CX,0000             ;Full screen
            MOV     DX,184FH
            INT     10H                 ;Call BIOS
            RET
Q10CLEAR    ENDP

;                   Set cursor row:column:
;                   ----------------------
Q20CURS     PROC    NEAR
            MOV     AH,02H
            MOV     BH,00               ;Page 0
            MOV     DH,ROW              ;Row
            MOV     DL,COL              ;Column
            INT     10H
            RET
Q20CURS     ENDP
            END     A10MAIN
```

شکل پ ۲-۱۱ انتخاب یک عنصر از یک منو

- C10INPT از INT 16H برای ورودی استفاده می‌کند: <Down Arrow> برای حرکت به سمت پایین در منو <Up Arrow> برای حرکت به سمت بالا در منو، <Enter> برای پذیرش یک عنصر منو و <ESC> برای اینکه از همه ورودیهای صفحه کلید صرفنظر شود. روتین مکان‌نما را در منو حرکت می‌دهد، بنابراین سعی در حرکت

مکان نما به سمت بالای اولین منو، مکان نما را در آخرین
خط تنظیم می‌کند و برعکس روتین برای تنظیم مجدد خط منوی قبلی به حالت معمولی ویدئو و خط منوی (انتخابی) جدید به نمایش معکوس، D10DISP را فراخوانی می‌نماید.

- D10DISP خط انتخابی‌جاری را برطبق یک صفت (معمول یانمایش‌معکوس)که فراهم خواهدشد، نمایش‌می‌دهد.
- Q10CLEAR صفحه ورودی را پاک می‌کند و آن را با پس زمینه قهوه‌ای و پیش زمینه آبی تنظیم می‌نماید برنامه انتخاب منو را با یک روش ساده مشخص می‌سازد، یک برنامه کامل برای هر عنصر انتخابی یک روتین اجرا می‌نماید. برای آنکه این برنامه را بهتر درک نمایید، آن را در قالب یک فایل ASM. تایپ نموده، اسمبل کنید و آن را بررسی نمایید.



۱) صفحه کلید وقفهٔ INT 09H را ایجاد می‌کند.
۲) وقفه INT 09H کد دریافتی را بررسی کرده و کاراکتر مربوط به آن را مشخص می‌کند.
۳) وقفه INT 09H کاراکتر و کد مربوطه را به بافر صفحه کلی ارسال می‌کند.
۵ و ۴)برنامه به صورت مستقیم یا از طریق INT 21H وقفه 16H را درخواست می‌کند.
۶) وقفهٔ 16H به بافر دسترسی پیدا کرده و کاراکتر را به AL و کد مربوط به آن را به AH ارسال می‌کند.

شکل ۳-۱۱ بافر صفحه کلید

BIOS INT 09H و بافر صفحه کلید

وقتی یک کلید را بفشارید، پردازشگر صفحه کلید، کد پیمایش کلید را تولید می‌کند و INT 09H را درخواست می‌نماید. این وقفه (در موقعیت ۳۶ از جدول سرویس وقفه) به روتین دستکاری وقفه در ROM BIOS اشاره می‌کند. روتین درخواستی برای ورودی، از درگاه ۹۶ (60H) صادر می‌کند. IN AL,60H روتین BIOS کد پیمایش را می‌خواند و آن را با ورودی در جدول کد پیمایش به منظور کاراکتر ASCII مرتبط (اگر وجود داشته باشد) مقایسه می‌کند. روتین کد پیمایش را با کاراکتر ASCII مرتبط ترکیب می‌کند و آن را در ۲ بایت، بافر صفحه کلید ارائه می‌دهد. شکل ۳-۱۱ این روال را مشخص می‌سازد.

توجه کنید که INT 09H بایت وضعیت صفحه کلید در 40:17H ، 40:18H و 40:96H برای <Shift>، <Alt> و <Ctrl> دستکاری می‌نماید. اگر چه فشردن این کلیدها INT 09H تولید می‌کند، روتین وقفه بیت‌های مناسب در بایت وضعیت را تنظیم می‌کند، اما هیچ کاراکتری به بافر صفحه کلید ارائه نمی‌دهد. همچنین INT 09H ترکیبات کلیدهای فشرده تعریف نشده را، در نظر نمی‌گیرد.

وقتی یک کلید را ***بفشارید***، پردازشگر صفحه کلید بطور خودکار یک کد پیمایش تولید می‌کند و INT 09H را صادر می‌نماید. وقتی کلید را در یک دوم ثانیه ***رها نمایید***، یک کد پیمایش اضافی تولید می‌کند و (مقدار اولین کد بعلاوه 100 00000B که بیت سمت چپ را تنظیم می‌کند) و یک INT 09H دیگر صادر می‌نماید. دومین کد پیمایش به روتین وقفه می‌گوید کلید را رها نموده‌اید. اگر کلید را بیشتر از یک دوم ثانیه نگهدارید، پردازش صفحه کلید دوباره تکرار خواهد شد و عملیات کلید بطور خودکار انجام می‌شود.

بافر صفحه کلید یک آدرس نیاز دارد (در بالای بافر) تا به INT 16H بفهماند از کجا کاراکتر بعدی را بخواهد و آدرس دیگری (در ادامه بافر) که کاراکتر بعدی را INT 09H در آن ذخیره نماید. این دو آدرس افست‌هایی در داخل سگمنت 40[0]H هستند. موارد زیر محتویات بافر را توصیف می‌کند:

آدرس	توضیح
41AH	آدرس بالای جاری بافر ، موقعیت بعدی که INT 16H باید از آن یک کاراکتر بخواند.
41CH	آدرس ادامه جاری بافر، موقعیت بعدی که INT 09H باید در آن یک کاراکتر بنویسد.
41EH	آدرس شروع خود بافر صفحه کلید، این بافر حاوی ۱۶ کلمه (۳۲ بایت) است، اگر چه می‌تواند طولانی‌تر باشد، و کاراکترهای صفحه کلید و کدهای پیمایش آنها را وقتی وارد می‌شود نگه دارد. سپس، INT 16H هر کاراکتر و کد پیمایش آن را می‌خواند و آنها را به برنامه ارائه می‌دهد.

برای هر کاراکتر و کد پیمایش آن ۲ بایت مورد نیاز است.

وقتی شما یک کاراکتر را تایپ کنید، INT 09H ادامه را پیش می‌برد. وقتی INT 16H یک کاراکتر بخواند، بالای بافر را پیش می‌برد. با این روش، پردازش دایره‌وار خواهد بود، که بالای بافر مدام بدنبال ادامه بافر خواهد بود.

وقتی بافر ***خالی*** باشد (INT 16H) همه کاراکترهای ذخیره شده را خوانده باشد، بالای بافر و ادامه آن یک آدرس خواهد بود.در مثال زیر ، کاربر کاراکترهای <Enter> abc را از قبل تایپ نموده است INT 09H کاراکترها را به فرم زیر ذخیره می‌سازد.

- a در 41EH و کد پیمایش 1EH در 41FH در بافر خواهد بود.
- b در 420H و کد پیمایش 30H در 421H در بافر خواهد بود.
- c در 422H و کد پیمایش 2EH در 423H در بافر خواهد بود.
- <Enter> در 424H و کد پیمایش E0H در 425H در بافر خواهد بود.

در اینجا INT 09H ادامه را به 426H پیش می‌برد.

a	1EH	b	30H	C	2EH	<Etr>	EOH	. . .
41E	41F	420	421	422	423	424	425	429

برنامه که INT 16H را ۵ مرتبه صادر نمود، همه کاراکترها را می‌خواند و تا ادامه در 426H پیش می‌رود. وقتی کاربر ۱۵ کاراکتر را وارد کند بافر **پر** خواهد بود و ادامه بلافاصله بعد از بالای بافر خواهد بود. برای دیدن این موضوع، فرض کنید کاربر از قبل تایپ کرده است "fghijkhlmnopqrs <Enter>". INT 09H کاراکترها را در ادامه در 426H ذخیره می‌کند و می‌چرخد تا <Enter> را در 422H ذخیره سازد. ادامه، در 424H بلافاصله قبل از بالای بافر در 426H خواهد بود.

r	s	<Ent>	*	f	g	h	i	j	k	l	m	n	o	p	q
41E	420	422	424	426	428	42A	42C	42E	430	432	434	436	438	43A	43C

در اینجا INT 09H کاراکترهای بیشتری نمی‌پذیرد، در هر بار فقط ۱۵ تا می‌پذیرد، اگر چه بافر ۱۶ مکان نگه می‌دارد. (می‌توانید بگویید چرا؟ اگر INT 09H یک کاراکتر دیگر بپذیرد، آدرس ادامه با آدرس بالای بافر یکی خواهد شد و INT 16H اشتباهاً فرض می‌کند که بافر خالی است.)

```
TITLE     A11KBSTA (COM) Testing Alt, Shift, & Ctrl
BIODATA   SEGMENT AT 40H          ;Locate BIOS data area
          ORG  17H                ;  and
KBSTATE   DB   ?                  ;  status byte
BIODATA   ENDS

CODESG    SEGMENT PARA
          ASSUME  CS:CODESG,DS:BIODATA
          ORG  100H
BEGIN:    JMP  A10MAIN
ALTKEY    DB   'Alt key pressed', '$'
CTRLKEY   DB   'Ctrl key pressed', '$'
SHIFKEY   DB   'Shift key pressed', '$'

A10MAIN   PROC
          MOV  AX,BIODATA         ;Initialize seg. address
          MOV  ES,AX              ;  of BIODATA in ES
A30:
          MOV  AH,10H             ;Request keyboard entry
          INT  16H
          CMP  AL,0DH             ;User requests end?
          JE   A90                ;  yes, exit
          CALL B10TEST
          JMP  A30                ;Repeat
A90:
          MOV  AX,4C00H           ;End processing
          INT  21H
A10MAIN   ENDP

B10TEST   PROC
          MOV  BL,ES:KBSTATE      ;Get keyboard status byte
          TEST BL,00000011B       ;Either Shift pressed?
          JZ   B20                ;  no, bypass
          LEA  DX,SHIFKEY
          CALL C10DISP
B20:
          TEST BL,00000100B       ;Ctrl key pressed?
          JZ   B30                ;  no, bypass
          LEA  DX,CTRLKEY
          CALL C10DISP
B30:
          TEST BL,00001000B       ;Alt key pressed?
          JZ   B90                ;  no, bypass
          LEA  DX,ALTKEY
          CALL C10DISP
B90:
          RET
B10TEST   ENDP

C10DISP   PROC
          MOV  AH,09H             ;Request display
          INT  21H
          RET
C10DISP   ENDP
CODESG    ENDS
          END  BEGIN
```

شکل ۴-۱۱ بررسی بایت وضعیت صفحه کلید

کلیدهای Shift ، Ctrl و Alt

- INT 09H بایت وضعیت صفحه کلید را در 40: 17H در ناحیه داده BIOS و نیز در 40:18H و 40:69H برای صفحه کلید گسترش یافته، دستکاری می‌نماید. وقتی شما <Shift>، <Ctrl> یا <Alt> را بفشارید، روتین BIOS بیت مناسب را با یک تنظیم می‌کند، و وقتی کلید را رها نمایید، آن بیت را با صفر پاک می‌کند. برنامه شما امکان بررسی، هر یک از کلیدهای کنترلی فشرده شده را توسط INT 16H تابع 02H یا با ارجاع مستقیم به بایت وضعیت دارد.

- برنامه COM. در شکل ۴-۱۱ استفاده از ارجاع مستقیم به بایت وضعیت در 40: 17H را مشخص می‌سازد. برنامه از SEGMENT AT برای تعریف ناحیه داده BIOS در حقیقت، یک سگمنت فرضی استفاده می‌کند. KBSTATE موقعیت بایت وضعیت صفحه کلید را در 40:17H مشخص می‌سازد. سگمنت کد حاوی روال‌های زیر می‌باشد.
- A10MAIN آدرس BIOSDATA در ES را مقدار دهی می‌کند (زیرا DS حاوی آدرس سگمنت داده‌های تعریف شده است). منتظر می‌شود تا کاربر یک کلید را بفشارد و اگر بغیر از <Enter> باشد، B10TEST فراخوانی می‌شود. فشردن <Enter> به برنامه خاتمه را اعلام می‌کند.
- B10TEST بایت وضعیت صفحه کلید را به BL می‌فرستد. MOV BL,ES : KBTATE

عملیات بازنویسی سگمنت ES به اسمبلر می‌گوید که آدرس افست از KBSTATE را با آدرس سگمنت در ES مرتبط سازد. اگر یکی از کلیدهای <Shift>، <Ctrl> یا <Alt> فشرده شده باشد روال C10DISP را فراخوانی می‌کند. توجه کنید که فشردن یک کلید کنترلی به تنهایی INT 16H را آگاه نمی‌کند، شما باید آن را مشترکاً با کلید دیگری بفشارید تا یک ورودی صفحه کلید معتبر ایجاد شود، مانند <A> + <Shift>، <F1> + <Ctrl> و غیره (ضمیمه F را ببینید). بایت وضعیت صفحه کلید عمل کلید کنترلی را منعکس می‌کند.

- C10DISP پیغام مناسب برای فشردن کلید کنترلی فشرده شده را نمایش می‌دهد. برای بررسی این برنامه با بایت وضعیت صفحه کلید پیشرفته در 40:18H و 40:96H می‌توانید آن را تغییر دهید.

وارد کردن مجموعه کامل کاراکترهای ASCII

مجموعه ASCII شامل ۲۵۶ کاراکتر از شماره ۰ تا ۲۵۵ (FFH) می‌باشد. خیلی از این مجموعه، کاراکترهای قابل نمایش استاندارد هستند، از ۲۰H (فاصله) تا ASCII 7EH (کاراکتر سه). بدلیل محدودیت صفحه کلید، خیلی از ۲۵۶ کاراکتر ASCII قابل بیان با آن نیستند. اما می‌توانید هر یک از کدهای ASCII از ۰۱ تا ۲۵۵ را، با نگهداشتن <Alt> و وارد کردن مقدار دهدهی آن با اعداد سمت راست صفحه کلید، ایجاد نمایید. سیستم مقدار داده شده شما را با دو بایت در بافر صفحه کلید نگه می‌دارد. اولین، کاراکتر ASCII و دومین بایت صفر است. برای مثال، 001+ <Alt>، 01H را ارائه می‌دهد و ۲۵۵ + <Alt> ، FFH را ارائه می‌دهد. با استفاده از INT 16H در فرمان A از DEBUG می‌توانید تاثیر وارد کردن مقادیر مختلف را بررسی کنید.

```
100   MOV   AH,10
102   INT   16
104   JMP   100
```

ضمیمه B را ببینید، جدول کاملی از مقادیر ASCII می‌باشد.

نکات کلیدی

بایت وضعیت شیفت در ناحیه داده BIOS در 40:17H و 40:18H بر وضعیت جاری کلیدهایی مانند <ctrl>، <Alt>، <Shift>، <Capslock> و <Numlock> <scrollLock> دلالت دارد.

- INT 21H عملیات صفحه کلید را فراهم می‌سازد که سرویس‌های مختلفی برای انعکاس یا عدم انعکاس کاراکتر بر صفحه نمایش، تشخیص یا رد <Break> + <Ctrl> و پذیرش کدهای پیمایش دارد.
- INT 16H عملیات صفحه کلید BIOS پایه برای پذیرش کاراکترها از بافر صفحه کلید را، فراهم می‌سازد. برای یک کلید کاراکتر، عملیات کاراکتر را در AL و کد پیمایش آن را در AH ارائه می‌دهد. برای یک کلید تابعی توسعه یافته، عملیات 00H یا E0H را به AL و کد پیمایش آن کلید را به AH ارائه می‌دهد.
- کد پیمایش یک عدد منحصر بفرد است که برای هر کلید تعیین شده است و سیستم را توانا می‌سازد تا منبع کلید فشرده شده را مشخص سازد و برنامه می‌تواند برای کلیدهای تابعی توسعه یافته مانند <Home>، <Pgup> و <Arrow> بررسی نماید.
- ناحیه داده BIOS در 40:1EH حاوی بافر صفحه کلید است که مجوز ورود داده تا ۱۵ کاراکتر را قبل از درخواست ورودی برنامه، می‌دهد.

- وقتی یک کلید را بفشارید، پردازشگر صفحه کلید کد پیمایش را تولید و INT 09H را درخواست می‌کند. وقتی کلید را رها نمایید ، یک کد پیمایش دومی تولید می‌کند (اولین کد بعلاوه 10000000B، که بیت سمت چپ را تنظیم نماید) تا به INT 09H بفهماند که کلید رها شده است.
- BIOS INT 09H کد پیمایش صفحه کلید را اصلاح می‌کند. عملیات با استفاده از کد پیمایش یک کاراکتر ASCII مرتبط با آن تولید می‌کند، که به ناحیه بافر صفحه کلید ارائه می‌دهد. BIOS همچنین وضعیت را برای <Ctrl> ، <Alt> یا <Shift> تنظیم می‌نماید.

پرسش‌ها

۱۱-۱. (الف) موقعیت اولین بایت وضعیت شیفت صفحه کلید در ناحیه داده BIOS چیست؟ (ب) منظور از محتویات 00000010 چیست؟ (ج) منظور از محتویات 00001100 چیست ؟

۱۱-۲. جزئیات توابع زیر برای ورودی صفحه کلید INT 21H را توضیح دهید. (الف) 01H، (ب) 07H، (ج) 08H، (د) 0AH.

۱۱-۳. تفاوت بین توابع 00H، 10H و 11H از INT 16H را توضیح دهید.

۱۱-۴. کد پیمایش برای توابع صفحه کلید زیر چیست؟ (الف) Home، (ب) PgDn، (ج) Down Arrow، (د) کلید تابعی F8.

۱۱-۵. از DEBUG برای بررسی تاثیر کلید فشرده شده استفاده نمایید. برای ورود جملات اسمبلی تایپ کنید 100 A و دستورات زیر را وارد نمایید :

```
MOV   AH,10 (or Ah,00)
INT   16
JMP   100
```

از فرمان 104 ، 100 U برای unassemble کردن برنامه و از فرمان P برای اجرای INT در DEBUG استفاده نمایید. اجرا متوقف شده و منتظر ورودی شما خواهد بود. هر کلیدی را بفشارید و ثبات AL و AH را بررسی کنید. کلیدهای مختلفی را امتحان کنید و برای خروج Q را بفشارید.

۱۱-۶. دستوراتی برای INT 16H بنویسید که کلید را بپذیرد، اگر <PgUp> بود مکان‌نما را در سطر و ستون صفر تنظیم نماید.

۱۱-۷. برنامه شکل ۲-۱۱ را برای فراهم ساختن موارد زیر بازنویسی نمایید: (الف) سایه را از سایه کامل به نقطه سه چهارم با (B2H) تغییر دهید. (ب) بعد از پاک کردن اولیه صفحه نمایش، اعلانی نشان دهد تا از کاربر بخواهد تا برای صفحه منو کلید <F1> را بفشارد. (ج) وقتی <F1> فشرده شد، منو نمایش داده شود. (د) به کاربر اجازه دهد تا بتواند با فشردن اولین کاراکتر (حرف بزرگ یا کوچک) هر عنصر نیز آن را انتخاب نماید. (ه) پس از درخواست یک عنصر، پیغامی برای آن انتخاب خاص نمایش دهد، مانند 'Procedure to Delete Records' (و) یک خط انتهایی به منو اضافه کند که حاوی یک عنصر 'Exit from program' باشد تا کاربر برای خاتمه آن را انتخاب نماید. باید روال B10MENU را برای دستکاری نمایش در سطر دیگر بازنویسی کنید.

۱۱-۸. تحت چه شرایطی INT 09H رخ می‌دهد؟

۱۱-۹. با عبارت ساده توضیح دهید چطور INT 09H، <Alt> و <Ctrl> را متفاوت با روش کلیدهای صفحه کلید استاندارد دستکاری می‌کند.

۱۱-۱۰. (الف) موقعیت بافر صفحه کلید در حافظهٔ BIOS کجاست؟ (ب) اندازه بافر، چند بایت است؟ (ج) چه تعداد کاراکتر صفحه کلید می‌تواند شامل باشد؟

۱۱-۱۱. تاثیر رخدادهای زیر در بافر صفحه کلید چیست؟ (الف) آدرس بالای بافر و ادامه آن یکی باشد، (ب) آدرس ادامه بافر بلافاصله بعد از بالای آن باشد.

۱۱-۱۲. شکل ۴-۱۱ را برای موارد مورد نیاز زیر بازنویسی کنید. (الف) بررسی برای <CapsLock> و <NumLock>، (ب) انتقال محتویات بایت دوم وضعیت شیفت صفحه کلید به BH، (ج) بررسی برای فشردن <Left Alt> و <Left Ctrl> نیز انجام شود و یک پیغام مناسب نمایش داده شود.

فصل ۱۷

پردازش داده‌های رشته‌ای

هدف : توضیح دستورات ویژه جهت استفاده در پردازش داده‌های رشته‌ای

مقدمه

تا این فصل، دستورات داده‌هایی را دستکاری می‌کنند که با یک بایت، کلمه، یا دو کلمه‌ای تعریف شده‌اند. اما اغلب لازم است که فیلد داده‌هایی منتقل یا مقایسه شوند که طول آن‌ها از این حد تجاوز می‌نماید. برای مثال ممکن است بخواهید توضیحات یا نام‌هایی را مقایسه کنید و به ترتیب صعودی آنها را مرتب کنید.

عناصر چنین نوع داده‌ای با ***داده‌های رشته‌ای*** شناخته می‌شوند و ممکن است در قالب کاراکتر یا اعداد باشند. زبان اسمبلی این دستورات رشته‌ای را برای پردازش داده‌های رشته‌ای فراهم ساخته است.

MOV انتقال یک بایت، کلمه، یا دو کلمه‌ای از یک موقعیت حافظه به دیگر موقعیت.

LODS بارگذاری یک بایت از حافظه در AL، یک کلمه در AX، یا دو کلمه‌ای در EAX .

STOS ذخیره سازی محتویات ثبات‌های AL، AX یا EAX در حافظه.

CMPS مقایسه بایت، کلمه، یا دو کلمه‌ای موقعیت حافظه.

SCAS مقایسه محتویات AL، AX یا EAX با محتویات یک موقعیت حافظه.

در دستور دیگر از عملیات رشته‌ای INS و OUTS در فصل ۱۲ گفته خواهد شد. یک دستور مرتبط پیشوند REP است، که سبب می‌شود یک دستور رشته‌ای مکرر اجرا شود تا تعدادی بایت، کلمه یا دو کلمه‌ای به تعداد خاصی پردازش شود.

جزئیات عملیات رشته‌ای

یک دستور رشته‌ای می‌تواند تکرار پردازش یک بایت، کلمه، یا دو کلمه‌ای (در 80386 و بعد) را مشخص سازد. بنابراین می‌توانید یک عملیات بایتی را برای یک رشته با طول فردی از بایت‌ها و یک عملیات کلمه‌ای برای رشته‌ای با طول زوجی از بایت‌ها را انتخاب نمایید. هر دستور رشته‌ای یک بایت، کلمه و دوکلمه‌ای دارد و استفاده از زوج ثبات ES:DI یا DS:SI در نظر می‌گیرد.

دستورات رشته‌ای انتظار دارند که DI و SI حاوی آدرس افست معتبر باشند که بایت‌های حافظه را ارجاع دهند. ثبات SI معمولاً با ثبات DS در ارتباط است (سگمنت داده) به شکل DS:SI. ثبات DI همیشه با ثبات ES (سگمنت اضافی) به شکل ES:DI در ارتباط است. به این دلیل MOVS، STOS، CMPS، SCAS نیاز دارند که یک برنامه EXE. ثبات ES را مقداردهی کند، اغلب، با همان آدرس که در ثبات DS است.

```
MOV  AX,@data  ; گرفتن آدرس سگمنت داده
MOV  DS,AX     ; ذخیره در DS
MOV  ES,AX     ; و در ES
```

شکل ۱-۱۲ ثبات‌های مرتبط با هر دستور رشته‌ای را نشان می‌دهد. دو روش پایه برای کد نویسی دستورات رشته‌ای وجود دارد :

۱) در شکل ۱-۱۲، ستون‌دوم قالب پایه برای هرعملیات را نشان‌می‌دهد، که از عملوندهای ستون‌سوم استفاده می‌کند اگر دستوری مانند MOVS را کد نمائید، عملوندهای آن را نیز می‌نویسید - برای مثال MOVS BYTE1,BYTE2 در حالیکه تعریف عملوندها بر طول انتقال دلالت دارد). بخش بعدی کدنویسی متغیر برای دستورات رشته‌ای این قالب را با توضیحات بیشتر ارائه می‌دهد.

۲) دومین روش برای کدنویسی دستورات رشته‌ای، عمل استانداردی است که در ستون چهار، پنج و شش از شکل ۱-۱۲ نشان داده شده است. می‌توانید آدرس عملوندها را در ثبات DI و SI بارگذاری کنید و دستورات را بدون عملوندها کد نمایید، برای مثال :

```
LEA   DI,BYTE2   ; آدرس BYTE 2
LEA   SI,BYTE1   ; آدرس BYTE 1
MOVSB            ; انتقال BYTE1 به BXTE2
```

REP: پیشوند تکرار رشته

پیشوند REP بلافاصله قبل از یک دستور رشته‌ای، مانند REP MOVSB، برای تکرار اجرا بر پایه شمارشگر خالی که در ثبات CX تنظیم شده، فراهم گردیده است. REP دستور رشته‌ای را اجرا می‌کند، CX را کاهش می‌دهد، و این عملیات را تا شماره CX صفر شود تکرار می‌کند. با این روش، شما می‌توانید رشته‌هایی با هر طول مجازی را پردازش نمایید.

Operation	Basic Instruction	Implied Operands	Byte Operation	Word Operation	Doubleword Operation
Move	MOVS	ES:DI,DS:SI	MOVSB	MOVSW	MOVSD
Load	LODS	AX,DS:SI	LODSB	LODSW	LODSD
Store	STOS	ES:DI,AX	STOSB	STOSW	STOSD
Compare	CMPS	DS:SI,ES:DI	CMPSB	CMPSW	CMPSD
Scan	SCAS	ES:DI,AX	SCASB	SCASW	SCASD

شکل ۱-۱۲ قالب دستورات رشته‌ای

پرچم جهت (DF) جهت تکرار عملیات را مشخص می‌سازد :

- برای پردازش از چپ به راست (عمل معمول)، از CLD برای پاک کردن DF با صفر استفاده کنید.
- برای پردازش از راست به چپ از STD برای تنظیم DF با یک استفاده کنید.

در مثال زیر، فرض کنید که DS و ES هر دو با آدرس سگمنت داده همانطور که قبلاً نشان داده شد مقدار دهی شده‌اند. یک عملیات REP MOVSB (یا ترجیحاً کپی) ۲۵ بایت از DATASTR1 را به DATASTR2 منتقل می‌کند :

```
DATASTR1 DB   25 DUP ('*')   ; فیلد ارسالی
DATASTR2 DB   25 DUP(' ')    ; فیلد دریافتی
         ...
         CLD                 ; پاک کردن پرچم جهت
         MOV  CX,25          ; مقدار دهی برای ۲۵ بایت
         LEA  D1,DATASTR2    ; مقدار دهی آدرس دریافتی
         LEA  S1,DATASTR1    ; مقدار دهی آدرس ارسالی
         REP  MOVSB ; DATASTR2 کپی DATASTR1 به
```

در طی اجرا، SCAS , CMPS نیز پرچم وضعیت را تنظیم می‌کند، بنابراین عملیات می‌تواند بلافاصله با یافتن شرط خاصی خاتمه یابد. انواع مختلف REP برای این منظور در زیر آورده شده است.

- REP، عملیات تا زمانی که CX کاهش یافته و صفر شود ادامه می‌یابد.
- REPE یا REPZ عملیات را تا زمانی که پرچم صفر مبنی بر مساوی یا صفر باشد تکرار می‌کند. وقتی ZF مبنی بر صفر نبودن یا مساوی نبودن شود یا CX کاهش یافته و صفر شود خاتمه می‌یابد.
- REPNE یا REPNZ عملیات را تا زمانی که ZF مبنی بر عدم صفر یا نامساوی باشد تکرار می‌کند. وقتی ZF مبنی بر صفر یا مساوی بودن شود یا CX کاهش یافته و صفر شود خاتمه می‌یابد.

استفاده از عملیات کلمه و دو کلمه‌ای پردازش سریع‌تری را فراهم می‌سازد. اکنون هر یک از عملیات رشته‌ای را با جزئیات آن بررسی می‌کنیم.

MOVS: دستور انتقال رشته

MOVSB ، MOVSW و MOVSD با یک پیشوند REP ترکیب شده و یک طول در CX، می‌تواند هر تعداد کاراکتر را منتقل نماید. اگر چه عملوندها را کد نویسی نمی‌کنید، دستور چنین خواهد بود.

[Label:]	REP MOVSn	[ES:DI, DS:SI]

برای رشته دریافتی، ثبات‌های افست : سگمنت، ES:DI هستند، برای رشته ارسالی، ثبات‌های افست : سگمنت، DS:SI می‌باشند. در نتیجه در شروع یک برنامه EXE. ثبات ES و ثبات DS مقدار دهی شده و قبل از اجرای MOVS از LEA برای مقدار دهی ثبات‌های DI و SI استفاده کنید. بسته به پرچم جهت، MOVS، ثبات‌های DI و SI را با یک برای بایت، ۲ برای کلمه و ۴ برای ۲ کلمه‌ای کاهش یا افزایش می‌دهد مثال زیر انتقال کلمه را مشخص می‌سازد.

```
MOV   CX,10              ; تعداد کلمه
LEA   DI,STRING 2        ; آدرس رشته دوم
LEA   SI,STRING 1        ; آدرس رشته اول
REP   MOVSW              ; انتقال ده کلمه
```

دستورات معادل با عملیات REP MOVSW چنین خواهد بود :

```
      JCXZ  J90          ; اگر CX صفر است بگذر
J30 : MOV   AX,[SI]      ; گرفتن کلمه از رشته اول
      MOV   [DI],AX      ; ذخیره کلمه در رشته دوم
      ADD   DI,2         ; افزایش برای کلمه بعدی
      ADD   SI,2         ; افزایش برای کلمه بعدی
      LOOP  J30          ; کاهش CX و تکرار
J90 : ...
```

شکل ۲-۶ انتقال یک فیلد ۹ بایتی را مشخص نموده است. این برنامه می‌تواند از MOVSB بدین منظور استفاده نماید. در شکل ۲-۱۲ روال B10MVSB از MOVSB برای انتقال فیلد ده بایتی HEADG1، یک بایت در هر بار به HEADG 2 استفاده می‌کند. دستور اول، CLD، پرچم جهت را با صفر پاک می‌کند بنابراین MOVSB داده‌ها را از چپ به راست پردازش می‌کند. پرچم جهت معمولاً در شروع اجرا صفر است ، اما CLD برای احتیاط، در اینجا کد می‌شود.

دو دستور LEA، ثبات‌های SI و DI را بترتیب با آدرس افست HEADG 1 و HEADG 2 مقدار دهی می‌کنند. از آنجائیکه بارگذار برای یک برنامه COM. بطور اتوماتیک ثبات‌های DS و ES را مقدار دهی می‌کند. آدرس افست : سگمنت برای EI:DI و DS:SI صحیح خواهد بود.

یک دستور MOV ، CX را با ۱۰ مقدار دهی می‌کند (طول HEADG 1 و HEADG 2). دستور REP MOVSB حال چنین عمل می‌کند.

```
TITLE     A12MOVST (COM)  Use of MOVSn string operations
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ---------------------------------------------------------
HEADG1    DB      'Cybernauts'          ;Data items
HEADG2    DB      10 DUP(' ')
HEADG3    DB      10 DUP(' ')
; ---------------------------------------------------------
A10MAIN   PROC    NEAR                  ;Main procedure
          CALL    B10MVSB               ;MVSB subroutine
          CALL    C10MVSW               ;MVSW subroutine
          MOV     AX,4C00H              ;End of processing
          INT     21H
A10MAIN   ENDP
;                 Use of MOVSB:
;                 -------------
B10MVSB   PROC    NEAR
          CLD                           ;Left to right
          MOV     CX,10                 ;Move 10 bytes,
          LEA     DI,HEADG2             ;  HEADG1 to HEADG2
          LEA     SI,HEADG1
          REP MOVSB
          RET
B10MVSB   ENDP
;                 Use of MOVSW:
;                 -------------
C10MVSW   PROC    NEAR
          CLD                           ;Left to right
          MOV     CX,05                 ;Move 5 words,
          LEA     DI,HEADG3             ;  HEADG2 to HEADG3
          LEA     SI,HEADG2
          REP MOVSW
          RET
C10MVSW   ENDP
          END     BEGIN
```

شکل ۲-۱۲ استفاده از عملیات رشته‌ای MOVS

- بایت سمت چپ HEADG 1 (با DS:SI آدرس دهی شده) را به بایت سمت چپ HEADG 2 (با ES:DI آدرس دهی شده) منتقل می‌کند.
- افزایش DI , SI برای بایت‌های بعدی (به سمت راست).
- کاهش CX با یک
- این عملیات را، ۱۰ بار تکرار می‌کند تا CX صفر شود.
- از آنجائیکه پرچم جهت صفر است و MOVSB، DI و SI را افزایش می‌دهد، هر بار تکرار یک بایت را پردازش می‌کند، مانند HEADG1+1 با HEADG2+1 و همینطور الی آخر.... در انتهای اجرا CX حاوی صفر، DI حاوی آدرس HEADG2+10 و SI حاوی آدرس HEADG1+10 است - هر دو یک بایت بعد از انتهای نام قرار دارند.

اگر پرچم جهت یک باشد، MOVSB ، DI و SI را کاهش خواهد داد و سبب خواهد شد که پردازش از راست به چپ انجام شود. اما در این حالت، برای انتقال صحیح محتویات، باید SI را با HEADHG1+9 و DI را با HEADG2+9 مقدار دهی کنید.

در روال بعدی شکل ۲-۱۲ C10MVSW از MOVSW به HEADG3 استفاده می‌کند. در انتها اجرا، CX صفر خواهد شد، DI آدرس HEADG3+10 برای انتقال پنج کلمه از HEADG2 رادارد، و SI حاوی آدرس HEADG2+10 می‌شود.

از آنجائیکه MOVSW ثبات‌های DI و SI را با ۲ کاهش می‌دهد، عملیات فقط ۵ بار حلقه نیاز دارد، برای پردازش ***راست به چپ*** پرچم جهت را تنظیم و SI را با HEADG1+8 و DI را با HEADG2+8 مقدار دهی کنید.

LODS: دستور بارگذاری رشته

LODS، یک بایت را در AL قرار می‌دهد، یک کلمه را در AX، یا یک دو کلمه از حافظه را در EAX قرار می‌دهد. آدرس حافظه چیزی است که باید در ثبات‌های DS:SI قرار گیرد، اگر چه شما SI را مجدداً بازنویسی کنید وابسته به پرچم جهت، عملیات نیز SI را افزایش یا کاهش می‌دهد، با یک برای بایت، با ۲ برای کلمه، و با ۴ برای دوکلمه‌ای.

از آنجائیکه یک عملیات LODS ثبات را پر می‌کند، دلیلی برای استفاده از پیشوند REP با آن وجود ندارد. در اغلب موارد، یک دستور MOV ساده کافی است. اما MOV ، ۳ بایت کد ماشین تولید می‌کند در حالیکه LODS فقط یک بایت تولید می‌کند، اگر چه لازم است که برای آن SI را مقدار دهی کنید. شما می‌توانید از LODS برای مراحل یک رشته، بایت، کلمه یا دو کلمه در هر بار بترتیب برای یک کاراکتر ویژه استفاده کنید. دستور معادل LODSB چنین است.

```
MOV   AL,[SI]          ; انتقال بایت به AL
INC   SI               ; افزایش SI برای بایت بعدی پرچم را تنظیم می‌کند
```

در شکل ۳-۱۲ ناحیه داده یک فیلد ۱۰ بایتی با نام HEADG1 حاوی مقدار "Cybenauts" و فیلد ۱۰ بایتی دیگری با نام HEADG2 تعریف می‌کند. هدف، انتقال بایت‌های HEADG1 به HEADG2 به ترتیب معکوس است، بنابراین HEADG2 حاوی "Stuanrebyc" خواهد بود LODSB برای دستیابی یک بایت در هر زمان از HEADG1 به AL استفاده می‌شود و دستور MOV [DI],AL بایت‌ها را به HEADG2 از راست به چپ منتقل خواهد کرد.

STOS: دستور ذخیره سازی رشته

STOS محتویات AL، AX یا EAX را در یک بایت، کلمه یا دو کلمه‌ای در حافظه ذخیره می‌کند. آدرس حافظه همیشه در ثبات‌های ES:DI قرار دارد. وابسته به پرچم جهت، STOS همچنین ثبات DI را یکی برای بایت، ۲تا برای کلمه، ۴ تا برای دو کلمه‌ای افزایش یا کاهش خواهد داد.

```
TITLE     A12LODST (COM)  Use of LODSB string operation
          .MODEL  SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A1OMAIN
; ---------------------------------------------------------
HEADG1    DB      'Cybernauts'     ;Data items
HEADG2    DB      10 DUP(20H)
; ---------------------------------------------------------
A1OMAIN   PROC    NEAR             ;Main procedure
          CLD                      ;Left to right
          MOV     CX,10
          LEA     SI,HEADG1        ;Load address of HEADG1
          LEA     DI,HEADG2+9      ;Load address of HEADG2+9
A20:      LODSB                    ;Get character in AL,
          MOV     [DI],AL          ;  store in HEADG2,
          DEC     DI               ;  right to left
          LOOP    A20              ;Ten characters?
          MOV     AX,4C00H         ;  yes, exit
          INT     21H
A1CMAIN   ENDP
          END     BEGIN
```

شکل ۳-۱۲ استفاده از عملیات رشته‌ای LODSB

یک مورد استفاده عملی از STOS با پیشوند REP برای مقدار دهی ناحیه داده با هر مقدار خاص است، مانند پاک کردن یک ناحیه داده با جای خالی. شما تعداد بایت‌ها و کلمه‌ها یا دو کلمه‌ای‌ها را در CX تنظیم می‌کنید.

دستورات معادل REP STOSB چنین هستند :

```
        JCXZ    P30         ; پرش اگر CX صفر است
P20 :   MOV     [DI],AL     ; ذخیره سازی AL در حافظه
        INC/DEC DI          ; افزایش یا کاهش
        LOOP    P20         ; کاهش CX و تکرار
P30 :   ...
```

دستور STOSW در شکل ۴-۱۲ یک کلمه حاوی 2020H (جای خالی) را ۵ مرتبه در HEADG1 ذخیره می‌کند. عملیات AL را در اولین بایت و AH را در دومین بایت (که معکوس شده است) ذخیره می‌کند. در انتها، همهٔ HEADG1 فاصله خواهد بود، CX حاوی 00 و DI حاوی آدرس HEADG1+10 است.

برنامه : استفاده از LODS و STOS برای انتقال داده

برنامه شکل ۵-۱۲ استفاده از دستورات LODS و STOS را مشخص می‌سازند. مثال شبیه برنامه شکل ۴-۱۰ است، که کاراکترها را منتقل می‌کند و صفت‌ها مستقیماً به ناحیه نمایش ویدئو، منتقل می‌شود، بجز آنکه شکل ۵-۱۲ حاوی این تفاوت‌ها است :

- برای ناحیه ویدئو، برنامه بیشتر از صفحه ۲ استفاده می‌کند تا صفحه ۱.
- در B10PROC از STOSW برای ذخیره کاراکترها و صفات مرتبط، در ناحیه ویدئویی، بجز دستورات استفاده می‌کند و با دستور DEC، DI را کاهش می‌دهد : `MOV WORD PTR [VIDAREA+DI],AX`
- یک عنصر با نام PROMPT در سگمنت داده تعریف می‌کند، که در انتهای پردازش برای اعلان به کاربر که "Press any key ..." استفاده شود.
- با تکمیل پردازش، روال C10PROMPT اعلان تعریف شده را به ناحیه نمایش ویدئو منتقل می‌کند. برای خاتمه با استفاده از LODSB کاراکترها را یکی در هر زمان از PROMPT در AL دستیابی می‌کند و با استفاده STOSW هر کاراکتر و صفت آن را از AX در ناحیه ویدئویی ذخیره می‌سازد.

```
TITLE     A12STOST (COM)  Use of STOSW string operation
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ------------------------------------------------------
HEADG1    DB      'Cybernauts'          ;Data item
; ------------------------------------------------------
A10MAIN   PROC    NEAR                  ;Main procedure
          CLD                           ;Left to right
          MOV     AX,2020H              ;Move
          MOV     CX,05                 ;5 blank words
          LEA     DI,HEADG1             ;to HEADG1
          REP STOSW
          MOV     AX,4C00H              ;End of processing
          INT     21H
A10MAIN   ENDP
          END     BEGIN
```

شکل ۴-۱۲ استفاده از عملیات رشته‌ای STOSW

CMPS : دستور مقایسه رشته

CMPS محتویات یک موقعیت حافظه (آدرسدهی شده با DS:SI) را با یک موقعیت دیگر حافظه (آدرسدهی شده با ES:DI) مقایسه می‌کند. وابسته به پرچم جهت، CMDS ، SI و DI را کاهش یا افزایش می‌دهد، یکی برای بایت، ۲ تا برای کلمه، ۴ تا برای دو کلمه‌ای. عملیات پرچم‌های SF ، PF ، OF ، AF ، CF و ZF را تنظیم می‌کند. وقتی با یکی پیشوند REP ترکیب شود و یک طول در CX گذاشته شود، CMPS می‌تواند بترتیب رشته‌ای با هر طول را مقایسه کند.

```
TITLE     A12DRVID (EXE)  Direct video display
          MODEL SMALL
; ------------------------------------------------------
VIDSEG    SEGMENT AT 0BA00H          ;Page 2 of video area
VIDAREA   DB  1000H DUP(?)
VIDSEG    ENDS
; ------------------------------------------------------
          DATA
PROMPT    DB      'Press any key...
; ------------------------------------------------------
          .STACK 64
; ------------------------------------------------------
          .CODE
A10MAIN   PROC    FAR
          MOV     AX,@data           ;Addressability for
          MOV     DS,AX              ;  data segment
          MOV     AX,VIDSEG          ;  and for
          MOV     ES,AX              ;  video area
          ASSUME  ES:VIDSEG
          MOV     AH,0FH             ;Request get
          INT     10H                ;  and save
          PUSH    AX                 ;  current mode
          PUSH    BX                 ;  and page
          MOV     AH,00H             ;Request set
          MOV     AL,03              ;  mode 03, clear screen
          INT     10H
          MOV     AH,05H             ;Request set
          MOV     AL,02H             ;  page #02
          INT     10H
          CALL    B10PROC            ;Process display area
          CALL    C10PROMPT          ;Display user prompt
          CALL    D10INPT            ;Provide for input
          MOV     AH,05H             ;Restore
          POP     BX                 ;original
          MOV     AL,BH              ;page number
          INT     10H
          POP     AX                 ;Restore video
          MOV     AH,00H             ;mode (in AL)
          INT     10H
          MOV     AX,4C00H           ;End of processing
          INT     21H
A10MAIN   ENDP
;                 Store character & attribute in video area:
;                 ------------------------------------------
B10PROC   PROC    NEAR
          MOV     AL,41H             ;Character to display
          MOV     AH,01H             ;Attribute
          MOV     DI,660             ;Start of display area
B30:      MOV     CX,60              ;Characters per row
          REP STOSW                  ;Repeat 60 times
          INC     AH                 ;Next attribute
          INC     AL                 ;Next character
          ADD     DI,40              ;Indent for next row
          CMP     AL,51H             ;Last character to display?
          JNE     B30                ;  no, repeat
          RET                        ;  yes, return
B10PROC   ENDP
;                 Move prompt to video area:
;                 --------------------------
C10PROMPT PROC    NEAR
          MOV     AH,03H             ;New attribute in AH
          MOV     CX,16              ;Characters to display
          MOV     DI,3840            ;Location in display area
          LEA     SI,PROMPT          ;Address of prompt
C20:      LODSB                      ;Character into AL
          STOSW                      ;Store in display area
```

شکل الف ۵-۱۲ استفاده از نمایش مستقیم ویدئو

```
        LOOP   C20                    ;16 times
        RET                           ;Return
C10PROMPT ENDP
;              Accept input:
;              -------------
D10INPT  PROC  NEAR
         MOV   AH,10H                 ;Request keyboard
         INT   16H                    ;  input
         RET
D10INPT  ENDP
         END   A10MAIN
```

شکل ب ۵-۱۲ استفاده از نمایش مستقیم ویدئو

اما توجه کنید که CMPS یک مقایسهٔ ***الفبایی عددی*** انجام می‌دهد، که مقایسه‌ای بر طبق مقادیر ASCII خواهد بود. عملیات یک مقایسه جبری نخواهد بود، که حاوی مقادیر عددی علامتدار باشد.

فرض کنید دو رشته حاوی 'Jean' و 'Joan' مقایسه می‌شوند. مقایسه از چپ به راست، هر بار یک بایت به شکل زیر خواهد بود : J : J مساوی a : a مساوی

e : o نامساوی (e کمتر است). n : n مساوی

مقایسه ۴ بایت با مقایسه n با n (مساوی) خاتمه می‌پذیرد. حال، از آنجائیکه دو نام مساوی نیستند، هنگامیکه مقایسه بین دو کاراکتر مختلف را انجام داد عملیات باید خاتمه یابد. بدین منظور، REPE (تکرار در هنگام مساوی بودن)، تا هنگامیکه مقایسه بین دو کاراکتر مساوی است، یا ثبات CX مساوی صفر است، عملیات را تکرار می‌کند. برای تکرار مقایسه یک بایت چنین کد می‌شود. : REPE CMPSB شکل ۶-۱۲ حاوی دو مثال است که از CMPSB استفاده می‌کند. اولین مثال ،HEADG1 را با HEADG2 مقایسه می‌کند، که حاوی همان مقادیر است. CMPSB برای ده بایت ورودی ادامه می‌یابد. در انتهای اجرا، CX حاوی 00 است، DI حاوی آدرس HEADG2+10، SI حاوی آدرس HEADG1+10 است، پرچم علامت مثبت است، و پرچم صفر مبنی بر مساوی یا صفر است.

مثال دوم HEADG2 را با HEADG3 مقایسه می‌کند، که حاوی مقادیر مختلفی است. عملیات CMPSB بعد از مقایسه اولین بایت خاتمه می‌یابد و نتیجه آن به شرط نامساوی / بزرگتر بودن چنین است. CX حاوی 09، DI حاوی آدرس HEADG3+1، SI حاوی آدرس HEADG2+1، پرچم علامت مثبت و پرچم صفر مبنی بر نامساوی بودن می‌باشد.

مثال اول مساوی یا صفر را نتیجه می‌دهد و (فقط برای مشخص ساختن نتیجه) 01 را به ثبات BH منتقل می‌کند. مثال دوم نامساوی بودن را نتیجه می‌دهد و 00 را در ثبات BL می‌گذارد. اگر از DEBUG برای پیگیری دستورات استفاده نمایید، در انتهای اجرا مقدار 0102 را در BX خواهید دید.

توجه ۱ : این مثال از CMPSB برای مقایسه داده‌ها یک بایت در هر بار استفاده می‌کند. اگر از CMPSW برای مقایسه داده‌ها یک کلمه در هر بار استفاده کنید، باید CX را با 5 مقدار دهی کنید که مشکلی نیست وقتی کلمات مقایسه می‌شوند، CMPSW بایت‌ها را معکوس می‌کند. برای مثال، نام‌های SAMUEL و ARNOLD را با هم مقایسه می‌کنیم. در مقایسه اولین کلمه‌ها، بجای مقایسه AS با AR، عملیات AS را با RA مقایسه می‌کند. بنابراین، بجای آن که SAMUEL، مقدار بزرگتر باشد، اشتباهاً کوچکتر خواهد بود. CMPSW وقتی صحیح خواهد بود که فیلدهای داده مقادیر عددی بدون علامت تعریف شده با DW ، DD یا DQ باشند. (که در این موارد داده‌ها به ترتیب بایت معکوس ذخیره می‌شود.)

```
TITLE     A12CMPST (COM)  Use of CMPS string operations
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ---------------------------------------------------------
HEADG1    DB      'Cybernauts'      ;Data items
HEADG2    DB      'Cybernauts'
HEADG3    DB      10 DUP(' ')
; ---------------------------------------------------------
A10MAIN   PROC    NEAR              ;Main procedure
          CLD                       ;Left to right
          MOV     CX,10             ;Initialize for 10 bytes
          LEA     DI,HEADG2
          LEA     SI,HEADG1
          REPE CMPSB                ;Compare HEADG1 : HEADG2
          JNE     A20               ;  not equal, bypass
          MOV     BH,01             ;  equal, set BH
A20:
          MOV     CX,10             ;Initialize for 10 bytes
          LEA     DI,HEADG3
          LEA     SI,HEADG2
          REPE CMPSB                ;Compare HEADG2 : HEADG3
          JE      A30               ;  equal, exit
          MOV     BL,02             ;  not equal, set BL
A30:
          MOV     AX,4C00H          ;End of processing
          INT     21H
A10MAIN   ENDP
          END     BEGIN
```

شکل ۶-۱۲ استفاده از عملیات رشته‌ای CMPS

SCAS: دستور پیمایش رشته‌ای

SCAS مشخصاً با CMPS تفاوت دارد چرا که SCAS یک رشته را برای یافتن یک بایت خاص کلمه، یا دو کلمه‌ای پیمایش می‌نماید. SCAS محتویات موقعیت حافظه (آدرسدهی شده با ES:DI) با محتویات AL، AX یا EAX مقایسه می‌کند. وابسته به پرچم جهت، SCAS، ثبات DI را برای بایت با یک، برای کلمه با ۲، و برای دو کلمه‌ای با ۴ افزایش یا کاهش می‌دهد. در خاتمه اجرا، SCAS پرچم‌های AF ، CF ، PF ، SF ، ZF و OF را تنظیم می‌کند. وقتی با یک پیشوند REPnn ترکیب شود و طول در CX گذاشته شود هر رشته‌ای به طول مورد نظر پیمایش می‌شود.

SCAS در عمل برای کاربردهای ویرایش متن مفید است. بطوریکه برنامه باید برای نقطه‌گذاری مانند نقطه، ویرگول، جای خالی پیمایش شود.

برنامه شکل ۷-۱۲، HEADG1 را برای یافتن حرف کوچک 'r' پیمایش می‌کند. زیرا عملیات SCASB به پیمایش ادامه می‌دهد تا مقایسه مساوی نباشد یا CX صفر شود، عملیات در این حالت چنین است. REPNE SCASB وقتی رشته 'Cybernauts' در HEADG1 پیمایش می‌شود، SCASB یک مورد موافق در پنجمین مقایسه می‌یابد. اگر از DEBUG برای پیگیری دستورات استفاده نمایید، در خاتمه اجرای REP SCASB خواهید دید که پرچم صفر، صفر را نشان خواهد داد، CX با 05 کاهش می‌یابد و DI با 05 افزایش یافته است (DI پس از موقعیت واقعی 5، یکی افزوده شده است).

برنامه در ثبات AL، (برای مشخص ساختن حاصل) ۰۳ را ذخیره می‌کند مبنی بر اینکه کاراکتر پیدا شده SCASW برای یافتن یک کلمه در حافظه که با کلمهٔ ثبات AX جور باشد، پیمایش می‌کند، اگر از LODW یا MOV برای انتقال یک کلمه به ثبات AX استفاده کنید اولین بایت در AL و دومین بایت در AH خواهد بود از آنجائیکه SCASW بایت‌ها را بصورت معکوس مقایسه می‌کند، عملیات درست کار می‌کند.

```
TITLE     A12SCAST (COM)  Use of SCASB string operation
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ------------------------------------------------------
HEADG1    DB      'Cybernauts'          ;Data item
; ------------------------------------------------------
A10MAIN   PROC    NEAR                  ;Main procedure
          CLD                           ;Left to right
          MOV     AL,'r'
          MOV     CX,10                 ;Scan HEADG1
          LEA     DI,HEADG1             ;  for 'r'
          REPNE   SCASB
          JNE     A20                   ;If found,
          MOV     AL,03                 ;  store 03 in AL
A20:
          MOV     AH,4CH
          INT     21H                   ;End of processing
A10MAIN   ENDP
          END     BEGIN
```

شکل ۷-۱۲ استفاده از عملیات رشته‌ای SCASB

مثال : استفاده از پیمایش و جایگزینی

ممکن است بخواهید یک کاراکتر خاص را با کاراکتر دیگری جایگزین نمائید، برای مثال، کاراکترهای ویرایشی را، مانند علامت پاراگراف و سمبول انتهای صفحه را از یک مدرک پاک کنید. مثال زیر TSTDATA را برای یافتن یک (*) پیمایش می‌کند و با یک فاصله جایگزین می‌کند. اگر SCASB در موقعیت یک '*' قرار بگیرد، عملیات را خاتمه می‌دهد. TSTDATA حاوی یک ستاره در TSTDATA+5 است، که باید جای خالی جایگزین آن شود، اگر چه SCASB ثبات DI را افزوده و TSTDATA+6 خواهد شد. کاهش از DI به [DI-1] آدرس صحیح برای جایگزینی کاراکتر فاصله را فراهم می‌سازد.

```
DATALEN   EQU     13              ; طول TSTDATA
TSTDATA   DB      'Extra * innings'
          ...
          CLD                     ; تنظیم چپ به راست
          MOV     AL,'*'          ; جستجوی کاراکتر
          MOV     CX,DATALEN      ; طول TSTDATA
          LEA     DI,TSTDATA      ; آدرس TSTDATA
          REPNE   SCASB           ; پیمایش TSTDATA
          JNE     E20             ; یافتن کاراکتر؟
          MOV BYTE PTR [DI-1],20H ; بله، جایگزینی با فاصله
E20 :     ...
```

روش دیگر درکدنویسی برای دستورات رشته‌ای

همانطور که بحث شد، اگر یک دستور رشته‌ای را با یک پسوند B، W یا D کد نویسی کنید مانند MOVSB، MOVSW یا MOVSD اسمبلر طول صحیح را فرض می‌کند و نیاز به عملوند ندارد. همچنین می‌توانید قالب دستور اصلی را برای عملیات رشته‌ای استفاده کنید. برای یک دستور مانند MOVS که هیچ پسوندی مبنی بر بایت، کلمه یا دو کلمه‌ای ندارد، عملوند طول را مشخص می‌سازد. برای مثال، اگر CHAR1 و CHAR2 به شکل DB تعریف شده باشد، دستور

```
REP  MOVS  CHAR1,CHAR2
```

تکرار انتقال بایت‌هایی است که با CHAR2 آغاز می‌شوند به بایت‌هایی که با CHAR1 آغاز می‌شوند. قالب دیگری است که اجازه می‌دهد تا صریحاً به ثبات سگمنت اشاره کنید و از پیش پردازنده PTR استفاده کنید. اگر ثبات‌های DI و SI را با آدرس CHAR1 و CHAR2 بارگذاری کنید، می‌توانید دستور MOV را چنین کد نویسی کنید :

```
        LEA     DI,CHAR2
        LEA     SI,CHAR1
REP     MOVS    ES:BYTE PTR [DI], DS:[SI]
```

کمی از برنامه‌ها با این روش کد نویسی می‌شود و این قالب برای ثبت در اینجا آورده شده است.

کپی کردن یک الگو

دستور STOS برای مقدار دهی یک ناحیه مطابق با یک ارزش مشخص بایت یا کلمه، مفید می‌باشد. برای تکرار یک الگو که بیش از یک کلمه است، می‌توانید دستور MOVS را با تغییرات جزئی استفاده کنید. فرض کنید می‌خواهید یک خط نمایش با الگوی زیر برقرار سازید. . . . |****| |****| |****| |****| |****| |****|

به جای تعریف تمام الگوی تکراری، فقط نیاز دارید اولین شش بایت را تعریف کنید. الگو را بلافاصله قبل از خط نمایش کد نمایید. برنامه نویسی مورد نیاز به صورت زیر است.

```
PATTERN  DB    '|****|'        ; الگویی که باید کپی شود
DISAREA  DB    42 DUP (?)      ; ناحیه نمایش
         CLD                   ; چپ به راست
         MOV   CX,21           ; ۲۱ کلمه
         LEA   DI,DISAREA      ; مقصد
         LEA   SI,RATTERN      ; مبدأ
         REP   MOVSW           ; انتقال کاراکترها
```

در موقع اجرا، دستورالعمل MOVSW اولین کلمه از PATTERN (یعنی *) را به اولین کلمه از DISAREA انتقال می‌دهد و سپس دومین (**) و سومین کلمه (|*) را انتقال می‌دهد.

```
|****|    |****|
  ↑         ↑
PATTERN   DISAREA
```

در اینجا، ثبات DI حاوی آدرس DISAREA+6 و ثبات SI حاوی آدرس PATTERN+6 ـ که آدرس DISAREA نیز هست ـ می‌باشند. حال، عمل به طول خودکار الگو را به وسیله انتقال اولین کلمه از DISAREA به DISAREA+6 ، DISAREA+2 به DISAREA+8 ، DISAREA+4 به DISAREA+10 و الی آخر کپی برداری می‌کند. بالاخره الگو تا انتهای DISAREA کپی برداری می‌شود.

```
|****| |****| |****| |****| |****|  . . .  |****|
  ↑      ↑      ↑      ↑                     ↑
PATTERN  DISAREA+6  DISAREA+12          DISAREA+24
```

می‌توانید از این تکنیک برای کپی‌برداری هر الگو به هر تعداد بار استفاده نمایید. الگو ممکن است هر طولی داشته باشد، اما باید بلافاصله قبل از میدان دریافت کننده قرار گرفته باشد.

برنامه : تنظیم از سمت راست روی صفحه تصویر

برنامه شکل ۸-۱۲ اکثر مفاهیم توصیف شده در این فصل را تشریح می‌کند. روال‌ها اعمال زیر را انجام می‌دهند :

- A10MAIN ثبات‌های سگمنت را مقدار دهی می‌کند و روالهای B10INPT ، C10SCAS ، D10RGHT و E10CLNM را فراخوانی می‌کند.

```
TITLE     A12RIGHT (EXE)  Right adjust displayed names
          .MODEL SMALL
          .STACK 64
          .DATA
NAMEPAR   LABEL   BYTE              ;Name parameter list
MAXNLEN   DB      31                ;Maximum length
ACTNLEN   DB      ?                 ;No. of chars entered
NAMEINP   DB      31 DUP(' ')       ;Name

PROMPT    DB      'Name?', '$'
NAMEDIS   DB      31 DUP(' '), 13, 10, '$'
ROW       DB      00
          .CODE
.386
A10MAIN   PROC    FAR               ;Main procedure
          MOV     AX,@data          ;Initialize
          MOV     DS,AX             ;  data segment
          MOV     ES,AX
          MOV     AX,0600H
          CALL    Q10SCR            ;Clear screen
          SUB     DX,DX             ;Set cursor 00,00
          CALL    Q20CURS
A20LOOP:
          CALL    B10INPT           ;Request input of name
          TEST    ACTNLEN,0FFH      ;No name? (indicates end)
          JZ      A90               ;  yes, exit
          CALL    C10SCAS           ;Scan for asterisk
          CMP     AL,'*'            ;Found?
          JE      A20LOOP           ;  yes, bypass
          CALL    D10RGHT           ;Right adjust name
          CALL    E10CLNM           ;Clear name
          JMP     A20LOOP
A90:      MOV     AX,4C00H          ;End of processing
          INT     21H
A10MAIN   ENDP
;                 Prompt for input:
;                 ----------------
B10INPT   PROC
          MOV     AH,09H
          LEA     DX,PROMPT         ;Display prompt
          INT     21H
          MOV     AH,0AH
          LEA     DX,NAMEPAR        ;Accept input
          INT     21H
          RET
B10INPT   ENDP
;                  Scan name for asterisk:
;                  ----------------------
C10SCAS   PROC
          CLD                       ;Left to right
          MOV     AL,'*'            ;Character for scan
          MOV     CX,30             ;Set 30-byte scan
          LEA     DI,NAMEINP
          REPNE SCASB               ;Asterisk found?
          JE      C20               ;  no,  exit
          MOV     AL,20H            ;  yes, clear * in AL
C20:      RET
C10SCAS   ENDP

;                 Right adjust and display name:
;                 -----------------------------
D10RGHT   PROC
          STD                       ;Right to left
          MOVZX   CX,ACTNLEN        ;Length in CX for REP
          LEA     SI,NAMEINP        ;Calculate rightmost
          ADD     SI,CX             ;  position
          DEC     SI                ;  of input name
```

شکل الف ۸-۱۲ تنظیم از راست بر روی صفحه تصویر

```
                LEA     DI,NAMEDIS+30       ;Right pos'n of display name
                REP MOVSB                   ;Move string right to left
                MOV     DH,ROW
                MOV     DL,48
                CALL    Q20CURS             ;Set cursor
                MOV     AH,09H
                LEA     DX,NAMEDIS          ;Display name
                INT     21H

                CMP     ROW,20              ;Bottom of screen?
                JAE     D20                 ;  no,
                INC     ROW                 ;  increment row
                JMP     D90
D20:
                MOV     AX,0601H            ;  yes,
                CALL    Q10SCR              ;  scroll and
                MOV     DH,ROW              ;  set cursor
                MOV     DL,00
                CALL    Q20CURS
D90:            RET
D10RGHT         ENDP
;                       Clear name:
;                       ----------
E10CLNM         PROC
                CLD                         ;Left to right
                MOV     AX,2020H
                MOV     CX,15               ;Clear 15 words
                LEA     DI,NAMEDIS
                REP STOSW
                RET
E10CLNM         ENDP
;                       Scroll screen:
;                       -------------
Q10SCR          PROC                        ;AX set on entry
                MOV     BH,30               ;Color attribute
                MOV     CX,00
                MOV     DX,184FH
                INT     10H
                RET
Q10SCR          ENDP
;                       Set cursor row/col:
;                       -------------------
Q20CURS         PROC                        ;DX set on entry
                MOV     AH,02H
                SUB     BH,BH
                INT     10H
                RET
Q20CURS         ENDP
                END     A10MAIN
```

شکل ب ۸-۱۲ تنظیم از راست بر روی صفحه تصویر

- B10INPT پذیرفتن اسامی حداکثر ۳۰ کاراکتری و نمایش آنها در بالای صفحه نمایش.
- C10SCAS از SCASB برای پیمایش نام و عدم پذیرش هر ورودی حاوی یک *.
- D10RGHT استفاده از طول ACTNLEN در لیست پارامتر ورودی برای محاسبه کاراکتر سمت راست نام و MOVSB هر نام وارد شده را از راست تنظیم می‌کند، یکی زیر دیگری به شکل زیر :

Willie Mays
Mickey Mantle
Framk Robinson

- E10CLNM از STOSW برای پاک کردن ناحیه ورودی صفحه کلید استفاده می‌کند.

نکات کلیدی

- برای دستورهای رشته‌ای MOVS ، STOS ، CMPS و SCAS از اینکه برنامه EXE. ثبات ES را مقدار دهی نموده است اطمینان حاصل کنید.
- برای دستورهای رشته‌ای، از پسوند B، W یا D برای دستکاری بایت ، کلمه یا دو کلمه‌ای استفاده کنید.
- برای جهت مورد نیاز پردازش پرچم جهت را مقدار دهی نمایید : صفر (CLD) برای چپ به راست یا یک (SID) برای راست به چپ.
- مقدار دهی ثبات‌های DI و SI را مجدداً بررسی نمایید. برای مثال، MOVS عملوندهای DI و SI را شامل می‌شود، در حالیکه CMP عملوندهای SI و DI را شامل می‌شود.
- ثبات CX را برای REP جهت پردازش تعداد مورد نیاز بایت‌ها، کلمات، یا دو کلمه‌ای‌ها، مقدار دهی نمایید.
- برای پردازش معمولی، در REP با MOVS و STOS استفاده کنید و از REP شرطی (REPNE یا REPE) با CMPS و SCAS استفاده نمایید.
- CMPSW و SCASW بایت‌های کلماتی که مقایسه می‌شوند را معکوس می‌نمایند.
- پردازش راست به چپ، آدرس از بایت سمت راست فیلد آغاز می‌شود. برای مثال، اگر یک فیلد با نام COTITLE، ۱۰ بایت طول داشته باشد، سپس برای پردازش بایت‌ها آدرس قرار گرفته در LEA و COTITLE+9 است. اما برای پردازش کلمات باید آدرس COTITLE+8 باشد زیرا عملیات رشته‌ای تلویحاً دستیابی به COTITLE+9 و COTITLE+8 را خواهد داشت.

پرسش

۱-۱۲. عملیات رشته‌ای فرض می‌کند که عملوندها با ثبات‌های ES:DI یا DS:SI مرتبط است ثبات‌های موارد زیر را مشخص کنید : (الف) CMPS (عملوند ۱ و ۲) (ب) MOVS (عملوند ۱ و ۲) (ج) LODS (عملوند ۱)

۲-۱۲. برای عملیات رشته‌ای که از REP استفاده می‌کنند، (الف) چگونه تعداد دفعات تکرار را تنظیم می‌کنید؟ (ب) چگونه پردازش را برای از راست به چپ تنظیم می‌کنید؟

۳-۱۲. این فصل معادل دستورات زیر را با پسوند REP مشخص کرده است. (الف) MOVSB، (ب) LODSB و (ج) STOSB. برای هر کدام کد معادل را جهت پردازش کلمات مشخص کنید.

۴-۱۲. برنامه شکل ۲-۱۲ را باز نویسی کنید : (الف) برنامه را از COM. به قالب EXE. تبدیل کنید، (ب) ثبات ES را مقدار دهی کنید (ج) عملیات MOVSB و MOVSW را تغییر دهید به نحویکه از راست به چپ انتقال می‌دهد. از DEBUG برای پیگیری روال‌ها استفاده کنید و به محتویات ثبات‌ها و سگمنت داده توجه کنید.

۵-۱۲. هر یک از دستورات زیر چه شروطی را بررسی می‌کنند؟ (الف) REPE COMPSB (ب) REPNE SCASB.

۶-۱۲. از تعاریف داده زیر استفاده نمائید و برای قسمت‌های (الف)... (و) عملیات رشته‌ای نامرتبط راکدنویسی کنید :

```
DESCRIP    DB    'Data Tech Advisors'
OUTAREA    DB    18DUP (' ')
```

الف) DESCRIP را به OUTAREA از چپ به راست منتقل کنید.

ب) DESCRIP را به OUTAREA از راست به چپ منتقل کنید.

ج) بایت پنج و شش از DESCRIP را در AX قرار دهید.

د) AX را در OUTAREA+10 ذخیره نمایید.

ه) DESCRIP را با OUTAREA مقایسه کنید (این دو مساوی نخواهند بود).

و) DESCRIP را برای یافتن اولین کاراکتر فاصله پیمایش گروه و اگر پیدا شد آن را به BH منتقل نمایید.

۷-۱۲. برنامه ۷-۱۲ را بازنویسی کنید به نحویکه عملیات، HEADG1 را برای یافتن 'na' به مانند یک جفت کاراکتر، پیمایش کند. بررسی HEADG1 مشخص می‌سازد که 'na' مانند یک کلمه ظاهر نشده است، همانطورکه نشان داده شده است /cy/be/rn/au/ts/. دو راه حل ممکن چنین هستند. (الف) استفاده از SCASW دو بار اولین CASW از HEADG1 آغاز می‌شود و دومین SCASW از HEADG1+1 آغاز می‌شود، (ب) با استفاده از SCASB و با یافتن یک 'n' مقایسه بایت بعد از آن برای یافتن a.

۸-۱۲. یک فیلد ۵ بایتی حاوی مقدار شانزدهی C9CDCDCD BB تعریف کنید. با استفاده از MOVSB این فیلد را ۲۰ مرتبه در یک ناحیه ۱۰۰ بایتی کپی کرده و نتیجه را نمایش دهید.

فصل ۱۳

محاسبات ۱ : پردازش داده‌های دودویی

هدف : حاوی ضروریات جمع، تفریق، ضرب و تقسیم داده‌های دودویی

مقدمه

این فصل مطالبی راجع به جمع، تفریق و تقسیم و استفاده از داده‌های عددی علامتدار و بدون علامت را در بر دارد. این فصل همچنین مثالهای فراوانی دارد و آگاهی‌هایی به مسافرین ناآگاه در حیطه محاسبات کامپیوتری می‌دهد. فصل ۱۴ ضروریات خاص را با نحوه تبدیل بین دودویی و ASCII در بر دارد.

اگر چه ما عادت داریم که محاسبات را در قالب دهدهی (مبنای ده) انجام دهیم، یک میکروکامپیوتر محاسباتش را فقط در مبنای دو انجام می‌دهد. علاوه بر آن، محدودیت ثبات‌های ۱۶ بیتی در پردازشگرهای ماقبل 80386، مستلزم رفتارهای خاص برای مقادیر بزرگ خواهد شد.

دستوراتی که در این فصل توصیف خواهد شد :

ADC	جمع با رقم نقلی	IDIV	تقسیم علامتدار
ADD	جمع	IMUL	ضرب علامتدار
CBW	تبدیل بایت به کلمه	MUL	ضرب بدون علامت
CDQ	تبدیل دو کلمه‌ای به چهار کلمه	NEG	منفی کردن
CWD	تبدیل کلمه به دو کلمه‌ای	SBB	تفریق با فرض
CWDE	تبدیل کلمه به دو کلمه‌ای توسعه یافته	SUB	تفریق
DIV	تقسیم بدون علامت		

پردازش داده‌های علامتدار و بدون علامت

برخی فیلدهای عددی برای مثال، تعداد مشتری و روز ماه - بدون علامتند. برخی فیلدهای عددی علامتدار - برای مثال بدهکاری تراز مشتریان و اعداد جبری - ممکن است حاوی مقادیر مثبت یا منفی باشند. دیگر فیلدهای علامتدار - برای مثال، پرداختی به کارمندان و مقدار عدد پی - همیشه مثبت فرض می‌شود.

برای داده‌های بدون علامت، وقتی همه بیت‌ها، بیت داده است، یک ثبات ۱۶ بیتی می‌تواند حداکثر تا ۶۵۵۳۵ مقدار بگیرد. برای داده‌های علامت دار، وقتی بیت سمت چپ یک بیت علامت است ، ثبات حداکثر می‌تواند حاوی ۳۲۷۶۷+ باشد. اما توجه کنید که، دستورات ADD و SUB تفاوتی بین داده‌های علامت دار و بدون علامت قائل نیستند و فقط جمع و تفریق را انجام می‌دهند.

مثال زیر جمع دو عدد دودویی را نشان می‌دهد، با مقادیری که هم بدون علامت و هم علامتدار نشان داده شده است. عدد بالایی حاوی ۱ بیت ۱ در سمت چپ است، برای داده‌های بدون علامت، بیت‌ها ۲۴۹ هستند، در حالیکه برای داده‌های علامتدار، بیت‌ها ۷- را بیان می‌کنند. جمع پرچم سرریز و نقلی را تنظیم نمی‌کند.

دودویی	بدون علامت دهدهی	علامتدار دهدهی	OF	CF
11111001	249	-7		
+00000010	+2	+2		
11111011	251	-5	0	0

در این مثال، نتیجه دودویی جمع برای داده‌های علامت دار و بدون علامت یکسان است. اما، بیت‌ها در یک فیلد بدون علامت مقدار دهی ۲۵۱ را بیان می‌کنند، در حالیکه بیت‌ها در یک فیلد علامت دار مقدار دهدهی ۵- را بیان می‌کنند. در واقع، محتویات یک فیلد معنی می‌شوند هرآنطور که شما بخواهید بر طبق آن معنی کنید.

رقم نقلی محاسبات

یک عملیات محاسبات بیت علامت منتج را (0 یا 1) به پرچم نقلی می‌فرستد. اگر بیت علامت یک باشد، پرچم نقلی یک می‌شود. وقتی یک رقم نقلی در داده‌های بدون علامت رخ دهد نتیجه نامعتبر است. مثال زیر یک رقم نقلی ایجاد می‌کند. عملیات بر روی داده‌های بدون علامت نامعتبر است زیرا رقم‌نقلی خروجی از بیت داده، در حالیکه عملیات بر روی داده‌های علامت دار است معتبر است.

دودویی	بدون علامت دهدهی	علامتدار دهدهی	OF	CF
11111100	252	-4		
+00000101	+5	+5		
(1) 00000001	251	-5	0	1

سرریز محاسباتی

یک عملیات محاسباتی پرچم سرریز را وقتی یک می‌کند که رقم نقلی **به** بیت علامت، رقم نقلی خروجی نداشته باشد، یا در صورت عدم وجود رقم نقلی ورودی ،یک رقم نقلی **خروجی** رخ دهد. اگر یک سرریز بر روی داده‌های علامت دار رخ دهد، نتیجه نامعتبر است (زیرا یک سرریز بر روی بیت علامت داریم) مثال زیر، در جمع سبب ایجاد یک سرریز خواهد شد :

دودویی	بدون علامت دهدهی	علامتدار دهدهی	OF	CF
01111001	121	+121		
+00001011	+11	+11		
10000100	132	-124	1	0

در یک عملیات جمع ممکن است هم پرچم سرریز و هم پرچم رقم نقلی، یک شوند. در مثال بعد، رقم نقلی باعث نامعتبر شدن عملیات داده بدون علامت شده و سرریز، داده علامت دار را تخریب می‌کند. به طور خلاصه ، باید ایده خوبی از بزرگی اعدادی که برنامه دستکاری می‌کند داشته باشید و فیلدها را بر طبق آن تعریف و پردازش کنند.

دودویی	بدون علامت دهدهی	علامتدار دهدهی	OF	CF
01111001	121	+121		
+00001011	+11	+11		
10000100	132	-124	1	0

جمع و تفریق

دستورات ADD و SUB جمع و تفریق ساده داده‌های دودویی را انجام می‌دهند. قالب کلی برای SUB و ADD چنین است.همانطور که در بخش قبل گفته شد، عملیات ADD یا SUB بر روی پرچم‌های نقلی و سرریز تاثیر می‌گذارند. مانند دیگر دستورات، عملیات حافظه با حافظه مجاز نیست. مثال زیر با استفاده از ثبات AX را با CURRAIN را با RAINFILL

[label:]	ADD/SUB	register,register
[label:]	ADD/SUB	memory,register
[label:]	ADD/SUB	register,memory
[label:]	ADD/SUB	register,immediate
[label:]	ADD/SUB	memory,immediate

جمع می‌کند.

```
CURRAIN   DW   123              ; تعریف CURRAIN
RAINFALL  DW   25               ; تعریف RAINFALL
          MOV  AX,CURRAIN   ; انتقال CURRAIN به AX
          ADD  AX,RAINFALL  ; جمع RAINFALL با AX
          MOV  RAINFALL     ; انتقال AX به RAINFALL
```

همانطورکه در فصل ۱ توضیح داده شد، یک عدد دودویی منفی به شکل مکمل ۲ بیان می‌شود که با معکوس کردن همه بیت‌ها و جمع با یک حاصل می‌شود. شکل ۱-۱۳ مثالهایی از ADD به SUB برای پردازش مقادیر بایت و کلمه فراهم می‌سازد. روال B10ADD از ADD برای پردازش بایت‌ها و روال CIOSNB از SUB برای پردازش کلمات، استفاده می‌کنند.

```
TITLE     A13ADD (COM)ADD and SUB operations
          .MODEL SMALL
          .CODE
          ORG    100H
BEGIN:    JMP    SHORT A10MAIN
; ------------------------------------------------------
BYTE1     DB     64H              ;Data items
BYTE2     DB     40H
BYTE3     DB     16H
WORD1     DW     4000H
WORD2     DW     2000H
WORD3     DW     1000H
; ------------------------------------------------------
A10MAIN   PROC   NEAR             ;Main procedure:
          CALL   B10ADD           ;Call ADD routine
          CALL   C10SUB           ;Call SUB routine
          MOV    AX,4C00H         ;End processing
          INT    21H
A10MAIN   ENDP

;                 Examples of ADD bytes:
;                 ----------------------
B10ADD    PROC
          MOV    AL,BYTE1
          MOV    BL,BYTE2
          ADD    AL,BL            ;Register-to-register
          ADD    AL,BYTE3         ;Memory-to-register
          ADD    BYTE1,BL         ;Register-to-memory
          ADD    BL,10H           ;Immediate-to-register
          ADD    BYTE1,25H        ;Immediate-to-memory
          RET
B10ADD    ENDP

;                 Examples of SUB words:
;                 ----------------------
C10SUB    PROC
          MOV    AX,WORD1
          MOV    BX,WORD2
          SUB    AX,BX            ;Register-from-register
          SUB    AX,WORD3         ;Memory-from-register
          SUB    WORD1,BX         ;Register-from-memory
          SUB    BX,1000H         ;Immediate-from-register
          SUB    WORD1,256H       ;Immediate-from-memory
          RET
C10SUB    ENDP
          END    BEGIN
```

شکل ۱-۱۳ مثالهایی از *ADD* و *SUB*

سرریز

در عملیات محاسباتی مراقب سرریز باشید، مخصوصاً برای داده‌های علامت دار از آنجائیکه یک بایت برای فقط یک بیت علامت و ۷ بیت داده (از ۱۲۸- تا ۱۲۷+) فراهم شده است، یک عملیات محاسباتی براحتی از ظرفیت یک

ثبات یک بایتی تجاوز می‌کند و عمل جمع در ثبات AL که از ظرفیت آن تجاوز می‌کند ممکن نتایج خارج از انتظار داشته باشد. برای مثال، اگر AL حاوی 60H باشد و دستور زیر اجرا شود. ADD AL,20H

80H در AL تولید می‌شود. در جمع دو مقدار مثبت، ما انتظار مجموع مثبت داریم، اما عملیات پرچم سرریز و علامت را یک می‌کند. حاصل مقدار 80H، یا دودویی 10000000، یک مقدار منفی است، بجای ۱۲۸ حاصل ۱۲۸- است. مشکل اینجاست که ثبات AL برای جمع کوچک است که باید در ثبات AX کامل، همانطور که در بخش بعد نشان داده شده باشد.

بسط مقادیر در یک ثبات

در بخش قبلی، دیدیم که جمع 20H با مقدار 60H در AL سبب ایجاد یک حاصل نادرست گردید.

راه حل بهتر، در جمع کامل استفاده از AX است، دستور CBW (تبدیل بایت به کلمه) بطور اتوماتیک بیت علامت را از AL را (0 یا 1) به AH منتقل می‌کند و AX کامل را در اختیار می‌گذارد، توجه کنید که CBW به استفاده از AX محدود شده است. در مثال بعدی، CBW علامت (0) را از AL به AH توسعه می‌دهد که 0060H را در AX تولید می‌کند. سپس مثال 20H را با AX (نه AL) جمع می‌شود و نتیجه درست در AX تولید می‌شود : 0080H یا ۱۲۸+

```
                                                AX   AL
...                                             XX   60H
CBW                 ; AH در AL گسترش علامت     00   60
ADD   AX,20H        ; AX اضافه کردن به         00   80
```

این مثال ، همان نتیجه عددی در بخش قبلی را دارد، اما جمع در AX سرریز یا علامت منفی ندارد. هنوز، اگر چه یک کلمه کامل در AX برای یک بیت علامت و ۱۵ بیت داده دارد ولی AX نیز به مقادیر ۳۲۷۶۷- تا ۳۲۷۶۷+ محدود می‌شود.

بسط کلمات و دو کلمه‌ای‌ها

دستور CWD (تبدیل کلمه به کلمه) برای بسط یک کلمه مقدار علامت دار به دو کلمه‌ای و کپی کردن بیت علامت AX در DX استفاده می‌شود. در اینجا یک مثال آورده شده است :

```
MOV   AX,WORD1      ; AX انتقال کلمه به
CWD                 ; DX:AX توسعه کلمه به
```

دستور CWDE (تبدیل کلمه به دوکلمه‌ای برای 80386 و مابعد) برای توسعه یک کلمه علامت دار به دو کلمه‌ای با کپی کردن بیت علامت AX در EAX استفاده می‌شود. در اینجا یک مثال آورده شده است :

```
MOV   AX,WORD1      ; AX انتقال کلمه به
CWDE                ; EAX توسعه کلمه به
```

دستور CDQ (تبدیل دو کلمه‌ای به چهار کلمه‌ای، برای 80386 و مابعد) برای توسعه دو کلمه‌ای علامتدار به چهار کلمه‌ای با کپی کردن بیت علامت EAX به EDX استفاده می‌شود. در اینجا یک مثال آورده شده است :

```
MOV   EAX,DBWORD    ; EAX انتقال دو کلمه‌ای به
CDQ                 ; EDX:EAX توسعه دو کلمه‌ای به
```

محاسبات مقادیر دو کلمه‌ای

همانطور که دیدیم، مقادیر عددی بزرگ ممکن است از ظرفیت یک کلمه نیز تجاوز کند، در حقیقت به ظرفیت چندین کلمه نیاز داشته باشد. یک ضرورت مهم در محاسبات چند کلمه‌ای ترتیب بایت معکوس و کلمه معکوس است.

به یاد دارید که اسمبلر به طور خودکار محتویات کلمات عددی تعریف شده را به صورت ترتیب بایت معکوس تبدیل می‌کند، برای مثال، تعریف عدد 0134H، خواهد شد 3401H. ساده‌ترین روش تعریف یک دو کلمه‌ای DD است. مثال زیر مقادیر دو کلمه‌ای را جمع و ذخیره می‌کند :

```
DBWORD1  DD    0123BC62H      ; عریف دو کلمه‌ای
DBWORD2  DD    0012553AH      ;
DBWORD3  DD    0              ;
         . . .
         MOV   EAX,DBWORD1    ; جمع و
         ADD   EAX,DBWORD2    ; ذخیره
         MOV   DBWORD3,EAX    ; دو کلمه‌ای
```

اسمبلر به طور خودکار داده‌های تعریف شده را بترتیب بایت معکوس (و کلمه) مرتب می‌کند. اما برای برخی کاربردها، داده‌ها ممکن است به صورت مقادیر کلمه تعریف شوند. برای DBWORD1 که در مثال قبل تعریف شد، باید کلمات را مجاور ولی به ترتیب معکوس تعریف کنید.

```
DW  0BC62H
DW  0123H
```

سپس اسمبلر این تعاریف را به ترتیب بایت معکوس به شکل 162BC123011 تبدیل می‌کند ، که برای محاسبات دو کلمه‌ای مناسب است. بیایید دو روش انجام محاسبات بر روی این مقادیر را بررسی کنیم، که اولی ساده و خاص است، در حالیکه دومی علمی‌تر و عمومی‌تر می‌باشد.

در شکل ۲-۱۳، روال B10DWD جمع دو جفت کلمه (WORD1A و WORD1B) با یک جفت دوم (WORD2A و WORD2B) و ذخیره آن در جفت سوم (WORD3A و WORD3B) را مشخص می‌کند در حقیقت عملیات مقادیر را با هم جمع می‌کند ، مانند زیر :

```
مقدار اولیه   0123  BC62H
جمع           0012  553AH
حاصل          0136  119CH
```

به دلیل ترتیب بایت معکوس در حافظه، برنامه مقادیر با کلمات مجاور ولی معکوس تعریف می‌کند. BC620123 و 553A0012 سپس اسمبلر این مقادیر دو کلمه‌ای را در حافظه به ترتیب بایت معکوس صحیح ذخیره می‌سازد.

```
WORD1A و WORD1B :   62BC 2301
WORD2A و WORD2B :   3A55 1200
```

روال B10DWD ابتدا WORD2A را با WORD1A در AX جمع می‌کند (بخش پایین رتبه) و حاصل جمع را در WORD3A ذخیره می‌سازد. سپس WORD2B را با WORD1B جمع می‌کند (بخش بالا رتبه) و در AX می‌گذارد و البته رقم نقلی از جمع قبلی را نیز در نظر می‌گیرد. سپس حاصل جمع را در WORD3B ذخیره می‌سازد. حال بیایید عملیات را با جزئیاتش بررسی کنیم. اولین عملیات MOV و ADD بایت‌های AX را معکوس و کلمات سمت چپ را با هم جمع می‌کنند.

```
WORD1A :    BC62H
WORD2A :  + 553AH
جمع       (1)119CH  (ذخیره می‌شود WORD3A در 9C11)
```

چون مجموع WORD1A و WORD2A از ظرفیت ثبات AX تجاوز می‌کند، رقم نقلی رخ می‌دهد و پرچم نقلی یک می‌شود، سپس مثال کلمه سمت را جمع می‌کند، اما در این حالت بجای ADD از ADC (جمع با رقم نقلی) استفاده می‌کند. دستور ADC دو مقدار را جمع می‌کند و اگر پرچم CF قبلاً یک شده باشد، یک داده به حاصل جمع می‌افزاید :

```
TITLE     A13DBADD (COM)  Adding doublewords
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; -----------------------------------------------
WORD1A    DW      0BC62H          ;Data items
WORD1B    DW      0123H
WORD2A    DW      553AH
WORD2B    DW      0012H
WORD3A    DW      ?
WORD3B    DW      ?
; -----------------------------------------------
A10MAIN   PROC    NEAR            ;Main procedure
          CALL    B10DWD          ;Call 1st ADD
          CALL    C10DWD          ;Call 2nd ADD
          MOV     AX,4C00H        ;End processing
          INT     21H
A10MAIN   ENDP
;                 Example of ADD doublewords:
;                 ---------------------------
B10DWD    PROC
          MOV     AX,WORD1A       ;Add leftmost word
          ADD     AX,WORD2A
          MOV     WORD3A,AX
          MOV     AX,WORD1B       ;Add rightmost word
          ADC     AX,WORD2B       ;  with carry
          MOV     WORD3B,AX
          RET
B10DWD    ENDP
;                 Generalized add operation:
;                 --------------------------
C10DWD    PROC
          CLC                     ;Clear carry flag
          MOV     CX,02           ;Set loop count
          LEA     SI,WORD1A       ;Leftmost word
          LEA     DI,WORD2A       ;Leftmost word
          LEA     BX,WORD3A       ;Leftmost word of sum
C20:
          MOV     AX,[SI]         ;Move word to AX
          ADC     AX,[DI]         ;Add with carry to AX
          MOV     [BX],AX         ;Store word
          INC     SI              ;Adjust addresses for
          INC     SI              ;  next word to right
          INC     DI
          INC     DI
          INC     BX
          INC     BX
          LOOP    C20             ;Repeat for next word
          RET
C10DWD    ENDP
          END     BEGIN
```

شکل ۲-۱۳ جمع مقادیر چند کلمه‌ای

```
WORD1B     0123H
WORD2B     0012H
Plus carry + 1H
جمع        0136H   (در WORD3B به شکل 3901H ذخیره می‌شود)
```

با استفاده از DEBUG برای پیگیری محاسبات، می‌توانید حاصل 0136 را در ثبات AX و مقدار بایت معکوس 9C11H را در WORD3A و 3901H را در WORD3B ببینید.

همانطور که در شکل ۲-۱۳ می‌بینید، روال خبره‌تر C10DWD دستیابی به جمع مقادیر با هر طولی را ارائه می‌دهد، اگر چه در اینجا همان جفت کلماتی را که قبلاً بوده با هم جمع کرده است. روال از SI و DI و BX مانند ثبات‌هایی پایه

برای آدرسدهی WORD1A ، WORD2A و WORD3A استفاده می‌کند. این روال یک بار حلقه دستورات را برای هر زوج کلمه‌ای که باید جمع شوند انجام می‌دهد در این حالت دو بار - اولین حلقه دو کلمه سمت چپ را با هم جمع می‌کند و دومین حلقه دو کلمه سمت راست را با هم جمع می‌کند. چون دومین حلقه برای پردازش کلمات سمت راست است، آدرس ثبات‌های SI ، DI و BX، ۲ عدد افزایش می‌یابد. دو دستور INC این عملیات را برای هر ثبات انجام می‌دهند. INC به این دلیل بجای ADD استفاده می‌شود دستور 02 و ADD reg پرچم نقلی را پاک می‌کند و سبب تولید جواب نادرست می‌گردد، در حالیکه INC بر پرچم نقلی تاثیر ندارد.

در حلقه فقط یک دستور جمع ADC وجود دارد. در شروع یک دستور CLC (پاک کردن نقلی) برای صفر بودن پرچم در ابتدای حلقه تاکید دارد. نتیجه‌ای که در WORD3A و WORD3B ، AX می‌باشد همان نتیجه مثال قبلی است.

برای کار با این روش، (۱) کلمات را حتماً مجاور یکدیگر تعریف کنید، (۲) CX را با تعداد کلماتی که باید جمع شوند مقدار دهی نمایید و (۳) کلمات را از چپ به راست پردازش کنید.

برای تفریق چند کلمه‌ای، دستور معادل ADC , SBB (تفریق با رقم قرضی) است. براحتی می‌توان ADC را با SBB در روال C10DWD جایگزین نمود.

در اینجا قالب کلی ADC و SBB آورده شده است.

می‌توانید چهار کلمه‌ای‌ها را نیز با استفاده از روش جمع چند کلمه، جمع کنید، که دو جفت دو کلمه‌ای مجاور تعریف کنید و از ثبات EAX نیز استفاده نمایید.

[label:]	ADC/SBB	register,register
[label:]	ADC/SBB	memory,register
[label:]	ADC/SBB	register,memory
[label:]	ADC/SBB	register,immediate
[label:]	ADC/SBB	memory,immediate

ضرب

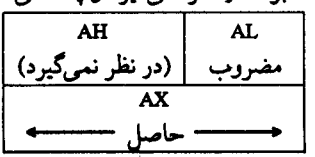
برای ضرب، دستور MUL بر داده‌های بدون علامت و IMUL (ضرب صحیح) بر داده‌های علامت دار عمل می‌کند. هر دو دستور بر رقم نقلی و سرریز تاثیر می‌گذارد. مانند برنامه نویسان، باید بر قالب داده‌هایی که پردازش می‌کنید کنترل داشته باشید و پاسخگوی نحوه انتخاب دستور ضرب مناسب باشید. قالب کلی دستور MUL چنین است :

[Label]:	MUL/IMUL	register/memory

عملیات ضرب پایه، بایت در بایت، کلمه در کلمه، و (در ۸۰۳۸۶ و مابعد) دو کلمه‌ای در دو کلمه‌ای است.

بایت در بایت

برای ضرب مقادیر یک بایتی، مضروب در ثبات AL و مضروب فیه در یک بایت حافظه یا یک ثبات قرار دارد. در دستور MUL DL، عملیات محتویات AL را در محتویات DL ضرب می‌کند. حاصل تولید شده در ثبات AX قرار دارد. عملیات هر داده‌ای که در AH بوده در نظر نمی‌گیرد و پاک می‌کند.



کلمه در کلمه

برای ضرب مقادیر یک کلمه‌ای، مضروب در ثبات AX و مضروب فیه در یک کلمه حافظه یا یک ثبات قرار دارد. در دستور MUL DX، عملیات محتویات AX را در محتویات DX ضرب می‌کند. حاصل یک دو کلمه‌ای است که به دو ثبات نیاز دارد : بخش بالا رتبه (سمت چپ) در DX و بخش پایین رتبه (سمت راست) در AX قرار دارد. عملیات داده‌ای که در DX بوده در نظر نمی‌گیرد و از بین می‌برد.

	DX	AX
قبل از ضرب	(در نظر نمی‌گیرد)	مضروب
بعد از ضرب	بالا رتبه	پایین رتبه

دو کلمه در دو کلمه

برای ضرب مقادیر دو کلمه‌ای مضروب در ثبات EAX و مضروب فیه یک دو کلمه‌ای در حافظه با یک ثبات قرار دارد.حاصل در جفت‌ثبات EDX:EAX تولیدمی‌شود. عملیات داده‌ای که در EDX بوده درنظرنمی‌گیرد و از بین‌می‌برد.

	EDX	EAX
قبل از ضرب	(در نظر نمی‌گیرد)	مضروب
بعد از ضرب	بالا رتبه	پایین رتبه

اندازه فیلدها

عملوند MUL یا IMUL فقط مضروب فیه را ارجاع می‌دهد، که اندازه فیلد را تعیین می‌کند. دستور فرض می‌کند که AL، AX یا EAX مضروب هستند، که به اندازه مضروب فیه بستگی دارد و در مثالهای زیر مضروب فیه در ثبات قرار دارد :

دستور	مضروب فیه	مضروب	حاصل
MUL CL	بایت	AL	AX
MUL BX	کلمه	AX	DX:AX
MUL EBX	دو کلمه	EAX	EDX:EAX

در مثالهای بعد ، مضروب فیه در حافظه تعریف شده است :

```
BYTE1   DB  ?
WORD1   DW  ?
DWORD1  DD  ?
```

دستور	مضروب فیه	مضروب	حاصل
MUL BYTE1	BYTE1	AL	AX
MUL WORD1	WORD1	AX	DX:AX
MUL WORD1	WORD1	EAX	EDX:EAX

ضرب بدون علامت MUL

هدف از دستور MUL ضرب داده‌های بدون علامت است، در شکل ۱۳-۱۳، B10MUL سه مثال استفاده از MUL را ارائه می‌دهد. بایت در بایت، کلمه در کلمه و کلمه در بایت. اولین مثال، 80H (۱۲۸) را در 40H (۶۴) ضرب می‌کند. حاصل در 2000H,AX (۸۱۹۲) است دومین مثال 10000000H را در ثبات‌های DX:AX تولید می‌کند.

سومین مثال ضرب کلمه در بایت است و باید BYTE1 در کلمه بسط داده شود. چون مقادیر بدون علامت در نظر گرفته می‌شود، مثال بیت‌های ثبات AH را صفر فرض می‌کند. (مشکل استفاده از CBW در اینجا این است که بیت سمت چپ AL می‌تواند یک باشد و انتقال بیت‌های یک به AH، یک مقدار بدون علامت بزرگتر را نتیجه می‌دهد)

حاصل در DX:AX مقدار 00400000H می‌باشد.
چهارمین مثال از EAX برای ضرب دو کلمه‌ای استفاده می‌کند.

ضرب علامت دار : IMUL

هدف دستور IMUL (ضرب صحیح)، ضرب داده‌های علامتدار است. در شکل ۳-۱۳، C10IMUL همان سه مثال مانند B10MUL را ارائه می‌دهد، اما MUL را با IMUL جایگزین نموده است. مثال اول 80H (یک عدد منفی) را در 40H(یک عدد مثبت) ضرب می‌کند. حاصل در ثبات AX مقدار E000H است. با استفاده از همان داده‌ها، MUL حاصل 2000H را ایجاد کرد، پس می‌توانید تفاوت بین استفاده از MUL و IMUL را ببینید. MUL، 80H را مانند ۱۲۸+ فرض می‌کند، در حالیکه IMUL، 80H را ۱۲۸- در نظر می‌گیرد. حاصل ۱۲۸- ضربدر ۶۴+ عبارت است از ۸۱۹۲- که برابر E000H است. (تبدیل E000H به بیتها را انجام دهید، بیتها را معکوس کنید، کمیت ۱ را بیفزائید و مقادیر بیتها را جمع کنید.)

دومین مثال 8000H (یک مقدار منفی) را در 2000H (مقدار مثبت) ضرب می‌کند. حاصل در DX:AX عبارتست از F0000000H که مقدار منفی است که MUL تولید کرده است.

سومین مثال، BYTE1 را به یک کلمه در ثبات AX توسعه می‌دهد. از آنجا که مقادیر علامت دار فرض می‌شود، مثال CBW را برای توسعه سمت چپ‌ترین بیت علامت به داخل ثبات AH به کار می‌برد : 80H در ثبات AL برابر FF80H در AX خواهد شد. چون مضروب فیه، WORD1، نیز منفی است، حاصل باید مثبت باشد و در واقع عبارت است از 004000000H در ثباتهای DX:AX - همان مقداری که از MUL حاصل شده بود - که طوری در نظر گرفته شده بود که دو عدد بدون علامت را ضرب کند. چهارمین مثال از EAX برای ضرب دو کلمه‌ای استفاده می‌کند.

در حقیقت، اگر مضروب و مضروب فیه علامت یکسانی دارند، MUL و IMUL یک حاصل را تولید می‌کنند. اما اگر مضروب و مضروب فیه، علامت مخالف داشته باشند، MUL یک حاصل مثبت و IMUL حاصل منفی تولید می‌کند. در نتیجه اینکه، برنامه شما قالب داده‌ها را باید بداند و از دستورات مناسب استفاده کند. ارزنده است اگر از DEBUG برای پیگیری این مثالها استفاده کنید.

انجام ضرب دو کلمه‌ای

ضرب مرسوم شامل ضرب بایت در بایت، کلمه در کلمه، یا دو کلمه‌ای در دو کلمه‌ای است. همانطور دیدیم، حداکثر مقدار علامت دار در یک کلمه ۳۲۷۶۷+ است. ضرب مقادیر بزرگتر در پردازشگرهای ما قبل ۸۰۳۸۶ شامل مراحل اضافی بود. روش مرسوم در این پردازشگرها ضرب هر کلمه بطور مجزا و سپس جمع حاصل با یکدیگر بود. مثال زیر یک عدد دهدهی چهار رقمی را در یک عدد دو رقمی ضرب می‌کند :

```
 1365
×  12
------
16380
```

چطور می‌توانید فقط اعداد دو رقمی را در هم ضرب کنید؟ سپس می‌توانید ۱۳ و ۶۵ را جداگانه در ۱۲ ضرب کنید، مانند این :

```
 65        13
×12       ×12
----      ----
780       156
```

حاصل دو حاصل را با هم جمع می‌کنیم، اما بخاطر بسپارید که ۱۳ در موقعیت صد قرار دارد و حاصل آن در حقیقت 15600 خواهد بود :

```
15600   (13×12×100)
 +980   (65×12)
-----
16380
```

یک برنامه اسمبلی در این تکنیک می‌تواند استفاده کند، بجز آنکه داده‌های حاوی کلماتی (۴ رقمی) در مبنای شانزده هستند. حال می‌خواهیم ضروریات ضرب دو کلمه‌ای در کلمه و دو کلمه‌ای در دو کلمه‌ای را بررسی نماییم.

```
TITLE     A13MULT (COM)  MUL and IMUL operations
          .MODEL SMALL
          .CODE
          ORG    100H
BEGIN:    JMP    SHORT A10MAIN
; ---------------------------------------------------------
BYTE1     DB     80H
BYTE2     DB     40H
WORD1     DW     8000H
WORD2     DW     2000H
DWORD1    DD     80000000
DWORD2    DD     20000000
; ---------------------------------------------------------
.386
A10MAIN   PROC   NEAR              ;Main procedure
          CALL   B10MUL            ;Call MUL  routine
          CALL   C10IMUL           ;Call IMUL routine
          MOV    AX,4C00H          ;End processing
          INT    21H
A10MAIN   ENDP
;                Examples of MUL:
;                ---------------
B10MUL    PROC
          MOV    AL,BYTE1          ;Byte x byte
          MUL    BYTE2             ;  product in AX

          MOV    AX,WORD1          ;Word x word
          MUL    WORD2             ;  product in DX:AX

          MOV    AL,BYTE1          ;Byte x word
          SUB    AH,AH             ;  extend multiplicand in AH
          MUL    WORD1             ;  product in DX:AX

          MOV    EAX,DWORD1        ;Doubleword x
          MUL    DWORD2            ;  doubleword
          RET
B10MUL    ENDP
;                Examples of IMUL:
;                ----------------
C10IMUL   PROC
          MOV    AL,BYTE1          ;Byte x byte
          IMUL   BYTE2             ;  product in AX

          MOV    AX,WORD1          ;Word x word
          IMUL   WORD2             ;  product in DX:AX

          MOV    AL,BYTE1          ;Byte x word
          CBW                      ;  extend multiplicand in AH
          IMUL   WORD1             ;  product in DX:AX

          MOV    EAX,DWORD1        ;Doubleword x
          IMUL   DWORD2            ;  doubleword
          RET
C10IMUL   ENDP
          END    BEGIN
```

شکل ۴-۱۳ ضرب مقادیر بدون علامت و علامتدار

دو کلمه‌ای در کلمه

در شکل ۴-۱۳، B10XMUL یک دو کلمه‌ای را در یک کلمه ضرب می‌کند. مضروب MULTCAN، شامل دو کلمه حاوی مقادیر 3206H و 2521H است. دلیل تعریف در DW به جای یک DD تسهیل آدرسدهی برای دستورالعمل‌های MOV است که کلمات را به ثبات AX منتقل می‌کند.

```
TITLE     A13DWMUL (COM)  Multiplication of doublewords
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ------------------------------------------------
MULTCAN   DW      2521H             ;Data items
          DW      3206H
MULTPLR   DW      0A26H
          DW      6400H
PRODUCT   DW      0
          DW      0
          DW      0
          DW      0
; ------------------------------------------------
A10MAIN   PROC    NEAR              ;Main procedure
          CALL    B10XMUL           ;Call 1st multiply
          CALL    C10CLEAR          ;Clear product
          CALL    D10XMUL           ;Call 2nd multiply
          MOV     AX,4C00H          ;End processing
          INT     21H
A10MAIN   ENDP
;                 Doubleword x word:
;                 ------------------
B10XMUL   PROC
          MOV     AX,MULTCAN        ;Multiply left word
          MUL     MULTPLR+2         ;  of multiplicand
          MOV     PRODUCT,AX        ;Store product
          MOV     PRODUCT+2,DX

          MOV     AX,MULTCAN+2      ;Multiply right word
          MUL     MULTPLR+2         ;  of multiplicand
          ADD     PRODUCT+2,AX      ;Add to stored product
          ADC     PRODUCT+4,DX
          RET
B10XMUL   ENDP
;                 Clear product area:
;                 -------------------
C10CLEAR  PROC
          MOV     PRODUCT,0000      ;Clear words
          MOV     PRODUCT+2,0000    ;  left to right
          MOV     PRODUCT+4,0000
          MOV     PRODUCT+6,0000
          RET
C10CLEAR  ENDP
;                 Doubleword x doubleword:
;                 ------------------------
D10XMUL   PROC
          MOV     AX,MULTCAN        ;Multiplicand word 1
          MUL     MULTPLR           ;  x multiplier word 1
          MOV     PRODUCT+0,AX      ;Store product
          MOV     PRODUCT+2,DX

          MOV     AX,MULTCAN        ;Multiplicand word 1
          MUL     MULTPLR+2         ;  x multiplier word 2
          ADD     PRODUCT+2,AX      ;Add to stored product
          ADC     PRODUCT+4,DX
          ADC     PRODUCT+6,00      ;Add any carry
          MOV     AX,MULTCAN+2      ;Multiplicand word 2
          MUL     MULTPLR           ;  x multiplier word 1
          ADD     PRODUCT+2,AX      ;Add to stored product
          ADC     PRODUCT+4,DX
          ADC     PRODUCT+6,00      ;Add any carry

          MOV     AX,MULTCAN+2      ;Multiplicand word 2
          MUL     MULTPLR+2         ;  x multiplier word 2
          ADD     PRODUCT+4,AX      ;Add to product
          ADC     PRODUCT+6,DX
          RET
D10XMUL   ENDP
          END     BEGIN
```

شکل ۴-۱۳ ضرب مقادیر چند کلمه‌ای

مقادیر به ترتیب کلمه معکوس تعریف می‌شوند و اسمبلر هر کلمه را به ترتیب بایت معکوس ذخیره می‌سازد. بنابراین MULTCAN، که یک مقدار تعریف شده 32062521H است بصورت 21250632H ذخیره می‌شود.

مضروب، MULTPLR+2 حاوی 6400H است. فیلدی که برای تولید حاصل در نظر گرفته شده، PRODUCT برای سه کلمه فراهم شده است. اولین عملیات MULTPLR +2,MUL را در کلمه سمت چپ از MULTCAN ضرب می‌کند، حاصل عبارت 0E80 E400H می‌باشد و در PRODUCT +2 و PRODUCT +4 ذخیره می‌شود. دومین MUL ، MULTPLR +2 را در کلمه سمت راست MULTCAN ضرب می‌کند. حاصل 138A 5800H می‌باشد. روال، دو حاصل ضرب را به صورت زیر با یکدیگر جمع می‌کند.

حاصل ضرب ۱ :	0000	0E80	E400
حاصل ضرب ۲ :	138A	5800	
مجموع :	138A	6680	E400

از آنجا که اولین جمع ممکن است یک رقم نقلی ایجاد کند، دومین جمع ADC (جمع با رقم نقلی) می‌باشد. چون داده‌های عددی در حافظه به صورت بایت معکوس ذخیره می‌شوند، PRODUCT، واقعاً حاوی 00E4 8066 8A13 می‌باشد. روال نیاز دارد که اولین کلمه از PRODUCT حاوی صفر باشد.

دو کلمه‌ای در دو کلمه‌ای

ضرب دو دوکلمه‌ای در پردازشگرهای ماقبل ۸۰۳۸۶ شامل چهار ضرب زیر می‌باشند :

مضروب فیه		مضروب
کلمه ۲	×	کلمه ۲
کلمه ۱	×	کلمه ۲
کلمه ۲	×	کلمه ۱
کلمه ۱	×	کلمه ۱

باید هر حاصل را در DX و AX جمع نمایید تا کلمه صحیح در حاصل انتهایی بوجود آید. در شکل ۴-۱۳ D10XMUL یک‌مثال‌ارائه‌می‌دهد. MULTCAN حاوی 32062521H است، MULTPLR حاوی 6400 0A26H و PRODUCT برای چهار کلمه در نظر گرفته شده است.

اگر چه منطق مشابه یا ضرب دو کلمه‌ای در کلمه است ولی این مسئله به یک خصیصه اضافی نیاز دارد. بعد از جفت ADD/ADC، دستورالعمل ADC دیگری وجود دارد که 0 را با PROPUCT جمع می‌کند.

اولین ADC خودش می‌تواند یک رقم نقلی ایجاد کند، که دستورات بعدی آن را پاک می‌کنند. دومین ADC، اگر رقم نقلی نباشد 0، اگر باشد یک را جمع خواهد کرد. آخرین زوج ADD/ADC ، ADC اضافی نیاز ندارد. از آنجائیکه PRODUCT به قدر کافی برای جواب نهایی بزرگ است، هیچ رقم نقلی نیاز ندارد. حاصلضرب نهایی 138A 687C 8E5C CCE6H می‌باشد، که در PRODUCT به ترتیب بایت معکوس ذخیره شده است. از DEBUG برای پیگیری این مثال استفاده نمایید.

دستورات ضرب خاص

پردازشگرهای ۸۰۲۸۶ و ما بعد قالب IMUL اضافی دارند که برای عملوندهای بلافصل فراهم شده‌اند و اجازه می‌دهد تا حاصل در پردازشگرهایی بجز AX تولید شوند. می‌توانید این دستورات را برای ضرب علامتدار یا بدون علامت استفاده نمایید، زیرا حاصل یکی خواهد بود. مقادیر باید یا طول یکسانی داشته باشند : ۱۶ یا (برای پردازشگرهای ۸۰۳۸۶ و ما بعد) ۳۲ بیت.

عملیات ۱۶ بیتی IMUL

برای IMUL ۱۶ بیتی ، اولین، عملوند (یک ثبات) حاوی مضروب است، و دومین عملوند (یک مقدار بلافصل مضروب می‌باشد. حاصل در اولین عملوند تولید می‌شود. یک حاصل که از ثبات تجاوز کند، سبب یک شدن پرچم نقلی و سرریز خواهد شد. قالب عمومی برای عملیات IMUL ۱۶ بیتی چنین است :

[Label:]	IMUL	register, immadiate

عملیات ۳۲ بیتی IMUL

عملیات IMUL ۳۲ بیتی سه عملوند دارند. دومین عملوند (حافظه) حاوی مضروب و سومین عملوند (یک مقدار بلافصل) حاوی مضروب فیه می‌باشد. حاصل در اولین عملوند تولید می‌شود (یک ثبات) قالب کلی برای عملیات ۳۲ بیتی IMUL چنین است.

[Label:]	IMUL	register, memary, immediate

عملیات IMUL ۳۲/۱۶ بیتی

پردازشگرهای ۸۰۳۸۶ و ما بعد قالب IMUL دیگری برای عملیات ۱۶ یا ۳۲ بیتی فراهم نموده‌اند. اولین عملوند (یک ثبات) حاوی مضروب و دومین عملوند (ثبات / حافظه) حاوی مضروب فیه است. حاصل در این عملوند تولید می‌شود.

[Label:]	IMUL	register, register/memory

در این جا مثالهایی از سه دستور IMUL آمده است :

اندازه	دستور	مضروب	مضروب فیه	حاصل
۱۶ بیت	IMUL DX,25	DX	25	DX
۳۲ بیت	IMUL ECX,MULTCAND,25	MULTCAND	25	DX
۱۶/۳۲بیت	IMUL BX,CX	BX	CX	BX

ضرب با شیفت دادن

برای ضرب کردن در توانی از ۲ (۲، ۴ ، ۸ و غیره) می‌توانید پردازش را بسادگی سریع‌تر انجام دهید با شیفت دادن به چپ تعداد بیت‌های لازم. در 8088/8086 یک شیفت بزرگتر از یک، نیاز دارد که تعداد شیفت را در ثبات CL قرار دهید. در مثالهای زیر، مضروب در AX است :

```
SHL   AX,01      ضرب در ۲ (یک شیفت به چپ) ;
MOV   CL,03      ضرب در ۸ (۳ شیفت به چپ) ;
SHL   AX,CL      ضرب در ۸ ۸۰۸۶/۸۰۸۸ ;
SHL   AX,03      ضرب در ۸ (۳ شیفت به چپ) در ۸۰۲۸۶ و مابعد ;
```

روتین زیر برای یک ۸۰۲۸۶ یا پردازشگر ما بعد می‌تواند جهت شیفت به چپ یک دو کلمه‌ای حاصل در ثبات‌های DX:AX مفید باشد. اگر چه برای ۴ بیت شیفت دادن در نظر گرفته شده، برای دیگر مقادیر نیز قابل تطبیق است.

```
SHL   DX,04      شیفت DX ۴ بیت به چپ ;
MOV   BL,AH      ذخیره AH در ; BL
SHL   AX,04      شیفت AX به چپ ۴ بیت ;
SHR   BL,04      شیفت BL به راست ۴ بیت ;
OR    DL,BL      جایگزینی ۴ بیتی از BL در DL ;
```

دقت کنید که اعداد نماد علمی را شیفت ندهید.

تقسیم

برای تقسیم، دستور DIV داده‌های بدون علامت و دستور IDIV داده‌های علامت دار را دستکاری می‌نماید. شما مسئول انتخاب دستور مناسب می‌باشید. قالب کلی برای DIV/IDIV چنین است :

[Label:]	DIV/IDIV	register/memory

دو عملیات تقسیم اصلی، بایت به کلمه، کلمه به دو کلمه و (در ۸۰۳۸۶ و ما بعد) دو کلمه‌ای به چهار کلمه می‌باشد.

بایت به کلمه

مقسوم در ثبات AX قرار دارد و مقسوم علیه عبارت از یک بایت در یک ثبات یا در حافظه است. بعد از تقسیم، باقیمانده در ثبات AH و خارج قسمت در ثبات AL می‌باشد. از آنجا که یک خارج قسمت تک بایتی خیلی کوچک است - اگر بدون علامت باشد (حداکثر FFH) ۲۵۵+ و اگر علامت دار باشد حداکثر 127(7FH)+ می‌باشد این عمل کاربرد محدودی دارد.

	AX	
	← مقسوم →	
قبل از تقسیم		
بعد از تقسیم	AH باقیمانده	AL خارج قسمت

کلمه به دوکلمه‌ای

مقسوم در جفت ثبات DX:AX و مقسوم علیه یک کلمه در یک ثبات یا حافظه می‌باشد. باقیمانده در ثبات DX و خارج قسمت در AX ذخیره می‌شود. خارج قسمت یک کلمه، چنانچه بدون علامت باشد می‌تواند حداکثر ۳۲۷۶۷+ (یا FFFFH) و در صورت علامت دار بودن حداکثر ۱۶۳۸۳+ (یا 7FFF) باشد.

	DX	AX
قبل از ضرب	بالا رتبه	پایین رتبه
بعد از ضرب	باقیمانده	خارج قسمت

دو کلمه‌ای به چهار کلمه‌ای

برای تقسیم یک دو کلمه‌ای به چهار کلمه‌ای ، مقسوم ثبات EDX:EAX و مقسوم علیه در یک دو کلمه‌ای در حافظه یا ثبات دیگر قرار دارد. عملیات باقیمانده را در EDX و خارج قسمت را در EAX ذخیره می‌سازد.

	EDX	EAX
قبل از ضرب	بالا رتبه	پایین رتبه
بعد از ضرب	باقیمانده	خارج قسمت

اندازه فیلدها

عملوند DIV/IDIV مقسوم را ارجاع می‌دهند، که اندازه فیلد را تعیین می‌کند. در مثالهای DIV زیر، مقسوم در یک ثبات است، که نوع عملیات را تعیین می‌کند :

عملیات	مقسوم	مقسوم علیه	خارج قسمت	باقیمانده
DIV C	byte	AX	AL	AH
DIV CX	Word	DX:AX	AX	DX
DIV EBX	doubleword	EDX:EAX	EAX	EDX

در مثالهای تقسیم زیر، مقسوم در حافظه تعریف شده است :

```
BYTE1   DD ?
WORD1   DW ?
DWORD1  DD ?
```

		مقسوم	مقسوم علیه	خارج قسمت	باقیمانده
...					
DIV	BYTE1	BYTE1	AX	AL	AH
DIV	WORD1	WORD1	DX:AX	AX	DX
DIV	DWORD1	DWORD1	EDX:EAX	EAX	EDX

```
TITLE     A13DIV (COM)  DIV and IDIV operations
,         .MODEL  SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ------------------------------------------------------
BYTE1     DB      80H             ;Data items
BYTE3     DB      16H
WORD1     DW      2000H
WORD2     DW      0010H
WORD3     DW      1000H
; ------------------------------------------------------
A10MAIN   PROC    NEAR            ;Main procedure
          CALL    B10DIV          ;Call DIV  routine
          CALL    C10IDIV         ;Call IDIV routine
          MOV     AX,4C00H        ;End processing
          INT     21H
A10MAIN   ENDP
;                 Examples of DIV:
;                 ---------------
B10DIV    PROC
          MOV     AX,WORD1        ;Word / byte
          DIV     BYTE1           ;  rmdr:quot in AH:AL
          MOV     AL,BYTE1        ;Byte / byte
          SUB     AH,AH           ;  extend dividend in AH
          DIV     BYTE3           ;  rmdr:quot in AH:AL

          MOV     DX,WORD2        ;Doubleword / word
          MOV     AX,WORD3        ;  dividend in DX:AX
          DIV     WORD1           ;  rmdr:quot in DX:AX
          MOV     AX,WORD1        ;Word / word
          SUB     DX,DX           ;  extend dividend in DX
          DIV     WORD3           ;  rmdr:quot in DX:AX
          RET
B10DIV    ENDP
;                 Examples of IDIV:
;                 ----------------
C10IDIV   PROC
          MOV     AX,WORD1        ;Word / byte
          IDIV    BYTE1           ;  rmdr:quot in AH:AL
          MOV     AL,BYTE1        ;Byte / byte
          CBW                     ;  extend dividend in AH
          IDIV    BYTE3           ;  rmdr:quot in AH:AL

          MOV     DX,WORD2        ;Doubleword / word
          MOV     AX,WORD3        ;  dividend in DX:AX
          IDIV    WORD1           ;  rmdr:quot in DX:AX
          MOV     AX,WORD1        ;Word / word
          CWD                     ;  extend dividend in DX
          IDIV    WORD3           ;  rmdr:quot in DX:AX
          RET
C10IDIV   ENDP
          END     BEGIN
```

شکل ۵-۱۳ تقسیم مقادیر بدون علامت و علامت دار

باقیمانده.

اگر ۱۳ را بر ۳ تقسیم کنید، حاصل $4\frac{1}{3}$ خواهد بود، در حالیکه خارج قسمت ۴ است و باقیمانده صحیح ۱ می‌باشد، توجه کنید که یک محاسبه‌گر (و یک زبان برنامه نویسی سطح بالا)، خارج قسمت ...4.33 را ارائه می‌دهد که حاوی یک بخش صحیح (۴) و یک بخش‌اعشاری (.۳۳۳) است. مقادیر ۱/۳ و ۰۳۳۳ اعشاری‌هستند، وقتی‌که باقیمانده یک است.

استفاده از DIV برای تقسیم بدون علامت

دستور DIV به منظور تقسیم داده‌های بدون علامت بکار می‌رود. در شکل ۵-۱۳، روال B10DIV چهار مثال از تقسیم ارائه می‌دهد بایت به کلمه، بایت به بایت، کلمه به دو کلمه، و کلمه به کلمه. اولین مثال 2000H (۸۰۹۲) را بر 80H (۱۲۸) تقسیم می‌کند، باقیمانده 00H در AH و خارج قسمت در AL ،40H (۶۴) قرار می‌گیرد.

در دومین مثال لازم است که BYTE1 در یک کلمه بسط داده‌شود. زیرامقدار بدون علامت فرض‌می‌شود، بیت‌های ثبات AH همگی صفر در نظر گرفته می‌شود. باقیمانده در AH ، 12H می‌باشد و خارج قسمت در AL ، 05H است.

در سومین مثال، باقیمانده در DX، 1000H و خارج قسمت در AX ، 0080H می‌باشد. در چهارمین تقسیم لازم است که WORD1 به یک دو کلمه‌ای در ثبات DX بسط داده شود. بعد از تقسیم، باقیمانده در DX، 0000H و خارج قسمت در AX ، 0002H است.

استفاده از IDIV برای تقسیم علامتدار

دستور IDIV به منظور تقسیم کردن داده‌های علامتدار بکار می‌رود. در شکل ۵-۱۳، C10IDIV همان چهار مثال مانند B10DIV را آورده است، اما DIV با IDIV جایگزین گردیده است. اولین مثال 2000H (مثبت) را بر 80H (منفی) تقسیم می‌کند. باقیمانده در AH ، 00H است و خارج قسمت در AL ، C0H (۶۴-) می‌باشد. (با استفاده از همین داده‌ها DIV خارج قسمت ۶۴+ را نتیجه می‌دهد) در سه مثال IDIV مقادیر زیر را داریم :

مثال IDIV	(DX) باقیمانده	(AX) خارج قسمت
2	EE (-18)	FB (-5)
3	1000 (4096)	0080 (128)
4	0000	0002

فقط مثال ۴ حاصلی مانند جواب DIV تولید می‌کند. در حقیقت، اگر مقسوم و مقسوم علیه بیت علامت مشابهی داشته باشند، DIV و IDIV یک نتیجه تولید می‌کنند، اما اگر مقسوم و مقسوم علیه بیت‌های علامت متفاوتی داشته باشند، DIV یک خارج قسمت مثبت و IDIV یک خارج قسمت منفی ایجاد می‌کند. ممکن است ارزنده باشد اگر با استفاده از DEBUG این مثالها را پیگیری نمایید.

سرریز و وقفه‌ها

عملیات DIV و IDIV فرض می‌کند که خارج قسمت کوچک‌تر از مقسوم اصلی است. در نتیجه، عملیات براحتی می‌تواند یک سرریز ایجاد کند، وقتی چنین باشد وقفه رخ می‌دهد، با نتایجی غیر قابل پیش بینی. تقسیم بر صفر همیشه سبب یک وقفه خواهد شد.اما تقسیم بر یک خارج قسمتی مساوی با مقسوم ایجاد می‌کند و می‌تواندسبب‌یک‌وقفه‌شود.

یک قانون مفید : اگر مقسوم علیه یک بایت است، باید بزرگتر از بایت سمت چپ مقسوم (AH) باشد، اگر مقسوم علیه یک کلمه است باید محتویات آن بزرگتر از کلمه سمت چپ (DX) مقسوم باشد، اگر مقسوم علیه یک دو کلمه‌ای است، محتویات آن باید بزرگتر از دو کلمه‌ای سمت چپ (EDX) مقسوم باشد. مثالهای زیر استفاده از یک مقسوم علیه ۱ را مشخص می‌سازد :

عملیات تقسیم	مقسوم	مقسوم علیه	خارج قسمت
: کلمه بر بایت	0123	01	(1) 23
: دو کلمه‌ای بر کلمه	0001 4026	0001	(1) 4026

در هر دو حالت، خارج قسمت تولید شده از فضای در دسترسش تجاوز می‌کند ، شما باید قبل از عملیات DIV یا IDIV بررسی داشته باشید، همانطور که در دو مثال بعدی دیده می‌شود. در اولین، DIVBYTE یک مقسوم علیه یک بایتی و مقسوم در AX می‌باشد.

```
CMP   AH,DIVBYTE    ; مقایسه AH با مقسوم علیه
JNB   F50           ; بگذر اگر کوچکتر نیست
DIV   DIV BYTE      ; تقسیم کلمه بر بایت
```

در دومین مثال، DIVWORD یک مقسوم علیه یک کلمه‌ای و مقسوم در DX:AX قرار دارد.

```
CMP   DX,DIVWORD    ; مقایسه DX با مقسوم علیه
JNB   F70
DIV   DIVWORD       ; تقسیم دو کلمه‌ای بر کلمه
```

برای IDIV، منطق باید جوابگوی این حقیقت باشد که هر یک از مقسوم یا مقسوم علیه می‌تواند منفی باشد چون ارزش مطلق مقسوم علیه باید کوچکتر باشد، می‌توانید دستور NEG را برای برقراری موقتی مقادیر منفی به مثبت استفاده نمایید.

تقسیم بوسیله تفریق

اگر یک خارج قسمت برای مقسوم علیه بسیار بزرگ باشد، می‌توانید تفریق متوالی انجام دهید. بدین صورت که، مقسوم علیه را از مقسوم کم کنید، یک واحد به خارج قسمت بیفزایید و تفریق کردن را تا زمانی که مقسوم کوچکتر از مقسوم علیه گردد، ادامه دهید. در مثال زیر، مقسوم در ثبات AX، مقسوم علیه در ثبات BX و خارج قسمت در CX ظاهر شده است.

```
        SUB   CX,CX       ; پاک کردن خارج قسمت
D20 :   CMP   AX,BX       ; اگر مقسوم کوچکتر از مقسوم علیه است
        JB    D30         ; خروج
        SUB   AX,BX       ; مقسوم‌علیه را از مقسوم کم کن
        INC   CX          ; به خارج قسمت یکی اضافه کن
        JMP   D20         ; تکرار
D30 :   . . .             ; خارج قسمت در CX، باقیمانده در AX
```

در انتهای روال، CX حاوی خارج قسمت و AX حاوی باقیمانده است. مثال، برای مشخص ساختن روش، عمداً ابتدایی انتخاب شده است. اگر خارج قسمت در DX:AX است، دو عملیات زیر را انجام دهید :

۱) در D20، فقط اگر DX صفر است AX را با BX مقایسه کنید.

۲) بعد از دستور SUB ، SBB DX,00 قرار دهید.

توجه : یک خارج قسمت بسیار بزرگ و یک مقسوم علیه کوچک ممکن است سبب هزاران حلقه شود.

برای تقسیم بر توانی از ۲ (۲، ۴، ۸ و غیره) ممکن است بسادگی پردازش سریع‌تری داشته باشید، توسط شیفت به راست تعداد لازم بیت‌ها. برای ۸۰۸۶/۸۰۸۸، عمل شیفت بیشتر از یک مرتبه باید در ثبات CL تعداد لازم گذاشته شود. مثال زیر فرض می‌کند که مقسوم در AX است :

```
SHR   AX,01               ; تقسیم بر ۲ (شیفت به راست یک مرتبه)
MOV   CL,03   : ۸۰۸۸/۸۰۸۶ ; تقسیم بر ۸ (شیفت به راست سه مرتبه)
SHR   AX,CL   ;
SHR   CL,03   ; ۸۰۲۸۶ و بعد ; تقسیم بر ۸ (شیفت به راست سه مرتبه)
```

روتین زیر برای پردازشگرهای ۸۰۲۸۶ و بعد می‌تواند برای شیفت به راست دو کلمه‌ای جفت AX و DX، مفید باشد. اگر چه برای شیفت ۴ بیت مشخص شده، می‌تواند برای دیگر مقادیر زیر تطبیق داده شود.

```
SHR   AX,04    ; شیفت AX به راست ۴ بیت
MOV   BL,DL    ; ذخیره DL در BL
SHR   DX,04    ; شیفت DX به راست ۴ بیت
SHL   BL,04    ; شیفت B به چپ ۴ بیت
OR    DL,BL    ; جایگزینی ۴ بیت BL در Dl
```

معکوس کردن علامت

دستور NEG (منفی) علامت یک مقدار دودویی را معکوس می‌کند، مثبت به منفی یا برعکس. در واقع NEG بیتها را معکوس می‌کند و سپس یک واحد به آن می‌افزاید. قالب کلی دستور NEG چنین است :

[Label:]	NEG	register/memory

مثالهای زیر را محاسبه کنید :

```
NEG   CL      ; 8 bits
NEG   BX      ; 16 bits
NEG   EDX     ; 32 bits
NEG   BINVAL  ; بایت یا کلمه در حافظه
```

معکوس سازی علامت یک مقدار ۳۲ بیتی (یا بزرگتر) مستلزم مراحل بیشتری است فرض کنید که جفت ثبات DX:AX دارای یک عدد دودویی ۳۲ بیتی است. از آنجا که NEG نمی‌تواند به طور همزمان روی جفت DX:AX عمل نماید، استفاده از آن سبب نتایج غیر صحیح می‌گردد.

از NOT برای معکوس ساختن بیت‌ها و از ADD و ADC برای جمع با ۱ استفاده نمایید.

```
NOT   DX      ; معکوس کردن بیتها
NOT   AX      ; معکوس کردن بیتها
ADD   AX.1    ; افزودن ۱ به AX
ADC   DX,0    ; افزودن رقم نقلی به DX
```

یک مسأله کوچک باقی می‌ماند. بسیار خوب است تا اعمال حسابی را روی داده‌های دودویی که خود برنامه تعریف می‌کند، اجرا نمایید. اما، داده‌هایی که از یک صفحه کلید وارد می‌شوند در قالب ASCII هستند. اگر چه داده‌های ASCII برای نمایش و چاپ مناسبند، لیکن برای اعمال محاسباتی نیاز به تنظیمات ویژه دارند - عنوانی که در فصل بعدی بحث خواهد شد.

پردازشگرهای داده عددی

این بخش مقدمه‌ای کلی بر پردازشگرهای داده عددی ارائه می‌دهد، بحث کامل خارج از حد این کتاب می‌باشد. برد سیستم حاوی یک سوکت برای یک پردازشگر داده عددی است که ما آن را کوپروسسور[1] می‌شناسیم. کوپروسسور ۸۰۸۷ با الحاق به یک ۸۰۸۸/۸۰۸۶ عمل می‌کند، یک ۸۰۲۸۷ با یک ۸۰۲۸۶، ۸۰۳۸۷ با یک ۸۰۳۸۶ و غیره.

یک کوپروسسور مجموعه دستورات خودش و سخت‌افزار ممیز شناور برای انجام عملیاتی مانند توان و لگاریتم و تریگنومتریک را داراست. ۸ ثبات ۸۰ بیتی ممیز شناور می‌توانند مقادیری بیشتر از توان ۱۰ تا ۴۰۰ را بیان نمایند. پردازش ریاضی کوپروسسور ۱۰۰ برابر سریع‌تر از پردازشگرهای عادی است. ۸۰۸۷ حاوی ۸ ثبات ۸۰ بیتی است، R1 تا R8، که شکل زیر را دارند :

۱- Coprocessor، مترجم

S	exponent		significand	
79	78	64	63	0

هر ثبات یک برچسب ۲ بیتی مرتبط دارد که مبنی بر وضعیت آن می‌باشد :

00 حاوی یک عدد معتبر

01 حاوی یک مقدار صفر

10 حاوی یک عدد نامعتبر

11 خالی است

کوپروسسور ۷ نوع از داده‌های عددی را تشخیص می‌دهد :

۱) کلمه صحیح : ۱۶ بیت از داده‌های دودویی.

S	number	
15	14	0

۲) صحیح کوتاه : ۳۲ بیت از داده‌های دودویی.

S	number	
31	30	0

۳) صحیح بزرگ : ۶۴ بیت از داده‌های دودویی

S	number	
63	62	0

۴) اعشاری کوتاه : ۳۲ بیت داده ممیز شناور

S	exponent		significand	
31	30	23	22	0

۵) اعشاری بزرگ : ۶۴ بیت داده ممیز شناور

S	exponent		significand	
63	62	52	51	0

۶) اعشاری کمکی : ۸۰ بیت داده ممیز شناور

S	exponent		significand	
79	78	64	63	0

۷) دهدهی فشرده، ۱۸ رقم دهدهی علمی

S	zeros		significand	
79	78	72	71	0

انواع ۱، ۲ و ۳ شکل مکمل دو، دودویی عمومی هستند. انواع ۴، ۵ و ۶ اعداد ممیز شناور را بیان می‌کنند. نوع ۷ حاوی ۱۸ رقم دهدهی هستند. می‌توانید هر یک از این قالب‌ها را از حافظه در یک ثبات کوپروسسور قرار دهد و می‌تواند محتویات ثبات را در حافظه ذخیره کند. اما، برای محاسبات، پردازشگر همه قالب‌ها را در ثبات‌هایش به قالب اعشاری کمکی تبدیل می‌کند. داده‌ها در حافظه به ترتیب بایت معکوس ذخیره می‌شوند.

پردازشگر یک عملیات خاص درخواست، می‌کند و داده‌های عددی را به کوپروسسور تحویل می‌دهد، که عملیات انجام می‌دهند و حاصل را بر می‌گردانند. برای اسمبل کردن، از پیش‌پردازنده مناسب 80X86 استفاده نمایید.

دستور INT 11H می‌تواند در تشخیص حضور یک کوپروسسور کمک نماید. عملیات وضعیت تجهیزات را در AX قرار می‌دهد، که بیت ۱ به معنی حضور یک، کوپروسسور است.

نکات کلیدی

- حداکثر مقادیر علامت‌دار برای انباشتگر ۱ بایتی ۱۲۷+ و ۱۲۸- است.
- برای جمع چند کلمه‌ای، از ADC برای محاسبهٔ هر رقم نقلی از ADD قبلی استفاده نمایید، اگر عملیات یک حلقه انجام می‌دهد، از CLC برای مقدار دهی اولیه پرچم نقلی با صفر استفاده نمایید.
- از MUL برای داده‌های بدون علامت و IMUL برای داده‌های علامت دار استفاده کنید.
- با MUL، اگر مضروب فیه یک بایت باشد، مضروب در AL است، اگر مضروب فیه یک کلمه باشد، مضروب AX است، اگر مضروب فیه یک دو کلمه‌ای باشد، مضروب در EAX است.
- شیفت به چپ (SHL یا SAL) می‌تواند بجای ضرب در توان ۲ استفاده شود.
- از DIV برای داده‌های بدون علامت و IDIV برای داده‌های علامت دار استفاده کنید.
- برای تقسیم، مراقب سرریز باشید. مقسوم علیه باید بزرگتر از محتویات AH باشد، اگر مقسوم علیه یک بایتی است، DX اگر مقسوم علیه یک کلمه است، یا EDX اگر مقسوم علیه یک دو کلمه‌ای است.
- برای DIV، اگر یک مقسوم علیه یک بایتی تعریف شود، مقسوم در AX است، اگر مقسوم علیه یک کلمه تعریف شود، مقسوم در DX:AX است، اگر مقسوم علیه یک دو کلمه‌ای تعریف شود، مقسوم در EDX:EAX خواهد بود.
- شیفت به راست برای تقسیم بر توانی از ۲، SHR برای داده‌های بدون علامت و SAR برای داده‌های علامت دار بکار می‌رود.

پرسش‌ها

۱-۱۳. برای هر دو داده‌های بدون علامت و علامت دار (الف) حداکثر مقدار در یک بایت چیست؟ و (ب)حداکثر مقدار یک کلمه چیست ؟

۲-۱۳. تفاوت بین رقم نقلی و سرریز در حاصل یک عملیات محاسباتی را توضیح دهید.

۳-۱۳. در جمع دودویی زیر جمع اعداد دودویی را نشان دهید، هم برای داده‌های علامت دار و هم بدون علامت، بعلاوه آنکه نحوه تنظیم پرچم نقلی و سرریز را نیز نشان دهید :

الف)		(ب)		(ج)
00110011		01110110		11010110
+00011000		+00011001		+01011001

۴-۱۳. برنامه شکل ۲-۱۳ را طوری بازنویسی کنید که بجای دو کلمه، سه جفت کلمه را با هم جمع نماید، کلمات اضافی را با نام WORD3A و WORD3B تعریف کنید و WORD3A و WORD3B قبلی را به WORD4A و WORD4B تغییر دهید.

برای پرسش ۸-۵، به داده‌های زیر اشاره شده است، با کلماتی که به ترتیب معکوس تعریف شده‌اند :

```
VALUE 1   DW   0153H
          DW   1624H
VALUE 2   DW   0328H
          DW   3C44H
RESULT    DW   0
          DW   0
          DW   0
```

۵-۱۳. دستوراتی برای جمع موارد زیر بنویسید : (الف) کلمه VALUE1 با کلمه VALVE2، (ب) دو کلمه‌ای شروع شده در VALUE1 با دو کلمه‌ای شروع شده در VALUE2.

۶-۱۳. تاثیر دستورات مرتبط زیر را توضیح دهید :

```
STC
MOV   DX,VALUE1
ADC   DX,VALUE2
```

۷-۱۳. دستوراتی برای ضرب موارد زیر کد نمایید (MUL) : الف) کلمه VALUE1 در کلمه VALUE2، (ب) دوکلمه‌ای که با VALUE1 آغاز می‌شود در کلمه‌ای که با VALUE2 آغاز می‌شود و حاصل برابر RESULT ذخیره کنید.

۸-۱۳. دستوراتی برای تقسیم (DIV) موارد زیر کدنویسی کنید (الف) کلمه VALUE1 بر ۳۶، (ب) دو کلمه‌ای که با VALUE1 آغاز می‌شود بر کلمه VALUE2.

۹-۱۳. کدام مقسوم علیه بغیر از صفر مسبب خطای سرریز می‌شود؟

۱۰-۱۳. به بخش "ضرب توسط شیفت" مراجعه کنید که شیفت به چپ ۴ بیت را مشخص می‌سازد. مثال را برای شیفت به چپ ۲ بیت بازنویسی کنید.

فصل ۱۶

محاسبات ۲ : پردازش داده‌های تعریف ASCII و BCD

هدف: بررسی قالب‌های داده ASCII و BCD، جهت انجام محاسبات بر روی این قالب‌ها و ارائه تبدیلات بین این قالب‌ها و دودویی .

مقدمه

سیستم متداول در محاسبات یک کامپیوتر، معمولاً سیستم دودویی است. همانطورکه در فصل ۱۳ ملاحظه گردید، تا وقتی یک برنامه خودش داده‌ها را تعریف می‌کند، در قالب دودویی مشکل عمده‌ای ایجاد نمی‌شود. اما، بسیاری از داده‌هایی که یک برنامه باید آنها را پردازش نماید به شکلی غیر از دودویی هستند. برای مثال، داده‌های عددی که یک برنامه از صفحه کلید وارد می‌کند به صورت کاراکترهای ASCII هستند، و در مبنای ده می‌باشند، بهمین صورت نیز نمایش مقادیر عددی روی صفحه در قالب ASCII می‌باشند.

یک قالب عددی مرتبط، ***کد دودویی دهدهی*** *(BCD)* است که گاهی‌اوقات مورداستفاده قرارگرفته و به شکل فشرده و غیر فشرده ظاهر می‌شود. کامپیوتر تعدادی دستورات که محاسبات ساده را سهولت می‌بخشد و تبدیلات بین قالب‌ها را انجام می‌دهد دارا است. در این فصل همچنین تکنیک‌هایی برای تبدیل داده ASCII به قالب دودویی برای انجام محاسبات ارائه شده است و تکنیک‌هایی برای تبدیل نتایج دودویی به قالب ASCII برای دیدن نیز وجود دارد. برنامه انتهای فصل بسیاری از موارد مطرح شده در فصول ۱ تا ۱۳ را در بر دارد. اگر در یک زبان سطح بالا مانند C برنامه نوشته باشید حسابرسی کامپایلر را برای مکانهای مبنا (دهدهی یا دودویی) استفاده کردید. اما کامپیوتر مکان مبنا در یک فیلد محاسباتی را تشخیص نمی‌دهد، بنابراین شما به عنوان یک برنامه نویس زبان اسمبلی باید این مکان را محاسبه کنید.

دستوراتی که در این فصل معرفی می‌شوند :

AAA تنظیم ASCII بعد از جمع	AAD تنظیم ASCII قبل از تقسیم
AAS تنظیم ASCII بعد از تفریق	DAA تنظیم دهدهی بعد از جمع
AAM تنظیم ASCII بعد از ضرب	DAS تنظیم دهدهی بعد از تفریق

داده‌ها در قالب دهدهی

تا اینجا، مقادیر عددی که با آنها سر و کار داشتیم دودویی یا ASCII بودند. سیستم PC قالب BCD را نیز پشتیبانی می‌کند، که برای برخی عملیات محاسباتی معدود مجاز می‌باشد. دو مورد استفاده BCD چنین است:

۱) BCD تقریب صحیح اعداد را بدون از دست دادن دقت ارائه می‌دهد، نکته‌ای که برای دستکاری بویژه دلار و سنت مفید است. (تقریب مقادیر اعداد دودویی که دلار و سنت را بیان می‌کنند مسبب از دست دادن دقت خواهد شد.)

۲) BCD اغلب یک قالب ساده‌تری برای انجام محاسبات بر مقادیر کوچک وارد شده از صفحه کلید یا خروجی بر صفحه یا چاپگر می‌باشد.

یک رقم BCD شامل ۴ بیت است که ارقام دهدهی ۰ تا ۹ را بیان می‌کنند :

Binary	BCD digit	Binary	BCD digit
0000	0	0101	5
0001	1	0110	6
0010	2	0111	7
0011	3	1000	8
0100	4	1001	9

می‌توانید ارقام BCD را بصورت فشرده، یا غیر فشرده استفاده کنید:

۱) *BCD غیر فشرده* حاوی یک رقم BCD تستها در ۴ بیت پایین (سمت راست) برای هر بایت، که چهار بیت بالایی صفر هستند. توجه کنید که قالب ASCII نیز غیر فشرده است.

۲) *BCD فشرده* حاوی دو رقم BCD، یک در ۴ بیت بالایی و دیگری در ۴ بیت پایینی است. این قالب عموماً برای محاسبات در کمک پردازنده‌های رقمی استفاده می‌شود و با پیش پردازنده DT در ۱۰ بایت تعریف می‌شود.

نمایش عدد دهدهی ۱۵۲۷ را در سه قالب دهدهی بررسی می‌کنیم :

طول	محتویات	قالب
چهار بایت	31 35 32 37	ASCII
چهار بایت	01 05 02 07	BCD غیر فشرده
۲ بایت	15 27	BCD فشرده

پردازشگرها محاسبات بر روی مقادیر ASCII و BCD را بصورت یک رقم در هر بار انجام می‌دهند. باید از دستورات خاص برای تبدیل دو قالب استفاده نمایید.

پردازش داده‌های ASCII

چون داده‌هایی را که از طریق صفحه کلید وارد می‌کنید به صورت ASCII هستند، نمایش یک مقدار دهدهی وارد شده مانند ۱۲۳۴ بصورت 31 32 33 34 H می‌باشد. اما انجام محاسبات بر روی مقادیر ASCII، مشمول استفاده از دستورات AAA و AAS می‌باشد :

```
[label] :   AAA   ; تنظیم ASCII بعد از جمع
[label] :   AAS   ; تنظیم ASCII بعد از تفریق
```

این دستورات بدون عملوند کد نویسی می‌شوند و بطور خودکار مقدار ASCII در ثبات AX را تنظیم می‌کنند. تنظیم باید انجام شود زیرا یک مقدار ASCII یک عدد مبنای ۱۰ غیر فشرده را بیان می‌کند، در حالیکه پردازشگر محاسبات مبنای ۲ را انجام می‌دهد.

جمع اعداد ASCII

نتیجه جمع اعداد ASCII ، 8(38H) و 4(34H) را ملاحظه کنید :

```
  38H
+ 34H
------
  6CH
```

حاصل جمع 6CH نه یک مقدار ASCII درست و نه یک مقدار دودویی صحیح می‌باشد. اما، ۶ سمت چپ را نادیده بگیرید و آن را با C سمت راست جمع کنید. C شانزدهی بعلاوه ۶ مساوی است با 12 H، که جواب درست به

صورت عدد دهدهی است. چرا جمع با ۶ ؟ زیرا اختلاف بین دهدهی (۱۰) و شانزدهی (۱۶) است. خوب، قدری ساده شد، و مشخص کننده روشی است که AAA عمل تنظیم را انجام می‌دهد. عملیات AAA، ۴ رقم سمت راست ثبات AL را بررسی می‌کند. اگر ارقام بین A و F بود یا پرچم نقلی معین (AF) یک بود، ۶ را با AL جمع می‌کند، ۱ را با ثبات AH جمع می‌کند و پرچم‌نقلی (CF) و نقلی‌معین را یک می‌کند. درهمه حالت‌ها، AAA ارقام سمت‌چپ AL را صفرمی‌کند.

بعنوان مثال، فرض کنید که AX حاوی 38H 00 و BX حاوی 34H 00 است. 38 در AL و 34 در BL بیانگر دو بایت ASCII است که باید جمع شوند. جمع ۸ و ۴ باید ۱۲ باشد. جمع و تنظیم چنین است.

```
ADD  AL,BL    ; جمع 34H با 38H، مساوی با 6CH
AAA           ; تنظیم جمع ASCII مساوی با 01 02H
```

چون بعد از ADD ارقام سمت راست در AL و C می‌باشد، AAA، ۶ را با AL جمع می‌کند، یک را با AH جمع می‌کند، پرچم‌های CF و AF را یک می‌کند، و ارقام سمت چپ AL را صفر می‌کند. حاصل در AX، 0102H است .

برای بیان ASCII، ۳ را در ارقام سمت چپ AH و AL قرار دهید تا 3132H یا ۱۲ دهدهی داشته باشید.

```
OR AX,3030H
```

در مثال بالا، پرچم AF بر نتیجه تاثیر می‌گذارد. عملیات ADD ، 39H را با 39H در AL جمع می‌کند، 72H حاصل می‌شود و AF را تنظیم می‌کند (زیرا رقم نقلی از بیت ۳، به بیت ۴ وجود دارد)، اگر چه مقادیر چهار بیت سمت راست، فقط ۲ هستند، زیرا AF یک شده ،AAA، ۶ را با AL و ۱ را با AH جمع می‌کند و ارقام شانزدهی سمت چپ در AL را پاک می‌کند. حاصل در AX، 0108 می‌باشد (یا 18).

عملیات فوق برای جمع اعداد ASCII یک بایتی مناسب است. اما جمع اعداد ASCII چندین بایتی، نیاز به حلقه دارند تا پردازش از چپ به راست (پایین رتبه، به بالا رتبه) انجام می‌گیرد و ارقام نقلی محاسبه شود. مثال در شکل ۱-۱۴ دو عدد ۳ بایتی ASCII را با هم جمع می‌کند، ASCVAL1 ، ASCVAL2 و حاصل چهار بایتی ASCTOT را تولید می‌کند. به نکات زیر توجه کنید :

- دستور CLC در شروع B10ADD پرچم CF را صفر می‌کند.
- بعد از B20 ، ADC برای جمع استفاده می‌شود زیرا یک ADD سبب ایجاد رقم نقلی می‌شود که باید با بایت بعدی (چپ) جمع شود.
- یک دستور MOVZX ، AH را در هر حلقه پاک می‌کند زیرا هر AAA ممکن است L را با AH جمع کند. اما ADC حساب هر رقم نقلی را دارد. توجه کنید که استفاده از XOR یا SUB برای صفر کردن AH پرچم CF را تغییر خواهد داد.
- وقتی حلقه کامل شود، AH (مساوی 00 یا 01) به بایت سمت چپ ASCTOT منتقل می‌شود. در انتها، ASCTOT حاوی 01020702H است. برای جایگزینی ASCII3 در هر بایت، برنامه C10CONV را فراخوانی می‌کند، که بر ASCTOT در حافظه حلقه دارد و هر بایت را با 30H، OR می‌کند. حاصل 32H 37 32 31، یا 72 12 دهدهی است، که برنامه قبل از خاتمه نشان خواهد داد. روتین از OR بعد از AAA برای جایگزین 3 سمت چپ استفاده نمی‌کند، زیرا OR پرچم نقلی را تنظیم می‌کند و تاثیر دستور ADC را تغییر می‌دهد. یک راه حل این است که تنظیمات پرچم را با گذاشتن بر روی پشته ذخیره نموده ، OR را اجرا کنید، و سپس پرچم را بازیابی کنید:

```
ADC    AL,[DI]    ; جمع با رقم نقلی
AAA               ; تنظیم برای ASCII
PUSHF             ; ذخیره پرچم‌ها
OR     AL,30H     ; جایگزینی ۳ ASCII
POPF              ; بازیابی پرچم‌ها
MOV    [BX],AL    ; ذخیره حاصل
```

```
TITLE     A14ASCAD (COM)  Adding ASCII numbers
          .MODEL SMALL
          .CODE
.386
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ---------------------------------------------------
ASCVAL1   DB      '548'             ;ASCII items
ASCVAL2   DB      '724'
ASCTOT    DB      '0000', '$'
; ---------------------------------------------------
A10MAIN   PROC    NEAR
          CALL    B10ADD
          CALL    C10CONV
          MOV     AH,09H            ;Request display
          LEA     DX,ASCTOT         ;  total
          INT     21H
          MOV     AX,4C00H          ;End processing
          INT     21H
A10MAIN   ENDP
;                 Add ASCII values:
;                 ----------------
B10ADD    PROC    NEAR
          CLC                       ;Clear carry flag
          LEA     SI,ASCVAL1+2      ;Initialize ASCII numbers
          LEA     DI,ASCVAL2+2
          LEA     BX,ASCTOT+3
          MOV     CX,03             ;Initialize 3 loops
B20:
          MOVZX   AX,[SI]           ;Load ASCII byte in AX
          ADC     AL,[DI]           ;Add (with carry)
          AAA                       ;Adjust for ASCII
          MOV     [BX],AL           ;Store sum
          DEC     SI
          DEC     DI
          DEC     BX
          LOOP    B20               ;Loop 3 times
          MOV     [BX],AH           ;At end, store carry
          RET
B10ADD    ENDP
;                 Convert binary to ASCII:
;                 -----------------------
C10CONV   PROC    NEAR
          LEA     BX,ASCTOT+3       ;Convert ASCTOT
          MOV     CX,04             ;  to ASCII
C20:
          OR      BYTE PTR[BX],30H
          DEC     BX
          LOOP    C20               ;Loop 4 times
          RET
C10CONV   ENDP
          END     BEGIN
```

شکل ۱-۱۴ جمع اعداد *ASCII*

تفریق اعداد ASCII

دستور AAS مانند AAA عمل می‌کند، AAS ارقام سمت راست (۴ بیت) AL را بررسی می‌کند. اگر رقم بین A و F باشد یا نقلی معین یک باشد، عملیات ۶ را از AL و ۱ را از AH کم می‌کند، و پرچم AF و CF را تنظیم می‌کند. در هر حال، AAS رقم سمت چپ شانزدهی AL را صفر می‌کند.

دو مثال بعد فرض می‌کند که ASCVAL1 حاوی 39H و ASCVAL2 حاوی 35H است. اولین مثال ASCVAL2(35H) را از ASCVAL1(39H) کم می‌کند. AAS نیازی ندارد تا تنظیم کند، زیرا رقم شانزدهی سمت راست کمتر از A است.

```
                                AX    AF   CF
MOV     AL,ASCVAL 1          ; 0039
SUB     AL,ASCVAL 2          ; 0004    0    0
AAS                          ; 0004    0    0
OR      AL,30H               ; 0034
```

دومین مثال ASCAL1 (39H) را از ASCVAL2 (35H) تفریق می‌کند. زیرا رقم سمت راست نتیجه، C شانزدهی است، AAS، ۶ را از AL و یک را از AH تفریق می‌کند، و پرچم‌های CF و AF را تنظیم می‌کند:

```
                                AX    AF   CF
MOV     AL,ASCVAL2           ; 0035
SUB     AL,ASCVAL1           ; 00FC    1    1
AAS                          ; FF06    1    1
```

حاصل FF06H بصورت مکمل دو یا ۴- است.

پردازش داده‌های BCD غیر فشرده

در ضرب و تقسیم اعداد ASCII لازم است که ابتدا اعداد را به قالب BCD غیر فشرده تبدیل کنید. می‌توانید از دستورات AAM و AAD برای انجام محاسبات مستقیم بر اعداد BCD غیر فشرده استفاده کنید:

[label :]	AAM	تنظیم ASCII بعد از ضرب ;
[label :]	AAD	تنظیم ASCII قبل از تقسیم ;

ضرب اعداد ASCII

دستور AAM نتیجه ضرب داده ASCII در ثبات AX را تصحیح می‌کند. اما باید قبلاً ۳ رقم شانزدهی سمت چپ هر بایت را پاک کنید، بنابراین مقدار به BCD3 غیر فشرده تبدیل می‌شود. نیز، چون تنظیم فقط برای یک بایت در هر بار انجام می‌شود، می‌توانید فقط فیلدهای یک بایتی را ضرب کنید و باید عملیات را مکرر در حلقه انجام دهید. فقط از عملیات MUL (ضرب بدون علامت) استفاده کنید.

AAM ، AL را بر ۱۰ (0AH) تقسیم می‌کند و خارج قسمت در AH و باقیمانده در AL ذخیره می‌شود. برای مثال فرض کنید AL حاوی 35H و CL حاوی 39H است. کد زیر محتویات AL را در CL ضرب می‌کند و حاصل به قالب ASCII تبدیل می‌شود:

دستور		توضیح	AX	CL
AND	CL,0FH	; تبدیل CL به 09	0035	09
AND	AL,0FH	; تبدیل AL به 05	0005	09
MUL	CL	; ضرب AL در CL	002D	06
AAM		; تبدیل AX به BCD غیر فشرده	0405	
OR	AX,3030H	; تبدیل AX به ASCII	3435	

عملیات MUL ۴۵ (002DH) را در AX تولید می‌کند. AAM این مقدار را بر 10 تقسیم می‌کند، خارج قسمت ۴ در AH و باقیمانده ۵ را در AL تولید می‌کند. دستور OR سپس مقدار BCD غیر فشرده را به قالب ASCII تبدیل می‌کند.

شکل ۲-۱۴ ضرب یک مضروب ASCII چهار بایتی در یک مضروب فیه ASCII یک بایتی را شرح می‌دهد. چون AAM فقط عملیات یک بایتی انجام می‌دهد، روال B10MULT در مضروب یک بایتی در هر بار از راست به چپ انجام می‌دهد. در خاتمه، حاصل BCD غیر فشرده 0108090105 خواهد بود، که یک حلقه در C10CONV به قالب ASCII صحیح تبدیل می‌کند و خواهد شد 3138391135 یا 18915 دهدهی برنامه قبل از خاتمه پردازش، حاصل را نمایش می‌دهد. اگر مضروب فیه بزرگتر از یک بایت باشد، باید حلقه دیگری که مراحل را برای مضروب فیه تکرار کند، فراهم کنید. در این حالت، ممکن است ساده‌تر باشد که داده ASCII، به قالب باینری تبدیل شود که در بخش بعدی گفته خواهد شد.

```
TITLE     A14ASCMU (COM)  Multiplying ASCII numbers
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     A10MAIN
;-----------------------------------------
MULTCAN   DB      '3783'          ;ASCII items
MULTPLR   DB      '5'
ASCPROD   DB      5 DUP(0), '$'
;-----------------------------------------
A10MAIN   PROC    NEAR
          CALL    B10MULT
          CALL    C10CONV
          MOV     AH,09H          ;Request display
          LEA     DX,ASCPROD
          INT     21H
          MOV     AX,4C00H        ;End processing
          INT     21H
A10MAIN   ENDP
;                 Multiply ASCII numbers:
;                 -----------------------
B10MULT   PROC    NEAR
          MOV     CX,04           ;Initialize 4 loops
          LEA     SI,MULTCAN+3
          LEA     DI,ASCPROD+4
          AND     MULTPLR,0FH     ;Clear ASCII 3
B20:
          MOV     AL,[SI]         ;Load ASCII character
          AND     AL,0FH          ;Clear ASCII 3
          MUL     MULTPLR         ;Multiply
          AAM                     ;Adjust for ASCII
          ADD     AL,[DI]         ;Add to
          AAA                     ;  stored
          MOV     [DI],AL         ;  product
          DEC     DI
          MOV     [DI],AH         ;Store product carry
          DEC     SI
          LOOP    B20             ;Loop 4 times
          RET
B10MULT   ENDP
;                 Convert product to ASCII:
;                 -------------------------
C10CONV   PROC    NEAR
          LEA     BX,ASCPROD+4    ;Right to left,
          MOV     CX,05           ;  5 bytes
C20:
          OR      BYTE PTR[BX],30H
          DEC     BX
          LOOP    C20             ;Loop 4 times
          RET
C10CONV   ENDP
          END     BEGIN
```

شکل ۲-۱۴ ضرب اعداد *ASCII*

تقسیم اعداد ASCII

دستور AAD تصحیح یک مقسوم ASCII را قبل از تقسیم فراهم می‌سازد. مانند AAM، ابتدا باید ۳ سمت چپ را از بایت‌های ASCII پاک کنید تا قالب BCD غیر فشرده ایجاد کنید. AAD برای یک مقسوم ۲ بایتی در AX ایجاد شده است. مقسوم علیه می‌تواند فقط یک بایت تک حاوی 01 تا 09 باشد. فرض کنید که AX حاوی مقدار ASCII، ۲۸ (32 38H) است و CL حاوی مقسوم علیه، 7 ASCII می‌باشد. دستورات زیر تنظیم و تقسیم را انجام می‌دهند :

دستور		توضیح	AX	CL
AND	CL,0FH	; تبدیل BCD غیره فشرده	32 38	07
AND	AX,0F0FH	; تبدیل BCD غیره فشرده	02 08	
AAD		; تبدیل به دودویی	00 1C	
DIV	CL	; تقسیم بر ۷	0004	

AAD ، AH را در 10 (0AH) ضرب می‌کند، حاصل 20 (14H) را با AL جمع می‌کند و AH را پاک می‌کند، حاصل، 001CH، بیان شانزدهی ۲۸ دهدهی است.

```
TITLE     A14ASCDV (COM)  Dividing ASCII numbers
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
;-----------------------------------------------------------
DIVDND    DB      '3698'          ;ASCII items
DIVSOR    DB      '4'
ASCQUOT   DB      4 DUP(0), '$'
;-----------------------------------------------------------
A10MAIN   PROC    NEAR
          CALL    B10DIV
          CALL    C10CONV
          MOV     AH,09H          ;Display
          LEA     DX,ASCQUOT      ;  quotient
          INT     21H
          MOV     AX,4C00H        ;End processing
          INT     21H
A10MAIN   ENDP
;                 Divide ASCII numbers:
;                 --------------------
B10DIV    PROC    NEAR
          MOV     CX,04           ;Initialize 4 loops
          SUB     AH,AH           ;Clear left byte of dividend
          AND     DIVSOR,0FH      ;Clear divisor of ASCII 3
          LEA     SI,DIVDND
          LEA     DI,ASCQUOT
B20:
          LODSB                   ;Load ASCII byte
          AND     AL,0FH          ;Clear ASCII 3
          AAD                     ;Adjust for divide
          DIV     DIVSOR          ;Divide
          STOSB                   ;Store quotient
          LOOP    B20             ;Four times?
          RET                     ;  yes, exit
B10DIV    ENDP
;                 Convert product to ASCII:
;                 ------------------------
C10CONV   PROC    NEAR
          LEA     BX,ASCQUOT      ;Leftmost byte,
          MOV     CX,04           ;  4 bytes
C20:
          OR      BYTE PTR[BX],30H  ;Clear ASCII 3
          INC     BX              ;Next byte
          LOOP    C20             ;Loop 4 times
          RET                     ;Exit
C10CONV   ENDP
          END     BEGIN
```

شکل ۳-۱۴ تقسیم اعداد *ASCII*

شکل ۳-۱۴ تقسیم یک مقسوم چهار بایتی را به یک مقسوم علیه یک بایتی است، روال B10DIV، مقسوم را از چپ به راست پیمایش می‌کند. LODSB یک بایت را از DIVDND می‌گیرد در AL (بر طبق SI) قرار می‌دهد، و STOSB بایت‌ها را در AL در QUOTNT (بر طبق DI) ذخیره می‌سازد. باقیمانده در ثبات AH مقیم می‌گردد، به طوری که ADD آن را در AL تنظیم خواهد کرد. خارج قسمت نهایی 00090204 است و باقیمانده در ثبات AH عبارت از 02 می‌باشد. روال C10CONV خارج قسمت در قالب ASCII را بصورت 30393234 (از چپ به راست) تبدیل می‌کند، برنامه خارج قسمت ASCII را، 0924، قبل از خاتمه نمایش می‌دهد.

اگر مقسوم علیه بزرگتر از یک بایت باشد، باید حلقهٔ دیگری برای مقسوم علیه در نظر بگیرید. بهتر است که، بخش بعدی "تبدیل ASCII به قالب باینری" را ملاحظه کنید.

پردازش داده‌های BCD فشرده

در مثال قبلی از تقسیم ASCII، خارج قسمت 0204 0009 خواهد بود. اگر این مقدار را فشرده کنید، و رقم سمت راست هر بایت را نگه دارید، حاصل 0924 خواهد شد، حال در قالب BCD فشرده شده است. می‌توانید جمع و تفریق را بر روی داده‌های BCD فشرده انجام دهید. بدین منظور، در دستور تنظیم DAA و DAS وجود دارد:

[label :]	DAA	تنظیم دهدهی بعد از جمع ;
[label :]	DAS	تنظیم دهدهی بعد از تفریق ;

DAA حاصل جمع دو مقدار BCD را در AL تصحیح می‌کند، و DAS حاصل تفریق آنها را تصحیح می‌کند. بار دیگر، باید فیلدهای BCD یک بایت در هر بار (دو رقم) را پردازش کنید.

جمع BCD در AL حاوی دو رقم ۴ بیتی است. اگر مقدار رقم سمت راست بیشتر از ۹ باشد یا پرچم AF یک باشد، DAA، ۶ را با AL جمع می‌کند و AF را تنظیم می‌کند. اگر مقدار در AL از 99H بیشتر باشد یا CF یک باشد، DAA ، 60H را با AL جمع می‌کند و CF را یک می‌کند. در غیر اینصورت، AF ، CF را پاک می‌کند. مثال زیر این روال را روشن می‌سازد.

فرض کنید مقادیر BCD ، 057836 و 069427 جمع می‌شوند. با صفر شدن پرچم CF، جمع با زوج ارقام سمت راست آغاز می‌شود.

	BCD	HEX	BINARY	STORED BCD
First ADC, clears CF	36	36	0011 0110	
	27	27	0010 0111	
	63	5D	0101 1101	
DAA adds 06H, sets AF			0000 0110	
			0110 0011	63
Second ADC, sets CF	78	78	0111 1000	
	94	94	1001 0100	
	(1)72	(1)0C	(1)0000 1100	
DAA adds 06H, sets AF			0000 0110	
			0001 0010	
DAA adds 60H (because			0110 0000	
CF set), sets CF			0111 0010	72
Third ADC, adds 1	05	05	0000 0101	
(because CF set),	06	06	0000 0110	
clears CF	1	1	0000 0001	
	12	0C	0000 1100	
DAA adds 06H, sets AF			0000 0110	
			0001 0010	12

جمع BCD، حال بطور صحیح بصورت 63 72 12 ذخیره می‌شود.

برنامه شکل ۴-۱۴ مثال جمع BCD که در بالا گفته شد، مشخص می‌سازد. روال B10CONV مقادیر ASCII و ASCVAL1 و ASCVAL2 را به مقادیر BCD، فشرده شده BCDVAL1 و BCDVAL2 تبدیل می‌کند. پردازش که از راست به چپ انجام می‌گیرد، می‌تواند از چپ به راست تنظیم شود. همچنین، پردازش کلمات، ساده‌تر از پردازش بایت‌ها است زیرا به دو بایت ASCII برای تولید یک بایت BCD فشرده نیاز دارید. اما استفاده از کلمات نیاز به تعداد زوجی از بایت در فیلدهای ASCII دارد.

روال C10ADD یک حلقه را سه بار برای جمع اعداد BCD فشرده در BCDSUM انجام می‌دهد. مجموع نهایی 00127263H خواهد بود، که می‌توانید با DEBUG - از D DS:114 استفاده کنید آن را پیگیری نمایید.

```
TITLE     A14BCDAD (COM)  Convert ASCII to BCD and add
          .MODEL SMALL
          .CODE
.386
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
;-----------------------------------------------
ASCVAL1   DB      '057836'          ;ASCII data items
ASCVAL2   DB      '069427'
BCDVAL1   DB      '000'             ;BCD data items
BCDVAL2   DB      "000'
BCDSUM    DB      4 DUP(0)
;-----------------------------------------------
A10MAIN   PROC    NEAR
          LEA     SI,ASCVAL1+4      ;Initialize for ASCVAL1
          LEA     DI,BCDVAL1+2
          CALL    B10CONV           ;Call convert routine
          LEA     SI,ASCVAL2+4      ;Initialize for ASCVAL2
          LEA     DI,BCDVAL2+2
          CALL    B10CONV           ;Call convert routine
          CALL    C10ADD            ;Call add routine
          MOV     AX,4C00H          ;End processing
          INT     21H
A10MAIN   ENDP
;                 Convert ASCII to BCD:
;                 --------------------
B10CONV   PROC
          MOV     CX,03             ;No. of words to convert
B20:      MOV     AX,[SI]           ;Get ASCII pair
          XCHG    AH,AL
          SHL     AL,04             ;Shift off
          SHL     AX,04             ;  ASCII 3s
          MOV     [DI],AH           ;Store BCD digits
          SUB     SI,02
          DEC     DI
          LOOP    B20               ;Three times?
          RET                       ;  yes, return
B10CONV   ENDP
;                 Add BCD numbers:
;                 ---------------
C10ADD    PROC
          XOR     AH,AH             ;Clear AH
          LEA     SI,BCDVAL1+2      ;Initialize
          LEA     DI,BCDVAL2+2      ;  BCD
          LEA     BX,BCDSUM+3       ;  addresses
          MOV     CX,03             ;3-byte fields
          CLC
C20:
          MOV     AL,[SI]           ;Get BCDVAL1 (or LODSB)
          ADC     AL,[DI]           ;Add BCDVAL2
          DAA                       ;Decimal adjust
          MOV     [BX],AL           ;Store in BCDSUM
          DEC     SI
          DEC     DI
          DEC     BX
          LOOP    C20               ;Loop 3 times
          RET
C10ADD    ENDP
          END     BEGIN
```

شکل ۴-۱۴ تبدیل و جمع اعداد *BCD*

تبدیل داده‌های ASCII به قالب دودویی

انجام محاسبات در قالب ASCII یا BCD فقط برای فیلدهای کوتاه مناسب است. برای اغلب اهداف محاسباتی، تبدیل چنین اعدادی به قالب دودویی عملی‌تر خواهد بود. در حقیقت، تبدیل از ASCII مستقیماً به دودویی ساده‌تر از تبدیل ASCII به BCD و سپس دودویی است.

تبدیلات ASCII به دودویی بر پایه این حقیقت است که یک عدد ASCII در مبنای ده می‌باشد و کامپیوتر محاسبات را در مبنای ۲ را انجام می‌دهد. در اینجا روال است ذکر شده است.

۱) بابایت سمت راست در فیلد ASCII آغاز کنید و پردازش را از دست به چپ انجام دهید.

۲) ۳ را از رقم شانزدهی سمت چپ هر بایت ASCII جدا کنید، که یک عدد BCD فشرده شده شکل می‌گیرد.

۳) اولین رقم BCD را در یک ضرب کنید، دومین را در 10 (0AH)، سومین را در 100 (64H)، و غیره و مجموع حاصل خواهد شد.

مثال زیر عدد ASCII، ۱۲۳۴ را به دودویی تبدیل می‌کند:

دهدهی		شانزدهی	
مرحله	حاصل	مرحله	حاصل
4 × 1 =	4	4 × 01H =	4H
3 × 10 =	30	3 × 0AH =	1EH
2 × 100 =	200	2 × 64H =	C8H
1 × 1000 =	1000	1 × 3E8H =	3E8H
مجموع	1234		04D2H

```
TITLE     A14ASCBI (COM)  Convert ASCII to binary format
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ------------------------------------------------------
ASCVAL    DB      '1234'          ;Data items
BINVAL    DW      0
ASCLEN    DW      4
MULFACT   DW      1
; ------------------------------------------------------
A10MAIN   PROC    NEAR            ;Main procedure
          CALL    B10CONV
          MOV     AX,4C00H
          INT     21H             ;End processing
A10MAIN   ENDP

B10CONV   PROC    NEAR
          MOV     BX,10           ;Mult factor
          MOV     CX,04           ;Count for loop
          LEA     SI,ASCVAL+3     ;Address of ASCVAL
B20:
          MOV     AL,[SI]         ;Select ASCII character
          AND     AX,000FH        ;Remove 3-zone
          MUL     MULFACT         ;Multiply by 10 factor
          ADD     BINVAL,AX       ;Add to binary
          MOV     AX,MULFACT      ;Calculate next
          MUL     BX              ;  10 factor
          MOV     MULFACT,AX
          DEC     SI              ;Last ASCII character?
          LOOP    B20             ;  no, continue
          RET                     ;  yes, return
B10CONV   ENDP
          END     BEGIN
```

شکل ۵-۱۴ تبدیل اعداد *ASCII* به قالب دودویی

بررسی کنید که آیا مجموع 04 D2H مساوی با عدد دهدهی ۱۲۳۴ می‌باشد. در شکل ۵-۱۴ روال BLOCONV عدد، ASCII، ۱۲۳۴ را به معادل دودویی تبدیل می‌کند. یک دستور LEA آدرس بایت سمت راست فیلد ASCII را مقدار دهی می‌کند، ASCVA + 3 در ثبات SI قرار می‌گیرد.

دستوری که در B20 است، بایت ASCII را به AL منتقل می‌کند MOV AL,[SI] عملیات با استفاده از آدرس ASCVAL+3 بایت سمت راست از ASCVAL را به AL کپی می‌کند. هر بار تکرار حلقه، SI را یکی می‌کاهد و بایت بعدی در سمت چپ را ارجاع می‌دهد. حلقه برای هر چهار بایت ASCVAL تکرار می‌شود. همچنین، هر تکرار

MULFACT را در 10 (0AH) ضرب می‌کند، مضروب فیه داده شده ۱ (10(0AH ، 100(64H) و غیره است. در خاتمه، BINVAL حاوی مقدار دودویی صحیح است، D209+H به ترتیب بایت معکوس، که با DEBUG می‌توانید آن را پیگیری نمایید.

در روتینی که برای وضوح بیشتر کدنویسی شده‌است، برای پردازش سریع‌تر، مضروب‌فیه باید در ثبات DI ذخیره شود.

تبدیل داده‌های باینری به قالب ASCII

برای چاپ یا نمایش حاصل محاسبات دودویی، باید آنها را به قالب ASCII تبدیل کنید. عملیات لازم مراحل قبلی است. بجای ضرب کردن، تقسیم اعداد باینری بر ۱۰ (0AH) تا هنگامیکه خارج قسمت کمتر از 10 شود ادامه می‌یابد. هر باقیمانده می‌تواند فقط 0 تا 9 باشد، بترتیب عدد ASCII را تولید می‌کند. بعنوان مثال 4D2H را به قالب دهدهی تبدیل می‌کنیم.

	تقسیم بر 10	خارج قسمت	باقیمانده
A	4D2	7B	4
A	7B	C	3
A	C	1	7

از آنجا که خارج قسمت (۱) کمتر از مقسوم علیه (0AH) است. عملیات کامل است. باقیمانده از راست به چپ، با آخرین خارج قسمت، حاصل BCD است: ۱۲۳۴. همه این ارقام با افزودن ۳ به صورت ASCII، 31323334 ذخیره خواهد شد.

```
TITLE     A14BINAS (COM)  Convert binary data to ASCII
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ------------------------------------------------------
ASCVAL    DB      4 DUP(' '), '$' ;Data items
BINVAL    DW      04D2H
; ------------------------------------------------------
A10MAIN   PROC    NEAR            ;Main procedure
          CALL    B10CONV
          MOV     AH,09H          ;Display
          LEA     DX,ASCVAL       ;  ASCII value
          INT     21H
          MOV     AX,4C00H        ;End processing
          INT     21H
A10MAIN   ENDP

B10CONV   PROC    NEAR
          MOV     CX,0010         ;Division factor
          LEA     SI,ASCVAL+3     ;Address of ASCVAL
          MOV     AX,BINVAL       ;Get binary amount
B20:.
          CMP     AX,CX           ;Value < 10?
          JB      B30             ;  yes, exit
          XOR     DX,DX           ;Clear upper quotient
          DIV     CX              ;Divide by 10
          OR      DL,30H
          MOV     [SI],DL         ;Store ASCII character
          DEC     SI
          JMP     B20
B30:
          OR      AL,30H          ;Store last quotient
          MOV     [SI],AL         ;  as ASCII character
          RET
B10CONV   ENDP
          END     BEGIN
```

شکل ٦-۱۴ تبدیل اعداد باینری به قالب اصلی

برنامه شکل ۶-۱۴ عدد دودویی 04 D2H را به قالب ASCII تبدیل می‌کند. روال B10CONV عدد دودویی مرتباً بر ۱۰ تقسیم می‌کند ، تا آخرین خارج قسمت کمتر از 10 (0AH) شود، و رقم شانزدهی تولید شده در قالب ASCII را بصورت 34 33 32 31 ذخیره می‌شود، که برنامه آن را قبل از خاتمه نشان می‌دهد. اگر برنامه را ایجاد و پیگیری کنید، ممکن است مفید باشد.

تغییر مکان و تقریب کردن یک حاصل

فرض کنید حاصلی که بدست آورده‌اید شامل ۳ رقم اعشار باشد و بخواهید آن را تقریب کنید و به دو رقم اعشار تقلیل دهید. بعنوان مثال، اگر حاصل ۱۷۳۸۵ باشد ، ۵ را با موقعیت اعشار سمت راست جمع کنید، و یک رقم به راست تغییر مکان دهید :

حاصل:	17.385
جمع با ۵ :	+0.005
حاصل تقریب می‌باشد:	17.390 = 17.39

اگر (الف) حاصل ۱۷۳۸۵۵ بود، ۵۰ را جمع کنید و ۲ رقم تغییر مکان دهید، و اگر (ب) حاصل ۱۷۳۸۵۵۵ بود، ۵۰۰ را جمع کنید و ۳ رقم تغییر مکان دهید:

(الف)		(ب)	
17.3855		17.38555	
0.0050		0.00500	
17.3905	= 17.39	17.39055	= 17.39

یک عدد با ۶ رقم اعشار نیاز به جمع 5.000 و تغییر مکان ۴ رقم دارد و الی آخر. حال، چون یک کامپیوتر به طول معمول داده‌های دودویی را پردازش می‌کند، عدد ۱۷.۳۸۵ بصورت 43E9H ظاهر می‌شود. جمع ۵ با 43E9H حاصلی برابر با 43EEH خواهد داشت، یا ۱۷۳۹۰ قالب دهدهی خواهد شد. اما تغییر مکان یک رقم دودویی 21F7H یا ۸۶۹۵ را نتیجه می‌دهد - در واقع، تغییر مکان مقدار را نصف می‌کند. چیزی که ما نیاز داریم عبارت از تغییر مکانی است که معادل با تغییر مکان یک رقم دهدهی به سمت ***راست*** باشد. می‌توانید این تغییر مکان را با تقسیم بر ۱۰ یا A مبنای شانزده انجام دهید: 43EEH تقسیم بر 0AH مساوی است با 6CBH. حاصل تبدیل 6CBH به مبنای ۱۰، عدد ۱۷۳۹ خواهد بود. حاصل کافیست نقطهٔ اعشار را در مکان صحیح ۱۷.۳۹ درج کنید و می‌توانید یک مقدار تغییر مکان داده شده و تقریب شده را نمایش دهید.

با این روش، می‌توانید هر عدد دودویی را گرد کنید و تغییر مکان دهید. برای سه رقم اعشاری، ۵ را اضافه کنید و بر ۱۰ تقسیم نمایید. برای چهار رقم اعشار، ۵۰ را اضافه کنید و بر ۱۰۰ تقسیم نمایید. ممکن است به یک الگو توجه کرده باشید. فاکتور گرد کردن (۵ ، ۵۰، ۵۰۰ و غیره) همواره، نصف فاکتور تغییر مکان (۱۰، ۱۰۰، ۱۰۰۰ و غیره) است.

البته یک نقطهٔ اعشار بر یک عدد دودویی اعمال می‌شود و به طور معمول وجود ندارد.

برنامه: تبدیل داده‌های ASCII

برنامه شکل ۷-۱۴ به کاربران امکان‌می‌دهد تاتعداد ساعات کارکرد و نرخ‌مزد کارمندان را واردکنند و دستمزد محاسبه شده را نمایش دهد. برای اختصار، برنامه برخی از تست‌های خطا را حذف می‌کند. روال‌ها به صورت زیر می‌باشند :

A10MAIN روالهایی را برای ورود داده‌ها و محاسبات نرخ‌ها، مقدار دهی می‌کند.

B10INPT ساعت‌ها و نرخ پرداختی را در قالب ASCII از صفحه کلید می‌پذیرد. این مقادیر ممکن است یک نقطه اعشار داشته باشد.

C10HOUR مقدار دهی تبدیل ساعات ASCII به دودویی .

D10RATE مقدار دهدهی تبدیل نرخ ASCII به دودویی .

```
TITLE     A14SCREMP (EXE) Enter hours and rate, display wage
          .MODEL SMALL
          .STACK 64
          .DATA
LEFCOL    EQU    28                ;Equates for screen
RITCOL    EQU    52                ;  locations
TOPROW    EQU    10
BOTROW    EQU    14

HRSPAR    LABEL  BYTE              ;Hours parameter list:
MAXHLEN   DB     6                 ;
ACTHLEN   DB     ?                 ;
HRSFLD    DB     6 DUP(?)          ;

RATEPAR   LABEL  BYTE              ;Rate parameter list:
MAXRLEN   DB     6                 ;
ACTRLEN   DB     ?                 ;
RATEFLD   DB     6 DUP(?)          ;

MESSG1    DB     'Hours worked? '
MESSG2    DB     'Rate of pay?  '
MESSG3    DB     'Wage =  '
ASCWAGE   DB     10 DUP(30H), 13, 10
MESSG4    DB     'Press any key to continue or Esc to quit'

ADJUST    DW     ?                 ;Data items
BINVAL    DW     00
BINHRS    DW     00
BINRATE   DW     00
COL       DB     00
DECIND    DB     00
MULT10    DW     01
NODEC     DW     00
ROW       DB     00
SHIFT     DW     ?
TENWD     DW     10
; ------------------------------------------------------------
.386
          .CODE
A10MAIN   PROC   FAR
          MOV    AX,@data          ;Initialize DS
          MOV    DS,AX             ;  and ES registers
          MOV    ES,AX
          CALL   Q10SCR            ;Clear screen
A20LOOP:
          CALL   Q30WIN            ;Clear window
          CALL   Q20CURS           ;Set cursor
          CALL   B10INPT           ;Accept hours & rate
          CALL   C10HOUR           ;Convert hours to binary
          CALL   D10RATE           ;Convert rate to binary
          CALL   E10MULT           ;Calculate wage, round
          CALL   F10WAGE           ;Convert wage to ASCII
          CALL   G10DISP           ;Display wage
          CALL   H10PAUS           ;Pause for user
          CMP    AL,1BH            ;Esc pressed?
          JNE    A20LOOP           ;  no, continue
          CALL   Q10SCR            ;  yes, clear screen
          MOV    AX,4C00H          ;End processing
          INT    21H
A10MAIN   ENDP
;                  Input hours & rate:
;                  -------------------
B10INPT   PROC   NEAR
          MOV    ROW,TOPROW+1      ;Set cursor
          MOV    COL,LEFCOL+3
          CALL   Q20CURS
          INC    ROW
          MOV    AH,40H            ;Request display
```

شکل الف ۷-۱۴ نمایش دستمزد کارمندان

```
                MOV     BX,01           ;File handle
                MOV     CX,14           ;No. of characters
                LEA     DX,MESSG1       ;Prompt for hours
                INT     21H
                MOV     AH,0AH
                LEA     DX,HRSPAR       ;Accept hours
                INT     21H
                MOV     COL,LEFCOL+3    ;Set column
                CALL    Q20CURS
                INC     ROW
                MOV     AH,40H          ;Request display
                MOV     BX,01           ;File handle
                MOV     CX,14           ;No. of characters
                LEA     DX,MESSG2       ;Prompt for rate
                INT     21H
                MOV     AH,0AH
                LEA     DX,RATEPAR      ;Accept rate
                INT     21H
                RET
B10INPT         ENDP
;                       Process hours:
;                       -------------
C10HOUR         PROC    NEAR
                MOV     NODEC,00
                MOVZX   CX,ACTHLEN
                LEA     SI,HRSFLD-1     ;Set right position
                ADD     SI,CX           ;  of hours
                CALL    J10ASBI         ;Convert to binary
                MOV     AX,BINVAL
                MOV     BINHRS,AX
                RET
C10HOUR         ENDP
;                       Process rate:
;                       ------------
D10RATE         PROC    NEAR
                MOVZX   CX,ACTRLEN
                LEA     SI,RATEFLD-1    ;Set right position
                ADD     SI,CX           ;  of rate
                CALL    J10ASBI         ;Convert to binary
                MOV     AX,BINVAL
                MOV     BINRATE,AX
                RET
D10RATE         ENDP
;                       Multiply, round, and shift:
;                       --------------------------
E10MULT         PROC    NEAR
                MOV     CX,05
                LEA     DI,ASCWAGE      ;Set ASCII wage
                MOV     AX,3030H        ;  to 30s
                CLD
                REP STOSW
                MOV     SHIFT,10
                MOV     ADJUST,00
                MOV     CX,NODEC
                CMP     CL,06           ;If more than 6
                JA      E40             ;  decimals, error
                DEC     CX
                DEC     CX
                JLE     E30             ;Bypass if 0, 1, 2 decs
                MOV     NODEC,02
                MOV     AX,01
E20:
                MUL     TENWD           ;Calculate shift factor
                LOOP    E20
                MOV     SHIFT,AX
                SHR     AX,1            ;Calculate round value
                MOV     ADJUST,AX
```

شکل ب ۷-۱۴ نمایش دستمزد کارمندان

```
E30:
            MOV     AX,BINHRS
            MUL     BINRATE             ;Calculate wage
            ADD     AX,ADJUST           ;Round wage
            ADC     DX,00
            CMP     DX,SHIFT            ;Product too large
            JB      E50                 ;  for DIV?
E40:
            XOR     AX,AX               ;Clear AX
            JMP     E70
E50:
            CMP     ADJUST,00           ;Shift required?
            JZ      E80                 ;  no, bypass
            DIV     SHIFT               ;Shift wage
E70:        XOR     DX,DX               ;Clear remainder
E80:        RET
E10MULT     ENDP
;                   Convert wage to ASCII:
;                   ----------------------
F10WAGE     PROC    NEAR
            LEA     SI,ASCWAGE+7        ;Set decimal point
            MOV     BYTE PTR[SI],'.'
            ADD     SI,NODEC            ;Set right start pos'n
F30:
            CMP     BYTE PTR[SI],'.'
            JNE     F40                 ;Bypass if at dec pos'n
            DEC     SI
F40:
            CMP     DX,00               ;If DX:AX < 10,
            JNZ     F50
            CMP     AX,0010             ;  operation finished
            JB      F60
F50:
            DIV     TENWD               ;Remainder is ASCII digit
            OR      DL,30H
            MOV     [SI],DL             ;Store ASCII character
            DEC     SI
            SUB     DX,DX               ;Clear remainder
            JMP     F30
F60:
            OR      AL,30H              ;Store last ASCII
            MOV     [SI],AL             ;  character
            RET
F10WAGE     ENDP
;                   Display wage:
;                   -------------
G10DISP     PROC    NEAR
            MOV     COL,LEFCOL+3        ;Set column
            CALL    Q20CURS
            MOV     CX,09
            LEA     SI,ASCWAGE
G20:                                    ;Clear leading zeros
            CMP     BYTE PTR[SI],30H
            JNE     G30                 ;  to blanks
            MOV     BYTE PTR[SI],20H
            INC     SI
            LOOP    G20
G30:
            MOV     AH,40H              ;Request display
            MOV     BX,01               ;File handle
            MOV     CX,19               ;No. of characters
            LEA     DX,MESSG3           ;Wage
            INT     21H
            RET
G10DISP     ENDP
;                   Pause for user:
                    --------------
```

شکل پ ۷-۱۴ نمایش دستمزد کارمندان

```
;
H10PAUS   PROC   NEAR
          MOV    COL,20              ;Set cursor
          MOV    ROW,22
          CALL   Q20CURS
          MOV    AH,40H              ;Request display
          MOV    BX,01               ;File handle
          MOV    CX,40               ;No. of characters
          LEA    DX,MESSG4           ;Display pause
          INT    21H
          MOV    AH,10H              ;Request reply
          INT    16H
          RET
H10PAUS   ENDP
;                  Convert ASCII to binary:
;                  ------------------------
J10ASBI   PROC   NEAR
          MOV    MULT10,0001
          MOV    BINVAL,00
          MOV    DECIND,00
          XOR    BX,BX               ;Clear BX
J20:
          MOV    AL,[SI]             ;Get ASCII character
          CMP    AL,'.'              ;Bypass if dec point
          JNE    J40
          MOV    DECIND,01
          JMP    J50
J40:
          AND    AX,000FH
          MUL    MULT10              ;Multiply by factor
          ADD    BINVAL,AX           ;Add to binary
          MOV    AX,MULT10           ;Calculate next
          MUL    TENWD               ;  factor x 10
          MOV    MULT10,AX
          CMP    DECIND,00           ;Reached decimal point?
          JNZ    J50
          INC    BX                  ;  yes, add to count
J50:
          DEC    SI
          LOOP   J20
          CMP    DECIND,00           ;End of loop
          JZ     J90                 ;Any decimal point?
          ADD    NODEC,BX            ;  yes, add to total
J90:      RET
J10ASBI   ENDP
;                  Scroll whole screen:
;                  --------------------
Q10SCR    PROC   NEAR
          MOV    AX,0600H
          MOV    BH,30H              ;Attribute
          MOV    CX,00
          MOV    DX,184FH
          INT    10H
          RET
Q10SCR    ENDP
;                  Set cursor row:column:
;                  ----------------------
Q20CURS   PROC   NEAR
          MOV    AH,02H
          MOV    BH,00               ;Page 0
          MOV    DH,ROW              ;Set row,
          MOV    DL,COL              ;  column
          INT    10H
          RET
Q20CURS   ENDP
;                  Scroll display window:
;                  ----------------------
```

شکل ت ۷-۱۴ نمایش دستمزد کارمندان

```
;
Q30WIN    PROC    NEAR
          MOV     AX,0605H          ;Five rows
          MOV     BH,16H            ;Attribute
          MOV     CH,TOPROW
          MOV     CL,LEFCOL
          MOV     DH,BOTROW
          MOV     DL,RITCOL
          INT     10H
          RET
Q30WIN    ENDP
          END     A10MAIN
```

شکل ث ۷-۱۴ نمایش دستمزد کارمندان

E10MULT ضرب، تقریب کردن، و تغییر مکان را انجام می‌دهد. یک دستمزد با صفر، یک، یا دو رقم اعشار نیازی به تقریب کردن یا تغییر مکان ندارد.

F10WAGE نقطه اعشار را جایگزین می‌کند، موقعیت سمت راست برای شروع کاراکترهای ASCII را مشخص می‌کند و دستمزد دودویی را به ASCII تبدیل می‌کند.

G10PSP صفرهای دستمزد را پاک می‌کند و آن را نمایش می‌دهد.

H10PAUS دستمزد محاسبه شده را تا زمانی که کاربر یک کلید را بفشارد نمایش می‌دهد. فشردن (ESC) به برنامه اعلام می‌کند که پردازش را ادامه ندهد.

J10ASBI تبدیل ASCII به دودویی (یک روتین کلی برای ساعات و نرخ) و تعیین تعداد ارقام اعشاری در مقادیر وارد شده.

Q10SCR صفحه کامل را حرکت طوماری می‌دهد و آن را سیاه بر روی فیروزه‌ای تنظیم می‌کند.

Q30WIN یک پنجره در میان صفحه، جایی که ساعات، نرخ و دستمزد بصورت قهوه‌ای بر روی آبی نمایش داده می‌شود، را حرکت طوماری می‌دهد.

محدودیت‌ها. یک محدودیت برنامه این است که فقط شش رقم اعشار را مجاز می‌شمارد. محدودیت دیگر عبارتست از مقدار خود دستمزد و این حقیقت که تغییر مکان مستلزم تقسیم بر چند ۱۰، و تبدیل به ASCII مستلزم تقسیم بر ۱۰ می‌باشد. اگر ساعات و نرخ حاوی یک مقدار کل باشد که از شش رقم اعشار تجاوز کند یا دستمزد از ۶۵۵۳۵۰ تجاوز نماید. برنامه دستمزد را با صفر مقدار می‌دهد. در عمل، یک برنامه یک پیغام اخطار دهنده خواهد نوشت یا شامل زیر رُوال‌هایی برای غلبه کردن بر این محدودیت‌ها باشد.

بررسی خطا. یک برنامه که علاوه بر برنامه نویس، برای سایر استفاده کنندگان نیز طراحی شده است، بایستی نه تنها پیغامهای آگاهی دهنده تولید کند بلکه باید ساعات و نرخ را نیز از نظر اعتبار ارزشیابی نماید. تنها کاراکترهای معتبر ارقام ۰ تا ۹ و یک نقطه اعشار می‌باشد. برای هر کاراکتر دیگر، برنامه باید یک پیغام را نشان دهد و به موقعیت اخذ ورودی باز گردد. یک دستور مفید برای تعیین اعتبار XLAT است که در فصل ۱۵ آمده است. برنامه خود را به طور کامل برای تمام شرایط ممکن از قبیل ارزش صفر، ارزشهای بینهایت کوچک و بزرگ و مقادیر منفی تست نماید.

مقادیر منفی. بعضی از کاربردها، به خصوص برای معکوس کردن و اصلاح وارده‌ها، مستلزم مقادیر منفی می‌باشند. شما می‌توانید یک علامت منها را بعد از یک مقدار، مثل ۱۲.۳۴، یا قبل از مقدار، مثل ۱۲.۳۴ - مجاز بدانید. برنامه می‌تواند در حین تبدیل به دودویی برای یک علامت منها تست انجام دهد. ممکن است بخواهید تا عدد دودویی را به عنوان مثبت رها کرده و علامتی که بیانگر منفی بودن عدد است مشخص کنید. وقتی عمل محاسبات کامل شد، در صورت احتیاج، برنامه می‌تواند یک علامت منها در فیلد ASCII درج نماید.

اگر بخواهید عدد دودویی منفی باشد ، ورودی ASCII را به طور معمول به دودویی تبدیل کنید. (در فصل ۱۳ "معکوس نمودن علامت" را برای تغییر علامت یک فیلد دودویی ملاحظه کنید.) و به نحوه استفاده از IMUL و IDIV، برای دستکاری داده‌های علامت دار دقت کنید. برای تقریب کردن یک مقدار منفی ، بجای افزودن ۵، آن را کم کنید.

نکات کلیدی

یک فیلد ASCII، یک بایت برای هر کاراکتر نیاز دارد. برای یک فیلد عددی، نیم بایت راست حاوی رقم، و نیم بایت چپ حاوی ۳ است.

- اگر ۳ سمت چپ را از رقم ASCII، به صفر تغییر دهید آن را به قالب BCD تبدیل کرده‌اید .
- فشرده کردن کاراکترهای ASCII به دو رقم در هر بایت، فیلد را به داده‌های BCD فشرده تبدیل می‌کند.
- بعد از جمع ASCII، از AAA برای تنظیم جواب استفاده کنید، بعد از تفریق ASCII از AAS برای تنظیم جواب استفاده کنید.
- قبل از ضرب ASCII، مضروب و مضروب فیه را بوسیله پاک کردن سمت چپ‌ترین ارقام ۳ به "دهدهی غیر فشرده" تبدیل نمایید بعد از ضرب، نتیجه را با AAM تنظیم نمایید.
- قبل از تقسیم ASCII، مقسوم و مقسوم‌علیه را به وسیله پاک کردن سمت چپ‌ترین ارقام ۳ به "دهدهی غیر فشرده" تبدیل کنید و مقسوم را با AAD تنظیم نمایید.
- برای اکثر اهداف محاسباتی، اعداد ASCII را به دودویی تبدیل کنید. موقع تبدیل ASCII به قالب دودویی بررسی کنید که کاراکترهای ASCII یعنی ۳۰ تا ۳۹، نقطه اعشار و احتمالاً یک علامت منها ، معتبر هستند.

پرسش‌ها

۱-۱۴) فرض کنید AX حاوی ASCII 8 (0038H) حاوی ASCII 6 (0036H) است. نتایج دقیق اعمال غیر مرتبط زیر را توضیح دهید.

(الف)
```
ADD AX,35H
AAA
```

(ب)
```
ADD AX,DX
AAA
```

(ج)
```
SUB AX,DX
AAS
```

(د)
```
SUB AX,0BH
AAS
```

۲-۱۴) از مبنای شانزده برای نمایش مقدار دهدهی 3796 در قالب‌های زیر استفاده کنید (الف) ASCII، (ب) BCD غیر فشرده، (ج) BCD فشرده.

۳-۱۴) یک فیلد BCD غیر فشرده با نام BCDAMT حاوی 02050904H است. حلقه‌ای را کد نویسی کنید که محتویات آن را به ASCII صحیح، 34H 39 35 32، تبدیل کند.

۴-۱۴) یک فیلد با نام ASCAMT1 حاوی مقدار دهدهی ASCII، ۲۱۵ است، و فیلد دیگری با نام ASCAMT2 حاوی ASCII 4 باشد. دستوراتی کد نویسی کنید که اعداد ASCII را در هم ضرب کند و حاصل را در ASCPROD ذخیره نماید.

۵-۱۴) با استفاده از همان فیلدهای پرسش ۴-۱۴ ، ASCAMT1 را بر ASCAMT2 تقسیم کنید و خارج قسمت در ASCQUOT ذخیره بشود.

۶-۱۴) محاسبات دستی برای موارد زیر را بنویسید (الف) مقدار دهدهی ۳۹۸۴۶ را به دودویی تبدیل کنید، و حاصل را در قالب شانزدهی نشان دهید، (ب) مقدار شانزدهی را به ASCII تبدیل کنید.

۷-۱۴) یک برنامه برای تعیین اندازه حافظه کامپیوتر بنویسید و آن را اجرا کنید (INT 21H، فصل ۳ را ببینید)، اندازه حافظه را به قالب ASCII تبدیل نمایید و آن را روی صفحه تصویر به صورت زیر نمایش دهید:

```
Memory size is nnn bytes
```

فصل ۱۵

تعریف و پردازش جداول

هدف: حاوی مطالبی راجع به تعریف جداول، انجام جستجو در جدول، و مرتب کردن ورودیهای آن.

مقدمه

بسیاری از کاربردهای برنامه نیازمند ***جداولی*** یا ***آرایه‌ای*** از داده‌ها مثل نام‌ها، توضیحات، کمیت‌ها و نرخ‌ها می‌باشند. این فصل با تعریف برخی جداول معمول و سپس روش برای جستجو در آنها آغاز می‌شود. تکنیک‌هایی برای جستجو در جدول تحت تسلط روشی است که جداول تعریف شده‌اند، و انواع بسیاری از تعاریف و جستجوی جدول موجود می‌باشد. تعریف و استفاده از جداول شامل بکار بردن آن چیزی است که شما در اینجا می‌آموزید. بقیه موارد شامل، مرتب نمودن ترتیب داده‌ها در جدول، و استفاده از لیست‌های پیوندی، که استفاده از اشاره‌گر به موقعیت عنصر در یک جدول می‌باشد.

تعریف جداول

برای سهولت جستجو در جدول، اغلب جداول به روشی مرتب می‌شوند که هر ورودی با همان قالب (کاراکتری یا عددی)، با همان طول، بر طبق ترتیب صعودی یا نزولی تعریف می‌شود جدولی که شما در این کتاب استفاده می‌کنید، پشته‌ای از ۶۴ کلمه ارزشدهی نشده است، در آن متغییر STACK به اولین کلمه جدول اشاره می‌کند:

```
STACK DW 64DUP(?)
```

در جدول زیر MONTABL و CUSTABL، کاراکترها و مقادیر عددی مقداردهی می‌شوند. MONTABL، خلاصه الفبایی از ماه‌ها را تعریف می‌کند، در حالیکه CUSTABL جدولی از شماره مشتریان را تعریف می‌کند :

```
MONTABL   DB   'Jan', 'Feb', 'Mar', . . ., 'Dec'
CUSTABL   DB   205, 208, 209, 212, 215, 229, . . .
```

همه ورودیها در MONTABL سه کاراکتر دارند، و همه ورودیها در CUSTABL سه رقم دارند. اما توجه کنید که اسمبلر اعداد دهدهی را به قالب دودویی تبدیل می‌کند و بیشتر از مقدار ۲۵۵ نخواهد شد و هر کدام در یک بایت ذخیره می‌شود.

یک جدول ممکن است حاوی ترکیبی از مقادیر عددی و کاراکتری باشد. جدول زیر از اقلام موجودی، هر ورودی عددی (تعداد ارقام) ۲ رقم (یک بایت) و هر کاراکتر ورودی (توضیح موجودی) ۹ بایت است :

```
STOKTBL  BL  12  'Computers', 14, 'Paper', . . . , 17, 'Diskettes', . . .
```

چهار نقطه بعد از توضیح 'Paper' برای نمایش آن است که فضای خالی وجود دارد، که، فضای خالی، نقطه نه، می‌تواند وارد شود. برای افزایش وضوح، می‌توانید هر ورودی جدول را در یک خط جداگانه کد نویسی کنید.

```
STOKTBL  DB   12,  'Computers'
         DB   14,  'Paper ....'
         DB   17,  'Disletres'
         . . .
```

مثال بعد جدولی با ۱۰۰ ورودی تعریف می‌کند، هر یک با ۱۵ کاراکتر فاصله (۱۵۰۰ بایت) مقدار دهی می‌شوند:

```
STORETBL  DB  100 DUP (IS DUP (' '))
```

جداول بر روی دیسک

در حالت‌های واقعی، بسیاری برنامه‌ها راه‌انداز جدول دارند. جداولی که بعنوان فایلهای دیسکی ذخیره می‌شوند، ممکن است هر تعدادی از برنامه برای پردازش نیاز داشته باشد. بدین منظور، برنامه می‌تواند یک جدول فایلی را در دیسک خوانده و در یک جدول خالی تعریف شده به همین منظور قرار دهد. چون محتویات جدول هر بار تغییر می‌کند چنین عمل می‌شود. اگر هر برنامه جدول خودش را تعریف کند، لازم است که با هر تغییری همه برنامه‌ها مجدداً جدول را تعریف کنند و اسمبل شوند. با جداول فایلی روی دیسک، یک تغییر از جدول شامل تغییر محتویات فایل نیز می‌باشد. فصل ۱۷ مثالی از یک جدول‌فایلی ارائه‌می‌دهد. می‌خواهیم روش‌های‌مختلف استفاده‌از جدول در برنامه را بررسی‌کنیم.

آدرس دهی مستقیم به ورودیهای جدول

فرض کنید یک کاربر شماره ماه مانند ۳ را وارد کرده و یک برنامه آن را به حروف الفبا مانند 'March' تبدیل می‌کند. روتینی که این تبدیل را انجام می‌دهد شامل تعریف جدولی است از ماه‌ها با حروف الفبایی که همگی طول یکسانی دارند. طول هر ورودی باید مساوی طولانی‌ترین نام باشد، September

```
MONTBL   DB   'January'
         DB   'February.'
         DB   'March . . .'
         . . .
         DB   'December'
```

ورودی 'January ...' در MONTBL+00 ، 'February' در MONTBL+09 ، 'March' در MONTBL+18 و الی آخر قرار دارد. برای قرار گرفتن در موقعیت ماه ۳، برنامه باید مراحل زیر را انجام دهد:

۱) ماه وارد شده را از ASCII 33 به دودویی ۳ تبدیل کنید.

۲) ۱ را از این عدد کم کنید: 2 = 1 - 3 (زیرا ماه ۰۱ در موقعیت MONTBL+00 قرار دارد).

۳) عدد جدید را در ۹ ضرب کنید (طول هر ورودی) : ۱۸ = ۹ × ۲

۴) این حاصل را (18) با آدرس MONTBL جمع کنید، حاصل در آدرس توضیح مورد نیاز قرار می‌گیرد : MONTBL+18 که ورودی 'March' آغاز می‌شود.

این تکنیک بعنوان ***آدرس دهی مستقیم جدول*** شناخته می‌شود. چون الگوریتم مستقیماً آدرس جدول مورد نیاز را محاسبه می‌کند، نیازی نیست که برنامه در ورودی جدول مرتباً جستجو نماید.

آدرسدهی مستقیم، مثال ۱: جدول ماه‌ها

برنامه شکل ۱-۱۵ مثالی است از دستیابی مستقیم به جدولی با نام‌های ماه. روال B10CONV فرض می‌کند 11(November) ورودی است و ماه را از ASCII به قالب باینری مانند زیر تبدیل می‌کند :

```
بارگزاری ماه ASCII در AX =          3131H
XOR کردن 3030 =                      0101H
اگر بایت سمت چپ غیر صفر است پاک کن =  0001H
و با 0AH (10) جمع کن =               000BH
```

روال C10LOC موقعیت واقعی ورودی‌های جدول را تعیین می‌کند :

000AH (10)	= AX در کاهش یک از ماه
005AH	= ضرب در ۹ (طول ورودی)
MONTBL+5AH	= جمع آوری جدول (MONTBL)

روال، توصیف ('November') را از جدول به ALFMON منتقل می‌کند، در محلی که D10DISP آن را نمایش دهد. یک روش اجرای برنامه پذیرش شماره‌ها، از صفحه کلید و تعیین مقدار در بین 01 و 12 است.

```
TITLE     A15DIREC (COM)  Direct table addressing
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ---------------------------------------------------------
NINE      DB      9
MONIN     DB      '11'
ALFMON    DB      9 DUP (20H)
MONTBL    DB      'January  ', 'February ', 'March    '
          DB      'April    ', 'May      ', 'June     '
          DB      'July     ', 'August   ', 'September'
          DB      'October  ', 'November ', 'December '
; ---------------------------------------------------------
.386
A10MAIN   PROC    NEAR              ;Main procedure
          CALL    B10CONV           ;Convert to binary
          CALL    C10LOC            ;Locate month
          CALL    D10DISP           ;Display alpha month
          MOV     AX,4C00H          ;End processing
          INT     21H
A10MAIN   ENDP
;                 Convert ASCII to binary:
B10CONV   PROC
          MOV     AH,MONIN          ;Set up month
          MOV     AL,MONIN+1
          XOR     AX,3030H          ;Clear ASCII 3s
          CMP     AH,00             ;Month 01-09?
          JZ      B20               ;  yes, bypass
          XOR     AH,AH             ;  no,  clear AH,
          ADD     AL,10             ;  correct for binary
B20:      RET
B10CONV   ENDP
;                 Locate month in table:
C10LOC    PROC
          LEA     SI,MONTBL
          DEC     AL                ;Correct for table
          MUL     NINE              ;Multiply AL by 9
          ADD     SI,AX
          MOVZX   CX,NINE           ;Initialize 9-char move
          CLD
          LEA     DI,ALFMON
          REP MOVSB                 ;Move 9 characters
          RET
C10LOC    ENDP
;                 Display alpha month:
D10DISP   PROC
          MOV     AH,40H            ;Request display
          MOV     BX,01             ;File handle
          MOV     CX,09             ;9 characters
          LEA     DX,ALFMON
          INT     21H
          RET
D10DISP   ENDP
          END     BEGIN
```

شکل ۱-۱۵ آدرس دهی مستقیم جدول: مثال ۱

آدرسدهی مستقیم، مثال ۲: جداول ماه و روز

برنامه شکل ۲-۱۵ تاریخ امروز را از سیستم بازیابی کرده و آن را نمایش می‌دهد. INT 21H تابع 2AH مقادیر دودویی زیر را ارائه می‌دهد:

```
TITLE     A15DISDA (EXE) Display day of week and month
          .MODEL  SMALL
          .STACK  64
          .DATA
SAVEDAY   DB      ?
SAVEMON   DB      ?
TEN       DB      10
ELEVEN    DB      11
TWELVE    DB      12
DAYSTBL   DB      'Sunday, $   ', 'Monday, $   '
          DB      'Tuesday, $  ', 'Wednesday, $'
          DB      'Thursday, $ ', 'Friday, $   '
          DB      'Saturday, $ '
MONTBL    DB      'January $   ', 'February $ ', 'March $    '
          DB      'April $     ', 'May $      ', 'June $     '
          DB      'July $      ', 'August $   ', 'September $'
          DB      'October $   ', 'November $ ', 'December $ '
; ---------------------------------------------------------
          .CODE
A10MAIN   PROC    FAR
          MOV     AX,@data      ;Initialize
          MOV     DS,AX         ;  segment registers
          MOV     ES,AX
          MOV     AX,0600H
          CALL    Q10SCR        ;Clear screen
          CALL    Q20CURS       ;Set cursor
          MOV     AH,2AH        ;Get today's date
          INT     21H
          MOV     SAVEMON,DH    ;Save month
          MOV     SAVEDAY,DL    ;Save day of month
          CALL    B10DAYWK      ;Display day of week
          CALL    C10MONTH      ;Display month
          CALL    D10DAYMO      ;Display day
          CALL    E10INPT       ;Wait for input
          CALL    Q10SCR        ;Clear screen
          MOV     AX,4C00H      ;End processing
          INT     21H
A10MAIN   ENDP
;                 Display day of week:
;                 --------------------
B10DAYWK  PROC    NEAR
          MUL     TWELVE        ;Day (in AL) x 12
          LEA     DX,DAYSTBL    ;Address of table
          ADD     DX,AX         ;  plus offset
          MOV     AH,09H        ;Request display
          INT     21H
          RET
B10DAYWK  ENDP
;                 Display month:
;                 --------------
C10MONTH  PROC    NEAR
          MOV     AL,SAVEMON    ;Get month
          DEC     AL            ;Decrement by 1
          MUL     ELEVEN        ;Multiply by entry length
          LEA     DX,MONTBL     ;Address of table
          ADD     DX,AX         ;  plus offset
          MOV     AH,09H        ;Request display
          INT     21H
          RET
C10MONTH  ENDP
```

شکل الف ۲-۱۵ آدرس دهی مستقیم جدول : مثال ۲

```
.386
;                   Display day of month:
;                   ---------------------
D10DAYMO    PROC    NEAR
            MOVZX   AX,SAVEDAY      ;Get day
            DIV     TEN             ;Convert from binary
            OR•     AX,3030H        ;  to ASCII
            MOV     BX,AX           ;Save ASCII day
            MOV     AH,02H          ;Request display
            MOV     DL,BL           ;  first digit
            INT     21H
            MOV     AH,02H          ;Request display
            MOV     DL,BH           ;Second digit
            INT     21H
            RET
D10DAYMO    ENDP
;                   Wait for keyboard input:
;                   ------------------------
E10INPT     PROC    NEAR
            MOV     AH,10H          ;Request input
            INT     16H             ;Call BIOS
            RET
E10INPT     ENDP
;                   Scroll screen:
;                   --------------
Q10SCR      PROC    NEAR
            MOV     AX,0600H        ;Request scroll
            MOV     BH,17H          ;White on blue
            MOV     CX,0000
            MOV     DX,184FH
            INT     10H             ;Call BIOS
            RET
Q10SCR      ENDP
;                   Set cursor:
;                   -----------
Q20CURS     PROC    NEAR
            MOV     AH,02H          ;Request set cursor
            MOV     BH,00           ;Page
            MOV     DH,10           ;Row
            MOV     DL,24           ;Column
            INT     10H             ;Call BIOS
            RET
Q20CURS     ENDP
            END     A10MAIN
```

شکل ب ۲-۱۵ آدرس دهی مستقیم جدول : مثال ۲

AL = روز هفته (مثلاً یک شنبه = ۰) DH = ماه (۱۲-۰۱)

CX = سال (با این برنامه استفاده نمی‌شود) DL = روز ماه (۳۱-۰۱)

این‌برنامه با استفاده ازاین‌مقادیر روزهفته و ماه رابصورت حروف‌الفبا نشان‌می‌دهد مثلاً 'Wedensday,September 12'. بدین منظور، برنامه جدولی را از روزهای هفته با نام DAYSTBL تعریف می‌کند، که با یکشنبه آغاز می‌شود، و یک جدول از ماه با نام MONTBL که با ژانویه آغاز می‌شود.

ورودیهای DAYSTBL ، ۱۲ بایت طول دارند، که هر توضیح یک ویرگول، فاصله و علامت $ دارد و با فاصله‌هایی در سمت راست، تطبیق داده شده است. INT 21H تابع ۰۹H همه کاراکترها تا علامت $ را نمایش می‌دهد، ویرگول و فاصله بعد از ماه بر روی صفحه قرار می‌گیرند. روال B10DAYWK روز هفته را در ۱۲ (طول هر ورودی در DAYSTBL) ضرب می‌کند. حاصل افستی از جدول است، که برای مثال یک‌شنبه در DAYSTBL+0 دوشنبه در DAYSTBL+12 و غیره قرار دارد. روز مستقیماً از جدول نمایش داده می‌شود.

ورودی MONTBL، ۱۱ بایت طول دارد، که هر توضیح یک فاصله و علامت $ دارد و با فاصله در سمت راست،

تنظیم شده است. روال C10MONTH ابتدا ماه را یکی کم می‌کند، برای مثال ماه ۰۱ ورودی صفر در MONTBL خواهد بود. سپس ماه را در ۱۱ ضرب می‌کند (طول هر ورودی در MONTBL) حاصل یک افست جدول است، برای مثال ژانویه در MONTBL+0 فوریه در MONTBL+11 وغیره قراردارد. روال ماه را مستقیماً ازجدول‌نمایش‌می‌دهد.

روال D10DAYMO روز ماه را بر ۰۱ تقسیم می‌کند تا از دودویی به قالب ASCII تبدیل نماید. چون حداکثر مقدار روز ۳۱ است، خارج قسمت و باقیمانده، یک رقم خواهد داشت. (برای مثال ۳۱ تقسیم بر ۱۰، خارج قسمت ۳ و باقیمانده ۱ خواهد داشت.) INT 21H تابع 02H هر یک از دو کاراکتر را نمایش می‌دهد، صفر برای روزهای کمتر از ۱۰، عدم نمایش صفر مستلزم ایجاد تغییرات کوچکی در برنامه است. در انتها، برنامه منتظر می‌شود تا کاربر یک کلید را قبل از خاتمه پردازش بفشارد.

اگر چه آدرسدهی مستقیم جدول بسیار کارایی دارد، بهتر از ورودیهایی مرتب و با ترتیبی قابل پیش‌بینی عمل می‌کند. این روش، برای ورودیهایی که بترتیب ۰۱، ۰۲، ۰۳، یا ۱۰۶، ۱۰۷، ۱۰۸، و.... یا حتی، ۰۵، ۱۰، ۱۵،... هستند خوب کار می‌کند. اما کاربردهای کمی هستند که این ترتیب مقادیر جدول را داشته باشند. بخش بعدی جداول با مقادیر ترتیبی ولی نه قابل پیش‌بینی را بررسی می‌کند.

جستجوی یک جدول

برخی جداول حاوی اعداد منحصربفردی هستند که هیچ الگوی ظاهری ندارند. یک نوع مثال، جدولی از موجودی اقلام است که اعداد متوالی مانند ۳۴، ۳۸، ۴۱، ۱۳۹، ۱۴۵ نیستند. یک نوع دیگر جدول مانند جدول مالیات - حاوی دامنه‌ای از مقادیر است. بخش بعد هر دو نوع این جداول را بررسی می‌کند و ضروریات جستجوی آن‌ها را عنوان می‌کند.

جداول با ورودیهای منحصربفرد

موجودی اقلام در اغلب مشاغل ترتیب متوالی ندارد. بیشتر، منجر به گروه‌بندی خلاصه می‌شود شاید با یک تعدادی که مبنی بر وسایل و اسباب است یا مبنی بر موقعیت آن در بخش خاصی است. همچنین، برخی اقلام از موجودی حذف یا برخی دیگر افزوده خواهد شد. بعنوان مثال، بیایید یک جدول با اقلام موجودی و توضیح مربوطه، تعریف نماییم. باید در جداول مجزا مانند این تعریف شود:

```
STOKNOS   DB  '05', '10', '12' . . .
STOKDESC  DB  'Excavators', 'Lifters . . .', 'Presses . . .', . . .
```

هر مرحله در جستجو می‌تواند آدرس اولین جدول را با ۲ افزایش دهد (طول هر ورودی در STOKNOS) و آدرس جدول دوم را با ده (طول هر ورودی در STOKDESC) بیفزاید. یا، یک روال می‌تواند شمارهٔ تعداد حلقه‌ای که اجرا شده نگه دارد و با یافتن یک مورد موافق با کلید خاص موجودی اقلام، شماره را در ده ضرب کند و این حاصل را بعنوان افست آدرس STUKDESK استفاده کند. به عبارت دیگر، ممکن است واضح‌تر باشد که موجودی و توضیح آن در یک جدول تعریف شود، با یک خط برای هر زوج از اقلام :

```
STOKTBL  DB  '05', 'Excavators'
         DB  '10', 'Lifters . . .'
         DB  '12', 'Dresses . . .'
         . . .
```

برنامه شکل ۳-۱۵ این جدول را با شش جفت از تعداد موجودی و توضیحات آن تعریف می‌کند. روتین جستجو در شروع از B10TABLE با مقایسه اولین بایت از تعداد موجودی ورودی STOKNIN، با اولین بایت از تعداد موجودی در جدول، آغاز می‌شود. نتیجه، مقایسه می‌تواند کمتر، بیشتر یا مساوی باشد.

۱) **کوچکتر**، اگر نتیجه، مقایسه اولین بایت یا دومین بایت، کمتر باشد، برنامه تعیین می‌کند که عدد موجودی در جدول نیست و در B40 یک پیغام خطا می‌تواند نشان دهد (کد نشده است). برای مثال، برنامه عنصر موجودی ۰۱ ورودی

```
TITLE     A15TABSR (COM)  Table search Using CMP
          .MODEL  SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ------------------------------------------------
STOKNIN   DB      '12'                  ;Input stock no.
STOKTBL   DB      '05','Excavators'     ;Start of table
          DB      '10','Lifters   '     ;
          DB      '12','Presses   '     ;
          DB      '15','Valves    '     ;
          DB      '23','Processors'     ;
          DB      '27','Pumps     '     ;End of table
DESCRIP   DB      10 DUP(?)             ;Save area
; ------------------------------------------------
A10MAIN   PROC    NEAR
          CALL    B10TABLE
          MOV     AX,4C00H          ;End processing
          INT     21H
A10MAIN   ENDP

B10TABLE  PROC    NEAR
          MOV     CX,06             ;Initialize compares
          LEA     SI,STOKTBL
B20:
          MOV     AL,STOKNIN
          CMP     AL,[SI]           ;Stock#(1) : table
          JNE     B30               ;Not equal, exit
          MOV     AL,STOKNIN+1      ;Equal:
          CMP     AL,[SI+1]         ;  stock#(2) : table
          JE      B50               ;  equal, found
B30:      JB      B40               ;Low, not in table
          ADD     SI,12             ;High, get next entry.
          LOOP    B20
B40:                                ;Not in table
;         ...                       ;Display error message
          JMP     B90
B50:
          CALL    C10DISP
B90:      RET
B10TABLE  ENDP

C10DISP   PROC    NEAR              ;Display description
          MOV     CX,05             ;Length of description
          LEA     DI,DESCRIP        ;Address of description
          INC     SI
          INC     SI                ;Extract description
          REP MOVSW                 ;  from table
          MOV     AH,40H            ;Request display
          MOV     BX,01             ;File handle
          MOV     CX,10             ;10 characters
          LEA     DX,DESCRIP        ;Stock description
          INT     21H
          RET
C10DISP   ENDP
          END     BEGIN
```

شکل ۳-۱۵ جستجو یک جدول با استفاده از *CMP*

را با عنصر ۰۵ جدول مقایسه می‌کند، اولین بایت مساوی است، اما چون دومین بایت کمتر است، برنامه تعیین می‌کند که عنصر در جدول نیست.

۲) **بزرگتر**، اگر نتیجه مقایسه اولین بایت یا دومین بایت، بزرگتر باشد، برنامه به جستجو ادامه می‌دهد، برای مقایسه اقلام موجودی ورودی با عنصر موجودی بعدی در جدول SI را می‌افزاید. که حاوی آدرس جدول است. برای مثال، برنامه عنصر موجودی ورودی ۰۶ را با عنصر جدول ۰۵ مقایسه می‌کند. اولین بایت مساوی است: موجودی ۰۶ با

موجودی جدول ۱۰. اولین بایت کمتر است، بنابراین برنامه تعیین می‌کند که عنصر در جدول نیست.

۳) **مساوی**، اگر هر دو بایت اول و دوم مساوی باشند، تعداد موجودی پیدا شده است. برای مثال ، برنامه موجودی ۱۰ را با عنصر جدول ۵ مقایسه می‌کند اولین بایت بزرگتر است، بنابراین با عنصر بعدی جدول مقایسه می‌کند. موجودی ۱۰ با عنصر جدول ۱۰ برابر است. چون اولین بایت و دومین بایت برابر است، برنامه عنصر را یافته است، در B50، روتین C10DISP را فراخوانی می‌کند، که توضیح را از جدول به DESCRN، جایی که نمایش داده خواهد شد کپی می‌کند.

حلقه جستجو اکثر ۶ مقایسه را انجام می‌دهد. اگر تکرار حلقه از ۶ تجاوز کند، اقلام مورد نظر در جدول وجود ندارند، جدول می‌تواند قیمت واحد را تعریف کند. کاربر تعداد موجودی و کمیت فروش را وارد می‌کند. برنامه می‌تواند در موقعیت عنصر موجودی در جدول قرار گیرد، میزان فروش را محاسبه کند (کمیت فروش در قیمت هر واحد) و توضیح و میزان فروش را نمایش دهد.

در شکل ۳-۱۵، شماره اقلام ۲ کاراکتر است و توضیح ۱۰ کاراکتر می‌باشد. برنامه نویسی با تعداد طولهای مختلفی از ورودی سرو کار دارد. برای مثال، برای مقایسه فیلدهای ۳ بایتی، می‌توانید از REPE CMPSB استفاده کنید، اگر چه REPE به ثبات CX نیاز دارد که LOOP هم اکنون از آن استفاده می‌کند.

جداول با دامنه‌ها

مالیات بر درآمد، یک نمونه از جدولی با محدوده‌هایی از مقادیر را ارائه می‌دهد. جدول فرضی زیر را از درآمد مالیات پذیر، نرخ‌ها و فاکتورهای اصلاحی در نظر بگیرید :

درآمد مالیات پذیر	نرخ	فاکتورهای اصلاحی
0-1000.00	.10	0.00
1000.01-2500.00	.15	050.00
2501.01-4250.00	.18	125.00
4250.01-6000.00	.20	260.00
و بیشتر 6000.01	.23	390.00

فاکتوراصلاحی، مالیات محاسباتی در نرخ‌بالا را جبران‌می‌کند، در حالیکه نرخ پایین برای سطح پایین درآمد بکارمی‌رود.

```
TAXTBL  DD  100 000, 10, 00000
        DD  250 000, 15, 05000
        DD  425000, 18, 12500
        DD  600000, 20, 26000
        DD  999999, 23, 39000
```

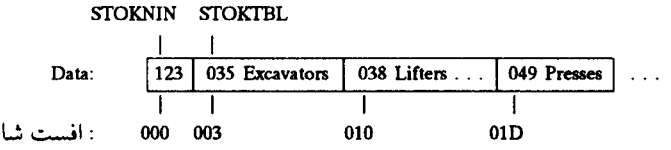
برای انجام یک جستجو از جدول، برنامه، درآمد مالیات پذیر مالیاتی را با درآمد مالیات پذیر جدول مقایسه می‌کند و بر طبق نتیجه مقایسه زیر عمل می‌کند:

- بزرگتر: هنوز پیدا نشده، برای ورودی بعدی در جدول، آدرس را افزایش می‌دهد.
- کوچکتر یا مساوی: پیدا شده، نرخ فاکتور اصلاحی مربوطه را استفاده کنید. کاهش مالیات به صورت (نرخ جدول × درآمد مالیات پذیر) محاسبه می‌شود. توجه کنید که آخرین ورودی جدول حاوی حداکثر مقدار ۹۹۹۹۹۹ است، که همیشه خاتمه جستجو را موجب می‌شود.

جستجوی جدول با استفاده از مقایسات رشته‌ای

اگر شماره‌های قلم کالا از دو بایت تجاوز کند از دستورالعمل REPE CMPS برای عمل مقایسه استفاده نمایید. برنامه شکل ۴-۱۵، STOKTBL را تعریف می‌کند، اما این بار با تعداد موجودی ۳ بایتی بازنویسی شده است. چون

STOKNIN در اولین فیلد ناحیه داده است و STOKTBL بعد از آن قرار می‌گیرد، در سگمنت داده چنین ظاهر می‌شود:




```
                                TITLE  A15STRSR (EXE)  Table search using CMPSB
                                       .MODEL  SMALL
                                       .STACK  64
                                ; ---------------------------------------------
                                       .DATA
0000  31 32 33                  STOKNIN  DB     '123'
0003 30 33 35 45 78 63          STOKTBL  DB     '035','Excavators' ;Define table
     61 76 61 74 6F 72
     73
0010 30 33 38 4C 69 66                   DB     '038','Lifters   '
     74 65 72 73 20 20
     20
001D 30 34 39 50 72 65                   DB     '049','Presses   '
     73 73 65 73 20 20
     20
002A 31 30 32 56 61 6C                   DB     '102','Valves    '
     76 65 73 20 20 20
     20
0037 31 32 33 50 72 6F                   DB     '123','Processors'
     63 65 73 73 6F 72
     73
0044 31 32 37 50 75 6D                   DB     '127','Pumps     '
     70 73 20 20 20 20
     20
0051 39 39 39                            DB     '999', 10 DUP(' ')   ;End table
     000A[ 20 ]
005E 000A[ ?? ]                 DESCRN   DB     10 DUP(?)            ;Save area
                                ; ---------------------------------------------
                                       .CODE
0000                            A10MAIN  PROC   FAR
0000  B8 ---- R                         MOV    AX,@data     ;Initialize
0003  8E D8                             MOV    DS,AX        ;  segment
0005  8E C0                             MOV    ES,AX        ;  registers
0007  E8 000F R                         CALL   B10TABLE
000A  B8 4C00                           MOV    AX,4C00H     ;End processing
000D  CD 21                             INT    21H
000F                            A10MAIN  ENDP

000F                            B10TABLE PROC  NEAR
000F  FC                                CLD
0010  8D 3E 0003 R                      LEA    DI,STOKTBL   ;Initialize table
0014                            B20:                        ;  address
0014  B9 0003                           MOV    CX,03        ;Compare 3 bytes
0017  8D 36 0000 R                      LEA    SI,STOKNIN   ;Init stock# addr
001B  F3/ A6                            REPE CMPSB          ;Stock# : table
001D  74 09                             JE     B30          ;  equal, exit
001F  72 0D                             JB     B40          ;  low, not entry
0021  03 F9                             ADD    DI,CX        ;Add CX to offset
0023  83 C7 0A                          ADD    DI,10        ;Next table item
0026  EB EC                             JMP    B20
0028                            B30:
0028  E8 002F R                         CALL   C10DISPL     ;Display descr'n
002B  EB 01 90                          JMP    B90
002E                            B40:
                                ;       <Display error message>
002E                            B90:
002E  C3                                RET
002F                            B10TABLE ENDP
```

شکل الف ۴-۱۵ جستجو در یک جدول با *CMPSB*

```
002F                    C10DISPL PROC  NEAR
002F  B9 0005                    MOV   CX,05        ;Set to move 5 words
0032  8B F7                      MOV   SI,DI
0034  8D 3E 005E R               LEA   DI,DESCRN    ;Addr of descr'n
0038  F3/ A5                     REP MOVSW          ;Get descr'n
003A  B4 40                      MOV   AH,40H       ;Request display
003C  BB 0001                    MOV   BX,01        ;File handle
003F  B9 000A                    MOV   CX,10        ;10 characters
0042  8D 16 005E R               LEA   DX,DESCRN    ;Stock description
0046  CD 21                      INT   21H
0048  C3                         RET
0049                    C10DISPL ENDP
                                 END   A10MAIN
```

شکل ب ۴-۱۵ جستجو در یک جدول با *CMPSB*

آخرین ورودی جدول حاوی موجودی "۹۹۹" است (بالاترین رقم موجودی ممکن) و جستجو را خاتمه می‌دهد. برنامه از LOOP برای خاتمه جستجو استفاده می‌کند، اما REPE از CX استفاده می‌کند و برای LOOP در دسترس نیست. روتین جستجو، STOKNIN را با هر ورودی جدول، بشرح زیر، مقایسه می‌کند:

نتیجه مقایسه	ورودی جدول	STOKNIN
بزرگتر، مقایسه ورودی بعدی	035	123
بزرگتر، مقایسه ورودی بعدی	038	123
بزرگتر، مقایسه ورودی بعدی	049	123
بزرگتر، مقایسه ورودی بعدی	102	123
مساوی، ورودی پیدا شد	123	123

روال B10TABLE ثبات DI را با آدرس افست STOKTBL (003)، مقدار دهی می‌کند، CX را با طول (۰۳) موجودی، و SI را با افست STOKNIN (000) مقدار دهی می‌کند. هرگاه که بایت‌ها حاوی مقادیر مساوی باشند، عملیات CMPSB بایت به بایت مقایسه می‌کند و بطور خودکار ثبات‌های DI و SI برای مقایسه بعدی افزایش می‌یابد. یک مقایسه با اولین ورودی جدول (۱۲۳:۰۳۵) خاتمه می‌یابد با مقایسه بالاتر بعد از اولین بایت، DI حاوی 004 ، SI حاوی 001 و CX حاوی 02 می‌باشد.

برای دومین مقایسه، DI باید حاوی 010 ، SI حاوی 000 باشد. تصحیح SI مستلزم دوباره مقدار دادن به STOKNIN می‌باشد. برای تصحیح آدرس ورودی جدول که باید در DI باشد، اما افزایش در این بستگی دارد که مقایسه بعد از یک، دو یا ۳ بایت خاتمه پذیرد. CX حاوی تعداد بایت‌های مقایسه نشده است، که در این حالت ۲ می‌باشد، افزودن مقدار CX با طول توصیف موجودی (که از مقایسه قلم موجودی قبلی است) افست قلم بعدی جدول را بدست می‌دهد، مانند زیر:

```
آدرس DI بعد از CMPSB :   004H
جمع طول باقیمانده در CX :  +02 H
جمع طول توصیف موجودی :   +0AH
آدرس افست بعدی در جدول :  010H
```

جداولی با ورودیهایی دارای طول متغیر

می‌توان یک جدول را با ورودیهایی دارای طول تغییر تعریف کرد. یک کاراکتر محدود کننده مخصوص مثل 00H می‌تواند بعد از هر ورودی قرار گیرد و کمیت FFH انتهای جدول را مشخص می‌کند. لیکن مطمئن باشید که هیچ بایتی در داخل یک ورودی، دارای ساختار بیتی شامل محدود کننده نباشد. برای مثال یک مقدار دودویی حسابی می‌تواند هر ساختار بیتی باشد دستورالعمل SCAS را برای جستجو استفاده کنید.

دستور انتقال XLAT

دستور XLAT محتویات یک بایت را به مقدار از پیش تعریف شده دیگری انتقال می‌دهد. شما می‌توانید از دستورالعمل XLAT برای تعیین اعتبار محتویات عناصر داده استفاده کنید. اگر داده‌ها را بین یک کامپیوتر شخصی و یک کامپیوتر پردازنده مرکزی IBM انتقال می‌دهید، از دستورالعمل XLAT برای انتقال داده‌ها بین قالب ASCII و EBCDIC استفاده کنید. شکل کلی XLAT چنین است:

[label :]	XLAT	بدون عملوند ;

XLAT آدرس جدول را در ثبات BX و بایتی که باید منتقل شود در AL می‌پذیرد. مثال بعد اعداد ASCII ۹-۰ را به قالب EBCDIC تبدیل می‌کند. چون بیان ۹-۰ در ASCII، 39-30 است در EBCDIC ، F0-F9 می‌باشد، می‌توانید از یک عملیات OR برای تغییرات استفاده کنید. اما بیایید تمام کاراکترهای دیگر را نیز به جای خالی EBCDIC، یعنی 40H تبدیل کنیم. برای XLAT، یک جدول انتقال که تمام ۲۵۶ کاراکتر ممکن را می‌شمارد، با کاراکترهای EBCDIC درج شده در موقعیت ASCII و جای خالی بجای همه موقعیت‌های دیگر، چون شماره صفر ASCII 30H است، اعداد در EBCDIC در جدول از موقعیت 30H یا ۴۸ دهدهی آغاز می‌شوند:

```
XLTABLE  DB  48DUP(40H)                ; جای خالی EBCDIC
         DB  0F0H, 0F1H, 0F2H, . . . , 0F9H  ; ۹ تا ۰ EBCDIC
         DB  198 DUP(40H)              ; جای خالی EBCDIC
```

مثال زیر مقدار دهی و انتقال عنصر ASCII با نام ASCNO را به قالب EBCDIC انجام می‌دهد:

```
         LEA   BX,XLTABLE   ; آدرس جدول در BX
         MOV   AL,ASCNO     ; کاراکتر انتقال در AL
         XLAT               ; انتقال به EBCDIC
```

```
            page   60,132
TITLE       A15XLATE (COM)  Translate ASCII to EBCDIC
            .MODEL SMALL
            .CODE
            ORG    100H
BEGIN:      JMP    A10MAIN
;-----------------------------------------------------
ASCNO       DB     '-31.5 '          ;ASCII item to convert
EBCNO       DB     6 DUP(' ')        ;Converted EBCDIC item
XLTABLE     DB     45 DUP(40H)       ;Translate table
            DB     60H, 4BH
            DB     40H
            DB     0F0H,0F1H,0F2H,0F3H,0F4H
            DB     0F5H,0F6H,0F7H,0F8H,0F9H
            DB     198 DUP(40H)
;-----------------------------------------------------
A10MAIN     PROC   NEAR
            LEA    SI,ASCNO          ;Address of ASCNO
            LEA    DI,EBCNO          ;Address of EBCNO
            MOV    CX,06             ;Length of items
            LEA    BX,XLTABLE        ;Address of table
A20:
            LODSB                    ;Get ASCII char in AL
            XLAT                     ;Translate character
            STOSB                    ;Store AL in EBCNO
            LOOP   A20               ;Repeat 6 times
;           ...
            MOV    AX,4C00H          ;End processing
            INT    21H
A10MAIN     ENDP
            END    BEGIN
```

شکل ۵-۱۵ تبدیل اعداد *ASCII* به *EBCDIC*

XLATS از مقدار AL بعنوان افست آدرس استفاده می‌کند، در واقع BX حاوی آدرس شروع جدول، و AL حاوی مقدار افست در جدول می‌باشد. بعنوان مثال اگر مقدار AL صفر باشد ، آدرس جدول XLTABLE+0 (اولین بایت از XLTABLE حاوی 40H است) خواهد بود. XLAT مقدار 00 از AL را با 40H جایگزین می‌نماید.

توجه کنید که اولی XLTABLE , DB، ۴۸ بایت را تعریف می‌کند، که با XLTABLE+00 تا XLTABLE+47 آدرسدهی می‌شود. دومین DB در XLTABLE داده‌هایی را با شروع از XLTABLE+48 تعریف می‌کند. اگر مقدار AL ، 32H باشد (۵۰ دهدهی)، آدرس جدول XLTABLE+50 خواهد بود. این موقعیت حاوی F2 (EBCDIC 2) است، که XLAT در ثبات AL قرار می‌دهد. برنامه در شکل ۵-۱۵، XLTABLE را تغییر می‌دهد بنابراین علامت منها در ASCII (2D) و نقطه اعشار (2E) را به EBCDIC (60 , 4B , 2E) تبدیل می‌کند و حال در یک فیلد ۶ بایتی ASCII حلقه می‌زند. ابتدا، ASCNO حاوی 31.5 - و بعد از آن یک جای خالی است ، یا 2D33312E3520. در پایان حلقه EBCNO باید حاوی 60F3F14BF540 باشد که توسط DEBUG می‌توانید آن را مشخص کنید.

برنامه، نمایش مبنای شانزده و کاراکترهای ASCII

برنامه شکل ۶-۱۵ همه ۲۵۶ مقادیر شانزدهی (00-FF) را نمایش می‌دهد، با اغلب سمبول‌های ASCII مربوطه، برای مثال، سمبول ASCII برای S و بیان شانزدهی آن ۵۳ را نشان می‌دهد. نمایش کامل بر روی صفحه بصورت یک ماتریس ۱۶ در ۱۶ ظاهر می‌شود:

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

همانطور که در شکل ۱-۸ نشان داده شده، نمایش سمبول‌های ASCII سبب مشکل نخواهد شد. اما نمایش بیان شانزدهی یک مقدار ASCII مشکل خواهد داشت. برای مثال، باید 00H را به 3030H و 01H را به 3031H تبدیل کنید و غیره.

```
TITLE     A15ASCHX (COM)  Display ASCII and hex characters
          .MODEL SMALL
          .CODE
.386
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ---------------------------------------------------------
DISPROW   DB      16 DUP(5 DUP(' ')), 13
HEXCTR    DB      00
XLATAB    DB      30H,31H,32H,33H,34H,35H,36H,37H,38H,39H
          DB      41H,42H,43H,44H,45H,46H
; ---------------------------------------------------------
A10MAIN   PROC    NEAR                ;Main procedure
          CALL    Q10CLR              ;Clear screen
          LEA     SI,DISPROW
A20LOOP:
          CALL    B10HEX              ;Translate
          CALL    C10DISPL            ;  and display
          CMP     HEXCTR,0FFH         ;Last hex value (FF)?
          JE      A50                 ;  yes, terminate
          INC     HEXCTR              ;  no,  incr next hex
          JMP     A20LOOP
A50:      MOV     AX,4C00H            ;End processing
          INT     21H
```

شکل الف ۶-۱۵ نمایش مقادیر ASCII و HEX

```
A10MAIN   ENDP
;                 Convert ASCII to hex:
;                 ---------------------
B10HEX    PROC    NEAR
          MOVZX   AX,HEXCTR          ;Get hex pair in AX
          SHR     AX,04              ;Shift off right hex digit
          LEA     BX,XLATAB          ;Set table address
          XLAT                       ;Translate hex
          MOV     [SI],AL            ;Store left character

          MOV     AL,HEXCTR
          AND     AL,0FH             ;Clear left hex digit
          XLAT                       ;Translate hex
          MOV     [SI]+1,AL          ;Store right character
          RET
B10HEX    ENDP
;                 Display as hex characters:
;                 --------------------------
C10DISPL  PROC    NEAR               ;
          MOV     AL,HEXCTR          ;Get hex character
          MOV     [SI]+3,AL
          CMP     AL,1AH             ;EOF character?
          JE      C20                ;  yes, bypass
          CMP     AL,07H             ;Lower than 7?
          JB      C30                ;  yes, ok
          CMP     AL,10H             ;Higher/equal 16?
          JAE     C30                ;  yes, ok
C20:                                 ;Else force blank
          MOV     BYTE PTR [SI]+3,20H
C30:
          ADD     SI,05              ;Next location in row
          LEA     DI,DISPROW+80
          CMP     DI,SI              ;Filled up row?
          JNE     C40                ;  no, bypass
          MOV     AH,40H             ;Yes, request display
          MOV     BX,01              ;  file handle
          MOV     CX,81              ;  entire row
          LEA     DX,DISPROW
          INT     21H
          LEA     SI,DISPROW         ;Reset display row
C40:      RET
C10DISPL  ENDP

;                 Clear screen:
Q10CLR    PROC    NEAR
          MOV     AX,0600H           ;Request scroll
          MOV     BH,61H             ;Attribute
          MOV     CX,0000
          MOV     DX,184FH
          INT     10H
          RET
Q10CLR    ENDP
          END     BEGIN
```

شکل ب ۶-۱۵ نمایش مقادیر *ASCII* و *HEX*

برنامه ابتدا HEXCTR را با 00 تعریف می‌کند و برای هر یک از ۲۵۶ کاراکتر ASCII آن را یک واحد می‌افزاید. روال B10HEX ، HEXCTR را به دو رقم شانزدهی قسمت می‌کند. برای مثال اگر HEXCTR حاوی 4FH باشد، روتین ۴ شانزدهی را با استفاده از XLAT برای انتقال، اقتباس می‌نماید، مقداری که در AL برمی‌گردد 34H است. سپس روتین F را اقتباس نموده و به 64H منتقل می‌کند. حاصل 3446H است، که بر روی صفحه تصویر بصورت 4F نمایش داده می‌شود.

روال C10DISPL کاراکترهای غیر ASCII را به جای خالی تبدیل می‌کند. زیرا INT 21H تابع 40H با 1AH مانند کاراکتر انتهای فایل برخورد می‌کند ، برنامه آن را نیز به جای خالی تبدیل می‌کند. روال یک سطر کامل از ۱۶ کاراکتر را

نمایش می‌دهد و بعد از نمایش سطر شانزدهم خاتمه می‌یابد. روشهای زیاد دیگری برای تبدیل ارقام شانزدهی به کاراکترهای ASCII وجود دارد، برای مثال، می‌توانید با شیفت دادن و مقایسه کردن بیازمایید.

مرتب نمودن ورودیهای جدول

معمولاً می‌خواهیم داده‌های یک جدول، به صورت صعودی یا نزولی **مرتب** شوند. برای مثال یک کاربر ممکن است لیستی از توصیف موجودی بترتیب صعودی خواسته باشد، یا یک لیست از کل فروش فروشندگان به ترتیب نزولی بخواهد. تعدادی روتین مرتب سازی جدول وجود دارد، که متنوع است از پردازش کند ولی واضح، تا پردازش سریع ولی مبهم. روتین این بخش بسیار کارآیی دارد و می‌تواند برای اغلب موارد مرتب سازی جدول بکار رود.

یک روش عمومی مرتب سازی جدول مقایسه یک ورودی جدول با ورودی بلافاصله بعد از خودش می‌باشد. اگر مقایسه بزرگتر باشد ، ورودی‌ها را تعویض کنید. به همین روش ادامه دهید، مقایسه ورودی ۱ با ورودیُ ۲، ورودی ۲ با ورودی ۳، و غیره تا انتهای جدول، و تعویض در هر جا که لازم باشد. اگر شما هر تعویضی را انجام دادید، پردازش ورودیها را از شروع حلقه آغاز نمایید. ورودی یک را با ورودی دو مجدداً مقایسه نمایید، و غیره. در هر نقطه، اگر شما ورودیهای جدول را مقایسه نمودید بدون آنکه تعویض انجام دهید خواهید دانست که جدول مرتب شده است.

در شبه کد زیر، SWAP عبارتی است مبنی بر تعویض در صورت (YES) یا مبنی بر عدم تعویض در صورت (NO) می‌باشد.

```
T10:    Initialize address of last entry in the table
T20:    Set SWAP to NO
        Initialize address of start of the table
T30:    Table entry > next entry?
            Yes: Exchange entries
                 Set SWAP to YES
        Increment for next entry in the table
        At end of the table?
            No:  Jump to T30
            Yes: Does SWAP = YES?
                 Yes:  Jump to T20 (repeat sort)
                 No:   End of sort
```

برنامه شکل ۷-۱۵ به کاربر این امکان را می‌دهد که تا ۱۰ نام از صفحه کلید وارد کند، که برنامهٔ آن‌ها را در جدول با نام NAMETABLE ذخیره می‌کند، و شامل روال‌های زیر می‌باشد:

- A10MAIN روال B10ENTR را برای پذیرش نام از صفحه کلید فراخوانی می‌کند، C10STOK را برای ذخیره نام در یک جدول فراخوانی می‌کند، وقتی همه نام‌ها وارد شده‌اند، D10SORT ، F10DISP را فراخوانی می‌کند.
- B10ENTER به کاربر اعلام می‌کند تا یک نام را وارد کند، آن را می‌پذیرد و سمت راست آن را با جای خالی پر می‌کند. وقتی همه نام‌ها وارد شدند، کاربر می‌تواند (Enter) را بفشارد و هیچ نامی وارد نکند.
- C10STOR هر نام را بترتیب در جدول ذخیره می‌کند.
- D10SORT ، E10XCHG جدول نام‌ها را بترتیب صعودی مرتب می‌کند.
- F10DISP جدول مرتب شده را نمایش می‌دهد.

توجه کنید که ورودیهای جدول همه طول ثابت ۲۰ بایت را دارند، روتینی که داده‌های جدول با طول متغیر را مرتب کند پیچیده‌تر خواهد بود.

لیست‌های پیوندی

یک *لیست پیوندی* حاوی داده است هر عنصر سلول نامیده می‌شود، مشابه ورودی جدول، اما با ترتیب مشخص شده نیست. برای سهولت جستجو به سمت جلو، هر سلول حاوی *اشاره‌گری* است که مبنی بر موقعیت بعدی ورودی در جدول است (یک سلول ممکن است حاوی اشاره‌گری به ورودی قبلی باشد بنابراین جستجو ممکن است در هر جهتی باشد). این روش، افزودن و حذف از یک لیست را بدون نیاز به توسعه یا تخفیف لیست، ساده می‌سازد.

```
            page 60,132
TITLE       A15NMSRT (EXE)  Sort names entered from terminal
            .MODEL SMALL
            .STACK 64
; ------------------------------------------------------
            .DATA
NAMEPAR     LABEL  BYTE                 ;Name parameter list:
MAXNLEN     DB     21                   ;  maximum length
NAMELEN     DB     ?                    ;  no. of chars entered
NAMEFLD     DB     21 DUP(' ')          ;  name

CRLF        DB     13, 10
ENDADDR     DW     ?
MESSG1      DB     'Name? '
NAMECTR     DB     00
NAMETAB     DB     30 DUP(20 DUP(' ')) ;Name table
NAMESAV     DB     20 DUP(?), 13, 10
SWAPPED     DB     00
; ------------------------------------------------------
            .CODE
A10MAIN     PROC   FAR
            MOV    AX,@data             ;Initialize DS and
            MOV    DS,AX                ;  ES registers
            MOV    ES,AX
            CLD
            CALL   Q10CLR               ;Clear screen
            CALL   Q20CURS              ;Set cursor
            LEA    DI,NAMETAB
A20LOOP:
            CALL   B10ENTER             ;Accept name from KB
            CMP    NAMELEN,00           ;Any more names?
            JZ     A30                  ;  no,  go to sort
            CMP    NAMECTR,30           ;30 names entered?
            JE     A30                  ;  yes, go to sort
            CALL   C10STOR              ;Store entered name in table
            JMP    A20LOOP
A30:                                    ;End of input
            CALL   Q10CLR               ;Clear screen
            CALL   Q20CURS              ;  and set cursor
            CMP    NAMECTR,01           ;One or no name entered?
            JBE    A40                  ;  yes, exit
            CALL   D10SORT              ;Sort stored names
            CALL   F10DISP              ;Display sorted names
A40:        MOV    AX,4C00H             ;End processing
            INT    21H
A10MAIN     ENDP
;           Accept name as input:
;           ---------------------
B10ENTER    PROC
            MOV    AH,40H               ;Request display
            MOV    BX,01                ;File handle
            MOV    CX,06                ;6 characters
            LEA    DX,MESSG1            ;Prompt
            INT    21H
            MOV    AH,0AH
            LEA    DX,NAMEPAR           ;Accept name
            INT    21H
```

شکل الف ۷-۱۵ مرتب کردن یک جدول از نام‌ها

```
            MOV     AH,40H              ;Request display
            MOV     BX,01               ;File handle
            MOV     CX,02               ;2 characters
            LEA     DX,CRLF             ;Return/line feed
            INT     21H

            MOV     BH,00               ;Clear characters after name
            MOV     BL,NAMELEN          ;Get count of chars
            MOV     CX,21
            SUB     CX,BX               ;Calc remaining length
B20:
            MOV     NAMEFLD[BX],20H     ;Set name to blank
            INC     BX
            LOOP    B20
            RET
B10ENTER    ENDP
;                   Store name in table:
;                   --------------------
C10STOR     PROC
            INC     NAMECTR             ;Add to number of names
            CLD
            LEA     SI,NAMEFLD
            MOV     CX,10               ;Ten words
            REP MOVSW                   ;Name (SI) to table (DI)
            RET
C10STOR     ENDP
;                   Sort names in table:
;                   --------------------
D10SORT     PROC
            SUB     DI,40               ;Set up stop address
            MOV     ENDADDR,DI
D20:
            MOV     SWAPPED,00          ;Set up start
            LEA     SI,NAMETAB          ; of table
D30:
            MOV     CX,20               ;Length of compare
            MOV     DI,SI
            ADD     DI,20               ;Next name for compare
            MOV     AX,DI
            MOV     BX,SI
            REPE CMPSB                  ;Compare name to next
            JBE     D40                 ;  no exchange
            CALL    E10XCHG             ;  exchange
D40:
            MOV     SI,AX
            CMP     SI,ENDADDR          ;End of table?
            JBE     D30                 ;  no, continue
            CMP     SWAPPED,00          ;Any swaps?
            JNZ     D20                 ;  yes, continue
            RET                         ;  no, end of sort
D10SORT     ENDP
;                   Exchange table entries:
;                   -----------------------
E10XCHG     PROC
            MOV     CX,10               ;Number of characters
            LEA     DI,NAMESAV
            MOV     SI,BX
            REP MOVSW                   ;Move lower item to save
            MOV     CX,10               ;Number of characters
            MOV     DI,BX
            REP MOVSW                   ;Move higher item to lower

            MOV     CX,10
            LEA     SI,NAMESAV
            REP MOVSW                   ;Move save to higher item
```

شکل ب ۷-۱۵ مرتب کردن یک جدول از نام‌ها

```
            MOV     SWAPPED,01          ;Signal exchange made
            RET
E10XCHG     ENDP
;                   Display sorted names:
;                   ---------------------
F10DISP     PROC
            LEA     SI,NAMETAB
K20:
            LEA     DI,NAMESAV          ;Init'ze start of table
            MOV     CX,10               ;Count for loop
            REP MOVSW
            MOV     AH,40H              ;Request display
            MOV     BX,01               ;File handle
            MOV     CX,22               ;20 characters + CR/LF
            LEA     DX,NAMESAV
            INT     21H
            DEC     NAMECTR             ;Is this last one?
            JNZ     K20                 ;  no, loop
            RET                         ;  yes, exit
F10DISP     ENDP
;                   Clear screen:
;                   ------------
Q10CLR      PROC
            MOV     AX,0600H
            MOV     BH,61H              ;Attribute
            MOV     CX,00               ;Full screen
            MOV     DX,184FH
            INT     10H
            RET
Q10CLR      ENDP
;                   Set cursor:
;                   ----------
Q20CURS     PROC
            MOV     AH,02H              ;Request set cursor
            MOV     BH,00               ;Page 0
            MOV     DX,00               ;Location 00:00
            INT     10H
            RET
Q20CURS     ENDP
            END     A10MAIN
```

شکل پ ۷-۱۵ مرتب کردن یک جدول از نام‌ها

یک لیست پیوندی را در نظر بگیرید که حاوی سلولهایی با شماره بخش (یک مقدار ASCII یک بایتی)، قیمت واحد (کلمه دودویی) و یک اشاره‌گر (کلمه دودویی) به شماره بخش بعدی است. پس ورودی ۸ بایت طول دارد. یک اشاره‌گر یک افست از شروع لیست است. لیست پیوندی از افست 0000 آغاز می‌شود، دومین عنصر از سری در 0024 قرار دارد و سومین در 0032 و الی آخر.

افست	شماره بخش	قیمت	موقعیت شماره بخش بعدی
0000	013	12.50	0024
0008	1720	08.92	0016
0016	1827	03.75	0000
0024	0120	13.80	0032
0032	0205	25.00	0008

عنصری که در افست 0016 حاوی 0000 در آدرس بعدی است، هم مبنی بر انتهای لیست است و هم برای دایره وار کردن لیست می‌باشد.

برنامه شکل ۸-۱۵ از محتویات تعریف شده در لیست پیوندی، LINKLIST استفاده می‌کند، تا در موقعیت یک شماره بخش خاص قرار گیرد، که در این حالت ۱۷۲۰ است. جستجو با اولین عنصر در جدول آغاز می‌شود. منطق استفاده از CMPSB مشابه شکل ۴-۱۵ می‌باشد. روال B10LINK شماره بخش (۱۷۲۰) را با هر عنصر جدول مقایسه می‌کند، حالت‌های منتج مقایسه :

- **کمتر:** عنصر در جدول نیست.

```
TITLE     A15LNKLS (EXE)  Use of a Linked List
          .MODEL SMALL
          .STACK 64                ;Define stack
          .DATA
PARTNO    DB      '1720'           ;Part number
LINKLST   DB      '0103'           ;Linked list table
          DW      1250, 24
          DB      '1720'
          DW      0895, 16
          DB      '1827'
          DW      0375, 00
          DB      '0120'
          DW      1380, 32
          DB      '0205'
          DW      2500, 08
;  ------------------------------------------------------
          .CODE                    ;Define code segment
A10MAIN   PROC    FAR
          MOV     AX,@data         ;Set address of data
          MOV     DS,AX            ;   segment in DS
          MOV     ES,AX            ;   and ES register
          CALL    B10LINK
          MOV     AX,4C00H         ;End processing
          INT     21H
A10MAIN   ENDP

B10LINK   PROC    NEAR
          CLD
          LEA     DI,LINKLST       ;Initialize table address
B20:
          MOV     CX,04            ;Set to compare 4 bytes
          LEA     SI,PARTNO        ;Init'ze part# address
          REPE    CMPSB            ;Part# : table
          JE      B30              ;  equal, exit
          JB      B40              ;  low, not in table
          ADD     DI,CX            ;Add CX value to offset
          ADD     DI,02            ;Get offset of next item
          MOV     DX,[DI]
          LEA     DI,LINKLST
          ADD     DI,DX
          CMP     DX,00            ;Last table entry?
          JNE     B20
          JMP     B40
B30:
;                 <Item Found>
          JMP     B90
B40:
;                 <Display error message>

B90:      RET
B10LINK   ENDP
          END     A10MAIN
```

شکل ۸-۱۵ استفاده از یک لیست پیوندی

- **بیشتر:** روال افست را از جدول برای عنصر بعدی که باید مقایسه شود می‌گیرد. اگر افست صفر نباشد، مقایسه برای عنصر بعدی تکرار می‌شود؛ اگر افست صفر است، جستجو بدون یافتن یک مورد موافق خاتمه می‌یابد.

- *مساوی:* عنصر پیدا شد.

یک برنامه کامل‌تر به کاربر این امکان را می‌دهد تا از صفحه کلید هر شماره بخش را وارد می‌کند و می‌تواند قیمت آن را در قالب مقادیر ASCII نمایش دهد:

عملگرهای نوع، طول و اندازه

اسمبلر تعدادی عملگر خاص دارد که ممکن است آن‌ها را مفید حس کنید. برای مثال، طول یک جدول در هر زمان ممکن است تغییر کند و باید برنامه را برای حسابرسی تعریف جدید تغییر دهید و روتین‌هایی برای بررسی انتهای جدول بیفزایید. با استفاده از عملگرهای TYPE ، LENGTH و SIZE می‌توانید تعداد دستوراتی که باید تغییر کند، کاهش دهید.

تعریف جدول زیر با ۱۲ کلمه را ملاحظه کنید :

```
RAINTBL  DW  12 DUP(?) ; جدول با ۱۲ کلمه
```

برنامه می‌تواند از عملگر TYPE برای تعیین تعریف (در این حالت DW)، عملگر LENGTH برای تعیین ضریب DUP (12) و عملگر SIZE برای تعیین تعداد بایت‌ها(24 = 12 × 12) استفاده کند. مثال‌های زیر سه عملگر را مشخص می‌سازد :

```
MOV  AX,TYPE RAINTBL     ; AX = 0002H (۲ بایت)
MOV  BX,LENGTH RAINTBL   ; BX = 000CH (۱۲ بایت)
MOV  CX,SIZE RAINTBL     ; CX = 0018H (۲۴ بایت)
```

می‌توانید از مقادیری که LENGTH و SIZE باز می‌گردانند، در جستجو یا مرتب نمودن جدول استفاده کنید. برای مثال اگر ثبات SI حاوی آدرس افست افزوده از یک جستجو باشد، می‌توانید این افست را چنین بررسی کنید:

```
CMP SI,SIZE RAINTBL
```

فصل ۲۶ عملگرهای TYPE ، LENGTH و SIZE را با جزئیات توضیح می‌دهد.

نکات کلیدی

- در اکثر موارد، یک جدول حاوی داده‌هایی مرتبط با طول و قالب داده‌ای یکسان می‌باشد.
- یک جدول بر پایه قالب داده است، برای مثال، ورودیها می‌توانند کاراکتری یا عددی یا هر نوع طول مشابه دیگری باشد.
- حداکثر مقدار عددی برای یک DB، ۲۵۶ است DW و DD بایت‌ها را معکوس می‌کنند. نیز CMPSW CMP کلمات را حاوی بایت‌هایی به ترتیب معکوس در نظر می‌گیرد.
- اگر یک جدول بطور مداوم تغییر می‌کند، یا اگر چندین برنامه به جدول مراجعه می‌کنند، آن را بر روی دیسک ذخیره نمایید. یک برنامه بروز رسانی شده می‌تواند تغییرات جدول را دستکاری نماید. هر برنامه می‌تواند جدول را از روی دیسک بارگذاری کند و دیگر برنامه‌ها نیازی به تغییر ندارند.
- با آدرس‌دهی مستقیم جدول ، برنامه آدرس ورودی جدول را محاسبه می‌کند و آن ورودی را مستقیماً دستیابی می‌نماید.
- در هنگام جستجوی جدول، یک برنامه مرتباً عناصر داده را با هر ورودی در جدول مقایسه می‌کند تا یک مورد موافق را بیابد.
- دستور XLAT تبدیل یک قالب داده به قالب دیگر را سهل می‌سازد.

پرسش‌ها

۱-۱۵. تفاوت بین پردازش جدول توسط آدرس‌دهی مستقیم و جستجو در چیست.

۲-۱۵. یک جدول با نام TEMPTBL با ۳۶۵ کلمه تعریف کنید، (الف) با صفر برای داده‌های دودویی مقدار دهی نمایید، (ب) با جای خالی برای داده‌های کاراکتری مقدار دهی کنید.

۳-۱۵. سه جدول مرتبط مجزا که حاوی داده‌های زیرمی‌باشد، تعریف کنید: (الف) عنصر ASCII عددی ۵، ۹، ۱۲، ۲۳ (ب) توضیح اقلام از diskettes ، videotape ، receivers ، modems ، keyboards (ج) قیمت عناصر 15.95 ، 79.35 ، 87.45 ، 93.75 و 12.50

۴-۱۵. پرسش ۳-۱۵ را طوری بازنویسی کنید که همه داده‌ها در یک جدول باشند. برای اولین عنصر، شماره و توضیح آن را در اولین خط و قیمت را در دومین خط تعریف کنید، برای عنصر دوم، آن را در خط سوم و چهارم و غیره تعریف، نمایید.

۵-۱۵. شکل ۱-۱۵ را طوری بازنویسی کنید که ماه را در قالب عددی (ASCII) از صفحه کلید بپذیرد. اگر ورودی معتبر (۱۲-۱) باشد، ماه را بصورت حروف الفبایی نمایش دهد، در غیر اینصورت یک پیغام خطا نمایش دهد. برای هر تعداد از ورودی صفحه کلید امکان داشته باشد، خاتمه پردازش وقتی باشد که کاربر در جلوی اعلان فقط <Enter> را تایپ کند همچنین، جدول را بازنویسی کند طوری که یک علامت $ بعد از هر ورودی باشد. بجای انتقال توضیح به ALFMON، برنامه آن را مستقیماً از جدول نمایش دهد.

۶-۱۵. برنامه‌ای بنویسید که به کاربر امکان ورود شماره عنصر (ITEMIN) و کمیت آن را (QTYIN) از صفحه کلید، فراهم سازد. از جدول تعریف شده در پرسش ۴-۱۵ استفاده کنید، و یک روتین جستجو داشته باشید که از ITEMIN، برای قرار گرفتن در شماره عنصر در جدول استفاده کند. توضیح و قیمت را از جدول اقتباس کنید. میزان فروش (قیمت × کمیت) هر یک را محاسبه، و توضیح و این مقدار را بر روی صفحه نمایش دهید.

۷-۱۵. از جدول توضیح تعریف شده در پرسش ۳-۱۵ استفاده کنید، برنامه‌ای بنویسید که (الف) محتویات جدول را به جدول خالی دیگری منتقل کند، (ب) محتویات این جدول جدید را به ترتیب صعودی مرتب کند. (ج) هر توضیح را در یک سطر در صفحه نمایش دهد. صفحه را حرکت طوماری دهد.

۸-۱۵. شکل ۵-۱۵ را طوری بازنویسی کنید که پردازش را بر عکس انجام دهد - که داده‌های EBCDIC را به قالب ASCII تبدیل کند. کاراکترهای EBCDIC که باید تبدیل شوند، علامت منها (60H)، نقطه اعشار (4BH) و اعداد ۹-۰ (F0H-F9H) می‌باشند. بقیه کاراکترها را باید به جای خالی ASCII تبدیل شوند. برای داده، از یک رشته کاراکترهای شانزدهی EBCDIC حاوی F0F0F1F24BF5F060 استفاده کنید، که باید به قالب ASCII تبدیل و نمایش داده شوند. حاصل شانزدهی خواهد بود: 303031322E35302D.

۹-۱۵. یک برنامه بنویسید که رمزگذاری ساده‌ای از داده‌ها را فراهم می‌سازد. یک ناحیه داده ۸۰ بایتی با نام CRYPDATA حاوی هر داده ASCII تعریف کنید. یک جدول تبدیل برای تبدیل داده‌ها بصورت تصادفی مرتب کنید، برای مثال، A به M، B به R ، C، به X و غیره. برای همه ۲۵۶ مقدار بایت ممکن فراهم باشد. یک جدول تبدیل دومی فراهم کنید که داده‌ها را معکوس کند. برنامه باید فعالیت‌های زیر را انجام دهد: (الف) محتویات اصلی CRYPDATA را بر روی یک خط نمایش دهد، (ب) CRYPDATA را کدگذاری نمایید و داده‌ها را بر روی خط دوم نمایش دهید. (ج) CRYPDATA را کد برداری نمایید و داده‌های کدگذاری شده را بر روی خط سوم نمایش دهید که باید با خط اول مساوی باشد.

فصل ۱۶

حافظه دیسک ۱: سازمان

هدف: بررسی قالب اصلی دیسک سخت و حافظه دیسک، رکورد راه‌انداز، شاخه‌ها و جدول اختصاص فایل

مقدمه

یک برنامه‌نویس جدی باید با جزئیات تکنیکی از سازمان دیسک آشنایی داشته باشد، بویژه برای نوشتن برنامه‌های سودمندی که محتویات دیسکت و دیسک سخت را بررسی می‌کنند، این آشنایی الزامی است.

این فصل مفاهیم تراک، سکتور و سیلندر را توضیح داده و ظرفیت برخی ابزارهای مورد استفاده عمومی را ارائه می‌دهد. همچنین سازماندهی داده‌های مهم ثبت شده در شروع یک دیسک، شامل رکورد راه انداز (که به سیستم کمک می‌کند تا سیستم عامل را از دیسک به حافظه بارگذاری کند) شاخه (که حاوی نام، موقعیت و وضعیت هر فایل روی دیسک است) و جدول تخصیص فایل (یا FAT، که فضای دیسک را برای فایل‌ها تخصیص می‌دهد) در این فصل مورد بررسی قرار می‌گیرند.

در این متن برای دیسک سخت یا فلاپی دیسک از واژه عمومی **دیسک** استفاده می‌شود

مشخصات دیسک

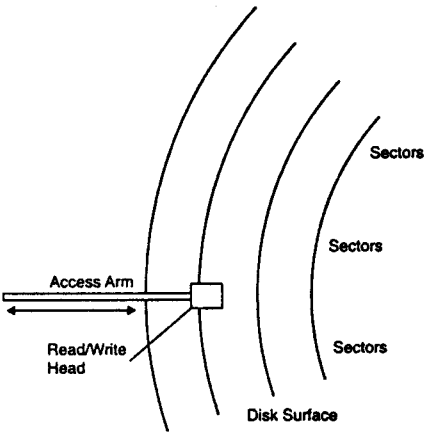
برای پردازش رکوردها برروی دیسک‌ها، باید باعبارات و مشخصات سازمان دیسک آشناشد. یک دیسکت دو طرف (یا سطح) دارد، در حالیکه یک دیسک سخت، شامل تعدادی دیسک دو طرفه است که بر روی یک محور قرار دارند.

تراک‌ها و سکتورها

هرطرف یک دیسکت یادیسک‌سخت شامل تعدادی ***تراک*** هم‌مرکز است‌که از 00 شماره‌گذاری‌شده‌اند. 00 بیرونی‌ترین تراک می‌باشد. هر تراک با ***سکتورهایی*** به اندازه ۵۱۲ بایت قالب بندی می‌شود داده‌ها در آن سکتورها ذخیره می‌شوند. هم دیسکت و هم دیسک سخت توسط یک ***کنترلر***[1] کنترل می‌شوند. وظیفه کنترلر، تنظیم هِدهای[2] خواندنی، نوشتنی، بر روی سطح دیسک و انتقال داده بین دیسک و حافظه می‌باشد. برای هر سطح دیسک یک هِد خواندن - نوشتن وجود دارد. در دیسک سخت و دیسکت، درخواست برای خواندن یا نوشتن سبب می‌شود تا کنترلر راه انداز دیسک، هد خواندن نوشتن را (در صورت لزوم) به تراک مورد نیاز منتقل کند. کنترلر سپس منتظر می‌شود تا هد به نقطه‌ای که باید عملیات خواندن یا نوشتن انجام گیرد، برسد. برای مثال در عملیات خواندن کنترلر هر بیت را از سکتور می‌خواند. شکل ۱-۱۶ این جزئیات را نشان می‌دهد.

1- Controller

2- Head



شکل ۱-۱۶ سطح دیسک و هد خواندن | نوشتن

دو تفاوت، عمده بین یک دیسک سخت و یک دیسکت وجود دارد. برای دیسک سخت، هد خواندن - نوشتن دقیقاً بالای سطح دیسک بدون لمس آن قرار می‌گیرد، در حالیکه برای دیسکت هد خواندن نوشتن دقیقاً سطح را لمس می‌کند. همچنین، یک دیسک سخت بطور پیوسته می‌چرخد، در حالیکه چرخش ابزار دیسکت، برای هر عملیات خواندن/نوشتن شروع و خاتمه می‌یابد.

سیلندرها

یک ***سیلندر*** یک مجموعه عمودی از همه تراک‌هایی است که بر روی هر سطح از یک دیسکت یا دیسک سخت با یک شماره قرار دارند. بنابراین سیلندر شامل کلیه تراک‌های صفر در هر طرف می‌باشد و همین موضوع در مورد دیگر سیلندرها به کار می‌رود. پس برای یک دیسکت، سیلندر 0 شامل تراک 0 بر روی طرف ۱ و تراک صفر بر روی طرف ۲، سیلندر ۱ شامل تراک ۱ بر روی طرف ۱ و تراک ۱ بر روی طرف ۲ و الی آخر می‌باشد. شماره طرف و هِد یکی می‌باشد، برای مثال هِد دیسک ۱ داده‌های طرف یک را دستیابی می‌کنند. وقتی یک فایل نوشته می‌شود، کنترلر همه تراک‌ها بر روی یک سیلندر را پر می‌کند و سپس بر روی هدهای خواندن - نوشتن سیلندر بعدی پیش می‌رود. برای مثال ، سیستم همه سیلندرهای صفر دیسکت (همه سکتورهای تراک 0، طرف ۱ و ۲، را پر می‌کند و سپس به سیلندر یک، طرف یک پیش می‌رود. همانطور که ملاحظه شد، مراجعه به هر طرف دیسک، تراک‌ها و سکتورها با اعداد انجام می‌گیرد. شمارهٔ طرف و تراک‌ها با صفر آغاز می‌شود، اما سکتورها با یکی از دو روش زیر شماره‌گذاری شوند:

۱) ***سکتور فیزیکی*** : شماره‌های سکتور بر روی هر تراک با یک شروع می‌شود، بنابراین اولین سکتور روی دیسک بصورت سیلندر 0، هد / طرف صفر، سکتور یک آدرسدهی می‌شود و بعدی بصورت سیلندر صفر، هد / طرف صفر، سکتور ۲ و غیره آدرسدهی می‌شود.

۲) ***سکتور رابطه‌ای*** : شماره سکتورها ممکن است با شروع دیسک ارتباط داشته باشند، بنابراین اولین سکتور روی دیسک، روی سیلندر 0، تراک 0 بصورت سکتور رابطه‌ای صفر آدرسدهی شود، بعدی بصورت سکتور رابطه‌ای شماره یک و تا آخرین سکتور روی دیسک.

عملیات دیسک مختلف، ممکن است یکی از این دو روش یا روش دیگری را استفاده نمایند، که به نحوه دستیابی انجام شده بستگی دارد.

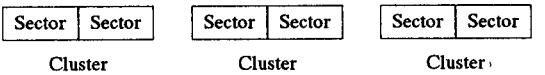
کنترلر دیسک

کنترلر دیسک، که بین پردازشگر و راه‌انداز دیسک قرار گرفته، همه ارتباطات بین آن‌ها را دستکاری می‌کند. کنترلر داده‌ها از پردازشگر می‌پذیرد و آن‌ها را به فرمی تبدیل می‌کند که توسط ابزارها قابل استفاده باشند. برای مثال، پردازشگر ممکن است از یک سیلندر، هد، سکتور خاص درخواست داده داشته باشد.

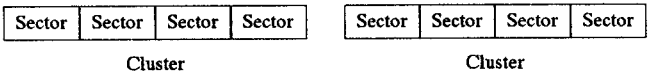
وظایف کنترلر عبارتند از: فراهم سازی فرمانهای مناسب برای انتقال بازوی دسترسی به سیلندر مورد نیاز، انتخاب هد خواندن / نوشتن و پذیرش داده از سکتور وقتی داده از هد خواندن - نوشتن می‌رسند. وقتی کنترلر، کارش را انجام می‌دهد، پردازشگر برای دیگر وظایف آزاد خواهد شد. با این روش دستیابی، کنترلر در هر بار فقط یک بیت را دستکاری می‌کند. اما، کنترلر می‌تواند با کنار گذاشتن پردازشگر و انتقال مستقیم داده به / یا از حافظه عملیات ورودی / خروجی را سریع‌تر انجام دهد. روش انتقال بلوک بزرگی از داده با استفاده از این تکنیک بعنوان دستیابی مستقیم حافظه (DMA) شناخته می‌شود. بدین منظور، پردازشگر، برای کنترلر تعدادی فرامین خواندن یا نوشتن ، آدرس بافر I/O در حافظه، تعداد سکتورهایی که باید منتقل شود، تعداد سیلندر، هد و سکتور شروع را فراهم نموده است. با این روش، پردازشگر تا زمانی که DMA کامل شود باید منتظر بماند، زیرا فقط یک جزء در هر زمان می‌تواند از مسیر حافظه استفاده نماید.

کلاسترها

یک **کلاستر** یک گروه از سکتورهاست که سیستم با آنها مانند یک واحد فضای حافظه برخورد می‌کند. اندازه کلاستر، همیشه توانی از ۲ است، مانند ۱، ۲، ۴، یا ۸ سکتور. بر روی ابزار دیسکی که یک سکتور در هر کلاستور دارد، سکتور و کلاستر یکی هستند. یک دیسک با دو سکتور در هر کلاستر می‌تواند مشابه این باشد:



و یک دیسک با چهار سکتور در هر کلاستر چنین خواهد بود:



معمولاً یک دیسک سخت چهار سکتور در هر کلاستر دارد. یک فایل از مرز یک کلاستر آغاز می‌شود و حتی اگر فایل فقط یکی از چهار سکتور را اشغال نماید حداقل به یک کلاستر نیاز دارد، یک کلاستر ممکن است از یک تراک به دیگری نیز سرریز داشته باشد.

یک فایل ۱۰۰ بایتی (آنقدر کوچک که در یک سکتور جا می‌گیرد) ذخیره شده بر روی دیسکی با چهار سکتور در هر کلاستر، از ۲۰۴۸=۵۱۲×۴ بایت حافظه استفاده می‌کند، اگر چه فقط یک سکتور حاوی داده است. سیستم کلاسترهای هر فایل را به ترتیب صعودی ذخیره می‌کند، اگر چه یک فایل ممکن است قطعه قطعه شود و برای مثال در کلاسترهای ۸، ۹، ۱۰، ۱۴، ۱۷ و ۱۸ قرار گیرد.

ظرفیت دیسک

در اینجا ظرفیت ذخیره سازی دیسکت‌های عمومی آورده شده است.

در مورد دیسک سخت، با توجه به حداکثر اندازه پارتیشن‌ها و نوع رسانه ظرفیت‌های متنوعی وجود دارند. عملیات مفید برای تعیین تعداد سیلندرها، سکتورها در هر تراک، یا هد خواندن نوشتن از طریق توابع 1FH و 1440DH در وقفه 21H و کد فرعی 60H انجام می‌شوند که در فصل ۱۸ توضیح داده خواهد شد.

Capacity	Tracks per Side (Cylinders)	Sectors per Track	Bytes per Sector	Total, Two Sides	Sectors per Cluster
5.25" 360KB	40	9	512	368,640	2
5.25" 1.2MB	80	15	512	1,228,800	1
3.5" 720KB	80	9	512	737,280	2
3.5" 1.44MB	80	18	512	1,474,560	1
3.5" 2.88MB	80	36	512	2,949,120	–

ناحیه سیستمی دیسک و ناحیه داده

سکتورهای خاص برای ذخیره سازی اطلاعاتی راجع به فایل روی دیسک در نظر گرفته شده است. سازمان دیسکت و دیسک سخت برطبق ظرفیت‌متفاوت‌است.یک دیسک سخت وبرخی‌دیسکت‌هابصورت‌خود راه‌انداز فرمت‌می‌شوند، که با آن‌ها می‌توان پردازش را در هنگام روشن شدن کامپیوتر یا هنگامی که کاربر کلیدهای Ctrl+Alt+Del را بفشارد، آغاز نمود. سازمان عمومی دیسک شامل یک ناحیه سیستم است، که پس از آن تا انتهای دیسک ناحیه داده قرار دارد.

ناحیه سیستم

ناحیه سیستم اولین ناحیه دیسک است که برروی خارجی‌ترین تراک با شروع از طرف صفر،تراک صفر، سکتور صفر قراردارد. اطلاعاتی که سیستم درناحیه سیستم ذخیره می‌سازد و نگه‌داری می‌کند برای تعیین موارد زیر استفاده‌می‌شود. برای مثال، موقعیت شروع هر فایل که بر روی دیسک ذخیره شده است. سه جزء اصلی ناحیه سیستم عبارتند از:

۱) رکورد راه انداز ۲) جدول تخصیص فایل (FAT) ۳) شاخه

ناحیه سیستم و ناحیه داده چنین سازماندهی می‌شوند :

Boot Record	FAT	Directory	System files	User files
⟵ System area ⟶			⟵ area Data ⟶	

لیست زیر سازمان ابزارهای دیسکت مختلف است، که شماره‌های سکتور شروع و خاتمه برای رکورد راه‌انداز، FAT و شاخه‌ها را نشان می‌دهد. سکتورها با شماره سکتور رابطه‌ای مشخص می‌شوند، در حالیکه سکتور رابطه‌ای 0 در سیلندر 0، تراک 0، سکتور ۱، قرار دارد، که اولین سکتور ابزار است (اخیراً در بخش سیلندرها توضیح داده شد) :

Device	Boot	FAT	Directory	Sectors/Cluster
5.25" 360KB	0	1-4	5-11	2
5.25" 1.2MB	0	1-14	15-28	1
3.5" 720KB	0	1-6	7-13	2
3.5" 1.44MB	0	1-18	19-32	1

برای دیسک سخت، موقعیت رکورد راه‌انداز و FAT اغلب مانند دیسکت است، در حالیکه اندازه FAT و موقعیت شاخه‌ها در ابزارها مختلف است. یک دیسکت فرمت شده در عبارت سکتورهای فیزیکی و رابطه‌ای شامل اطلاعات زیر است

File	720K (9 sectors/track)				1.44MB (18 sectors/track)			
	Cyl.	Side	Sector	Relative Sector	Cyl.	Side	Sector	Relative Sector
Boot record	0	0	1	0	0	0	1	0
FAT1	0	0	2	1	0	0	2	1
FAT2	0	0	4	4	0	0	11	10
Directory	0	0	8	7	0	1	2	19
Data area	0	1	6	1	0	1	16	33

فایلهای داده روی دیسکت 720K از سیلندر 0، طرف اول، سکتور ۶ تا ۹ آغاز می‌شود. سیستم رکوردهای بعدی را بر روی سیلندر یک، طرف 0، سپس سیلندر طرف ۱، سپس سیلندر ۲، طرف صفر و الی آخر... ذخیره می‌سازد. این نکتهٔ پرکردن داده‌ها روی طرف‌های متضاد (در یک سیلندر) قبل از پیش رفتن به سیلندر بعدی حرکت هد دیسک را می‌کاهد و روشی است که هم بر روی دیسکت‌ها و هم برای دیسک سخت استفاده می‌شود.

ناحیه داده

ناحیه داده برای دیسک راه انداز یا دیسکت، با دو فایل سیستم با نام‌های IO.SYS و MSDOS.SYS (برای MS-DOS) یا IBMBIO.COM تا IBMDOS.COM (برای IBMPC-DOS) آغاز می‌شود. وقتی از فرمان FORMAT/S برای فرمت کردن دیسک استفاده می‌کنید، DOS این فایلهای سیستمی را بر روی اولین سکتور ناحیه داده کپی می‌کند. فایلهای کاربران بلافاصله بعد از فایلهای سیستمی قرار می‌گیرد در صورت عدم وجود فایل‌های سیستمی فایل‌های کاربر در ابتدای ناحیه داده قرار می‌گیرند.
مبحث بعدی رکورد راه‌انداز، شاخه و FAT را توضیح می‌دهد.

رکورد راه انداز

رکورد راه انداز شامل دستوراتی است که فایل‌های سیستم IO.SYS ، MSDOS.SYS و COMMAND.COM (اگر باشد) را از روی دیسک بر حافظه قرار می‌دهد (یا راه‌اندازی می‌کند). همه دیسک‌های فرمت شده حتی اگر فایل‌های سیستمی بر روی آن ذخیره نشده باشند شامل یک رکورد راه انداز می‌باشند. رکورد راه انداز بترتیب آدرس افست حاوی اطلاعات زیر می‌باشد:

00H پرش کوتاه یا دور به روتین bootstrap در افست 1EH یا 3EH در رکورد راه‌انداز.
03H نام کارخانهٔ سازنده و شمارهٔ نسخه DOS وقتی که راه‌انداز ایجاد می‌شود.
0BH تعداد بایت‌های هر سکتور ، اغلب 200H (۵۱۲).
0DH تعداد سکتورهای هر کلاستر (۱، ۲ ، ۴ یا ۸)
0EH سکتورهای رزرو شده
10H تعداد کپی‌های FAT (۱ یا ۲)
11H تعداد ورودیهای شاخه ریشه[1]
13H مجموع تعداد سکتورها اگر Volume کمتر از ۳۲MB باشد.
15H بایت توصیف میانی (مانند بایت اول از FAT، قبلاً توضیح داده شد).
16H تعداد سکتورهای FAT
18H تعداد سکتورهای هر تراک
1AH تعداد هدهای خواندن - نوشتن (طرف‌ها یا سطح‌ها)
1CH تعداد سکتورهای پنهان
1EH روتین بارگذار Boot strap برای نسخه‌های DOS تا 3.3
20H مجموع تعداد سکتورها اگر volume بیشتر از ۳۲MB باشد.
24H شماره ابزار فیزیکی (برای دیسکت A=0، برای دیسکت سخت، درایو C=80H وغیره)
25H توسط سیستم رزور شده است.

1- Root Directory

26H علامت سکتور راه انداز توسعه داده شده (حاوی 29H)

27H Volume ID

2BH بر چسب Volume

36H توسط سیستم رزرو شده است

3EH-1FFH در DOS4.0 بارگذار bootstrap از اینجا آغاز می‌شود.

DOS 4.0 رکورد راه‌انداز را با فیلدهای اضافی از 20H تا 1FFH توسعه می‌دهد. رکورد راه‌انداز اصلی 20H (۳۲) بایت است، در حالیکه نسخه توسعه داده شده 200H (۵۱۲) بایت می‌باشد.

شاخه

هر فیلد در دیسک از مرز یک کلاستر آغاز می‌شود، که اولین سکتور از کلاستر است. برای هر فایل DOS یک ورودی شاخه ۳۲ بیتی (20H) ایجاد می‌کند که نام فایل، تاریخ ایجاد آن، اندازه آن و موقعیت کلاستر شروع را مشخص می‌سازد. ورودیهای شاخه شکل زیر را دارند :

بایت	منظور
00H-07H	***نام فایل***، در برنامه‌ای که فایل ایجاد شده تعریف می‌شوند. اولین بایت مبنی بر وضعیت فایل می‌باشد.
00H	فایلی که هرگز استفاده نشده است
05H	اولین کاراکتر نام فایل در عمل همیشه E5H است
2EH	ورودی است برای یک زیر شاخه
E5H	فایل حذف شده است
08H-0AH	***توسعه نام فایل***، مانند EXE یا ASM
0BH	***صفت فایل***، تعریف نوع فایل (توجه کنید که یک فایل ممکن است بیشتر از یک صفت داشته باشد)
00H	فایل معمولی
01H	فایلی که فقط خواندنی است
02H	فایل پنهان، با جستجوی شاخه نمایش داده نمی‌شود
04H	فایل سیستم، با جستجوی شاخه نمایش داده نمی‌شود
08H	برچسب، Volume (اگر یک رکورد برچسب volume وجود داشته باشد، در نام فایل و فیلد توسعه خواهد بود)
10H	زیر شاخه
20H	فایل آرشیو، مبنی بر این است که فایل از هنگام آخرین بروز رسانی، بازنویسی شده است. بعنوان مثال کد 07H به معنی این است یک فایل است (04H)، فقط خواندنی (01H) و پنهانی (02H) است.
0CH-15H	توسط سیستم رزرو شده است.
16H-17H	***زمان ایجاد*** فایل یا بروزرسانی آن، در ۱۶ بیت قالب دودویی بصورت hhhhh mmmmmm sssss ذخیره می‌شود.
18H-19H	***تاریخ ایجاد*** فایل یا آخرین بروز رسانی، در ۱۶ بیت در قالب دودویی به شکل yyyyyyy mmmm ddddd ذخیره می‌شود. سال می‌تواند 119-1BH000-1AHکه ۱۹۸۰ نقطهٔ شروع است باشد، ماه ممکن است ۱۲-۱ و روز ۳۱-۱ باشد.
1AH-1BH	***کلاستر شروع*** فایل. عدد آن به آخرین دو سکتور شاخه مرتبط است. در صورت عدم وجود فایل‌های

سیستمی اولین فایل داده از کلاستر رابطه‌ای 002 شروع می‌شود. طرف فعال، تراک و کلاستربه ظرفیت دیسک بستگی دارد. یک ورودی صفر به معنی آن است که فایل هیچ فضایی را اشغال نکرده است.

1CH-1FH ***اندازه فایل*** به بایت. وقتی یک فایل ایجاد می‌کنید. سیستم اندازه فایل را محاسبه و در این فیلد ذخیره می‌سازد.

برای فیلدهای عددی که از یک بایت در شاخه تجاوز می‌کنند، داده‌ها، بترتیب بایت معکوس ذخیره می‌شود.

جدول تخصیص فایل

منظور از FAT تخصیص فضای دیسک به فایل‌هاست. FAT حاوی ورودی برای هر کلاستر از دیسک است. وقتی شما یک فایل جدید ایجاد می‌کنید یا یک فایل موجود را اصلاح می‌نمایید، سیستم FAT را مطابق با مکان فایل روی دیسک اصلاح می‌کند. FAT از سکتور ۲، بلافاصله پس از رکورد راه انداز آغاز می‌شود. در دیسکی که یک کلاستر شامل چهار سکتور است، یک عدد ورودی از FAT می‌تواند چهار بار مقدار داده‌هایی را از دیسک ارجاع دهد در مقایسهٔ یک کلاستر که حاوی یک سکتور است. در نتیجه، استفاده از کلاسترها با چندین سکتور تعداد ورودیهای FAT را می‌کاهد و سیستم را توانا می‌سازد تا یک فضای حافظه دیسک بزرگتری را آدرسدهی نماید.

طراحان اصلی دو کپی از FAT فراهم نموده‌اند (FAT2 و FAT1) تا در صورت خراب شدن FAT1، FAT2 قابل استفاده باشد. اگر چه FAT2 تعبیه شده است ولی هیچگاه در عمل مورد استفاده قرار نمی‌گیرد در ناحیه سیستم دیسک و ناحیه داده، FAT1 و FAT2 در قسمت ضروریات حافظه FAT قرار می‌گیرد. همه مباحث این فصل بر FAT1 متمرکز شده است.

اولین ورودی در FAT

اولین بایت از FAT یعنی توصیف‌گر کد رسانه، نوع ابزار را مشخص می‌کند و (بایت 15H در رکورد راه انداز را ببینید) شامل موارد زیر :

F0H "3.5 دو طرفه، ۱۸ سکتور / تراک (1.44MB) و "3.5 دو طرفه، ۳۶ سکتور/ تراک (2.88MB)

F8H دیسک سخت (شامل دیسک RAM)

F9H "3.5 دو طرفه، ۹ سکتور / تراک (720KB) و "5.25 دو طرفه ۱۵ سکتور / تراک (1.2MB)

FCH "5.25 یک طرفه، ۹ سکتور / تراک (180KB)

FDH "5.25 دو طرفه، ۹ سکتور / تراک (360KB)

FFH "5.25 دو طرفه، ۸ سکتور / تراک (320KB)

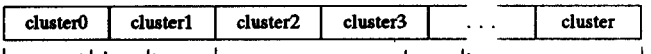
توجه کنید که F0H و F9H هر کدام دو نوع دیسک مختلف را مشخص می‌کند.

دومین ورودی در FAT

دومین ورودی FAT، حاوی FFFFH برای FAT دیسکت که ۱۲ بیت ورودیهای FAT را پشتیبانی می‌کند و FFFFFFH برای دیسک سخت که ۱۶ بیت ورودیهای FAT را پشتیبانی می‌کند. اولین دو ورودی FAT مشابه این خواهند بود.

دیسکت 1.44MB	F0	FF	FF	..	..	..	..	..	. . .	..
دیسکت سخت	F8	FF	FF	FF	..	..	..	..	. . .	..

همانطور که گفته شد، اولین فیلد روی دیسک، رکورد راه انداز است، پس از آن FAT و سپس شاخه ناحیه داده بعد از آن است. شکل ورودی چنین خواهد بود.



ممکن است گمان کنید که ناحیه داده در نقطه شروع کلاستر است، اما اولین دو کلاستر (۰ و ۱) به شاخه اشاره می‌کند، بنابراین ناحیه داده برای ذخیره سازی فایل‌های داده از شماره کلاستر ۲ آغاز می‌شود. دلیل این حالت فرد بزودی مشخص خواهد شد.

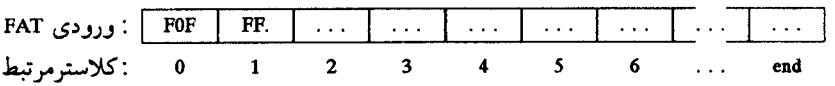
ورودیهای اشاره‌گر در FAT

بعد از ورودیهای اولین دو FAT، ***ورودیهای اشاره‌گر*** هستند که با هر کلاستر در ناحیه داده در ارتباط هستند. شاخه (در 1BH-1AH) حاوی موقعیت اولین کلاستر برای یک فایل است و FAT حاوی زنجیره‌ای از ورودیهای اشاره‌گر برای هر کلاستر بعد از آن است.

طول ورودی دیسکت ۲ رقم شانزدهی است (۱ ۱/۲ بایت و یا ۱۲ بیت) اما برای دیسک سخت ۴ رقم شانزدهی (۲ بایت یا ۱۶ بیت) است. هر اشاره‌گر FAT مبنی بر استفاده از کلاستر خاص بر طبق قالب زیر می‌باشد :

12 BITS	16 BITS	EXPLANATION
000	0000	Referenced cluster is currently unused
nnn	nnnn	Relative number of next cluster for a file
FF0–FF6	FFF0–FFF6	Reserved cluster
FF7	FFF7	Unusable (bad track)
FFF	FFFF	Last cluster of a file

اولین دو ورودی یک دیسکت 1.44MB (یک FAT ۱۲ بیتی است) مشابه این :

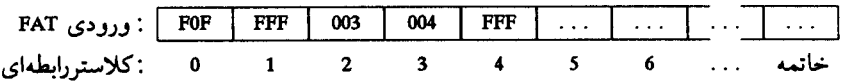


عبارت کلاستر مرتبط به معنی کلاستری است که ورودی FAT به آن اشاره می‌کند، در یک حالت، اولین دو ورودی FAT (۰ و ۱) به آخرین دو کلاستر در شاخه اشاره می‌کنند، که مانند شروع کلاستر تعیین شده‌اند، دایرکتوری مبنی بر اندازه و کلاستر شروع فایل‌هاست.

دایرکتوری حاوی شماره کلاستر شروع برای هر فایل است و یک زنجیره از ورودیهای اشاره‌گر FAT که بر موقعیت کلاستر بعدی دلالت دارد، اگر فایل ادامه داشته باشد. یک ورودی اشاره‌گر شامل FFFH(F) مبنی بر آخرین کلاستر یک فایل است.

نمونه‌هایی از ورودیهای FAT

مثال زیر به روشن شدن ساختار FAT کمک می‌کند. فرض کنید یک دیسکت شامل فقط یک فایل با نام CUSTOMER.FIL است که بطور کامل در کلاسترهای ۲، ۳ و ۴ ذخیره شده است. ورودی شاخه برای این فایل حاوی نام فایل CUSTOMER، توسعه FIL ، 00H مبنی بر یک فایل معمولی، تاریخ ایجاد، 0002H برای موقعیت اولین کلاستر رابطه‌ای از فایل، و یک ورودی اندازه فایل در بایت می‌باشد. ورودی 12 بیتی FAT به شکل زیر ظاهر خواهد شد، بجز آنکه جفت بایتها معکوس خواهند شد.



برای اولین دو ورودی FAT ، F0 مبنی بر دیسکت 1.44MB دو طرفه ۹ سکتور است که پس از FFFFFH قرار گرفته است. در برنامه‌ای که فایل COSTOMER.FIL را بطور ترتیبی از دیسک به حافظه، می‌خواهند سیستم مراحل زیر را طی می‌کند.

- برای اولین کلاستر، شاخه دیسک را برای نام فایل COSTOMER و توسعه FIL جستجو می‌کند، از شاخه موقعیت اولین کلاستر رابطه‌ای (۲) فایل را اقتباس می‌کند، و محتویات آن (داده‌ها از سکتورها) را برنامه در حافظه اصلی ارائه می‌دهد.
- برای کلاستر بعدی، ورودی اشاره‌گر FAT را دستیابی می‌کند که کلاستر رابطه‌ای ۲ را بیان می‌کند. در شکل فوق، این ورودی حاوی 003 است، به معنی این که فایل در کلاستر رابطه‌ای ۳ ادامه دارد. سیستم محتویات این کلاستر را به برنامه ارائه می‌دهد.
- برای کلاستر آخر ورودی اشاره‌گر FAT را دستیابی می‌کند که کلاستر رابطه‌ای ۳ را بیان می‌کند. ورودی حاوی 004 است، به معنی آن که فایل در کلاستر رابطه‌ای ۴ ادامه دارد. سیستم محتویات این کلاستر را به برنامه ارائه می‌دهد.

ورودی FAT در کلاستر رابطه‌ای ۴ حاوی FFFH است مبنی بر اینکه کلاسترهای بیشتری برای این فایل تخصیص داده نشده است. حال سیستم همه داده‌های فایل را از کلاستر ۲، ۳ و ۴ ارائه داده است. دیدیم که ورودیهای FAT در اصل چگونه عمل می‌کنند، حال چگونگی عمل آنها را وقتی به ترتیب بایت معکوس هستند، یعنی جایی که تسلطی بیشتر نیاز است، بررسی می‌کنیم.

کار با ورودیهای 12 بیتی FAT به ترتیب بایت معکوس

در زیر مثال ورودیهای FAT برای CUSTOMER.FIL این بار با ورودیهای اشاره‌گر بترتیب بایت معکوس ارائه شده است. ۱۲ بیت FAT برای این فایل چنین خواهد بود :

ورودی FAT :	F0F	FFF	034	000	FF0	FXX	. . .
کلاستر رابطه‌ای :	0	1	2	3	4	5	

اما چیزی که حال مورد نیاز است از رمز درآوردن ورودیهایی است که بیشتر بر طبق بایت مرتبط بیان می‌شوند تا کلاسترها :

ورودی FAT :	F0	FF	FF	03	40	00	FF	0F	. . .
بایت رابطه‌ای :	0	1	2	3	4	5	6	7	. . .

در اینجا مراحل دستیابی به کلاسترها ذکر شده است :

- برای پردازش اولین ورودی FAT، ۲ را (اولین کلاستر فایل که در شاخه ثبت شده است) در ۱۰۵ (طول ورودیهای FAT) ضرب‌کنید تا عدد ۳ بدست‌آید، (برای برنامه‌نویسی، در ۳ ضرب کنید و یک بیت به راست تغییر مکان دهید). کلمهٔ بایت‌های ۳ و ۴ در FAT را دستیابی کنید. این بایت‌ها حاوی 0340 است که وقتی معکوس شوند عدد 4003 بدست می‌آید. چون کلاستر ۲ یک عدد زوج است، از سه رقم آن استفاده کنید، بنابراین 003 کلاستر دوم فایل است.
- برای سومین کلاستر، شماره کلاستر ۳ را در ۱/۵ ضرب کنید حاصل ۴ است. بایت‌های ۴ و ۵ FAT را بازیابی کنید. این بایت‌ها حاوی 4000 هستند که اگر معکوس شوند حاصل 0040 خواهد شد. چون کلاستر ۳ یک عدد فرد است، از سه رقم اول استفاده کنید، که 004 کلاستر سوم فایل خواهد بود.
- برای چهارمین کلاستر، ۴ را در ۱.۵ ضرب کنید حاصل ۶ خواهد شد. بایت‌های ۶ و ۷ FAT را دستیابی نمایید. این بایت‌ها حاوی FF 0F هستند، که اگر معکوس شوند 0FFF خواهد شد. چون کلاستر ۴ یک عدد زوج است، از سه رقم آخر، FFF استفاده کنید که یعنی آخرین ورودی است.

کار با داده‌های ۱۶ بیتی FAT

همانطور که ذکر شد، توصیف گر رسانه برای دیسک سخت FFFFFFH خواهد بود، ورودی‌های اشاره‌گر FAT ۱۶ بیت طول دارند و با بایت‌های ۳ و ۴ آغاز می‌شوند، که کلاستر ۲ را بیان می‌کنند. ورودی شاخه کلاستر شروع فایل‌ها را فراهم می‌سازد و ورودی اشاره‌گر FFFFH مبنی بر انتهای فایل است. تعیین شُماره کلاستر از هر ورودی FAT ساده است، اگر چه بایت‌ها در هر ورودی بترتیب بایت معکوس هستند. در زیر مثالی از ورودیهای ۱۶ بیتی FAT بیان می‌کنیم. فرض کنید که تنها فایل روی یک دیسک سخت ویژه چهار کلاستر را اشغال می‌کنند (۴ سکتور بر هر کلاستر، یا ۱۶ سکتور در کل). بر طبق شاخه، فایل از کلاستر ۲ آغاز می‌شود. هر ورودی اشاره‌گر FAT یک کلمه کامل است، بنابراین معکوس بایت‌ها فقط یک ورودی را شامل می‌شود. در اینجا FAT با ورودیهای اشاره‌گر به ترتیب بایت معکوس آورده شده است :

ورودی FAT :	F8FF	FFFF	0300	0400	0500	FFFF	. . .
کلاستر رابطه‌ای :	0	1	2	3	4	5	

ورودی FAT برای کلاستر ۲ رابطه‌ای، 0300 است که برای کلاستر بعدی بصورت 0003 معکوس می‌شود. ورودی FAT برای کلاستر رابطه‌ای ۳، 0400 است که برای کلاستر بعدی بصورت 0004 معکوس می‌شود. با زنجیره‌ای از ورودیهای باقیمانده تا ورودی شماره کلاسترها ادامه می‌یابد. اگر برنامه شما باید نوع دیسک که نصب شده را تعیین نماید، می‌تواند توصیف گر رسانه در سکتور راه انداز را مستقیماً بررسی کند یا ترجیحاً تابع 1BH یا 1CH وقفه 21H استفاده نماید.

تمرین بررسی FAT

بیایید از DEBUG برای بررسی FAT، یک دیسک استفاده کنیم. برای این تمرین، به دو دیسکت خالی فرمت شده "3.5 با ظرفیت 720K و 1.44MB نیاز دارید فایل‌های سیستمی نباید روی این دو دیسک کپی شده باشند. دو فایل بر روی هر یک کپی کنید. اولین فایل بزرگتر از ۵۱۲ بایت و کوچکتر از 1,024 بایت باشد، تا بر روی دو سکتور قرار گیرد. فرض کنید این فایل A04ASM1.ASM باشد. فایل دوم باید بزرگتر ۱۵۳۶ بایت باشد و کوچکتر از 2048 بایت تا بر ۴ سکتور قرار گیرد فرض کنید نام فایل دوم A10DRVID.ASM باشد. خواهید دید که FAT در هر دو دیسکت مشابه است، اما یکسان نیست.

روال برای دیسک 720K

ابتدا دیسکت 720K را در درایو A قرار دهید (یا در صورت لزوم در درایو B). DEBUG را اجرا کنید و فرمان L را وارد کنید: (برای درایو B از 20 0 1 100 L استفاده کنید) 20 0 0 100 L. ورودی فرمان L چنین خواهد بود:

- 100H افست شروع در سگمنت DEBUG خواهد بود جایی که داده‌ها باید خوانده شوند.
- اولین 0 یعنی از درایو A استفاده شده است (یا برای درایو B)
- دومین 0 یعنی خواندن داده از سکتور رابطه‌ای صفر آغاز می‌شود.
- 20 یعنی 20H (۳۲) سکتور خواهد شد.

حال می‌توانید رکورد راه انداز شاخه و FAT این دیسکت را بررسی نمایید. برای شروع نمایش، فرمان 100 D را وارد کنید. چون فایل از افست 100H ذخیره شده است، می‌توانید این رکوردها را چنین قرار دهید :

۱) رکورد راه انداز از 100H آغاز می‌شود.

۲) FAT پس از سکتور راه انداز است : 100H + 200H (یک سکتور 200H یا ۵۱۲ بایت) = 300H

۳) دایرکتوری پس از FAT است: 300H + [200H × سکتور 6] = F00H

رکورد راه انداز. برخی فیلدهای رکورد راه انداز چنین هستند :

- افست سگمنت 103H نام کارخانه و نسخه DOS وقتی FAT ایجاد می‌شود را نشان می‌دهد.
- افست 10BH تعداد بایت‌های هر سکتور (در حالیکه 0002H بصورت 0200H معکوس می‌شود یا ۵۱۲ بایت) را نشان می‌دهد.
- در افست 115H توصیف گر رسانه است، برای این دیسکت F9H می‌باشد. فیلدهای دیگر را بررسی کنید.

شاخه. برای شاخه، فرمان D F00 را وارد کنید، که:

- افست F00H شامل نام فایل اول است A04ASM1.ASM
- افست F1AH شماره کلاستر شروع (0200 و یا 0002) را برای این فایل می‌دهد.
- افست F1CH - F1FH اندازه فایل به بایت را می‌دهد.
- افست F20H ورودی برای دومین فایل A10DRVID.ASM را آغاز می‌کند. توجه کنید که F3AH کلاستر شروع را بصورت 0300، یا 0003 نشان می‌دهد.

FAT. برای FAT، فرمان D 300 را وارد کنید، که باید چنین نمایش داده شود:

ورودی FAT :	F9	FF	FF	FF	4F	00	FF	OF	...
بایت مرتبط :	0	1	2	3	4	5	6	7	

- F9 توصیف گر رسانه است
- FFFF در بایت‌های ۱ و ۲ محتویات فیلد دوم است.

ورودیهای اشاره‌گر با شروع از بایت ۳ می‌توانند چنین محاسبه شوند:

- برای اولین فایل، ۲ (بر طبق شاخه اولین کلاستر) را در ۱.۵ ضرب کنید حاصل ۳ می‌باشد. افست بایت‌های ۳ و ۴ را در FAT دستیابی نمایید، که حاوی FF4F است و معکوس آن 4FFF خواهد بود. چون کلاستر ۲ یک عدد زوج است، از سه رقم آخر استفاده کنید. FFF، که می‌گوید هیچ کلاستر دیگری برای این فایل وجود ندارد.
- برای فایل دوم، ۳ (این بر طبق شاخه اولین کلاستر است) را در ۱.۵ ضرب کنید که ۴ حاصل می‌شود. بایت ۴ چون کلاستر ۳ یک عدد فرد است، از سه رقم اول استفاده کنید، 004، که کلاستر بعدی در سری را مشخص می‌سازد. ۴ را در ۵.۱ ضرب کنید، ۶ حاصل می‌شود. بایت‌های ۶ و ۷ در FAT را دستیابی کنید که حاوی FF0F است و بایت‌ها را معکوس کنید، 0FFF خواهد شد. چون کلاستر ۴ یک عدد زوج است از سه رقم آخر استفاده کنید، FFF، که مبنی بر خاتمه داده‌هاست.

روالی برای دیسک 1.44MB

حال یک دیسکت 1.44MB را در درایو A قرار دهید و فرمان DEBUG ، 030 0 100 L را وارد کنید. (30H سکتور را بارگذاری کنید زیرا چند FAT روی دیسکت 1.44MB قرار دارد). رکورد راه‌انداز از 100H آغاز می‌شود و FAT پس از آن در 300H = 200H + 100H قرار دارد. چون FAT برای یک دیسکت 1.44MB، ۱۸ سکتور طول دارد، شاخه در 2700H = (12H × 200H) + 300H قرار دارد.

رکورد راه انداز. از فرمان D 100 برای نمایش رکورد راه انداز، استفاده کنید. توجه کنید که توصیف‌گر میانی در 115H ، F0H است و تعداد سکتور هر کلاستر (در 10DH) یک می‌باشد.

شاخه. برای شاخه، فرمان D 2700 را وارد کنید. نام اولین فایل در 2700H قرار دارد و کلاستر شروع (۲) در 271A می‌باشد. نام دومین فایل در 2720H و کلاستر شروع (۴) در 273A می‌باشد. (کلاستر شروع برای دومین فایل در دیسکت 720K، ۳ می‌باشد زیرا در شکل کلی، دو سکتور در هر کلاستر دارد، در حالیکه یک دیسکت 1.44MB یک سکتور در هر کلاستر دارد.)

FAT. برای FAT فرمان D 300 را وارد کنید، که چنین نمایش داده می‌شود:

ورودی FAT :	F0	FF	FF	03	F0	FF	05	60	00	07	F0	FF
بایت رابطه‌ای :	0	1	2	3	4	5	6	7	8	9	10	11

بر طبق شاخه، اولین فایل از کلاستر ۲ شروع می‌شود، بنابراین ۲ را در ۱.۵ ضرب کنید تا بایت رابطه‌ای ۳ را بدست آورید. بایت ۳ و ۴ حاوی 03F0 است، که بصورت F003 معکوس می‌شوند. چون کلاستر ۲ یک عدد زوج است، از سه رقم آخر 003 استفاده کنید. کلاستر ۳ × ۱.۵ می‌شود ۴، که بایت‌های رابطه‌ای ۴ و ۵ F0FF می‌باشد و بصورت FFF0 معکوس می‌شوند. چون کلاستر ۳ یک عدد فرد است از سه رقم اول، FFF استفاده کنید، که مبنی بر انتهای فایل می‌باشد. حال می‌دانیم که فایل در کلاسترهای ۲ و ۳ بازنویسی شده است. از همین تکنیک برای پیگیری زنجیره فایل سوم استفاده کنید، که با کلاستر ۴ ، یا بایت رابطه‌ای ۶ آغاز می‌شود. INT 21H برخی سرویس‌های پشتیبانی برای برنامه به منظور دستیابی به اطلاعاتی راجع به شاخه و FAT فراهم می‌سازد، شامل توابع 47H (گرفتن شاخه جاری) و 1BH و 1CH (گرفتن اطلاعات FAT) که در فصل ۱۸ توصیف خواهد شد.

پردازش فایلها روی دیسک

داده‌ها بر روی دیسک به شکل ***فایل*** ذخیره می‌شوند، دقیقاً مانند آنکه شما برنامه خودتان را ذخیره نموده‌اید. اگر چه محدودیتی برای نوع داده‌هایی که در یک فایل نگه داری می‌شود وجود ندارد، یک نوع فایل کاربر شامل ***رکوردهایی*** برای مشتریان، فهرست موجودی، یا لیست‌های نام و آدرس می‌باشد. هر رکورد حاوی اطلاعاتی راجع به یک مشتری خاص یا فهرست اقلام می‌باشد. داخل یک فایل، همه رکوردها همیشه (اما نه الزاماً) یک قالب و طول را دارند. یک رکورد شامل یک یا چند فیلد است که اطلاعاتی راجع به رکورد فراهم می‌سازند. رکوردهای یک فایل مشتریان، برای مثال، می‌تواند شامل فیلدهایی مانند شماره مشتریان، نام مشتریان و میزان پرداختی باشد. رکوردها می‌توانند بترتیب صعودی شماره مشتریان بصورت زیر باشند :

#1	name	amt	#2		name	amt	#3	name	amt	. . .	#n	name	amt

پردازش فایلها روی هارد دیسک مشابه دیسکت است، برای هر دو باید نام مسیر دستیابی فایل‌ها در زیر شاخه‌ها را فراهم سازید.

سرویس‌های وقفه برای ورودی / خروجی دیسک

تعدادی از سرویس‌های وقفه خاص ورودی / خروجی دیسک را پشتیبانی می‌کنند. یک برنامه که یک فایل می‌نویسد (***یا ایجاد می‌کند***) ابتدا سبب می‌شود تا سیستم یک ورودی برای آن در شاخه ایجاد کند. وقتی تمام رکوردهای فایل نوشته شد، برنامه فایل را ***می‌بندد*** بنابراین سیستم می‌تواند ورودی شاخه را برای اندازه فایل تکمیل نماید. برنامه‌ای که یک فایل را می‌خواند ابتدا آن را ***باز می‌کند*** تا مطمئن شود چنین فایلی وجود دارد. وقتی برنامه همه رکوردها را خواند، فایل را می‌بندد و آن را در اختیار برنامه‌های دیگر قرار می‌دهد. بدلیل نوع طراحی شاخه، ممکن است رکوردها را در یک فایل دیسکی یا بطور ترتیبی (یک رکورد پس از دیگری، بطور متوالی) یا تصادفی (رکوردها بطوری که نیاز هست بازیابی شوند) پردازش نمایید.

بالاترین سطح. پردازش دیسک بر اساس INT 21H می‌باشد، که دستیابی دیسک را توسط شاخه و بلوکه کردن و عدم بلوکه کردن رکوردها پشتیبانی می‌کند. این روش برخی پردازش‌های مقدماتی را قبل از پیوند با BIOS انجام می‌دهد. فصل ۱۷ استفاده از عملیات DOS برای خواندن و نوشتن فایلهای دیسکی در بر دارد، فصل ۱۸ در مورد عملیات مختلفی که شاخه‌ها و فایلهای دیسکی را پشتیبانی می‌کنند، بحث می‌کند.

پایین‌ترین سطح. پردازش دیسک بر اساس وقفه INT 13H از BIOS است، که شامل آدرسدهی مستقیم تراک‌ها و شماره‌های سکتور است، در فصل ۱۹ توضیح داده خواهد شد.

نکات کلیدی

- هر طرف یک دیسکت یا دیسک سخت شامل تعدادی تراک‌های هم مرکز است، که با شماره 00 آغاز می‌شود. هر تراک به سکتورهای ۵۱۲ بایتی فرمت می‌شوند، که با شماره ۱ آغاز می‌شود.
- یک سکتور ممکن است با هد - سیلندر یا با شماره سکتور رابطه‌ای ارجاع شود.
- یک کلاستر، گروهی از سکتورهاست که با آنها مانند واحدی از فضای حافظه برخورد می‌کند. اندازه یک کلاستر توانی از ۲ است، مانند ۱، ۲، ۴، یا ۸ سکتور. یک فایل از مرز یک کلاستر آغاز شده و حداقل به یک کلاستر نیاز دارد
- صرف نظر از اندازه، همه فایل‌ها از مرز یک کلاستر آغاز می‌شوند.
- رکورد راه‌انداز، شامل دستوراتی است که فایل‌های سیستم MSDOS.SYS و COMMAND.COM و IO.SYS را از دیسک با حافظه بارگذاری می‌کنند.
- شاخه شامل یک ورودی برای هر فایل روی دیسک است که مبنی بر نام فایل، توسعه، صفت فایل، زمان، تاریخ، سکتور آغازین و اندازه فایل می‌باشد.
- هدف از جدول تخصیص فایل (FAT)، تخصیص فضای دیسک به فایل‌هاست. FAT از سکتور ۲ بلافاصله پس از رکورد راه انداز آغاز می‌شود و شامل یک ورودی برای هر کلاستر هر فایل در شاخه می‌باشد.

پرسش‌ها

۱-۱۶. طول استاندارد سکتور بر حسب بایت چقدر است؟

۲-۱۶. سیلندر چیست؟

۳-۱۶. هدف از کنترلر دیسک چیست؟

۴-۱۶. (الف) کلاستر چیست؟ (ب) هدف از آن چیست؟ ،(ج) فضای دیسک (با عبارت بایت) برای هر اندازه کلاستر ۱، ۲، ۴ و ۸ چیست؟

۵-۱۶. نشان‌دهید که چگونه برمبنای تعداد سیلندرها، سکتورهای هرتراک و بایت‌های هر سکتور ظرفیت یک دیسکت برای موارد زیر محاسبه می‌شود؟ (الف) یک دیسکت "3.5 ، 1.44MB و (ب) یک دیسکت "5.25، 360KB.

۶-۱۶. سه بخش ناحیه سیستم دیسک چیست؟

۷-۱۶. (الف) هدف از رکورد راه انداز چیست؟، (ب) در کجا قرار دارد؟، (ج) چگونه می‌توان از آن برای تعیین تعداد سکتورها در هر تراک استفاده کنید؟

۸-۱۶. چگونه یک شاخه بر حذف فایل دلالت می‌کند؟

۹-۱۶. در شاخه چه چیز مبنی‌بر موارد زیر می‌باشد، (الف) یک فایل معمولی، (ب) یک فایل فقط خواندنی، (ج)یک فایل سیستمی؟

۱۰-۱۶. تاثیراضافی بریک دیسک‌سخت یا دیسکت، وقتی‌از FORMAT/S برای فرمت کردن استفاده‌می‌کنید، چیست؟

۱۱-۱۶. یک فایل به اندازه ۳۱۶۵ بایت (دهدهی) را در نظر بگیرید، (الف) سیستم کجا اندازه را ذخیره می‌کند، (ب) اندازه آن در قالب شانزدهی چیست؟ مقداری را به طوری که سیستم آن را ذخیره می‌سازد نشان دهید.

۱۲-۱۶. کجا و چطور FAT مبنی بر ابزاری است که نصب شده بر روی (الف) یک دیسک سخت می‌باشد؟ (ب) یک دیسکت "3.5 ، 1.44MB می‌باشد، (ج) یک دیسکت "5.25 ، 360KB می‌باشد؟

۱۳-۱۶. چطور FAT مبنی بر وردیهای ۱۲ بیت و ۱۶ بیت می‌باشد؟

فصل ۱۷

حافظه دیسک ۲ : نوشتن و خواندن فایل‌ها

هدف: استفاده از دستگیره‌های فایل و عملیاتی برای نوشتن و خواندن فایلهای دیسکی بطور ترتیبی و تصادفی

مقدمه

سرویس‌های اصلی برای پردازش فایل‌های دیسکی از روشی به نام بلوک‌های کنترل فایل (FCBS) استفاده می‌کند. این روش، اگر چه هنوز توسط DOS پشتیبانی می‌شود، می‌تواند درایوها و نام‌های فایل را آدرسدهی کنند، اما قادر به آدرسدهی زیر شاخه‌ها نیستند. نسخه‌های موفق DOS تعدادی سرویس‌های پیشرفته معرفی می‌کنند که ساده‌ترند قابلیت بیشتری از نسخه‌های اصلی خود دارند. برخی از این عملیات مستلزم استفاده از رشته ASCIIZ برای تعیین اولیه درایو، مسیر و نام فایل می‌باشد، یک دستگیره فایل برای دستیابی مرتب به فایل و کدهای بازگشتی خاص برای تشخیص خطاها می‌باشد.

اگر چه هیچ دستور زبان اسمبلی جدیدی مورد نیاز نیست، این فصل تعدادی از سرویس‌های INT 21H برای پردازش فایل‌های دیسکی معرفی می‌کند. این دستورات در اینجا مرتب شده‌اند:

عملیات با استفاده از FCBS		عملیات با استفاده از دستگیره فایل	
0FH	بازکردن فایل	3CH	ایجاد فایل
10H	بستن فایل	3DH	بازکردن فایل
14H	خواندن رکورد	3EH	بست فایل
15H	نوشتن رکورد	3FH	خواندن رکورد
16H	ایجاد فایل	40H	نوشتن رکورد
21H	خواندن رکورد بطور تصادفی	42H	انتقال اشاره‌گر فایل
22H	نوشتن رکورد بطور تصادفی		دیگر سرویس‌ها :
27H	خواندن بلوک بطور تصادفی	INT 25H	خواندن مطلق
28H	نوشتن بلوک بطور تصادفی	INT 26H	نوشتن مطلق

این فصل سرویس‌هایی برای نوشتن و خواندن فایل‌های دیسکی در بر دارد و فصل ۱۸ سرویس‌های پشتیبان متنوع مورد نیاز برای دستکاری دیسک درایو، شاخه‌ها و فایل‌ها را در بر دارد. همانطور که به یاد دارید، یک کلاستر، گروهی از یک یا چندین سکتور داده است که به ابزار بستگی دارد.

رشته‌های ASCIIZ

در هنگام استفاده از سرویس‌های توسعه یافته برای پردازش دیسک، ابتدا برای سیستم آدرس یک **رشته** *ASCIIZ* شامل مشخصات فایل مهیا کنید: موقعیت دیسک درایو، مسیر شاخه و نام فایل (همه اختیاری هستند و داخل علائم نقل

قول قرار می‌گیرند) و بلافاصله بعد از آن یک بایت صفر شانزدهی، جای می‌گیرد بدین ترتیب رشته ASCIIZ نامیده می‌شود. حداکثر طول این رشته ۱۲۸ بایت است. مثال زیر یک درایو و نام فایل را تعریف می‌کند :

```
PATHNAM1 DB 'E:\A17RANRD.ASM' , 00H
```

این مثال یک درایو، زیر شاخه، و نام فایل تعریف می‌کند:

```
PATHNAME2 DB 'F:\UTILITY\A17RANRD.EXE' , 00H
```

یک کاراکتر \ مانند جداکننده مسیر عمل می‌کند. یک بایت صفر رشته را خاتمه می‌دهد. برای وقفه‌هایی که به رشته ASCIIZ نیاز دارند، آدرس افست آن را در ثبات DX قرار دهید، برای مثال، بدین شکل

```
LEA DX,PATHNAM1
```

دستگیره‌های فایل

همانطور که در فصل ۹ بحث شد، ممکن است از دستگیره‌های فایل مستقیماً جهت ابزارهای استاندارد خاص استفاده کنید. 00 = ورودی، 01 = خروجی، 02 = خروجی خطا، 03 = ابزار معین[1] و =04 چاپگر. دیگر سرویس‌های I/O مستلزم استفاده از دستگیره فایل برای عملیاتی که فایل را دستیابی می‌کنند، می‌باشد، بدین منظور باید یک شماره دستگیره فایل از سیستم تقاضا کنید. یک فایل دیسکی ابتدا باید باز شود، بر خلاف انتقال داده از صفحه کلید یا به صفحه نمایش، سیستم باید فایل دیسکی را شاخه ورودیهای FAT آدرسدهی کند و باید این ورودی بروز رسانی شود. در طی اجرای برنامه، هر فایل باید با دستگیره فایل منحصر بفرد خودش ارجاع شود.

در هنگام باز کردن فایل برای ورودی یا ایجاد یک فایل برای خروجی، سیستم یک دستگیره فایل ارائه می‌دهد. این عملیات مستلزم استفاده از رشته ASCIIZ و INT 21H تابع 3CH یا 3DH می‌باشد. دستگیره فایل یک عدد یک کلمه‌ای منحصر بفرد است که در AX باز گردانده می‌شود و می‌توانید در کلمه ذخیره نمایید و برای همه درخواست‌های بعدی برای دستیابی به فایل از آن استفاده نمایید. معمولاً، اولین دستگیره فایل بازگشتی ۰۵، و دومین ۰۶ و غیره می‌باشد. PSP حاوی یک جدول دستگیره فایل پیش فرض است که ۲۰ دستگیره را فراهم می‌سازد (بنابراین محدودهٔ معتبر برای تعداد فایل‌های باز شده ۲۰ می‌باشد) اما می‌توانید با استفاده از INT 21H تابع 67H این محدوده را افزایش دهید، این موضوع در فصل ۲۴ توضیح داده خواهد شد.

01	Invalid function number	20	Unknown unit
02	File not found	21	Drive not ready
03	Path not found	22	Unknown command
04	Too many files open	23	CRC data error
05	Access denied	24	Bad request structure length
06	Invalid handle	25	Seek error
07	Memory control block destroyed	26	Unknown media type
08	Insufficient memory	27	Sector not found
09	Invalid memory block address	28	Printer out of paper
10	Invalid environment	29	Write fault
11	Invalid format	30	Read fault
12	Invalid access code	31	General failure
13	Invalid data	32	Sharing violation
15	Invalid drive specified	33	Lock violation
16	Attempt to remove directory	34	Invalid disk change
17	Not same device	35	FCB unavailable
18	No more files	36	Sharing buffer overflow
19	Write-protected disk		

شکل ۱-۱۷ کدهای اصلی بازگشتی خطا برای دیسک

1 auxiliary

کدهای بازگشتی خطا

عملیات دستگیره فایل برای دیسک یک ***وضعیت تکمیل*** در پرچم نقلی و ثبات AX ارائه می‌دهد. یک عملیات موفق، پرچم نقلی را با صفر مقدار می‌دهد و دیگر عملیات مناسب را انجام می‌دهد. یک عملیات ناموفق پرچم نقلی رایک می‌کند و بسته به نوع عملیات یک کد خطا در AX باز می‌گرداند. شکل ۱-۱۷ کدهای خطا از ۱ تا ۳۶ را لیست می‌کند، دیگر کدها با شبکه در ارتباطند.

اگر این خطاها کافی نیست، می‌توانید INT 59H را برای اطلاعات اضافی راجع به خطا بیابید. (فصل ۱۸ را ببینید). بخش بعدی مورد ضروری جهت ایجاد، نوشتن و بستن فایل‌های دیسکی را در بر دارد.

اشاره گرهای فایل

سیستم، یک ***اشاره‌گر فایل*** مجزا برای هر فایل که برنامه پردازش می‌کند، نگه می‌دارد. عملیات ایجاد و باز کردن فایل اشاره‌گر فایل را با صفر مقدار دهی می‌کند که یعنی موقعیت شروع فایل است. اشاره‌گر فایل بعداً افست موقعیت جاری در فایل را محاسبه می‌کند.

هر عملیات خواندن / نوشتن سبب می‌شود تا سیستم اشاره‌گر فایل را با تعداد بایت‌های منتقل شده جابجا کند. پس اشاره‌گر فایل به موقعیت رکورد بعدی که باید دستیابی شود، اشاره می‌کند. استفاده از اشاره‌گر فایل باعث تسهیل پردازش ترتیب و تصادفی می‌شود. در پردازش تصادفی یک رکورد، از تابع 42H واقع در وقفه 21H (در بخش‌های بعدی توضیح داده خواهد شد) برای تنظیم اشاره‌گر فایل در هر موقعیت از فایل استفاده کنید.

استفاده از دستگیره فایل برای ایجاد فایل‌های دیسکی

روال نوشتن در یک فایل دیسکی بشرح زیر می‌باشد:

۱) با استفاده از یک رشته ASCIIZ یک دستگیره فایل از سیستم بگیرید.
۲) با استفاده از INT 21H تابع 3CH یک فایل ایجاد کنید.
۳) با استفاده از INT 21H تابع 40H یک رکورد در فایل بنویسید.
۴) در خاتمه با استفاده از INT 21H تابع 3EH فایل را ببندید.

INT 21H تابع 3CH: ایجاد فایل

برای ***ایجاد*** یک فایل جدید یا نوشتن بر روی فایل قدیم با همان نام، ابتدا از INT 21H تابع 3CH استفاده کنید. CX را با صفت فایل درخواستی (در فصل ۱۶ گفته شد) مقداردهید و DX را با آدرس رشته ASCIIZ (موقعیت دیسکی فایل جدید) مقدار دهی کنید. در اینجا یک مثال آورده شده که یک فایل معمولی بر روی درایو E با صفت 0 ایجاد می‌کند :

```
PATHNAM1  DB   ; 'E:\ACCOUNTS.F1L', 00H
FILHANDL  DW   ?                        دستگیر فایل ;
          MOV  AH,3CH                   درخواست ایجاد فایل ;
          MOV  CX , 0                   صفت معمولی ;
          LEA  DX,PATHNAME1             رشته ASCIIZ ;
          INT  21H                      فراخوانی سرویس وقفه ;
          JC   error                    عمل خاص در صورت بروز خطا ;
          MOV  FILHANDL,AX              ذخیره دستگیره در کلمه :
```

در یک عملیات معتبر ، سیستم یک ورودی شاخه با صفت داده شده ایجاد می‌کند، پرچم نقلی را پاک می‌کند و دستگیره را در در AX تنظیم می‌کند. از این دستگیره فایل برای همه دستیابی‌های بعدی به فایل استفاده کنید. فایل

نامبرده با اشاره‌گر فایل خودش که صفر است ، باز می‌شود و حال برای نوشتن در دسترس می‌باشد. اگر یک فایل با نام داده شده در مسیر وجود داشته باشد، یک فایل جدید صفر بایتی با همان نام روی فایل قبلی بازنویسی شده و آماده دریافت اطلاعات جدید خواهد شد.

در شرایط خطا، عملیات پرچم نقلی را یک می‌کند ، یک کد در AX باز می‌گرداند (شکل ۱-۱۷ را ببینید) کد ۰۵ یعنی شاخه پر است یا نام فایل ارجاع شده، صفت فقط خواندنی دارد. ابتدا پرچم نقلی را بررسی کنید. برای مثال ممکن است، ایجاد یک فایل دستگیره ۰۵ را در AX ارائه دهد، که براحتی با کد خطا لغو ۰۵ اشتباه می‌شود و دستیابی لغو می‌گردد. سرویس‌های مرتبط برای ایجاد یک فایل، توابع 5AH و 5BH از وقفه هستند، که در فصل ۱۸ توضیح داده خواهد شد.

تابع 40H از وقفه 21H : خواندن رکورد

برای نوشتن رکوردها روی دیسک، از تابع 40H واقع در وقفه 21H استفاده کنید، BX را با دستگیره فایل ذخیره شده مقدار دهید، CX را با تعداد بایت‌هایی که باید نوشته شود و DX را با آدرس ناحیه خروجی مقدار دهید. مثال بعدی در دستگیره فایل از عملیات ایجاد قبلی برای نوشتن یک رکورد ۲۵۶ بایتی از OVTAREA استفاده خواهد شد:

```
FILHAND1        DW          ; دستگیره فایل
OUTAREA  DB     256DUP( ' '); ناحیه خروجی
         MOV    AH,40H      ; درخواست خواندن رکورد
         MOV    BX,FILHAND1 ; دستگیره فایل
         MOV    CX,256      ; طول رکورد
         LEA    DX,OUTAREA  ; آدرس ناحیه خروجی
         INT    21H         ; فراخوانی سرویس وقفه
         JC     error2      ; عمل خاص در صورت بروزخطا
         CMP    AX,256      ; همه بایت‌ها نوشته شد؟
         JNE    error 3     ; اگر نه، بروز خطا
```

یک عملیات معتبر ، رکورد را بر روی دیسک می‌نویسد، اشاره‌گر فایل را می‌افزاید، پرچم نقلی را صفر می‌کند، و AX را با تعداد بایت‌هایی که واقعاً نوشته شده تنظیم می‌کند. در دیسک پر ممکن است تعداد نوشته شده با تعداد درخواستی تفاوت داشته باشد، چون سیستم این حالت را بعنوان خطا گزارش نمی‌دهد شما باید مقدار بازگشتی در AX را بررسی کنید. یک عملیات نامعتبر پرچم نقلی را یک می‌کند و کد خطای ۰۵ (دستیابی لغو شد) یا ۰۶ (دستگیره نامعتبر) را در AX باز می‌گرداند.

تابع 3EH از وقفه 21H : بستن فایل

وقتی نوشتن در یک فایل را تمام می‌کنید، باید آن را **ببندید**، دستگیره فایل را در BX قرار دهید و از تابع 3EH استفاده کنید :

```
MOV  AH,3EH       ; درخواست بستن فایل
MOV  BX,FILHAND1  ; دستگیره فایل
INT  21H          ; فراخوانی سرویس وقفه
```

یک عملیات بستن موفق، هر رکورد و باقیمانده در بافر حافظه را می‌نویسد و FAT و شاخه را با تاریخ و اندازه فایل بروز رسانی می‌کند. یک عملیات ناموفق، پرچم نقلی را یک می‌کند و در AX کد خطای ممکن را باز می‌گرداند، مثلاً 06 در حالت دستگیره نامعتبر.

برنامه: ایجاد یک فایل دیسکی

برنامه شکل ۲-۱۷ یک فایل با نامی که کاربرد وارد می‌کند، ایجاد می‌نماید. روال اصلی آن بشرح زیر است:

- A10MAIN، که B10CREAT، C10PROC را فراخوانی می‌کند، اگر در خاتمه ورودی است F10XCLSE را فرا می‌خواند.
- B10CREAT از تابع 3CH وقفهٔ 21H برای ایجاد فایل و ذخیره دستگیره آن در یک عنصر داده با نام HANDLE استفاده می‌کند.
- C10PROC ورودی را از صفحه کلیدمی‌پذیرد و موقعیت‌های پس‌از انتهای نام تاانتهای ناحیه ورودی راپاک می‌کند.
- D10SCRL وقتی به سطر انتهایی نزدیک می‌شود صفحه را حرکت طوماری می‌دهد.
- E10WRIT از تابع 40H وقفهٔ 21H برای نوشتن رکوردها استفاده می‌کند.
- F10CLSE در انتهای پردازش، جهت ایجاد ورودی شاخه صحیح، از تابع 3EH وقفهٔ 21H برای بستن فایل استفاده می‌کند.

ناحیه ورودی ۳۰ بایت است، پس از آن ۲ بایت برای Enter (0DH) و LineFeed (0AH) قرار دارد، که در مجموع ۳۲ بایت می‌شود، برنامه ۳۲ بایت را بعنوان طول ثابت رکورد می‌نویسد. می‌توانید کاراکترهای Enter/Line Feed را حذف کنید اما اگر بخواهید رکوردهای فایل را مرتب کنید، باید آنها را اضافه کنید، چون برنامه DOS SORT به این کاراکترها مبنی بر انتهای رکوردها نیاز دارد. برای این مثال، فرمان مرتب کردن رکوردها از فایل NAMEFILE.DAT به ترتیب صعودی در NAMEF1ZE.SRT چنین خواهد بود.

```
SORT  F:<NAMEFILE-DAT> NAMEFILE.SRT
```

(SORT از NAMEFILE.DAT به NAMEFILE.SRT پردازش می‌کند.)

```
            page 60,132
TITLE       A17CRFIL (EXE)  Create disk file of names
            .MODEL  SMALL
            .STACK  64
; ------------------------------------------------
            .DATA
NAMEPAR     LABEL   BYTE              ;Parameter list:
MAXLEN      DB      30                ;Maximum length
NAMELEN     DB      ?                 ;Actual length
NAMEREC     DB      30 DUP(' '), 0DH, 0AH ;Entered name,
                                      ;  CR/LF for writing
ERRCDE      DB      00                ;Error indicator
HANDLE      DW      ?                 ;File handle
PATHNAM     DB      'F:\NAMEFILE.DAT', 0
PROMPT      DB      'Name? '
ROW         DB      01
OPNMSG      DB      '*** Open error  ***', 0DH, 0AH
WRTMSG      DB      '*** Write error ***', 0DH, 0AH
; ------------------------------------------------
            .CODE
A10MAIN     PROC    FAR
            MOV     AX,@data          ;Initialize data
            MOV     DS,AX             ;  segment
            MOV     ES,AX
            MOV     AX,0600H
            CALL    Q10SCR            ;Clear screen
            CALL    Q20CURS           ;Set cursor
            CALL    B10CREAT          ;Create file
            CMP     ERRCDE,00         ;Create error?
            JZ      A20LOOP           ;  yes, continue
            JMP     A90               ;  no,  exit
```

شکل الف ۲-۱۷ استفاده از یک دستگیره برای ایجاد فایل

```
A20LOOP:
            CALL    C10PROC
            CMP     NAMELEN,00          ;End of input?
            JNE     A20LOOP             ;  no,  continue
            CALL    F10CLSE             ;  yes, close,
A90:        MOV     AX,4C00H            ;End processing
            INT     21H
A10MAIN     ENDP

;                   Create disk file:
;                   ----------------
B10CREAT    PROC    NEAR
            MOV     AH,3CH              ;Request create
            MOV     CX,00               ;Normal
            LEA     DX,PATHNAM
            INT     21H
            JC      B20                 ;Error?
            MOV     HANDLE,AX           ;  no, save handle
            RET
B20:                                    ;  yes,
            LEA     DX,OPNMSG           ;  error message
            CALL    X10ERR
            RET
B10CREAT    ENDP
;                   Accept input:
;                   ------------
C10PROC     PROC    NEAR
            MOV     AH,40H              ;Request display
            MOV     BX,01               ;Handle
            MOV     CX,06               ;Length of prompt
            LEA     DX,PROMPT           ;Display prompt
            INT     21H

            MOV     AH,0AH              ;Request input
            LEA     DX,NAMEPAR          ;Accept name
            INT     21H
            CMP     NAMELEN,00          ;Is there a name?
            JZ      C90                 ;  no, exit
            MOV     AL,20H              ;Blank for storing
            SUB     CH,CH
            MOV     CL,NAMELEN          ;Length
            LEA     DI,NAMEREC
            ADD     DI,CX               ;Address + length
            NEG     CX                  ;Calculate remaining
            ADD     CX,30               ;  length
            REP STOSB                   ;Set to blank
            CALL    E10WRIT             ;Write disk record
            CALL    D10SCRL             ;Check for scroll
C90:
            RET
C10PROC     ENDP
;                   Check for scroll:
;                   ----------------
D10SCRL     PROC    NEAR
            CMP     ROW,18              ;Bottom of screen?
            JAE     D20                 ;  yes, bypass
            INC     ROW                 ;  no,  add to row
            JMP     D90
D20:
            MOV     AX,0601H            ;Scroll one row
            CALL    Q10SCR
D90:        CALL    Q20CURS             ;Reset cursor
            RET
D10SCRL     ENDP
;                   Write disk record:
;                   -----------------
E10WRIT     PROC    NEAR
            MOV     AH,40H              ;Request write
```

شکل ب ۲-۱۷ استفاده از یک دستگیره برای ایجاد یک فایل

```
                MOV     BX,HANDLE
                MOV     CX,32           ;30 for name, 2 for CR/LF
                LEA     DX,NAMEREC
                INT     21H
                JNC     E20             ;Valid write?
                LEA     DX,WRTMSG       ;  no,
                CALL    X10ERR          ;  call error routine
                MOV     NAMELEN,00
E20:
                RET
E10WRIT         ENDP
;                       Close disk file:
;                       ---------------
F10CLSE         PROC    NEAR
                MOV     NAMEREC,1AH     ;Set EOF mark
                CALL    E10WRIT
                MOV     AH,3EH          ;Request close
                MOV     BX,HANDLE
                INT     21H
                RET
F10CLSE ENDP
;                       Scroll screen:
;                       -------------
Q10SCR          PROC    NEAR            ;AX set on entry
                MOV     BH,1EH          ;Set yellow on blue
                MOV     CX,0000
                MOV     DX,184FH
                INT     10H             ;Scroll
                RET
Q10SCR          ENDP
;                       Set cursor:
;                       ----------
Q20CURS         PROC    NEAR
                MOV     AH,02H          ;Request
                MOV     BH,00           ;  set cursor
                MOV     DH,ROW          ;Row
                MOV     DL,00           ;Column
                INT     10H
                RET
Q20CURS         ENDP
;                       Display disk error message:
;                       --------------------------
X10ERR          PROC    NEAR            ;DX contains
                MOV     AH,40H          ;  address of message
                MOV     BX,01           ;Handle for screen
                MOV     CX,21           ;Length
                INT     21H
                MOV     ERRCDE,01       ;Set error code
                RET
X10ERR          ENDP
                END     A10MAIN
```

شکل پ ۲-۱۷ استفاده از یک دستگیره برای ایجاد یک فایل

به دو نکته توجه کنید (۱) کاراکترهای Enter / Line Feed بعد از هر رکورد، فقط برای سهولت در مرتب کردن، قرار می‌گیرند و در غیر اینصورت می‌توانند حذف شوند. (۲) هر رکورد می‌تواند قالب طول متغیری داشته باشد، فقط تا انتهای نام، این مطلب مستلزم برنامه نویسی اضافی است که بعداً خواهید دید.

استفاده از دستگیره فایل برای خواندن فایل‌های دیسکی

این بخش حاوی مواد ضروری برای باز کردن و خواندن فایل‌های دیسکی با استفاده از دستگیره‌های فایل می‌باشد. روال خواندن یک فایل دیسکی بشرح زیر می‌باشد :

۱) از یک رشته ASCIIZ جهت گرفتن دستگیره فایل از سیستم، استفاده نمایید.

۲) از تابع 3DH وقفهٔ 21H برای باز کردن فایل استفاده نمایید.

۳) از تابع 3FH وقف 21H برای خواندن از فایل استفاده نمایید.
۴) در خاتمه از تابع 3EH وقفه 21H برای بستن فایل استفاده نمایید.

تابع 3DH از وقفهٔ 21H : باز کردن فایل

اگر برنامه شما، خواندن از یک فایل می‌باشد، ابتدا با استفاده از تابع 3DH آن را باز کنید. این عملیات وجود فایل مربوط را در مسیر داده شده بررسی می‌کند. در DX، آدرس رشته ASCIIZ مورد نظر را قرار دهید و AL را با کد دستیابی مقدار دهید.

BITS		REQUEST	BITS		REQUEST
0-2	000	read only	3	1	Reserved
	001	write only	4-6		Sharing mode
	010	read/write	7		Inheritance flag

قبل از خواندن یک فایل، حتماً با تابع 3DH آن را باز کنید بدین منظور استفاده از 3CH صحیح نیست زیرا این تابع باعث ایجاد فایل می‌شود. مثال زیر یک فایل را برای خواندن باز می‌کند:

```
FILEHAND2  DW   ?                    ; دستگیره فایل
           MOV  AH,3DH               ; درخواست باز کردن فایل
           MOV  AL,00                ; فقط خواندنی
           LEA  DX,PATHNAM1          ; رشته ASCIIZ
           INT  21H                  ; فراخوانی سرویس وقفه
           JC   error 4              ; عمل خاص در صورت بروز خطا
           MOV  FILHAND2,AX          ; ذخیره دستگیره در کلمه
```

اگر فایلی با نام داده شده وجود داشته باشد، عملیات طول رکورد را با ۱ تنظیم می‌کند که می‌توانید بر روی آن بنویسید)، صفت جاری فایل را در نظر می‌گیرد، اشاره‌گر را با 0 تنظیم می‌کند (شروع فایل)، پرچم نقلی را پاک می‌کند و دستگیره‌ای برای فایل در AX باز می‌گرداند. این دستگیره فایل را برای همه دستیابی‌های بعدی به فایل استفاده کنید.

اگر فایل وجود نداشته باشد، عملیات پرچم نقلی را تنظیم می‌کند و یک کد خطا در AX (02، 04، 05، 03، شکل ۱-۱۷ را ببینید) باز می‌گرداند. ابتدا، برای اطمینان پرچم نقلی را بررسی کنید. برای مثال، ایجاد یک فایل احتمالاً دستگیره ۰۵ را در AX ارائه می‌دهد، که به سادگی ممکن است با کد خطای ۰۵ اشتباه شود و دستیابی لغو گردد.

تابع 3FH از وقفه 21H : خواندن رکورد

برای خواندن رکورد از تابع 3FH وقفهٔ 21H استفاده کنید. دستگیره فایل را در BX قرار دهید، تعداد بایت‌هایی که باید خوانده شود در CX و آدرس ناحیه ورودی را در DX قرار دهید. مثال زیر از دستگیره فایل از مثال قبلی برای خواندن یک رکورد ۵۱۲ بایتی استفاده می‌کند:

```
F1LHAND2   DW   ?
INAREA     DB   512    PUP(' ')
                MOV   AH,3FH          ; درخواست خواندن رکورد
                MOV   BX,FILHAND2     ; دستگیره فایل
                MOV   CX,512          ; طول رکورد
                LEA   DX,INAREA       ; آدرس ناحیه ورودی
                INT   21H             ; فراخوانی سرویس وقفه
                JC    error5          ; عمل خاص در صورت بروز خطا
                CMP   AX,00           ; صفر بایت خوانده شد؟
                JE    endfile         ; بله، انتهای فایل
```

یک عملیات معتبر، رکورد را به برنامه تحویل می‌دهد، پرچم نقلی را صفر می‌کند و AX را با تعداد بایت‌هایی که واقعاً خوانده شده، مقدار می‌دهد. صفر در AX به معنی تلاش برای خواندن از انتهای فایل است ، وجود صفر در AX به عنوان خطا تلقی نمی‌شود، بلکه صرفاً به برنامه نویسی هشدار می‌دهد. یک خواندن نامعتبر، پرچم نقلی را یک می‌کند و در AX که خطا ۰۵ (دستیابی لغو شد) یا ۰۶ (دستگیره نامعتبر) را باز می‌گرداند.

چون سیستم تعداد فایل‌های باز در یک زمان را محدود می‌کند، برنامه‌ای که مرتباً تعدادی از فایل‌ها را می‌خواند باید آنهایی را که نیاز ندارد، ببندد.

برنامه: خواندن یک فایل دیسکی بطور ترتیبی

برنامه شکل ۳-۱۷، فایل ایجاد شده در برنامه شکل ۲-۱۷ را می‌خواند و با فرمان DOS SORT مرتب می‌کند. در اینجا روال‌های اصلی آورده شده است.

- A10MAIN ، روال‌های B10OPEN ، C10READ و D10DISP را فراخوانی می‌کند و در انتها فایل را می‌بندد و پردازش را خاتمه می‌بخشد.
- B10OPEN از INT 21H تابع 3DH برای باز کردن فایل استفاده می‌کند و دستگیره را در یک عنصر داده به نام HADLE ذخیره می‌کند.
- C10READ با استفاده از تابع 3FH واقع در وقفه 21H از دستگیره برای خواندن رکوردها استفاده می‌کند.
- D10DISP رکوردها را نمایش می‌دهد و صفحه را حرکت طوماری می‌دهد. چون کاراکترهای Enter و Line Feed بعد از هر رکورد قرار دارند، برنامه نباید در هنگام نمایش رکوردها، مکان نما را حرکت دهد.

استفاده از دستگیره فایل برای پردازش تصادفی

بحث قبلی راجع به پردازش فایل‌های ترتیبی دیسک، برای ایجاد یک فایل، چاپ محتویات آن و ایجاد تغییرات در فایل‌های کوچک کافی است. اما برخی کاربردها، مستلزم دستیابی به یک رکورد خاص در یک فایل هستند، مانند اطلاعاتی از یک کارمند خاص یا بخش‌های موجود.

برای بروز رسانی فایل با اطلاعات جدید، برنامه‌ای که محدود به پردازش ترتیبی است باید همگی رکوردها در فایل از بالا تا آن یکی که مورد نیاز است، بخواند. برای مثال، برای دستیابی به سیصدمین رکورد فایل، پردازش ترتیبی مستلزم خواندن ۲۹۹ رکورد قبلی، قبل از تحویل سیصدمین رکورد می‌باشد (اگر چه سیستم می‌تواند از یک شماره رکورد خاص شروع کند).

راه حل کلی استفاده از پردازش تصادفی است، بطوری که یک برنامه، مستقیماً می‌تواند هر رکورد داده شده در یک فایل را دستیابی نماید. اگر چه می‌توانید یک فایل را بطور ترتیبی ایجاد کند، امکان دستیابی رکوردها هم بصورت ترتیبی و هم به صورت تصادفی وجود دارد.

وقتی یک برنامه ابتدا در خواست یک رکورد تصادفی را دارد، عملیات خواندن از شاخه جهت قرار گرفتن در سکتوری که رکورد در آن باشد استفاده می‌کند، سکتور ورودی را از دیسک خوانده و در بافر قرار می‌دهد و رکورد مورد نیاز را به برنامه تحویل می‌دهد.

در مثال بعدی، رکوردهای ۱۲۸ بایت طول دارند و در هر سکتور ۴ تا قرار می‌گیرد، درخواست برای رکورد تصادفی شماره ۲۱ سبب می‌شود تا ۴ رکورد از سکتور در بافر خوانده شود:

record # 20	record # 21	record # 22	record # 23

وقتی برنامه درخواست رکورد تصادفی بعدی را داشته باشد، مثلاً شماره ۲۳، عملیات ابتدا بافر را بررسی می‌کنند. چون

رکورد در آنجا قرار دارد، مستقیماً به برنامه فرستاده می‌شود.

اگر برنامه رکوردی را درخواست کند که در بافر نیست، عملیات از شاخه برای قرارگرفتن در موقعیت سکتور حاوی رکورد استفاده می‌کند، سکتور ورودی را می‌خواند و در بافر قرار می‌دهد و رکورد را به برنامه تحویل می‌دهد. در این حالت درخواست شماره‌های رکورد تصادفی که در فایل با هم متصل هستند، دستیابی دیسک کمتری خواهد داشت.

```
TITLE     A17RDFIL (EXE)  Read disk records sequentially
          .MODEL  SMALL
          .STACK  64
          .DATA
ENDCDE    DB      00                ;End process indicator
HANDLE    DW      ?
IOAREA    DB      32 DUP(' ')
OPENMSG   DB      '*** Open error ***', 0DH, 0AH
PATHNAM   DB      'F:\NAMEFILE.SRT',0
READMSG   DB      '*** Read error ***', 0DH, 0AH
ROW       DB      00
;-------------------------------------------------------
          .CODE
A10MAIN   PROC    FAR
          MOV     AX,@data          ;Initialize
          MOV     DS,AX             ;  segment
          MOV     ES,AX             ;  registers
          MOV     AX,0600H
          CALL    Q10SCR            ;Clear screen
          CALL    Q20CURS           ;Set cursor
          CALL    B10OPEN           ;Open file
          CMP     ENDCDE,00         ;Valid open?
          JNZ     A90               ;  no, exit
A20LOOP:
          CALL    C10READ           ;Read disk record
          CMP     ENDCDE,00         ;Normal read?
          JNZ     A80               ;  no,  exit
          CALL    D10DISP           ;  yes, display name,
          JMP     A20LOOP           ;  continue
A80:      MOV     AH,3EH            ;Request close file
          MOV     BX,HANDLE         ;
          INT     21H
A90:      MOV     AX,4C00H          ;End processing
          INT     21H
A10MAIN   ENDP
;                 Open file:
;                 ---------
B10OPEN   PROC    NEAR
          MOV     AH,3DH            ;Request open
          MOV     AL,00             ;Normal file
          LEA     DX,PATHNAM
          INT     21H
          JC      B20               ;Error?
          MOV     HANDLE,AX         ;  no, save handle,
          JMP     B90               ;  return
B20:
          MOV     ENDCDE,01         ;  yes,
          LEA     DX,OPENMSG        ;  display
          CALL    X10ERR            ;  error message
B90:      RET
B10OPEN   ENDP
;                 Read disk record:
;                 ----------------
C10READ   PROC    NEAR
          MOV     AH,3FH            ;Request read
          MOV     BX,HANDLE
          MOV     CX,32             ;30 for name, 2 for CR/LF
          LEA     DX,IOAREA
          INT     21H
          JC      C20               ;Error on read?
```

شکل الف ۳-۱۷ خواندن رکوردها بصورت ترتیبی

```
                CMP     AX,00           ;End of file?
                JE      C30
                CMP     IOAREA,1AH      ;EOF marker?
                JE      C30             ;  yes, exit
                JMP     C90
C20:                                    ;  no,
                LEA     DX,READMSG      ;  invalid read
                CALL    X10ERR
C30:            MOV     ENDCDE,01       ;Force end
C90:            RET
C10READ         ENDP
;                       Display name:
;                       ------------
D10DISP         PROC    NEAR
                MOV     AH,40H          ;Request display
                MOV     BX,01           ;Set handle
                MOV     CX,32           ;  and length
                LEA     DX,IOAREA
                INT     21H
                CMP     ROW,20          ;Bottom of screen?
                JAE     D80             ;  yes, bypass
                INC     ROW             ;  no, increment row
                JMP     D90
D80:
                MOV     AX,0601H
                CALL    Q10SCR          ;Scroll
                CALL    Q20CURS         ;Set cursor
D90:            RET
D10DISP         ENDP
;                       Scroll screen:
;                       -------------
Q10SCR          PROC    NEAR            ;AX set on entry
                MOV     BH,1EH          ;Set color
                MOV     CX,0000
                MOV     DX,184FH        ;Request scroll
                INT     10H
                RET
Q10SCR          ENDP
;                       Set cursor:
;                       ----------
Q20CURS         PROC    NEAR
                MOV     AH,02H          ;Request set
                MOV     BH,00           ;  cursor
                MOV     DH,ROW          ;  row
                MOV     DL,00           ;  column
                INT     10H
                RET
Q20CURS         ENDP
;                       Display disk error message:
;                       --------------------------
X10ERR          PROC    NEAR
                MOV     AH,40H          ;DX contains address
                MOV     BX,01           ;Handle for screen
                MOV     CX,20           ;Length
                INT     21H             ;  of message
                RET
X10ERR          ENDP
                END     A10MAIN
```

شکل ب ۳-۱۷ خواندن رکوردها بصورت ترتیبی

تابع 42H از وقفه 21H: حرکت اشاره‌گر فایل

باز کردن فایل اشاره‌گر فایل را با صفر مقدار می‌دهد و خواندن و نوشتن‌های بعدی، برای هر پردازش رکورد آن را می‌افزاید. می‌توانید از تابع 42H (انتقال اشاره‌گر فایل) برای تنظیم اشاره‌گر فایل در هر جایی داخل فایل استفاده کنید و از دیگر سرویس‌های بازیابی تصادفی و یا بروز رسانی رکوردها استفاده نمایید. برای درخواست تابع 42H، دستگیره فایل را

در BX و افست مورد نیاز را در CX:DX قرار دهید. برای افستی تا ۶۵۵۳۵ بایت، CX را صفر و مقدار افست را در DX بگذارید. همچنین یک **کد روش** در AL تنظیم کنید که به عملیات می‌گوید نقطه‌ای که باید افست را در نظر بگیرد:

00 افست در شروع فایل است.

01 افست در موقعیت جاری اشاره‌گر فایل است که در هر جایی از فایل می‌تواند باشد، شامل ابتدای فایل.

02 افست در انتهای فایل است از این کد روش برای افزودن رکوردها در انتهای فایل می‌توانید استفاده کنید. یا می‌توانید اندازه فایل را با پاک کردن CX:DX با صفر تعیین کنید و کد روش 02 استفاده نمایید. مثال زیر اشاره‌گر را، ۱۰۲۴ بایت از شروع فایل حرکت می‌دهد.

```
MOV   AH,42H          ; درخواست انتقال اشاره‌گر
MOV   AL,00           ; به شروع فایل
MOV   BX,HANDLE1      ; تنظیم دستگیره فایل
MOV   CX,00           ; بخش بالایی افست
MOV   DX,1024         ; بخش پایینی افست
INT   21H             ; فراخوانی سرویس وقفه
JC    error           ; عمل خاص در صورت بروز خطا
```

یک عملیات معتبر پرچم نقلی را پاک می‌کند و موقعیت جدید اشاره‌گر را در DX: AX قرار می‌دهد، سپس ممکن است عملیات خواندن یا نوشتن برای پردازش تصادفی را انجام دهید. یک عملیات نامعتبر پرچم نقلی را یک می‌کند و در AX کد ۰۱ (کد روش نامعتبر) یا ۰۶ (دستگیره نامعتبر) را باز می‌گرداند.

برنامه: خواندن یک فایل دیسکی بطور تصادفی

برنامه شکل ۴-۱۷ فایل ایجاد شده در شکل ۲-۱۷ را می‌خواند. با وارد کردن یک شماره رکورد مرتبط که در فایل قرار دارد، کاربر می‌تواند درخواست نمایش هر رکورد از فایل را داشته باشد. اگر فایل حاوی ۲۴ رکورد باشد ، پس شماره رکورد معتبر از ۱ تا ۲۴ می‌باشد، عددی که از صفحه کلید وارد می‌شود در قالب ASCII است و در این حالت یک یا دو رقم است.

برنامه بدین شکل سازمان یافته است:

- A10MAIN ، روال‌های B10OPEN ، C10RECN ، D10READ و E10DISP را فراخوانی می‌کند، وقتی کاربر در خواست دیگری نداشته باشد خاتمه می‌یابد.
- B10OPEN فایل را باز می‌کند و دستگیره فایل را می‌گیرد.
- C10RECN یک شماره رکورد را در صفحه کلید می‌پذیرد و طول آن را در لیست پارامتر بررسی می‌کند. سه طول ممکن وجود دارد.

00 انتهای پردازش درخواستی

01 یک رقم درخواستی ، ذخیره شده در AL

02 دو رقم درخواستی ، در AX

روال اعداد ASCII را به دودویی تبدیل می‌کند. چون عدد در AX است، دستور ADD بدین منظور خوب عمل می‌کند. سیستم موقعیت صفر در شروع یک فایل را تشخیص می‌دهد. برنامه ۱ را از عدد واقعی کم می‌کند (بنابراین وقتی کاربر رکورد ۱ را درخواست کند رکورد صفر خواهد شد) این مقدار در ۱۶ ضرب می‌شود (طول رکورد در فایل) و حاصل در یک فیلد با نام RECINDX ذخیره می‌شود. برای مثال، اگر عدد وارد شده به ASCII، ۱۲ باشد، AX حاوی 3132 خواهد بود. یک دستور AND این مقدار را به 0102 تبدیل می‌کند، AAD بعدی آن را به 000C تبدیل

می‌نماید (12) و SHL در حقیقت این مقدار را در ۱۶ ضرب می‌کند حاصل خواهد شد C0 (192). یک پیشرفت برای روال، تعیین اعتبار اعداد (۱ تا ۲۴) است.

- D10 READ از تابع 42H استفاده می‌کند و از موقعیت رکورد مرتبط از RECINDX برای تنظیم اشاره‌گر فایل و صدور تابع 3FH برای ارائه رکورد درخواستی به برنامه در IOAREA، استفاده می‌نماید.
- E10DISP رکورد بازیابی شده را نمایش می‌دهد.

```
TITLE     A17RDRAN (EXE)  Read disk records randomly
          .MODEL  SMALL
          .STACK  64
          .DATA
HANDLE    DW      ?                 ;File handle
RECINDX   DW      ?                 ;Record index
ERRCDE    DB      00                ;Read error indicator
PROMPT    DB      'Record number? '
IOAREA    DB      32 DUP(' ')       ;Disk record area
PATHNAM   DB      'F:\NAMEFILE.SRT',0
OPENMSG   DB      '*** Open error ***', 0DH, 0AH
READMSG   DB      '*** Read error ***', 0DH, 0AH
ROW       DB      00
COL       DB      00

RECDPAR   LABEL   BYTE              ;Input parameter list:
MAXLEN    DB      3                 ;  maximum length
ACTLEN    DB      ?                 ;  actual length
RECDNO    DB      3 DUP(' ')        ;  record number

;-------------------------------------------------------
          .CODE
.386
A10MAIN   PROC    FAR
          MOV     AX,@data          ;Initialize
          MOV     DS,AX             ;  segment
          MOV     ES,AX             ;  registers
          MOV     AX,0600H
          CALL    Q10SCRN           ;Clear screen
          CALL    Q20CURS           ;Set cursor
          CALL    B10OPEN           ;Open file
          CMP     ERRCDE,00         ;Valid open?
          JNZ     A90               ;  no, exit
A20LOOP:
          CALL    C10RECN           ;Request record #
          CMP     ACTLEN,00         ;Any more requests?
          JE      A90               ;  no, exit
          CALL    D10READ           ;Read disk record
          CMP     ERRCDE,00         ;Normal read?
          JNZ     A30               ;  no,  bypass
          CALL    E10DISP           ;  yes, display name,
A30:
          JMP     A20LOOP           ;  continue
A90:      MOV     AX,4C00H          ;End processing
          INT     21H
A10MAIN   ENDP
;                   Open file:
;                   ---------
B10OPEN   PROC    NEAR
          MOV     AH,3DH            ;Request open
          MOV     AL,00             ;Normal file
          LEA     DX,PATHNAM
          INT     21H
          JC      B20               ;Error?
          MOV     HANDLE,AX         ;  no, save handle
          RET
```

شکل الف ۴-۱۷ خواندن یک فایل دیسکی بطور تصادفی

```
B20:        MOV     ERRCDE,01           ;  yes,
            LEA     DX,OPENMSG          ;  display
            CALL    X10ERR              ;  error message
            RET
B10OPEN     ENDP
;                   Get record number:
;                   ------------------
C10RECN     PROC    NEAR
            MOV     AH,40H              ;Request display prom
            MOV     BX,01               ;File handle
            MOV     CX,15               ;15 characters
            LEA     DX,PROMPT
            INT     21H
            MOV     AH,0AH              ;Request input
            LEA     DX,RECDPAR          ;  of record number
            INT     21H
            CMP     ACTLEN,01           ;Check length 0, 1, 2
            JB      C40                 ;Length 0, terminate
            JA      C20
            XOR     AH,AH               ;Length 1
            MOV     AL,RECDNO
            JMP     C30
C20:
            MOV     AH,RECDNO           ;Length 2
            MOV     AL,RECDNO+1
C30:
            AND     AX,0F0FH            ;Clear ASCII 3s
            AAD                         ;Convert to binary
            DEC     AX                  ;Adjust (1st record i
            SHL     AX,05               ;Multiply by 16
            MOV     RECINDX,AX          ;Save index
C40:
            MOV     COL,20
            CALL    Q20CURS
            RET
C10RECN     ENDP
;                   Read disk record randomly:
;                   --------------------------
D10READ     PROC    NEAR
            MOV     AH,42H              ;Request set file poi
            MOV     AL,00               ;  to start of file
            MOV     BX,HANDLE           ;File handle
            MOV     CX,00               ;Upper portion of off
            MOV     DX,RECINDX          ;Lower portion of off
            INT     21H
            JC      D20                 ;Error condition?
                                        ;  yes, bypass
            MOV     AH,3FH              ;Request read
            MOV     BX,HANDLE
            MOV     CX,32               ;30 for name, 2 for C
            LEA     DX,IOAREA
            INT     21H
            JC      D20                 ;Error on read?
            CMP     IOAREA,1AH          ;EOF marker?
            JE      D30                 ;  yes, exit
            JMP     D90
D20:        LEA     DX,READMSG          ;Invalid read
            CALL    X10ERR              ;Display message
```

شکل ب ۴-۱۷ خواندن یک فایل دیسکی بطور تصادفی

```
D30:        MOV     ERRCDE,01           ;Force end
D90:        RET
D10READ     ENDP
;                   Display name:
;                   ------------
E10DISP     PROC    NEAR
            MOV     AH,40H              ;Request display
            MOV     BX,01               ;Set handle
            MOV     CX,32               ;  and length
            LEA     DX,IOAREA
            INT     21H
            MOV     COL,00              ;Clear column
            CMP     ROW,20              ;Bottom of screen?
            JAE     E20                 ;  yes, bypass
            INC     ROW                 ;  no, increment row
            JMP     E90
E20:
            MOV     AX,0601H
            CALL    Q10SCRN             ;Scroll
            CALL    Q20CURS             ;Set cursor
E90:        RET
E10DISP     ENDP
;                   Scroll screen:
;                   -------------
Q10SCRN     PROC    NEAR                ;AX set on entry
            MOV     BH,1EH              ;Set color
            MOV     CX,0000
            MOV     DX,184FH            ;Request scroll
            INT     10H
            RET
Q10SCRN     ENDP
;                   Set cursor:
;                   ----------
Q20CURS     PROC    NEAR
            MOV     AH,02               ;Request set
            MOV     BH,00               ;  cursor
            MOV     DH,ROW              ;  row
            MOV     DL,COL              ;  column
            INT     10H
            RET
Q20CURS     ENDP
;                   Display disk error message:
;                   --------------------------
X10ERR      PROC    NEAR
            MOV     AH,40H              ;DX contains address
            MOV     BX,01               ;Handle
            MOV     CX,20               ;Length
            INT     21H                 ;  of message
            INC     ROW
            RET
X10ERR      ENDP
            END     A10MAIN
```

شکل پ ۴-۱۷ خواندن یک فایل دیسکی بطور تصادفی

برنامه: پردازش یک فایل ASCII

مثالهای قبلی فایل‌ها را ایجاد کردند و خواندند، لیکن ممکن است بخواهید فایلهای ASCII را که DOS با یک ویرایشگر ایجاد کرده است ، پردازش کنید. آنچه که لازم است بدانید، سازمان شاخه و FAT و روشی است که سیستم، داده‌ها را در یک سکتور ذخیره می‌کند. برای مثال، سیستم عامل DOS داده‌های شما را در یک فایل ASM، دقیقاً به همان روشی که آن را وارد کرده‌اید، شامل کاراکترهای tab (06H) بازگشت به ابتدای خط (0DH) و تعویض خط (0AH)، ذخیره می‌کند. برای صرفه‌جویی در فضای دیسک DOS فاصله‌هایی را که روی صفحه تصویر بلافاصله قبل از کاراکتر tab ظاهر می‌شوند و فاصله‌های سمت راست کاراکتر بازگشت واقع روی یک خط را، ذخیره نمی‌کند. مورد زیر

یک دستور اسمبلی است به همان صورتی که در صفحه کلید وارد می‌شود:

<Tab> MOV <Tab> AH,09 <Enter>

بیان شانزدهی داده‌های ASCII فوق می‌تواند چنین باشد:

094D4F560941482C30390D09

```
TITLE     A17RDASC (EXE)  Read/display an ASCII file
          .MODEL SMALL
          .STACK 64
          .DATA
DISAREA   DB      120 DUP(' ')        ;Display area
ENDCDE    DW      00                  ;End process indicator
FILESIZE  DW      0                   ;File size (low-order)
HANDLE    DW      0                   ;File handle
OPENMSG   DB      '*** Open error ***'
PATHNAM   DB      'F:\A20PRTAS.ASM', 0
ROW       DB      00
SECTOR    DB      512 DUP(' ')        ;Input area
; ---------------------------------------------------------
          .CODE
A10MAIN   PROC    FAR                 ;Main procedure
          MOV     AX,@data            ;Initialize
          MOV     DS,AX               ;  segment
          MOV     ES,AX               ;  registers
          MOV     AX,0600H
          CALL    Q10SCR              ;Clear screen
          CALL    Q20CURS             ;Set cursor
          CALL    B10OPEN             ;Open file
          CMP     ENDCDE,00           ;Valid open?
          JNE     A90                 ;  no,  exit
          CALL    C10READ             ;Yes, read 1st disk sector
          CMP     ENDCDE,00           ;End-file, no data?
          JE      A90                 ;  yes, exit
          CALL    D10XFER             ;Display/read
A90:
          MOV     AH,3EH              ;Request close file
          MOV     BX,HANDLE
          INT     21H
          MOV     AX,4C00H            ;End processing
          INT     21H
A10MAIN   ENDP
;                 Open disk file:
;                 --------------
B10OPEN   PROC    NEAR
          MOV     AH,3DH              ;Request open
          MOV     AL,00               ;Read only
          LEA     DX,PATHNAM
          INT     21H
          JNC     B20                 ;Test carry flag,
          CALL    X10ERR              ;  error if set
          RET
B20:      MOV     HANDLE,AX           ;Save handle
          MOV     AH,42H              ;Request set pointer
          MOV     AL,02               ;  to end of file
          MOV     BX,HANDLE           ;  to determine
          MOV     CX,0                ;  file size
          MOV     DX,CX               ;
          INT     21H
          MOV     FILESIZE,AX         ;Save size (low-order)
          MOV     AH,42H              ;Reset file pointer
          MOV     AL,00               ;  to start of file
          MOV     DX,CX               ;
          INT     21H
          RET
B10OPEN   ENDP
```

شکل الف ۵-۱۷ خواندن یک فایل ASCII

```
;                       Read disk sector:
;                       ----------------
C10READ     PROC        NEAR
            MOV         AH,3FH              ;Request read
            MOV         BX,HANDLE           ;Device
            MOV         CX,512              ;Length
            LEA         DX,SECTOR           ;Buffer
            INT         21H
            MOV         ENDCDE,AX           ;Save status
            RET
C10READ     ENDP
;                       Transfer data to display line:
;                       ------------------------------
D10XFER     PROC        NEAR
            CLD                             ;Set left-to-right
            LEA         SI,SECTOR
D20:        LEA         DI,DISAREA
D30:
            LEA         DX,SECTOR+512
            CMP         SI,DX               ;End of sector?
            JNE         D40                 ;  no,  bypass
            CALL        C10READ             ;  yes, read next
            CMP         ENDCDE,00           ;End of file?
            JE          D80                 ;  yes, exit
            LEA         SI,SECTOR
D40:
            LEA         DX,DISAREA+80
            CMP         DI,DX               ;End of DISAREA?
            JB          D50                 ;  no,  bypass
            MOV         [DI],0D0AH          ;  yes, set CR/LF,
            CALL        E10DISP             ;  and display
            LEA         DI,DISAREA
D50:
            LODSB                           ;[SI] to AL, INC SI
            STOSB                           ;AL to [DI], INC DI
            DEC         FILESIZE            ;All chars processed?
            JZ          D80                 ;  yes, exit
            CMP         AL,0AH              ;Line feed?
            JNE         D30                 ;  no,  loop
            CALL        E10DISP             ;  yes, display
            JMP         D20
D80:        CALL        E10DISP             ;Display last line
D90:        RET
D10XFER     ENDP
;                       Display line:
;                       ------------
E10DISP     PROC        NEAR
            MOV         AH,40H              ;Request display
            MOV         BX,01               ;Handle
            LEA         CX,DISAREA          ;Calculate
            NEG         CX                  ;  length of
            ADD         CX,DI               ;  line
            LEA         DX,DISAREA
            INT         21H
            CMP         ROW,22              ;Bottom of screen?
            JAE         E20                 ;  no, exit
            INC         ROW
            JMP         E90
E20:        MOV         AX,0601H            ;Scroll
            CALL        Q10SCR
            CALL        Q20CURS
E90:        RET
E10DISP     ENDP
;                       Scroll screen:
```

شکل ب ۵-۱۷ خواندن یک فایل ASCII

```
;                       -------------
Q10SCR      PROC    NEAR                ;AX set on entry
            MOV     BH,1EH              ;Set color attribute
            MOV     CX,0000             ;Scroll
            MOV     DX,184FH
            INT     10H
            RET
Q10SCR      ENDP
;                       Set cursor:
;                       ----------
Q20CURS     PROC    NEAR
            MOV     AH,02H              ;Request set
            MOV     BH,00               ;  cursor
            MOV     DH,ROW
            MOV     DL,00
            INT     10H
            RET
Q20CURS     ENDP
;                       Display disk error message:
;                       --------------------------
X10ERR      PROC    NEAR
            MOV     AH,40H              ;Request display
            MOV     BX,01               ;Handle
            MOV     CX,18               ;Length
            LEA     DX,OPENMSG
            INT     21H
            MOV     ENDCDE,01           ;Error indicator
            RET
X10ERR      ENDP
            END     A10MAIN
```

شکل پ ۵-۱۷ خواندن یک فایل *ASCII*

09H همان Tab ، 0DH همان Enter و OAH همان Linefeed می‌باشد، وقتی یک ویرایشگر یا پردازشگر کلمات فایل را می‌خواند، Tab، Enter و Linefeed بطور خودکار مکان‌نما را بر روی صفحه تنظیم می‌کنند.

حال بیایید برنامه شکل ۵-۱۷ که فایل A17RDFIL.ASM از شکل ۳-۱۷ را به صورت یک سکتور در هر زمان می‌خواند بررسی کنیم. برنامه مانند تابع DOS TYPE عمل می‌کند، که هر خط تا کاراکترهای Enter/Linefeed را نمایش می‌دهد.

- A10MAIN روال B10OPEN را فرا می‌خواند C10READ برای خواندن اولین سکتور D10XFER را فرا می‌خواند و در انتها فایل را می‌بندد.
- B10OPEN فایل را باز می‌کند. دستگیره فایل را ذخیره می‌سازد، اندازه فایل را تعیین می‌کند، (البته به بخش پایین رتبه از اندازه فایل در AX).
- C10READ یک سکتور کامل داده را می‌خواند و در SETOR قرار می‌دهد.
- D10XFER داده‌ها را از سکتور به خط نمایش منتقل می‌کند، E10DISP را برای نمایش آن فرا می‌خواند، C10READ را برای سکتور بعدی فرا می‌خواند و پردازش را تا رسیدن به انتهای فایل ادامه می‌دهد.

روال یک بایت را در هر زمان از SECTOR به DISAREA منتقل می‌کند، جایی که کاراکترها باید نمایش داده شوند. باید انتهای یک سکتور (برای خواندن سکتور دیگر) و خاتمه ناحیه نمایش را بررسی کند. برای فایل‌های مرسوم ASCII مانند فایل ASM. هر خط نسبتاً کوتاه است و با Enter/Line Feed ختم می‌شود. فایل‌های غیر ASCII، مانند یک فایل EXE. و OBJ. خط ندارند بنابراین برنامه باید انتهای DISAREA را برای اجتناب از انتقال داده به ناحیه پس از آن، بررسی نماید. برنامه قصد دارد تا فقط فایلهای ASCII را نمایش دهد، اما بررسی انتهای DISAREA بیمه‌ای است برای انواع فایل در نظر گرفته نشده.

این مراحل وجود دارد :

۱) مقدار دهی آدرس SECTOR و DISAREA.
۲) اگر در انتهای SECTOR است، سکتور بعدی را بخوان. اگر در انتهای فایل است، خارج شو، DISAREEA را مقدار دهی کن. در غیر اینصورت آدرس SECTOR را مقدار دهی کن.
۳) اگر در انتهای DISAREA است، یک LinFeed / Enter بگذار، خطر نمایش بده و
۴) یک کاراکتر از SECTOR بگیر و در DISAREA ذخیره کن.
۵) اگر همه کاراکترها پردازش شده‌اند، خارج شو.
۶) اگر کاراکتر Line Feed (0AH) است، خط را نمایش بده و به مرحله ۲ برو، در غیر اینصورت به مرحله ۳ برود.

- E10DISP داده‌ها را در خط نمایش تا و شامل Line Feed نمایش می‌دهد. چون خطوط یک فایل ASCII طول متغیری دارند، باید انتهای هر خط را قبل از نمایش آن پیمایش نمایید. (مونیتور کاراکتر Tab (09H) را می‌پذیرد و مکان نما را در موقعیت بعدی که قابل تقسیم بر ۸ باشد، تنظیم می‌کند). حرکت طوماری می‌تواند مُشکل باشد. اگر هیچ بررسی خاصی برای تعیین اینکه به خط انتهای صفحه رسیدید، انجام نداده باشید، عملیات بطور خودکار خطوط جدید را بر روی خطوط قبلی نشان می‌دهد، اگر خطوط قبلی طولانی‌تر باشند، کاراکترهای قبلی هنوز در سمت راست پدیدار هستند. برای حرکت طوماری صحیح، باید سطرها را بشمارید و بررسی کنید آیا به انتهای صفحه نمایش رسیده‌اید.
- X10ERR. یک پیغام برای خطای دیسک نمایش می‌دهد.

سعی کنید این برنامه را با یک شماره درایو مناسب و فایل ASCII مناسب تحت DEBUG اجرا کنید بعد از هر ورودی دیسک، محتویات ناحیه ورودی را نشان می‌دهد و ببینید که چطور رکوردهای شما قالب بندی می‌شوند. توسعه این برنامه می‌تواند اعلان به کاربر برای ورود نام فایل و توسعه آن و استفاده از DX:AX کامل برای اندازه فایل باشد.

عملیات I/O در دیسک

هدف از دستورات DOS INT 25H و 26H، توانایی خواندن و نوشتن‌های ***مطلق*** برای پردازش مستقیم یک دیسک است، از جمله این عملیات بازسازی یک فایل خراب شده، می‌باشد. در این حالت، دستگیره فایل یا FCBS تعریف نمی‌کنید و آسایش دستیابی به شاخه و بلوکه کردن یا بلوکه نکردن رکوردها را که INT 21H فراهم می‌سازد نخواهید داشت. توجه کنید که تابع 44H از وقفه 21H (در فصل ۱۸ گفته خواهد شد)، سرویس مشابهی را فراهم می‌سازد و ما را از وقفه‌های 25H و 26H بی‌نیاز می‌کند.

چون این عملیات با همه رکوردها برخورد دارد، اگر اندازه سکتور باشند، مستقیماً یک سکتور کامل یا بلوکی از سکتورها دستیابی خواهد شد. آدرسدهی دیسک به عبارتی شماره رکوردهای مرتبط (سکتورهای مرتبط) می‌باشد. برای تعیین یک شماره رکورد مرتبط روی دیسکت دو طرفه با ۹ سکتور در هر تراک، سکتور را از تراک 0، سکتور 1 بشرح زیر بشمارید :

شماره رکورد مرتبط	سکتور	تراک
0 (اولین سکتور روی دیسک)	1	0
1	2	0
9	1	1
17	9	1
26	9	2

یک فرمول برای تعیین شماره رکورد مرتبط روی دیسکتی با ۹ سکتور چنین است :

(۱- سکتور) + (۹× تراک) = شماره رکورد مرتبط

شماره رکورد مرتبط برای تراک ۲، سکتور ۹، چنین محاسبه می‌شود.

(۲×۹) + (۹-۱) = ۱۸ + ۸ = ۲۶

در اینجا کد نویسی لازم برای بخش‌های دیسکی که کوچکتر از ۳۲MB است آورده شده است :

```
MOV   AL,drive #      ; 0 برای A، ۱ برای B و غیره
MOV   BX,addr         ; آدرس بلوک پارامتر
MOV   CX,Sector #     ; تعداد سکتورهایی باید خوانده / نوشته شود
MOV   DX,Sector #     ; شماره سکتور مرتبط شروع
INT   25H یا 26H      ; خواندن یا نوشتن مطلق
JC    [error]         ; CF با بروز خطا یک می‌شود
POPF                  ; برداشتن پرچم‌ها از روی پشته
```

INT 25H,26H همه ثبات‌ها بجز ثبات‌های سگمنت را خراب می‌کنند و از پرچم نقلی برای دلالت بر، موفقیت (0) با عدم موفقیت استفاده می‌نمایند. یک عملیات ناموفق یکی از کدهای غیر صفر زیر را در AL بار می‌گرداند.

اتصال در پاسخ ناتوان است	10000000
عملیات پشته ناتوان است	01000000
CRC خراب خوانده شده از روی دیسک	00001000
سکتور خواسته شده پیدا نشد	00000100
کوشش برای نوشتن روی یک دیسکت محافظت‌شده از نوشتن	00000011
دیگر خطاها	00000010

عملیات INT پرچم را روی پشته می‌گذارد. چون پرچم‌های اصلی هنوز بر روی پشته حتی با بازگشت از عملیات قرار دارند، بعد از بررسی پرچم نقلی، از روی پشته، آنها را بردارید.

از DOS 4.0 و بعد می‌توانید با استفاده از 26H یا INT 25H برای دستیابی بخش‌های دیسک که از ۳۲ مگابایت تجاوز می‌کند استفاده نمایید، DX استفاده نمی‌شود و BX به یک بلوک پارامترهای 10 بایتی در قالب زیر اشاره دارد:

شماره سکتور ۳۲ بیتی	00H-03H
شماره سکتورهایی‌که باید خوانده/نوشته شود	04H-05H
آدرس افست بافر	06H-07H
آدرس سگمنت بافر	08H-09H

سرویس‌های دیسک با استفاده از بلوک‌های کنترل فایل

این بخش بطور خلاصه سرویس‌های FCB برای ایجاد فایل‌های دیسکی و پردازش ترتیبی و تصادفی آنها را در بر دارد. این سرویس‌ها با اولین نسخه از DOS معرفی شدند و هنوز تحت همهٔ نسخه‌ها در دسترس می‌باشند. پردازش دیسک برای سرویس‌های FCB، مستلزم تعریف یک بلوک کنترل فایل (FCB) است که فایل و یک ناحیه انتقال دیسک (DTA) برای تعریف رکوردها را تعریف می‌کند. شما سیستم را با آدرس DTA برای همه عملیات دیسک I/O را مهیا می‌سازید. چون روش FCB هیچ دستگیره فایل یا نام مسیر را پشتیبانی نمی‌کند، به پردازش فایل در شاخه جاری محدود می‌شود. نیز، عملیات FCB از کدهای خطی لیست شده در شکل ۱-۱۷ استفاده نمی‌کند و پرچم نقلی را مبنی بر موفقیت یا عدم موفقیت، صفر یا یک نمی‌کند. (FCBهای متنوعی در PSP وجود دارد)

بلوک کنترل فایل

FCB، که در ناحیه داده تعریف می‌شود، شامل اصطلاحات زیر راجع به فایل و رکوردهایش می‌باشد.

(شما بایت‌های ۰ تا ۱۵ و ۳۲ تا ۳۶ را مقدار دهی می‌کنید، در حالیکه سیستم بایت‌های ۱۷-۳۱ را تنظیم می‌نماید)

۰ ***دیسک درایو.*** برای اغلب عملیات FCB ، 00 درایو پیش فرض، 01 درایو A، 02 درایو B و غیره می‌باشد.

۱-۸ ***نام فایل.*** نام فایل، تنظیم شده از چپ، در با دنباله‌ای از جای خالی اگر باشد.

۱۱-۹ ***توسعه نام فایل***. تنظیم شده از چپ اگر کمتر از ۳ کاراکتر باشد.

۱۳-۱۲ ***شماره بلوک جاری***. یک بلوک حاوی ۱۲۸ رکورد. عملیات خواندن / نوشتن از شماره بلوک جاری و شماره رکورد جاری (بایت ۳۲) برای قرار گرفتن در موقعیت یک رکورد خاص استفاده می‌کند. عدد با شروع فایل مرتبط است، جایی که اولین بلوک ۰ است، دومین ۱ و الی آخر.

۱۵-۱۴ ***اندازه رکورد منطقی***. یک عملیات باز کردن اندازه رکورد را با ۱۲۸ (80H) مقدار دهی می‌کند، اگر چه شما می‌توانید آن را به اندازه رکورد مورد نیاز خود تغییر دهید.

۱۹-۱۶ ***اندازه فایل***. اندازه فایل از شاخه، که برنامه شما ممکن است بخواند، اما نباید تغییر کند.

۲۱-۲۰ ***تاریخ***. تاریخ شاخه، که برنامه شما ممکن است بخواند، اما نباید تغییر کند.

۳۱-۲۲ ***رزرو شده***. برای برنامه در دسترس نیست.

۳۲ ***شماره رکورد جاری***. برای خواندن / نوشتن تصادفی، این ورودی باید حاوی شماره رکورد، رابطه‌ای باشد. چون حداکثر اندازه فایل (۱۰۷۳۷۴۱۸۲۴ بایت) می‌باشد، یک فایل با اندازه رکوردهای کوتاه می‌تواند شامل رکوردهای بیشتری باشد و ممکن است حداکثر شماره رکوردهای مرتبط بیشتری داشته باشد نسبت به فایلی که اندازه رکوردهای طولانی‌تر دارد. اگر اندازه رکورد بزرگتر از ۶۴ باشد، بایت ۳۶ حاوی 00H خواهد بود.

FCB فوق الذکر یک توسعه ۷ بایتی انتخابی دارد، که می‌توانید برای پردازش فایل‌هایی با صفات خاص از آن استفاده کنید. با استفاده از توسعه، اولین بایت را با FFH کدنویسی کنید، بایت دوم را با صفت فایل و ۵ بایتی باقیمانده را با صفر شانزدهی مقدار دهید.

استفاده از FCB جهت ایجاد فایل‌های دیسک

یک برنامه با استفاده از این سرویس‌های دیسک یک FCB برای هر فایل ارجاع شده تعریف می‌کند. عملیات دیسک به آدرس FCB در ثبات DX نیاز دارد و با استفاده از این آدرس فیلدهای داخل FCB را دستیابی می‌کند. عملیات شامل ایجاد فایل، تنظیم ناحیه انتقال دیسک (DTA)، رکورد نوشتن و بستن فایل می‌باشد.

تابع 16H از وقفه 21H: ایجاد فایل. در مقدار دهی، یک برنامه با استفاده از تابع 16H یک فایل FCB جدید ایجاد می‌کند :

```
MOV   AH,16H          ; درخواست ایجاد
LEA   DX,FCB name     ; FCB فایل دیسکی
INT   21H             ; فراخوانی سرویس وقفه
```

عملیات شاخه را برای یک نام فایل که با ورودی FCB یکسان باشد، جستجو می‌کند. در صورت یافتن چنین نامی ، از فضای شاخه مجدداً استفاده می‌کند، و اگر پیدا نشد به دنبال یک ورودی خالی می‌گردد. سپس عملیات اندازه فایل را با صفر مقدار می‌دهد و فایل را باز می‌کند. مرحله باز کردن، فضای دیسک مورد دسترس را بررسی می‌کند و یکی از کدهای بازگشتی زیر را در AL تنظیم می‌کند: 00H = فضا در دسترس نیست. همچنین باز کردن، شماره بلوک جاری FCB را با صفر مقدار می‌دهد و یک مقدار پیش فرض در اندازه رکورد FCB با ۱۲۸ بایت، تنظیم می‌کند. قبل از نوشتن یک رکورد، شما ممکن است این بیش فرض‌ها را با مقادیر خود بازنویسی کنید.

ناحیه انتقال دیسک. ناحیه انتقال دیسک (DTA)، موقعیت ذخیرهٔ که رکوردها روی دیسک را، مشخص می‌سازد. چون FCB حاوی اندازه رکورد است، DTA نیازی به یک محدود کننده مبنی بر انتهای رکورد ندارد. فقط یک DTA ممکن است در هر زمان فعال باشد. از تابع 1AH برای مهیا کردن سیستم با آدرس DTA استفاده نمایید:

```
MOV   AH,1AH          ; در خواست تنظیم آدرس
LEA   DX,DTA name     ; از DTA
INT   21H             ; فراخوانی سرویس وقفه
```

اگر یک برنامه، فقط یک فایل دیسکی را پردازش نماید، نیاز دارد که DTA را یک بار برای اجرای ورودی مقدار دهی نماید. برای پردازش چندین فایل، باید بلافاصله از هر خواندن / نوشتن DTA مناسب را مقدار دهی نماید.

تابع 15H از وقفه 21H : نوشتن رکورد برای نوشتن یک رکورد دیسکی بطور ترتیبی با روش FCB از تابع 15H استفاده کنید.

```
MOV  AH,15H          ; درخواست نوشتن رکورد FCB
LEA  DX,FCB name     ; بطور ترتیبی
INT  21H             ; فراخوانی سرویس وقفه
```

عملیات نوشتن از اطلاعات FCB در آدرس DTA جاری استفاده می‌کند. اگر رکوردها به اندازه یک سکتور باشند، عملیات آن را می‌نویسد. در غیر اینصورت، عملیات رکوردها را در ناحیه بافر پر می‌کند که به طول یک سکتور است و وقتی بافر پر شد آن را می‌نویسد. برای مثال، اگر هر رکورد ۱۲۸ بایت طول داشته باشد، عملیات بافر را با ۴ رکورد پر می‌کند (۵۱۲ = ۱۲۸ × ۴) و سپس بافر را در یک سکتور دیسک ورودی می‌نویسد.

یک عملیات نوشتن موفق، اندازه رکورد را به فیلد اندازه فایل می‌افزاید و شماره رکورد جاری را یکی می‌افزاید. وقتی شماره رکورد جاری بیشتر را از ۱۲۷ باشد، عملیات آن را صفر می‌کند و شماره بلوک جاری را می‌افزاید. عملیات یک کد در AL باز می‌گرداند: 00H = نوشتن موفقیت‌آمیز است، 01H = دیسک پر است، DTA = 02H برای رکورد کوچک است.

INT 21H تابع 10H : بستن فایل. وقتی نوشتن رکوردها تمام شد، شما ممکن است یک علامت انتهای فایل بنویسید (1AH در اولین بایت از آخرین رکورد خاص) و سپس از تابع 10H برای بستن فایل استفاده کنید:

```
MOV  AH,10H          ; درخواست بستن
LEA  DX,FCB name     ; فایل FCB
INT  21H             ; فراخوانی سرویس وقفه
```

عملیات بستن، هر بخش داده که هنوز در بافر دیسک است، روی دیسک می‌نویسد، شاخه را با تاریخ و اندازه فایل بروز رسانی می‌کند و یک کد در AL باز می‌گرداند، 00H = بستن موفقیت‌آمیز بود، FFH = فایل در موقعیت مناسبی در شاخه نیست، ممکن است توسط یک کاربر دیسکت تغییر نموده باشد.

استفاده از FCBS برای خواندن فایل‌های دیسکی ترتیبی

یک برنامه که فایلی را از دیسک می‌خواند FCB را دقیقاً مانند آن یکی برای ایجاد فایل استفاده شده، تعریف می‌کند. عملیات خواندن ترتیبی شامل باز کردن فایل، تنظیم DTA، خواندن رکورد و بستن فایل است.

تابع 0FH از وقفه 21H : باز کردن تابع 0F یک فایل را برای ورودی باز می‌کند:

```
MOV  AH,0FH          ; درخواست باز کردن
LEA  DX,FCB name     ; فایل FCB
INT  21H             ; فراخوانی سرویس وقفه
```

عملیات باز کردن بررسی می‌کند که شاخه حاوی یک ورودی با نام فایل و توسعه تعریف شده در FCB هست. اگر ورودی در شاخه نباشد عملیات FFH را در AL باز می‌گرداند. اگر ورودی حاضر باشد، عملیات 00H را در AL باز می‌گرداند و اندازه واقعی فایل، تاریخ، شماره بلوک جاری (0) و اندازه رکورد (80H) در FCB را تنظیم می‌کند. شما ممکن است بعداً این اندازه رکورد را بازنویسی کنید.

ناحیه انتقال دیسک : DTA یک ناحیه برای دریافت رکوردهای ورودی مانند همان قالب مورد استفاده برای ایجاد فایل تعریف می‌کند. از تابع 1AH برای تنظیم آدرس DTA استفاده کنید همانطور که برای ایجاد فایل عمل کردید.

تابع 14H از وقفه 21H : از تابع 14H برای خواندن یک رکورد دیسک FCB بطور ترتیبی استفاده نمایید.

```
MOV  AH,14H          ; درخواست خواندن رکورد FCB
LEA  DX,FCB name     ; بطور ترتیبی
INT  21H             ; فراخوانی سرویس وقفه
```

عملیات یک کد در AL باز می‌گرداند. 00 = خواندن با موفقیت، 01 = انتهای فایل، داده‌ای خوانده نشد، 02 = DTA کوچک است، 03 = انتهای فایل، بخشی از رکورد خوانده شد و با صفر پر شد یک عملیات خواندن موفق از اطلاعات FCB برای ادامه به رکورد دیسک، با شروع در آدرس DTA استفاده می‌کند. کوشش جهت خواندن بعد از آخرین رکورد فایل سبب می‌شود، عملیات یک علامت انتهای فایل بفرستد که AL را با 01H تنظیم می‌کند، که می‌توانید آن را بررسی کنید.

استفاده از FCB برای پردازش تصادفی

پردازش تصادفی مستلزم جایگزینی شماره رکورد مورد نظر در فیلد رکورد رابطه‌ای (بایت‌های ۳۳-۳۶) و صدور یک فرمان خواندن / نوشتن تصادفی است. برای قرار گرفتن در موقعیت یک رکورد بطور تصادفی، سیستم شماره رکورد رابطه‌ای را به بلوک جاری (بایت‌های ۱۲-۱۳) و رکورد جاری (بایت ۳۲) تبدیل می‌کند.

تابع 21H از وقفه 21H : خواندن رکورد بطور تصادفی. عملیات باز کردن و تنظیم DTA برای پردازش تصادفی و ترتیبی یکسان هستند. برای مثال، برای یک برنامه که شماره رکورد رابطه‌ای ۰۵ را بطور تصادفی می‌خواند 05000000H را در شماره رکورد رابطه‌ای قرار دهید و تابع 21H را درخواست کنید.

```
MOV  AH,21H          ; درخواست FCB
LEA  DX,FCB name     ; خواندن تصادفی
INT  21H             ; فراخوانی سرویس وقفه
```

عملیات یک کد در AL باز می‌گرداند، 00 = خواندن موفقیت‌آمیز، 01 = انتهای فایل، داده بیشتری در دسترس نمی‌باشد، 02 = DTA کوچک است، 03 = بخشی از رکورد خوانده شده و با صفر پر شد. یک عملیات موفق شماره رکورد رابطه‌ای را به بلوک جاری و رکورد تبدیل می‌کند، این مقدار را برای قرار گرفتن در رکورد دیسک مورد نیاز استفاده می‌کند و آن را به DTA ارائه می‌دهد. یک پاسخ خطا می‌تواند به سبب یک شماره رکورد نامعتبر یا یک آدرس اشتباه در DTA یا FCB ایجاد شود.

تابع 22H از وقفه 21H : نوشتن رکورد بطور تصادفی. عملیات ایجاد و تنظیم DTA برای پردازش تصادفی و ترتیبی یکسان است. با شماره رکورد رابطه‌ای مقدار دهی شده در FCB و استفاده از تابع 22H یک رکورد را بطور تصادفی بنویسید.

```
MOV  AH,22H          ; درخواست FCB تصادفی
LEA  DX,FCB name     ; نوشتن
INT  21H             ; فراخوانی سرویس وقفه
```

عملیات یک کد در AL باز می‌گرداند. 00 = نوشتن موفق ، 01 = دیسک پر است، 02 = DTA کوچک است.

پردازش تصادفی بلوک

اگر یک برنامه فضایی کافی داشته باشد، یک عملیات بلوک تصادفی می‌تواند یک فایل ورودی را از DTA روی دیسک بنویسید یا می‌تواند فایل ورودی را از دیسک به DTA بنویسید. ابتدا باید فایل را باز کنید و DTA را مقدار دهی نمایید. سپس ممکن است با هر شماره رکورد رابطه‌ای معتبر و هر تعداد رکورد پردازش را آغاز نمایید، اگر چه بلوک باید داخل محدوده فایل رکوردها باشد.

تابع 28H از وقفه 21H : نوشتن بلوک بطور تصادفی از تابع 28H برای یک نوشتن بلوک تصادفی از فایل FCB استفاده کنید.

```
MOV   AH,28H          ; درخواست نوشتن بلوک تصادفی FCB
MOV   DX,records      ; تنظیم تعداد رکوردها
LEA   DX,FCB name     ; آدرس FCB
INT   21H             ; فراخوانی سرویس وقفه
```

عملیات شماره رکورد رابطه‌ای را به بلوک و رکورد جاری تبدیل می‌کند، که از آن برای تعیین شروع موقعیت دیسک و بازگشت یک کد در AL استفاده می‌کند: 00 = نوشتن موفقیت‌آمیز رکوردها، 01 = رکوردی نوشته نشد چون فضای دیسک کافی نیست، 02 = DTA کوچک می‌باشد. یک عملیات معتبر رکورد رابطه‌ای، بلوک جاری، و رکورد جاری را برای شماره رکورد بعدی تنظیم می‌نماید.

تابع 27H از وقفه 21H : خواندن بلوک بطور تصادفی. از تابع 27H برای خواندن یک بلوک تصادفی از یک فایل FCB استفاده کنید :

```
MOV   AH,27H          ; درخواست خواندن بلوک تصادفی FCB
MOV   CX,records      ; مقدار دهی تعداد رکوردها
LEA   DX,FCB name     ; آدرس FCB
INT   21H             ; فراخوانی سرویس وقفه
```

عملیات یک کد در AL باز می‌گرداند: 00 = خواندن موفق همه رکوردها، 01 = خواندن از انتهای فایل، آخرین رکورد کامل شده است، 02 = DTA کوچک است، خواندن کامل نشده است، 03 = انتهای فایل، بخشی از یک رکورد خوانده شد. عملیات تعداد واقعی رکوردهای خوانده شده را در CX ذخیره می‌کند و رکورد رابطه‌ای بلوک جاری و رکورد جاری را برای رکورد بعدی تنظیم می‌کند.

نکات کلیدی

- بسیاری از سرویس‌های دیسک به یک رشتهٔ ASCIIZ رجوع می‌کند که حاوی مسیر شاخه و پس از آن یک بایت صفر شانزدهی قرار دارد.
- در صورت بروز خطا، خیلی از توابع دیسک پرچم نقلی را تنظیم می‌کنند و یک کد خطا در AX باز می‌گردانند.
- سیستم یک اشاره‌گر فایل برای هر فایل که برنامه پردازش می‌کند نگه می‌دارد. عملیات ایجاد و باز کردن مقدار اشاره‌گر فایل را با صفر یعنی در موقعیت شروع فایل مقدار می‌دهد.
- تابع ایجاد 3CH برای نوشتن یک فایل و تابع باز کردن 3DH برای خواندن فایل استفاده می‌شود.
- توابع ایجاد و باز کردن، یک دستگیره فایل باز می‌گرداند که برای دستیابی‌های بعدی فایل می‌توانید استفاده کنید.
- یک برنامه که نوشتن در یک فایل را تکمیل نمود، باید آن را ببندد تا سیستم شاخه را بروز رسانی نماید.
- یک برنامه با استفاده از توابع INT 21H اصلی برای I/O دیسک، یک بلوک کنترل فایل (FCB) برای هر فایل که دستیابی می‌شود، تعریف می‌کند.
- یک بلوک FCB شامل ۱۲۸ رکورد می‌باشد. شماره بلوک جاری در ترکیب با شماره رکورد جاری بر رکورد دیسک که باید پردازش شود دلالت دارد.
- برای یک FCB، ناحیه انتقال دیسک (DTA) موقعیت رکوردی است که باید نوشته یا خوانده شود. هر DTA را قبل از اجرای عملیات خواندن / نوشتن مقدار دهی می‌کند.

پرسش‌ها

در پرسش‌های زیر ۱۰ پرش اول با عملیات دیسک با استفاده از دستگیره فایل و مابقی به استفاده از عملیات دیسک FCB متمرکز شده است.

۱-۱۷. کد بازگشتی خطا برای موارد زیر چیست؟ (الف) دستگیره نامعتبر ، (ب) مسیر پیدا نشد ، (ج) دیسک محافظت شده در مقابل نوشتن؟

۲-۱۷. یک رشته ASCIIZ با نام ASCPATH برای یک فایل با نام PATIENT.LST روی درایو D تعریف کنید.

۳-۱۷. برای فایل پرسش ۲-۱۷ هر رکورد شامل شماره بیمار (۵ کاراکتر)، نام (۲۰)، آدرس خیابان (۲۰)، شهر (۲۰)، تاریخ تولد (mm dd yy) کد MIF شماره اتاق (۲) و شماره تخت (۲) می‌باشد. دستوراتی برای موارد زیر فراهم کنید. (الف) یک عنصر با نام FHANDLE برای دستگیره فایل تعریف کنید، (ب) فیلدهایی برای رکورد بیمار تعریف کنید، (ج) فایل را ایجاد کنید، (د) یک رکورد از PATNTOUT بنویسید و (ه) فایل را ببندید. خطاها را بررسی کنید.

۴-۱۷. برای فایل پرش ۳-۱۷، دستوراتی کد نویسی کنید تا (الف) فایل را باز کنید، (ب) هر رکورد را در PATNTIN بخوانید و آن را نمایش دهید. برنامه را کامل کنید و داده بررسی آن را فراهم نمایید.

۵-۱۷. برنامه شکل ۵-۱۷ را بازنویسی کنید بطوریکه کاربر بتواند یک نام فایل را وارد کند، که برنامه با استفاده از این نام در موقعیت فایل قرار گیرد و محتویات آن را نمایش دهد. برای هر تعداد درخواستی عمل کند و برای خاتمه پردازش فقط <Enter> را وارد کند.

۶-۱۷. تحت چه شرایطی، فایلی که برای ورودی استفاده می‌شود باید ببندید؟

۷-۱۷. برنامه‌ای بنویسید که به کاربر این امکان را بدهد تا شماره بخش (۳ کاراکتر) توضیح بخش (۱۲ کاراکتر) و قیمت واحد (xx.xxx) را از ترمینال وارد کند. برنامه با استفاده از دستگیره فایل یک فایل دیسکی حاوی اطلاعات زیر ایجاد کند. به خاطر بسپارید که قیمت را از ASCII به دودویی تبدیل کنید. در زیر داده‌های ورودی ساده‌ای آورده شده است :

```
Part Description  Price    Part Description Price
|023|Assemblers  |00315|  |122|Lifters     |10520|
|024|Linkages    |00430|  |124|Processors  |21335|
|027|Compilers   |00525|  |127|Labelers    |00960|
|049|Compressors |00920|  |232|Bailers     |05635|
|114|Extractors  |11250|  |237|Grinders    |08250|
|117|Haulers     |00630|  |999|            |00000|
```

۸-۱۷. برنامه‌ای بنویسید که محتویات فایل ایجاد شده در پرسش ۷-۱۷ را نمایش دهد. در این برنامه باید مقادیر دودویی قیمت برای نمایش به قالب ASCII تبدیل شود.

۹-۱۷. با استفاده از فایل ایجاد شده در پرسش ۷-۱۷، برنامه‌ای بنویسید که موارد زیر را داشته باشد: (الف) همه رکوردها را در یک جدول و حافظه بخواند (ب) یک کاربر درخواست وارد کردن شماره بخش و کمیت را دارد، (ج) جدول را برای شماره بخش جستجو نماید، (د) اگر شماره بخش پیدا شد با استفاده از جدول قیمت نرخ بخش را محاسبه کند (قیمت × کمیت)، (ه) توضیح و مقدار محاسبه شده را نمایش دهد. به هر تعداد درخواست امکان پاسخ داشته باشد.

۱۰-۱۷. برنامه پرسش ۸-۱۷ را طوری بازنویسی کنید که پردازش دیسک تصادفی انجام دهد. جدولی از شماره‌های بخش معتبر تعریف کنید. کاربر یک شماره بخش وارد می‌کند، که برنامه در جدول قرار داده است. از افست جدول برای محاسبه افست فایل استفاده کنید و از تابع 12H وقفه 21H برای انتقال اشاره‌گر استفاده کنید.

توضیح و قیمت را نشان دهید. از کاربر بخواهید تا قیمت فروش را وارد کند، سپس میزان فروش را محاسبه کند و نمایش دهد (قیمت × کمیت).

۱۱-۱۷. مجموعه کامل دستورات (MOV تا INT 21H) برای عملیات FCB زیر را بنویسید: (الف) ایجاد، (ب) تنظیم DTA، (ج) نوشتن ترتیبی، (د) باز کردن، (ه) خواندن ترتیبی.

۱۲-۱۷. یک برنامه از اندازه رکوردی که عملیات باز کردن FCB فرض می‌کند، استفاده می‌نماید. (الف) طول این رکورد (چقدر است)، (ب) یک سکتور شامل چند رکورد است؟، (ج) یک دیسکت شامل چه تعداد رکورد است، فرض کنید روی چهار تراک با ۹ سکتور در هر تراک ذخیره شده است؟ (د) اگر فایل در بخش ج بطور ترتیبی خوانده شود، چه تعداد دستیابی فیزیکی به دیسک رخ خواهد داد؟

فصل ۱۷

حافظه دیسک ۳ : توابع INT 21H برای پشتیبانی دیسک‌ها و فایل‌ها

هدف : بررسی عملیات مختلف INT 21H که استفاده از دیسک درایوها و فایل‌ها را پشتیبانی می‌کنند.

مقدمه

دراین فصل مجموعه‌ای‌از عملیات مفید برای‌کار با دیسک درایو، شاخه، FAT و فایل‌های دیسکی بررسی می‌شوند.

عملیات قابل انجام دیسک درایوها

0DH : تنظیم مجدد دیسک درایو
0EH : انتخاب درایو پیش فرض
19H : گرفتن درایو پیش فرض
1CH،1BH : گرفتن اطلاعات درایو
IFH : گرفتن بلوک پارامترهای درایو پیش فرض (DPB)
2EH : تنظیم/ تنظیم مجدد تشخیص دیسک
32H : گرفتن بلوک پارامترهای درایو(DPB)
36H : گرفتن فضای آزاد دیسک
4400H : گرفتن اطلاعات ابزار
4401H : تنظیم اطلاعات ابزار
4404H : خواندن داده کنترلی از درایو
4405H : نوشتن داده کنترلی از درایو
4406H : بررسی وضعیت ورودی
4407H : بررسی وضعیت خروجی
4408H : تعیین قابل جابجایی بودن رسانه ذخیره سازی
440DH : کد فرعی 41H : نوشتن سکتور دیسک
440DH : کد فرعی 61H : خواندن سکتور دیسک
440DH : کد فرعی 42H : فرمت تراک
440DH : کد فرعی 46H : تنظیم ID رسانه
440DH : کد فرعی 60H : گرفتن پارامترهای ابزار
440DH : کد فرعی 66H : گرفتن ID رسانه
440DH : کد فرعی 68H: تشخیص نوع رسانه
54H : دریافت وضعیت تاثیر
59H : گرفتن خطای توسعه یافته

عملیاتی که روی فایل‌ها انجام می‌شود

29H : تشریح نام فایل
41H : حذف فایل
43H : گرفتن / تنظیم صفت فایل
45H , 46H : کپی برداری از دستگیره فایل
4EH , 4FH : یافتن فایل مرتبط
56H : نام‌گذاری مجدد فایل
57H : گرفتن / تنظیم تاریخ / زمان فایل
5AH , 5BH : ایجاد فایل جدید /کمکی

عملیاتی که روی شاخه وFAT انجام می‌شود

39H : ایجاد زیر شاخه
3AH : حذف زیر شاخه
3BH : تغییر شاخه جاری
47H : گرفتن شاخه جاری

کدهای خطای ذکر شده در این فصل، همان لیست شکل ۱-۱۷ هستند.

عملیات دستکاری دیسک درایو

تابع 0DH : تنظیم مجدد دیسک درایو

معمولاً، بستن یک فایل سبب می‌شود تا رکوردهای باقی مانده نوشته شده و شاخه را تحت شرایط خاص، به هنگام شود. مانند بین مراحل برنامه یا بروز شرایط خطا، برنامه ممکن است برای تنظیم مجدد دیسک درایو از تابع 0DH استفاده نماید.

```
MOV  AH,0DH  ; درخواست تنظیم مجدد دیسک
INT  21H     ; فراخوانی سرویس وقفه
```

این تابع همه بافرهای فایل را خالی می‌کند و هد خواندن / نوشتن را در سیلندر صفر تنظیم می‌کند؛ تابع 0DH بطور خودکار فایل را نمی‌بندد و مقداری را باز نمی‌گرداند.

تابع 0EH: انتخاب دیسک درایو پیش فرض

هدف اصلی تابع 0EH انتخاب یک درایو به عنوان درایو پیش فرض است. برای استفاده از آن، شماره درایو را در DL تنظیم کنید بطوریکه 0 = درایو A، 1 = درایو B و بنابراین :

```
MOV  AH,0EH  ; درخواست تنظیم پیش فرض
MOV  DL,03   ; درایو D
INT  21H     ; فراخوانی سرویس وقفه
```

عملیات شماره درایوها را در AL ارائه می‌دهد (هم انواع شامل دیسک RAM). چون سیستم به دو درایو منطقی A و B نیاز دارد، مقدار 02 را برای سیستم با یک درایو باز می‌گرداند. (از INT 11H برای تعیین تعداد واقعی درایوها استفاده کنید).

تابع 19H : گرفتن دیسک درایو پیش فرض

این تابع دیسک درایو پیش فرض را تعیین می‌کند:

```
MOV  AH,19H  ; درخواست درایو پیش فرض
INT  21H     ; فراخوانی سرویس وقفه
```

عملیات یک شماره درایو در AL باز می‌گرداند که A = 0 و B = 1 و غیره. می‌توانید این شماره را مستقیماً به برنامه منتقل کنید تا یک فایل از درایو پیش فرض دستیابی شود اگر در برخی از توابع دیسک شاخص درایو A عدد 1 و شاخص درایو B عدد 2 است.

تابع 1BH : گرفتن اطلاعات درایو پیش فرض

این عملیات، که اکنون توسط تابع 39H کنار گذاشته شده، اطلاعاتی راجع به درایو پیش فرض باز می‌گرداند.

```
MOV  AH,1BH  ; درخواست اطلاعات درایو
INT  21H     ; فراخوانی سرویس وقفه
```

چون عملیات DS را تغییر می‌دهد، باید قبل از وقفه آن را بر روی پشته PUSH و بعد از وقفه POP کنید. یک عملیات موفق اطلاعات زیر را باز می‌گرداند :

AL تعداد سکتورهای هر کلاستر

BX اشاره‌گر (DS:BX) به اولین بایت (توصیف‌گر رسانه) در FAT

CX اندازه سکتور فیزیکی (اغلب ۵۱۲ بایت)

DX تعداد کلاستر روی دیسک

حاصل AL ، CX و DX ظرفیت دیسک را بدست می‌دهد. یک عملیات ناموفق FFH را در AL باز می‌گرداند.

تابع 1CH : گرفتن اطلاعاتی راجع به درایو خاص

این عملیات، هم اکنون توسط تابع 36H جایگزین شده است، اطلاعاتی راجع به درایو خاص باز می‌گرداند. برای استفاده از آن، شماره درایو مورد نظر را در DL قرار دهید، طوریکه 0 = پیش فرض، 1 = A و غیره :

```
MOV  AH,1CH  ; درخواست اطلاعات دیسک
MOV  DL,drive        ; شماره ابزار
INT  21H             ; فراخوانی سرویس وقفه
```

این عملیات از دیگر جهات با تابع 1BH یکسان می‌باشد.

تابع 1FH : گرفتن بلوک پارامتر درایو پیش فرض (DPB)

بلوک پارامتر درایو (DPB) یک ناحیه داده است که حاوی اطلاعات سطح پایین راجع بـه سـاختار داده در درایـو می‌باشد:

افست	اندازه	محتویات
00H	بایت	شماره درایو (0 = A و غیره)
01H	بایت	واحد منطقی درایو
02H	کلمه	اندازه سکتور به بایت
04H	بایت	سکتورهای هر کلاستر منهای یک
05H	بایت	سکتورهای هر کلاستر (توان ۲)
06H	کلمه	اولین سکتور رابطه‌ای FAT
08H	بایت	تعداد کپی FAT
09H	کلمه	تعداد ورودی شاخهٔ ریشه[1]
0BH	کلمه	اولین سکتور رابطه‌ای از اولین کلاستر
0DH	کلمه	بالاترین شماره کلاستر بعلاوه ۱
0FH	کلمه	سکتورهایی که با هر FAT اشغال شده‌اند
11H	کلمه	اولین سکتور رابطه‌ای از شاخه
13H	دو کلمه‌ای	آدرس راه انداز ابزار
17H	بایت	توصیف‌گر رسانه
18H	بایت	پرچم‌دستیابی(۰اگردیسک‌دستیابی‌شده‌باشد)
19H	دو کلمه‌ای	اشاره‌گر به بلوک پارامتر بعدی
1DH	کلمه	آخرین کلاستر اشغال شده
1FH	کلمه	تعداد کلاسترهای آزاد

DS را قبل از این توابع PUSH و بعد از بازگشت POP کنید:

```
PUSH DS
MOV  AH,1FH          ; درخواست آدرس DPB
INT  21H             ; فراخوانی سرویس وقفه
...  (دستیابی DPB)
POP  DS
```

1- Root Directory

یک عملیات معتبر AL را پاک می‌کند و یک آدرس در DS:BX باز می‌گرداند که به DPB برای درایو پیش فرض اشاره می‌کند. برای حالت خطا AL با FFH تنظیم می‌شود. تابع 32H را نیز ببینید.

تابع 2EH : تنظیم مجدد تصدیق نوشتن دیسک

این تابع به شما امکان می‌دهد تا عملیات نوشتن دیسک را برای ارزیابی صحت نوشته شده دادها تصدیق کنید. این تابع یک سوئیچ دارد که از سیستم می‌خواهد تا (CRC)کنترلر دیسک را تأکید کند.
CRC یک شکل هوشمندانه‌تر برای بررسی توزان می‌باشد. قرار دادن 00 در AL تصدیق را غیر فعال و 01 تصدیق را فعال می‌کند. سوئیچ تا زمانی که عملیات دیگری آن را تعویض نکند فعال باقی می‌ماند. مثالی را در زیر مشاهده می‌کنید :

```
MOV  AH,2EH  ; (MOV AX,2E01H) درخواست تصدیق
MOV  AL,01   ; تنظیم فعال تصدیق
INT  21H     ; فراخوانی سرویس وقفه
```

این تابع مقاری را باز نمی‌گرداند بلکه فقط یک سوئیچ را تنظیم می‌کند. پس از اجرای تابع، سیستم به عملیات نوشتن نا معتبر پاسخ می‌دهد. چون یک دیسک درایو بندرت داده‌های نادرست را ثبت می‌کند و تصدیق سبب تاخیر خواهد شد، این عملیات اغلب برای ثبت داده‌هایی که بطور ویژه بحرانی هستند مفید می‌باشد. یک تابع مرتبط، 54H تنظیم جاری سوئیچ تصدیق را باز می‌گرداند.

تابع 32H : گرفتن بلوک پارامترهای درایو (DPB)

برای گرفتن DPB، شماره درایو را در DX (بطوریکه 0 = پیش فرض، 1 = A و غیره) قرار دهید. (تابع 1FH را ببینید).

تابع 36H : گرفتن فضای آزاد دیسک

این تابع اطلاعاتی راجع به فضای روی ابزار دیسک ارائه می‌دهد. برای استفاده از آن، شماره درایو (0 = پیش فرض، 1 = A و 2 = B، غیره) را در DL قرار دهید:

```
MOV  AH,36H  ; درخواست فضای آزاد دیسک
MOV  DL,0    ; برای درایو پیش فرض
INT  21H     ; فراخوانی سرویس وقفه
```

یک عملیات موفق مقادیر زیر را باز می‌گرداند: AX = شماره سکتورهای هر کلاستر، BX = تعداد کلاسترهای در دسترس، CX = تعداد بایت‌های هر سکتور، DX = مجموع تعداد کلاسترهای روی ابزار. حاصل AX ، CX و DX ظرفیت دیسک را بدست می‌دهد. برای یک شماره ابزار نامعتبر، عملیات FFFFH را در AX، باز می‌گرداند. عملیات پرچم نقلی را تنظیم یا پاک نمی‌کند.

تابع 44H : کنترل I/O برای ابزار

این سرویس پر کار، IOCTL، ارتباط اطلاعات بین یک برنامه و یک ابزار باز را برقرار می‌کند. برای استفاده از آن، یک مقدار زیر تابع در AL قرار دهید تا یکی از چند فعالیت را درخواست کنید. یک عملیات معتبر پرچم نقلی را صفر می‌کند. یک خطا، مانند یک دستگیره فایل نامعتبر، پرچم نقل را تنظیم می‌کند و یک کد خطای/استاندارد را در AX باز می‌گرداند. زیر تابع‌های اصلی IOCTL بشرح زیر می‌باشد.

تابع 4400H : گرفتن اطلاعات ابزار

این عملیات اطلاعاتی راجع به یک فایل یا ابزار باز می‌گرداند :

```
MOV   AX,4400H   ; درخواست اطلاعات ابزار
MOV   DX,handle  ; دستگیره فایل یا ابزار
INT   21H        ; فراخوانی سرویس وقفه
```

یک عملیات معتبر پرچم نقلی را پاک می‌کند و یک مقدار در DX باز می‌گرداند، که بیت ۷ = ۰ به معنی آن است که دستگیره بر یک فایل دلالت دارد و بیت ۷ = ۱ به معنی یک ابزار است. دیگر بیت‌ها برای فایل یا ابزار به معنی زیر می‌باشند:

فایل (بیت 7 = 0) :

0 - 5 شماره درایو (0 = A و 1 = B و غیره)

6 1 = فایل نوشته نشد

ابزار (بیت 7 = 1) :

0 ورودی کنسول استاندارد

1 خروجی کنسول استاندارد

2 ابزار خالی

3 ابزار ساعت

4 ابزار خاص

5 0 = حالت ASCII، 1 = حالت باینری

6 برای ورودی، 0 = انتهای فایل بازگشت

یک خطا پرچم نقلی را تنظیم می‌کند و کد ۰۱، ۰۵ یا ۰۶ را در AX باز می‌گرداند.

تابع 4401H : تنظیم اطلاعات ابزار

این عملیات اطلاعات ابزار را تنظیم می‌کند، همانطور که برای تابع 4400H نشان داده شد. برای استفاده از آن دستگیره فایل را در BX و تنظیم بیت را در DL برای بیت‌های ۰ تا ۷ قرار دهید. یک خطا پرچم نقلی را تنظیم می‌کند و کد ۰۱، ۰۵، ۰۶ یا 0DH را در DX باز می‌گرداند.

تابع 4404H : خواندن داده کنترل از درایو

این عملیات داده کنترلی را از یک درایور بلوک ابزار (دیسک درایو) می‌خواند. برای استفاده از آن، درایو را در BL (۰ = پیش فرض ۱ = A و غیره)، تعداد بایت‌هایی که باید خوانده شود در CX قرار دهید و آدرس ناحیه داده را در DX قرار دهید، یک عملیات موفق در AX تعداد بایت منتقل شده را باز می‌گرداند یک خطا پرچم نقلی را تنظیم می‌کند و کد ۰۱، ۰۵ یا 0DH در AX باز می‌گرداند.

تابع 4405H : نوشتن داده کنترلی در درایو

این عملیات داده کنترلی را در یک راه انداز ابزار بلوک می‌نویسد. تنظیم مانند تابع 4404H است.

تابع 4406H : بررسی وضعیت ورودی

این سرویس بررسی می‌کند که آیا یک فایل یا ابزار برای ورودی آماده است. برای استفاده از آن، دستگیره را در BX

قرار دهید، یک عملیات معتبر یکی از کدهای زیر را در AL باز می‌گرداند :

- ابزار 00H = آماده نیست یا FFH = آماده است.
- فایل 00H = به E0F رسیده است یا FFH = E0F نرسیده است

یک خطا پرچم نقلی را تنظیم می‌کند و کد ۰۱ و ۰۵ یا ۰۶ را در AX باز می‌گرداند.

تابع 4407H از وقفه 21H: بررسی وضعیت خروجی

این سرویس تعیین می‌کند که آیا یک فایل یا ابزار برای خروجی آماده است. یک عملیات معتبر یکی از کدهای زیر را در AL باز می‌گرداند.

- ابزار: 00H = آماده نیست یا FFH = آماده است.
- فایل: 00H = آماده است یا FFH = آماده است.

یک خطا پرچم نقلی را تنظیم می‌کند و کد ۰۱، ۰۵، یا ۰۶ را در AX باز می‌گرداند.

تابع 4408H :تعیین اینکه برای ابزار یک واسطه قابل انتقال وجود دارد.

این سرویس تعیین می‌کند که آیا ابزار شامل یک واسطه قابل انتقال مانند دیسکت وجود دارد. برای استفاده از آن BL را با شماره درایو بارگذاری کنید (0 = پیش فرض، 1 = A و غیره) یک عملیات معتبر پرچم نقلی را صفر می‌کند و یکی از کدهای زیر را در AX باز می‌گرداند. 00H = ابزار قابل انتقال یا 01H = ابزار ثابت. یک خطا پرچم نقلی را تنظیم می‌کند و کد ۰۱ یا 0FH را در AX باز می‌گرداند (شماره درایو نامعتبر).

تابع 440DH از وقفه 21H ، کد فرعی 41H : نوشتن سکتور دیسک

این عملیات داده را از یک بافر به یک یا چند سکتور روی دیسک می‌نویسد. برای استفاده از آن، این ثبات‌ها را مقدار دهید.

```
MOV   AX,440DH      ; درخواست نوشتن سکتور دیسک
MOV   BX,drive      ; درایو (0 = پیش فرض، 1 = A و غیره)
MOV   CH,08H        ; طبقه بندی ابزار = 08H
MOV   CL,41H        ; کد فرعی = نوشتن تراک
LEA   DX,devblock   ; آدرس بلوک ابزار
INT   21H           ; فراخوانی سرویس وقفه
```

آدرسی که در DX باز می‌گردد به یک بلوک ابزار در قالب زیر اشاره می‌کند :

```
devblock  LABEL BYTE       ;Device block:
specfunc  DB  0            ;  Special functions (zero)
rwhead    DW  head         ;  Read/write head
rwcyl     DW  cylinder     ;  Cylinder
rwsect1   DW  sector       ;  Starting sector
rwsects   DW  number       ;  Number of sectors
rwbuffr   DW  buffer       ;  Offset address of buffer
          DW  SEG _DATA    ;  Address of data segment
```

ورودی با نام rwbuffr آدرس بافر در سگمنت را فراهم می‌سازد: افست (DS:DX)، به ترتیب کلمه معکوس ذخیره شده است. عملگر SEG بر تعریف سگمنت دلالت دارد ، در این حالت سگمنت داده، DATA- می‌باشد. بافر ناحیه داده که باید نوشته شود، تعیین می‌کند و باید به طول تعداد سکتورها × ۵۱۲ باشد ، مانند :

```
WRBUFFER DB 1024 DUP(?) ; بافر خروجی
```

یک عملیات موفق پرچم نقلی را پاک می‌کند و داده‌ها را می‌نویسید. در غیر اینصورت، عملیات پرچم نقلی را تنظیم می‌کند. کد خطا 01، 02، یا 05 را در AX باز می‌گرداند.

تابع 440DH ، از وقفه 21 H کد فرعی 42 H : فرمت تراک‌ها

برای استفاده از این تابع جهت فرمت تراک‌ها، این ثبات‌ها را تنظیم کنید:

```
MOV   AX,440DH          ; درخواست تنظیم ID واسطه
MOV   BX,drive          ; درایو (0 = پیش فرض، 1 = A و غیره)
MOV   CH,08             ; طبقه بندی ابزار = 08
MOV   CL,46 H           ; کد فرعی = تنظیم ID واسطه
LEA   DX,block          ; آدرس بلوک (DS:DX)
INT   21H               ; فراخوانی سرویس وقفه
```

آدرس بازگشتی در DX به بلوک واسطه در قالب زیر اشاره می‌کند :

```
blkname  LABEL BYTE    ;Disk information block:
specfun  DB    0       ;  Special function, code 0
diskhd   DW    ?       ;  Disk head
cylindr  DW    ?       ;  Cylinder
tracks   DW    ?       ;  Number of tracks
```

فیلد filetyp حاوی مقادیر ASCII، FAT12 یا FAT 16 با دنباله‌ای از جای خالی است. یک عملیات موفق پرچم نقلی را پاک می‌کند و ID را تنظیم می‌نماید. در غیر اینصورت تابع پرچم نقلی را یک می‌کند و کد خطای 01، 02 یا 05 را در AX باز می‌گرداند. (تابع 440DH با کد فرعی 66H را نیز ببینید)

تابع 440DH ، از وقفه 21 H کد فرعی 46 H : تنظیم شماره شناسایی رسانه

کاربرد این تابع برای تنظیم شماره شناسایی رسانه، به شرح زیر است :

تقاضای تنظیم شماره شناسایی رسانه ;

```
MOV   AX,440DH          ; درایو (0 = پیش فرض، 1 = A و . . . )
MOV   BX,drive          ; طبقه‌بندی ابزار (08)
MOV   CH,08             ; کد فرعی = تنظیم شماره شناسایی رسانه
LEA   CL,block          ; آدرس بلاک (DS:DX)
INT   21H               ; فراخوانی سرویس وقفه
```

آدرس ارجاع شده در DX به یک بلاک رسانه با مشخصات زیر اشاره می‌کند :

```
blkname   LABEL   BYTE          ; بلاک رسانه
infolev   DW      0             ; سطح اطلاعات = 0
serialn   DD      ??            ; شماره سریال
volabel   DB      11 DUP (?)    ; برچسب
filetyp   DB      8 DUP (?)     ; نوع FAT
```

فیلد filetyp حاوی مقدار اسکی FAT12 یا FAT16 می‌باشد. در یک عملیات موفق پرچم carry پاک شده و شماره شناسایی تعیین می‌شود. در غیر اینصورت این پرچم یک شده و یک از کدهای خطای 01، 02 یا 05 در ثبات AX بازگردانده می‌شود (به تابع 66H رجوع کنید)

تابع 440DH از وقفه 21H : کدکوچکتر 60H : گرفتن پارامترهای ابزار

برای استفاده از این تابع جهت گرفتن پارامترهای ابزار، این ثبات‌ها را تنظیم نمایید:

```
MOV  AX,440DH      درخواست گرفتن پارامترهای ابزار ;
MOV  BX,drive      درایو (0 = پیش فرض، 1 = A و غیره) ;
MOV  CH,08         طبقه بندی ابزار = 08 ;
MOV  CL,60H        کد فرعی = گرفتن پارامترها ;
LEA  DX,block      آدرس بلوک (DS:DX) ;
INT  21H           فراخوانی سرویس وقفه ;
```

آدرس بازگشتی در DX ، به بلوک پارامتر ابزار (DPB) با قالب زیر اشاره دارد :

```
specfun  DB  ?    ;Special functions (0 or 1)
devtype  DB  ?    ;Device type
devattr  DW  ?    ;Device attribute
cylindr  DW  ?    ;Number of cylinders
medityp  DB  ?    ;Media type
bytesec  DW  ?    ;Bytes per sector
secclus  DB  ?    ;Sectors per cluster
ressect  DW  ?    ;Number of reserved sectors
fats     DB  ?    ;Number of FATs
rootent  DW  ?    ;Number of root directory entries
sectors  DW  ?    ;Total number of sectors
mediads  DB  ?    ;Media descriptor
fatsecs  DW  ?    ;Number of sectors per FAT
sectrak  DW  ?    ;Sectors per track
heads    DW  ?    ;Number of heads
hidsect  DD  ?    ;Number of hidden sectors
exsects  DD  ?    ;Number of sectors if sectors field = 0
```

اگر فیلد specfun صفر باشد ، اطلاعات راجع به واسطه پیش فرض در درایو است، اگر یک باشد اطلاعات راجع به واسطه جاری است، یک عملیات موفق پرچم نقلی را پاک می‌کند و داده را ارائه می‌دهد. در غیر اینصورت عملیات پرچم نقلی را یک می‌کند و کد خطا ۰۱، ۰۲، یا ۰۵ را در AX باز می‌گرداند.

تابع DH 440، کد فرعی 61H : خواندن سکتور دیسک

این عملیات داده را از یک یا چند سکتور روی دیسک خوانده و در بافر قرار می‌دهد، برای استفاده از آن، CL را با کد فرعی 61H تنظیم نمایید ، از دیگر جهات، جزئیات تکنیکی با کد فرعی 41 H کـه سکـتورها را مـی‌نویسد یکسـان می‌باشد. شکل ۱-۱۸، که بعداً خواهید دید، این تابع را مشخص می‌سازد.

تابع DH 440 از وقفه H 21 کد فرعی 66H : گرفتن ID واسطه‌ها

برای استفاده از این تابع جهت اخذ ID واسطه‌ها، این ثبات‌ها را تنظیم نمایید:

```
MOV  AX,440 DH     درخواست ID واسطه ;
MOV  BX,drive      درایو (0 = پیش فرض، 1 = A و غیره) ;
MOV  CH,08         طبقه بندی ابزار ;
MOV  CL,66H        کد فرعی = گرفتن ID واسطه ;
LEA  DX,block      آدرس بلوک (DS:DX) ;
INT  21H           فراخوانی سرویس وقفه ;
```

آدرس بازگشتی در DX به بلوک واسطه اشاره می‌کند :

```
blkname  LABEL  BYTE          ;Media block:
infolev  DW     0             ; Information level = 0
serialn  DD     ?             ; Serial number
volabel  DB     11 DUP (?)    ; Volume label
filetyp  DB     8 DUP (?)     ; Type of FAT
```

یک عملیات موفق پرچم نقلی را صفر می‌کند و ID را تنظیم می‌کند. فیلد filetyp حاوی مقدار ASCII، FAT 12 یا FAT 16 با دنباله‌ای از جای خالی است. در غیر اینصورت ، عملیات پرچم نقلی را تنظیم می‌کند و کد خطای 01، 02 یا 05 را در AX باز می‌گرداند (تابع 440DH کد فرعی 46H را نیز ببینید.)

تابع 440DH از وقفه 21 H کد فرعی 68 H : تشخیص نوع واسطه‌ها

```
MOV  AX,440 DH          ; درخواست نوع واسطه‌ها
MOV  BX,drive           ; درایو (0 = پیش فرض، 1 = A و غیره)
MOV  CH,08              ; طبقه بندی ابزار(08)
MOV  CL,68 H            ; کد فرعی = گرفتن نوع واسطه
LEA  DX,block           ; آدرس بلوک (DS:DX)
INT  21H                ; فراخوانی سرویس وقفه
```

آدرس بازگشتی در DX به یک بلوک ۲ بایتی، جهت دریافت داده‌ها اشاره می‌کند:

```
default  DB  ?   ; ۱ برای حالت پیش فرض ۲ برای دیگر حالات
medatyp  DB  ?   ; 02 = 720K , 07 = 1.44MB , 09 = 2.88MB
```

یک تابع موفق پرچم نقلی را صفر می‌کند و نوع را تنظیم می‌کند. در غیر اینصورت تابع پرچم نقلی را یک می‌کند و کد خطای 01 یا 05 را در Ax باز می‌گرداند.

دیگر توابع IOCTL برای تابع 44H، که در اینجا ذکر نخواهد شد، در مورد اشتراک فایل می‌باشند.

تابع 54 H از وقفه 21 H : گرفتن وضعیت تصدیق

این سرویس می‌تواند وضعیت پرچم تصدیق نوشتن دیسک را معین سازد. (تابع 2 EH برای تنظیم این سوئیچ را ببینید). این عملیات 00H را در AL در صورت غیر فعال بودن تصدیق یا 01H در صورت فعال بودن تصدیق بار می‌گرداند. هیچ حالت خطایی وجود ندارد.

تابع 59 H از وقفه 21 H : گرفتن خطای توسعه یافته

این عملیات اطلاعات اضافی راجع به خطاهای پس از اجرای سرویس‌های INT 21H که پرچم نقلی را تنظیم می‌کنند، سرویس‌های FCB که FFH باز می‌گرداند و دستیابی خطاهای INT 24 H فراهم می‌سازد.

عملیات موارد زیر را باز می‌گرداند.

فعالیتی که پیشنهاد می‌شود = BL	کد خطای توسعه یافته = AX
موقعیت = CH	کلاس خطا = BH

همچنین، عملیات پرچم نقلی را صفر می‌کند - بررسی کنید - محتویات ثبات‌های CL ، DI ، DS ، DX ، ES و SI را خراب می‌کند. همه این ثبات‌های مورد نیاز را قبل از وقفه PUSH و بعد از آن POP کنید. بخش‌های زیر، خطاها را توضیح می‌دهد :

کد خطای توسعه یافته (AX)

۹۰ یا تعداد بیشتری کد خطا وجود دارد، کد 00 یعنی عملیات INT 21 H قبلی هیچ خطایی ندارد.

کلاس خطا(BH). اطلاعات زیر را فراهم می‌سازد :

01H خارج از منبع، مانند کانال ذخیره سازی
02H وضعیت کمکی (نه یک خطا) مانند یک شرایط قفل شدن فایل که باید برداشته شود
03H عدم اختیار درست
04H خطای نرم‌افزاری سیستم، نه این برنامه
05H خطای سخت افزار
06H خطای جدی سیستم، نه این برنامه
07H خطا در این برنامه، مانند یک درخواست ناجور
08H عنصر خواسته شده پیدا نشد
09H فایل یا فرمت دیسک نادرست
0AH فایل یا عنصر قفل است
0BH خطای دیسک، مانند خطای CRC یا دیسک اشتباه
0CH فایل یا عنصر هم اکنون موجود است
0DH کلاس خطای ناشناخته

فعالیت (BL). اطلاعاتی راجع به فعالیتی که باید انجام بگیرد فراهم می‌سازد :

01 چند بار دیگر سعی مجدد ممکن است از کاربر درخواست شود تا خاتمه دهد.
02 ابتدا توقف و چند بار سعی مجدد
03 درخواست کاربر برای ورود مجدد درخواست مناسب
04 بستن فایل‌ها و خاتمه برنامه
05 ختم بلافاصله برنامه، فایل‌ها بسته نشد
06 نادیده گرفتن خطا
07 درخواست کاربر برای یک فعالیت (مانند تغییر دیسکت) و سعی مجدد در عملیات.

موقعیت (CH). اطلاعات اضافی برای قرار گرفتن یک خطا فراهم می‌سازد

01 حالت ناشناخته نمی‌توان کمک کرد
02 مشکل حافظه دیسک
03 مشکل شبکه
04 مشکل ابزار سریال
05 مشکل حافظه

برنامه: خواندن داده از سکتورها

برنامه شکل ۱-۱۸ استفاده از IOCTL تابع H 44 زیر تابع 0DH با کد فرعی H 61 را مشخص می‌سازد. این برنامه داده را از یک سکتور خوانده و در بافری در حافظه قرار می‌دهد و هر بایت ورودی را مانند یک جفت کاراکتر شانزدهی نمایش می‌دهد، همانطور که در شکل ۶-۱۵ نشان داده شد.

```
TITLE     A18RDSCT (EXE)  Read disk sector
          .MODEL  SMALL
          .STACK  64
          .DATA
ROW       DB      00
COL       DB      00
XLATAB    DB      30H,31H,32H,33H,34H,35H,36H,37H,38H,39H
          DB      41H,42H,43H,44H,45H,46H
READMSG   DB      '*** Read error ***', 0DH, 0AH

RDBLOCK   DB      0                   ;Block
RDHEAD    DW      0                   ;  structure
RDCYLR    DW      0                   ;
RDSECT    DW      8                   ;
RDNOSEC   DW      1                   ;
RDBUFFR   DW      IOBUFFR             ;
          DW      SEG _DATA           ;
IOBUFFR   DB      512 DUP(' ') ;Disk sector area
;-------------------------------------------------------
.386
          .CODE
A10MAIN   PROC    FAR
          MOV     AX,@data            ;Initialize
          MOV     DS,AX               ;  segment
          MOV     ES,AX               ;  registers
          CALL    Q10SCR              ;Clear screen
          CALL    Q20CURS             ;Set cursor
          CALL    B10READ             ;Get sector data
          JNC     A80                 ;If valid read, bypass
          LEA     DX,READMSG          ;If invalid, display
          CALL    X10ERR              ;  error message
          JMP     A90
A80:
          CALL    C10DISP             ;Convert and display
A90:      MOV     AX,4C00H            ;End processing
          INT     21H
A10MAIN   ENDP
;                 Read sector data:
;                 -----------------
B10READ   PROC    NEAR
          MOV     AX,440DH      ;IOCTL for block device
          MOV     BX,01         ;Drive A
          MOV     CH,08         ;Device category
          MOV     CL,61H        ;Read sector
          LEA     DX,RDBLOCK    ;Address of block structure
          INT     21H
          RET
B10READ   ENDP
;                 Display sector data:
;                 --------------------
C10DISP   PROC    NEAR
          LEA     SI,IOBUFFR
C20:
          MOV     AL,[SI]
          SHR     AL,04               ;Shift off right hex digit
          LEA     BX,XLATAB           ;Set table address
          XLAT                        ;Translate hex
          CALL    Q30DISPL
          INC     COL
          MOV     AL,[SI]
          AND     AL,0FH              ;Clear left hex digit
          XLAT                        ;Translate hex
          CALL    Q30DISPL
          INC     SI
          INC     COL
          CMP     COL,64              ;Rightmost screen loc
          JBE     C20                 ;No, repeat
```

شکل الف ۱-۱۸ خواندن سکتورهای دیسک

```
            INC     ROW                 ;Yes, increment row
            MOV     COL,00
            CALL    Q20CURS
            CMP     ROW,15              ;At last row?
            JBE     C20                 ;No, repeat
            RET
C10DISP     ENDP
;                   Scroll screen:
;                   --------------
Q10SCR      PROC    NEAR
            MOV     AX,0600H            ;Request scroll
            MOV     BH,1EH              ;Set attribute
            MOV     CX,0000
            MOV     DX,184FH
            INT     10H
            RET
Q10SCR      ENDP
;                   Set cursor:
;                   ----------
Q20CURS     PROC    NEAR
            MOV     AH,02H              ;Request set
            MOV     BH,00               ;  cursor
            MOV     DH,ROW              ;  row
            MOV     DL,COL              ;  column
            INT     10H
            RET
Q20CURS     ENDP
;                   Display character:
;                   ------------------
Q30DISPL    PROC    NEAR
            MOV     AH,02H              ;Request display
            MOV     DL,AL               ;  character
            INT     21H
            RET
Q30DISPL    ENDP
;                   Display disk error message:
;                   ---------------------------
X10ERR      PROC    NEAR
            MOV     AH,40H              ;DX contains address
            MOV     BX,01               ;Handle
            MOV     CX,20               ;Length
            INT     21H                 ;  of message
            INC     ROW
            RET
X10ERR      ENDP
            END     A10MAIN
```

شکل ب ۱-۱۸ خواندن سکتورهای دیسک

ساختار بلوک در سگمنت داده، RDBLOCK بطور اختیاری یک هد، سیلندر، سکتور شروع را مشخص می‌سازد که می‌توانید برای اهداف برنامه تغییر دهید. RDBUFFR دو آدرس تعریف می‌کند:

۱) IOBUFFR در آدرس افست از بافر ورودی است، که برای یک سکتور (۵۱۲ بایتی) از داده فراهم شده است.

۲) SEG - DATA از عملگر SEG برای تعیین آدرس سگمنت داده جهت، عملیات IOCTL استفاده می‌کند.

روال‌های اصلی در سگمنت کد بشرح زیر می‌باشد:

- A10MAIN ، B10READ را فرامی خواند، و اگر یک عملیات معتبر باشد، C10DISP را فرامی خواند.
- B10READ از عملیات CTL 10 برای خواندن داده از سکتور به IOBUFFR استفاده می‌کنند. (بررسی خواندن معتبر، زمان بازگشت از روال انجام می‌گیرد).
- C10DISP هر بایت در IOBUFFR را به دو کاراکتر شانزدهی برای نمایش تبدیل می‌کند. دو دستور XLAT

تبدیلات هر نیمه بایت را دستکاری می‌کند. روتین ۱۶ سطر با ۳۲ جفت کاراکتر در ستونهای ۰ تا ۶۳ و سطرهای ۰ تا ۱۵ نمایش می‌دهد.

می‌توانید این برنامه را توسعه دهید بطوریکه به کاربر این امکان را بدهید که از صفحه کلید هر سکتور را درخواست نماید.

عملیات دستکاری شاخه و FAT

تابع 39 H : از وقفهٔ 21H : ایجاد زیر شاخه

این سرویس یک زیر شاخه ایجاد می‌کند، دقیقاً مانند فرمان MKDIR از DOS می‌باشد. برای استفاده از آن، در DX آدرس یک رشته ASCIIZ حاوی درایو و نام مسیر شاخه قرار دهید:

```
ASCSTRYDB  d:'dathname' , 00H  ; رشته ASCIIZ
           MOV   AH,39H        ; درخواست ایجاد زیر شاخه
           LEA   DX,Ascstry    ; (DS:DX) آدرس رشته ASCIIZ
           INT   21H           ; فراخوانی سرویس وقفه
```

یک عملیات معتبر پرچم نقلی را صفر می‌کند، یک خطا پرچم نقلی را یک می‌کند و کد ۰۳ یا ۰۵ را به AX باز می‌گرداند.

تابع 3AH : پاک کردن زیر شاخه

این سرویس یک زیر شاخه را پاک می‌کند، دقیقاً مانند فرمان RMDIR از DOS می‌باشد. توجه کنید که نمی‌توان شاخه جاری یا یک زیر شاخه که فایل دارد پاک کرد. در DX آدرس رشته ASCIIZ حاوی درایو و مسیر شاخه را قراردهید:

```
AscstrgDB'd:\pathname',00H  ; ASCIIZ string
           MOV   AH,3AH     ; درخواست حذف زیر شاخه
           LEA   DX,Ascstry ; (DS:DX) آدرس رشته ASCIIZ
           INT   21H        ; فراخوانی سرویس وقفه
```

یک عملیات معتبر پرچم نقلی را پاک می‌کند، یک خطا پرچم نقلی را تنظیم می‌کند و کد ۰۳، ۰۵، یا 10H را در AX باز می‌گرداند.

تابع 38H از وقفه 21 H: تغییر شاخه جاری

این سرویس شاخه‌جاری را به شاخه‌ای که شما مشخص کرده‌اید تغییرمی‌دهد ودقیقاً مانند فرمان CHDIR از DOS عمل می‌کند. برای استفاده از آن، در DX آدرس رشته ASCIIZ حاوی درایو جدید و نام مسیر دایرکتوری قرار دهید :

```
Ascstrg  DB   'd:\pathname',00H  ; رشته ASCIIZ
         MOV  AH,3BH             ; درخواست تغییر زیر شاخه
         LEA  DX,Ascstry         ; (DS:DX)آدرس رشته ASCIIZ
         INT  21H                ; فراخوانی سرویس وقفه
```

یک عملیات معتبر پرچم نقلی را صفر می‌کند، یک خطا پرچم نقلی را یک می‌کند کد ۰۳ را در AX باز می‌گرداند.

تابع 47 H : گرفتن شاخه جاری

این سرویس شاخه جاری برای هر درایو را تعیین می‌کند. برای استفاده از آن، یک فضای با فربه اندازه کافی بزرگ

جهت بزرگترین نام مسیر ممکن (۶۴ بایت) تعریف کنید و آدرس آن را در SI قرار دهید. درایو را در DL مشخص کنید، ۰ = پیش فرض، A = 1 و B = 2 و غیره :

```
buffer  DB   64 DUP (20H)      ; فضای بافر ۶۴ بایت
        ...
        MOV  AH,47H            ; درخواست گرفتن شاخه
        MOV  DL,drive          ; درایو
        LEA  SI,buffer         ; آدرس بافر (DS:DI)
        INT  21H               ; فراخوانی سرویس وقفه
```

یک عملیات معتبر پرچم نقلی را صفر می‌کند و نام شاخه جاری (نه درایو) را در یک بافر مانند یک رشته ASCIIZ به شکل PCPROG S\TESTDATA0 ارائه می‌دهد. یک بایت حاوی 00 H انتهای نام مسیر را مشخص می‌سازد، اگر شاخه خواسته شده ریشه باشد ، مقدار بازگشتی فقط یک بایت 00H خواهد بود. با این روش می‌توانید نام مسیر جاری را برای دستیابی هر فایل در یک زیر شاخه، اخذ نمایید. یک شماره درایو معتبر پرچم نقلی را تنظیم می‌کند و کد خطای 0FH را در AX باز می‌گرداند.

تابع 56 H : نام‌گذاری مجدد فایل یا شاخه

بخش بعدی را برای این تابع ببینید.

برنامه: نمایش شاخه

برنامه شکل ۲-۱۸ استفاده از دو تابع توضیح داده شده در بخش‌های قبلی را توصیف می‌کند. روال‌ها چنین عمل می‌کنند:

- A10MAIN روال‌های B10DRIV و C10PATH را فرا می‌خواند و برای یک ورودی از صفحه کلید قبل از خاتمه منتظر می‌شود.
- B10DRIV از تابع 19H برای گرفتن درایو پیش فرض در ثبات AL استفاده می‌کند. درایو بصورت 0 برای A، ۱ برای B و غیره بازگردانده می‌شود، برای تنظیم شماره با معادل الفبایی، روال بسادگی 41H را با آن جمع می‌کند، بنابراین 00 خواهد شد 41 H (A) ، 01 خواهد شد 42 H (B) و غیره سپس حرف درایو را و پس از آن یک : و یک \ را نمایش می‌دهد (n:\)
- C10PATH از تابع 47 H برای گرفتن نام مسیر شاخه جاری استفاده می‌کند. این روال بلافاصله وجود کاراکتر 00H را در انتهای رشته بررسی می‌کند، زیرا در غیر اینصورت روتین، نام کاراکترها را تا رسیدن به محدود کننده بعدی نمایش می‌دهد.
- DI0DISP یک کاراکتر تک را نمایش می‌دهد.

برنامه عمداً حاوی جزئیات ضروری برای فقط کار کردن است ، یک برنامه کامل برای مثال می‌تواند شامل، پاک کردن صفحه و تنظیم رنگ‌ها باشد.

توابع مربوط به فایل‌ها

این بخش عملیات پردازش فایل‌های دیسکی را توصیف می‌کند.

```
TITLE     A18GETDR (COM)  Get current directory
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT A10MAIN
; ------------------------------------------------
PATHNAM   DB      64 DUP(' ')      ;Current pathname
; ------------------------------------------------
A10MAIN   PROC    NEAR
          CALL    B10DRIV          ;Get/display drive
          CALL    C10PATH          ;Get/display path
          MOV     AH,10H           ;Pause until user
          INT     16H              ;  presses a key
          MOV     AX,4C00H         ;End processing
          INT     21H
A10MAIN   ENDP
;                  Get default drive:
;                  -----------------
B10DRIV   PROC    NEAR;
          MOV     AH,19H           ;Request default drive
          INT     21H
          ADD     AL,41H           ;Change hex no. to letter
          MOV     DL,AL            ;  0=A, 1=B, etc.
          CALL    D10DISP          ;Display drive number,
          MOV     DL,':'
          CALL    D10DISP          ;  colon,
          MOV     DL,'\'
          CALL    D10DISP          ;  backslash
          RET
B10DRIV   ENDP
;                  Get pathname for current directory:
;                  -----------------------------------
C10PATH   PROC    NEAR;
          MOV     AH,47H           ;Request pathname
          MOV     DL,00
          LEA     SI,PATHNAM
          INT     21H
C20:
          CMP     BYTE PTR [SI],00H ;End of pathname?
          JE      C90              ;  yes, exit
          MOV     AL,[SI]          ;Display pathname
          MOV     DL,AL            ;  one byte at
          CALL    D10DISP          ;  a time
          INC     SI
          JMP     C20              ;Repeat
C90:      RET
C10PATH   ENDP
;                  Display single character:
;                  -------------------------
D10DISP   PROC    NEAR             ;DL set on entry
          MOV     AH,02H           ;Request display
          INT     21H
          RET
D10DISP   ENDP
          END     BEGIN
```

شکل ۲-۱۸ گرفتن شاخه جاری

تابع 29H : تشریح نام فایل

این سرویس یک فرمان خطی حاوی یک مشخصه فایل بشکل d:filename.ext را به قالب FCB تبدیل می‌کند. تابع یک مشخصه فایل را از کاربر دریافت می‌کند، بعنوان مثال، برای کپی کردن یا حذف کردن فایل‌ها د رثبات SI (با استفاده بصورت DS:SI) آدرس مشخصه فایل که باید تشریح شود، در DI (بصورت ES:DI استفاده می‌شود) آدرس ناحیه‌ای که عملیات باید قالب FCB را در آن تولید نماید و در AL یک مقدار بیت که روش تشریح را کنترل می‌کند، قرار دهید:

```
MOV   AH,29H        ; درخواست تشریح نام فایل
MOV   AL,code       ; روش تشریح
LEA   DI,FCB name   ; آدرس FCB
LEA   SI,filespec   ; آدرس مشخصه فایل
INT   21 H          ; فراخوانی سرویس وقفه
```

کدهای روش تشریح چنین می‌باشد:

بیت	مقدار	عمل
0	0	یعنی مشخصه فایل از اولین بایت آغاز می‌شود.
0	1	پیمایش پس از جداکننده (مانند جای خالی) برای یافتن مشخصه فایل.
1	0	تنظیم بایت ID درایو در FCB تولید شده: درایو پیش فرض B = 02 , 00 A = 01 و غیره.
1	1	تغیر بایت ID درایو در FCB تولید شده فقط در صورتیکه مشخصه فایل تشریح شده یک درایو را مشخص سازد. با این روش یک FCB می‌تواند درایو پیش فرض خودش را داشته باشد
2	0	تغییر نام فایل در FCB در صورت لزوم
2	1	تغییر نام فایل در FCB فقط در صورتیکه مشخصه فایل حاوی یک نام معتبر باشد.
3	0	تغییر توسعه نام فایل در صورت لزوم
3	1	تغییر توسعه نام فایل فقط در صورتیکه حاوی یک توسعه معتبر باشد.
4-7	0	باید صفر باشد.

برای داده‌های معتبر، تابع 29 H یک قالب FCB استاندارد جهت نام فایل و توسعه آن، با یک نام فایل ۸ کاراکتری با جای خالی در صورت نیاز پر می‌شود، یک توسعهٔ ۳ کاراکتری که با جای خالی در صورت نیاز پر می‌شود، ایجاد می‌کند که هیچ نقطه‌ای بین آنها نیست.

عملیات نقطه‌گذاری استاندارد را تشخیص می‌دهد و کاراکترهای جایگزین * و? را به رشته‌ای از یک یا چند کاراکتر تبدیل می‌کند. برای مثال *.PROG12 خواهد شد ??? bb 12 PROG (۲ جای خالی برای پر کردن نام فایل و ??? برای توسعه است). AL یکی از کدهای 00H = هیچ کاراکتر جایگزین وجود ندارد، =01H کاراکترهای جایگزین تبدیل شوند، یا FFH = درایو مشخص شده نامعتبر.

بعد از عملیات، DS:SI حاوی آدرس اولین بایت بعد از مشخصه فایل تشریح شده است و ES : DI حاوی آدرس اولین بایت FCB می‌باشد. در یک عملیات ناموفق، بایت DI+1 جای خالی است اگر چه عملیات سعی در تبدیل اغلب آنچیزی که قرار داده‌اید دارد. جهت انجام این عملیات با دستگیره‌های فایل، ویرایش بیشتری لازم است که شامل حذف جای خالی و جایگذاری یک نقطه بین نام فایل و توسعه آن می‌باشد.

تابع 41H : حذف فایل

این تابع یک فایل (غیر از فقط خواندنی) را از داخل یک برنامه حذف می‌کند. در DX آدرس رشته ASCIIZ حاوی مسیر ابزار و نام، بدون هیچ کاراکتر جایگزین قرار دهید:

```
Ascstrng  DB   'd:\pathname',00H        ; ASCIIA string
          . . .
          MOV  AH,41 H                 درخواست حذف فایل ;
          LEA  DX,Ascstrng             ; (DS:DX)ASCIIZ آدرس رشته
          INT  21H                     فراخوانی سرویس وقفه ;
```

یک عملیات موفق پرچم نقلی را پاک می‌کند، نام فایل در شاخه را بصورت حذف شده علامت می‌زند و فضای دیسک تخصیص داده شده به فایل در FAT را آزاد می‌سازد. یک خطا سبب تنظیم پرچم نقلی و باز گرداندن 02، 03 یا 05 در AX خواهد شد.

تابع 43H: گرفتن یا تنظیم صفت فایل

از این عملیات هم برای گرفتن یا تنظیم صفت فایل در مسیر شاخه می‌توانید استفاده نمایید. عملیات به آدرس رشته‌ای حاوی درایو ، مسیر و نام فایل برای فایل مورد نظر نیاز دارد.یا از شاخه پیش فرض اگر هیچ مسیری داده نشده، استفاده می‌کند.

گرفتن صفت. برای اخذ صفت در AL کد 00 قرار دهید، همانطور که در مثال زیر نشان داده شده است

```
Asc3trg   DB   'd:\pathname',00H        ; رشتهٔ ASCIIZ
          MOV  AH,43 H                 درخواست ;
          MOV  AL,000                  گرفتن صفت ;
          LEA  DX,ASC strng            ; (DS:DX) رشته ASCIIZ
          INT  21H                     فراخوانی سرویس وقفه ;
```

یک عملیات معتبر پرچم نقلی و CH را صفر می‌کند و صفت جاری را در CL باز می‌گرداند.

بیت	صفت	بیت	صفت	بیت	صفت
۰	فایل فقط خواندنی	۱	فایل پنهان	۲	فایل سیستم
۳	برچسب volume	۴	زیرشاخه	۵	فایل آرشیو

یک خطا پرچم نقلی را یک می‌کند و کد ۰۲ یا ۰۳ را در AX باز می‌گرداند.

تنظیم صفت فایل برای **تنظیم** صفت فایل، در AL کد 01 در AX بیت‌های صفت را قرار دهید. امکان تغییر فایل‌های فقط خواندنی ، پنهان و سیستم وجود دارد ولی امکان تغییر بر چسب volume یا زیر شاخه وجود ندارد. مثال زیر صفات آرشیو پنهان را برای یک فایل تنظیم می‌کند:

```
          MOV  AH,43 H                 فراخوانی ;
          MOV  AL,01H                  تنظیم صفات ;
          MOV  CX,22 H                 پنهان و آرشیو ;
          LEA  DX,ASCSTRNG             ; (DS:DX) رشته ASCIIZ
          INT  21H                     فراخوانی سرویس وقفه ;
```

یک عملیات معتبر پرچم نقلی را صفر می‌کند و ورودی شاخه را با صفت CX تنظیم می‌کند. یک عملیات ناموفق پرچم نقلی را یک می‌کند و کد ۰۲، ۰۳ یا ۰۵ را در AX باز می‌گرداند.

تابع 45 H: دو نسخه کردن یک دستگیره فایل

هدف از این سرویس تخصیص دادن بیشتر از یک دستگیره به یک فایل است. استفاده از دستگیره‌های قبلی در کنار دستگیره‌های جدید یکسان می‌باشد - دستگیره‌ها به یک فایل، اشاره‌گر فایل و ناحیه بافر ارجاع می‌کنند. یک مورد استفاده درخواست دستگیره فایل و دیگری دستیابی برای بستن یک فایل می‌باشد. این فعالیت سبب می‌شود تا سیستم

بافر را خالی کند و شاخه را بروز رسانی نماید. پس می‌توانید از دستگیره فایل اصلی برای ادامه پردازش فایل استفاده نمایید. در این جا مثالی از تابع 45 H آورده شده است :

```
درخواست نسخه‌ای از دستگیره ;  MOV   AH,45H
دستگیره جاری نسخهٔ دیگری خواهد داشت ;  MOV BX,handle
فراخوانی سرویس وقفه ;  INT   21H
```

یک عملیات موفق، پرچم نقلی را پاک می‌کند و دستگیره فایل بعدی را در AX باز می‌گرداند. یک خطا پرچم نقلی را تنظیم می‌کند و کد خطا 04 یا 06 را در AX باز می‌گرداند. (تابع 46 H را نیز ببینید.)

تابع 46 H : تحمیل دو نسخه کردن یک دستگیره فایل

این سرویس مشابه تابع 45 H است، بجز آنکه این یکی می‌تواند یک دستگیره فایل خاص را تعیین نماید. از این سرویس برای تغییر مسیر خروجی استفاده کنید، بعنوان مثال، برای یک مسیر دیگر. برای استفاده از آن در BX، دستگیره اصلی و در CX دومین دستگیره را قرار دهید. یک عملیات موفق پرچم نقلی را صفر می‌کند. بروز خطا پرچم نقلی را یک می‌کند و کد خطای 04 یا 06 را در AX باز می‌گرداند برخی ترکیبات ممکن است عمل نکند، بعنوان مثال، دستگیره 00 همیشه ورودی صفحه کلید است، 04 خروجی چاپگر و 03 (auxiliary) که نمی‌تواند تعیین شود. (تابع 45 H را نیز ببینید).

تابع 4 EH : یافتن اولین فایل موافق

می‌توانید از تابع 4EH برای شروع یک جستجو در شاخه جهت اولین (احتمالاً) فایل‌های مرتبط و استفاده از تابع 4FH برای ادامه جستجو جهت فایل‌های بعدی از گروه استفاده نمایید. باید یک بافر ۴۳ بایتی برای عملیات جهت بازگشت ورودی موقعیت شاخه استفاده کنید و یک تابع 1AH (تنظیم DTA) قبل از استفاده از این سرویس، صادر نمایید. برای آغاز جستجو، CX را با صفت فایل از نام فایلی (فایلهایی) که بازگردانده می‌شود تنظیم کنید - هر ترکیبی از فقط خواندنی (بیت ۰)، پنهان (بیت ۱)، سیستم (بیت ۲)، بر چسب volume (بیت ۳)، شاخه (بیت ۴) یا آرشیو (بیت ۵) می‌تواندباشد. در DX آدرس‌رشته ASCIIZ حاوی مشخصه‌فایل، رشته‌ای که ممکن‌است (و احتمالاً خواهدبود) حاوی کاراکترهای‌جایگزین ? و * باشد. برای‌مثال، یک درخواست برای‌مشخصه‌فایل E:\ASMPROGS\A12*.ASM سبب می‌شود تا عملیات برای جستجوی اولین فایلی که بااین رشته جور باشد آغاز شود. در اینجا یک مثال آورده شده است :

```
DTA name   DB   DUP (?)
ASCstring  DB   'ASCIIZ string',00 H
                MOV AH,1AH              ; درخواست تنظیم DTA
                LEA  DX,DTA name        ; ناحیه DTA (DS:DX)
                INT  21H                ; فراخوانی سرویس وقفه
                MOV AH,4EH              ; درخواست اولین مورد موافق
                MOV CX,00H              ; صفت معمولی
                LEA  DX,Ascstrng        ; رشته ASCIIZ (DS:DX)
                INT  21H                ; فراخوانی سرویس وقفه
```

یک عملیات که مورد موافق را می‌یابد پرچم نقلی را پاک می‌کند و ۴۳ بایت DTA (2BH) را مثل زیر پر می‌کند.

```
FILEDTA     LABEL   BYTE            ;File DTA:
            DB      21 DUP(20H)     ; Reserved for subsequent search
FILATTR     DB      0               ; File attribute
FILTIME     DW      0               ; File time
FILDATE     DW      0               ; File date
LOSIZE      DW      0               ; File size: low word
HISIZE      DW      0               ; File size: high word
FILNAME     DB      13 DUP(20H)     ; Name and extension as an ASCIIZ
                                      string, followed by hex 00
```

بروز خطا سبب تنظیم پرچم نقلی و بازگشت کد ۰۲، ۰۳، یا 12H خواهد شد. اگر می‌خواهید از تابع 4FH بعداً استفاده نمایید محتویات DTA را تغییر ندهید.

یک مورد استفاده بی‌نظیر از تابع 4EH تعیین این است که ارجاع به یک نام فایل است یا یک زیرشاخه. برای مثال، اگر صفت بازگشتی 10H باشد، ارجاع به یک زیر شاخه است. عملیات اندازه فایل را باز می‌گرداند، در شکل ۲-۱۰ مشخص خواهد شد، ممکن است از تابع 4EH جهت تعیین اندازه فایل و تابع 36H جهت بررسی فضای مورد دسترس برای نوشتن استفاده نمود. این عملیات با تابع مطلق 11H جایگزین شده است.

تابع 4FH : یافتن فایل موافق بعدی

قبل از استفاده از این سرویس، یک 4EH صادر کنید تا جستجو در شاخه آغاز گردد و سپس یک تابع 4FH صادر کنید تا جستجو ادامه یابد:

```
درخواست مورد موافق بعدی ;   MOV     AH,4FH
فراخوانی سرویس وقفه ;       INT     21H
```

یک عملیات موفق پرچم نقلی را صفر می‌کند و در AX کدهای 00 (نام فایل پیدا شد) یا 18 (فایل بیشتری نیست) باز می‌گردند. بروز خطا سبب تنظیم پرچم نقلی و بازگرداندن کد ۰۲، ۰۳، یا 12H در AX خواهد شد. این عملیات با تابع 12H مطلق جدید جایگزین می‌شود. شکل ۳-۱۸ اخیر تابع 4EH و 4FH را مشخص می‌سازد.

تابع 56H : نام گذاری مجدد فایل یا شاخه

این سرویس می‌تواند یک فایل یا شاخه را از داخل یک برنامه مجدداً نامگذاری کند. برای استفاده از آن، در DX آدرس رشته ASCIIZ که حاوی درایو قبلی، نام فایل یا شاخه‌ای که باید مجدداً نام‌گذاری گردد، قرار دهید. در DI (بصورت ES:DI ترکیب می‌شود) آدرس یک رشته ASCIIZ حاوی درایو جدید، مسیر و نام، بدون هیچ کاراکتر جایگزین قرار دهید. شماره درایو، اگر استفاده شد، در هر دو رشته باید یکی باشد. چون مسیرها نیازی نیست یکسان باشند، عملیات علاوه بر نام‌گذاری مجدد، فایل آن را به شاخه دیگر در درایو نامبرده منتقل می‌نماید.

```
oldstrng    DB      'd:\oldpath\old name' , 00H
newstrng    DB      'd:\newpath\newname' , 00H
            . . .
            MOV     AH,56H      درخواست نام‌گذاری مجدد فایل/شاخه ;
            LEA     DX,oldstnng                          ; DS:DX
            LEA     DI,newstrng                          ; ES:DI
            INT     21H                  فراخوانی سرویس وقفه ;
```

یک عملیات موفق پرچم نقلی را صفر می‌کند، یک حالت خطا پرچم نقلی را یک می‌کند و در AX کد ۰۲، ۰۳، ۰۵، یا 11H باز می‌گردند.

تابع 57 H : گرفتن / تنظیم یک تاریخ فایل و زمان

این سرویس برنامه می‌تواند یک تاریخ و زمان را از یک فایل باز گرداند یا تنظیم کند، قالب‌های این زمان و تاریخ در شاخه مانند موارد زیر هستند:

بیت‌ها برای زمان	بیت‌ها برای تاریخ
0FH-0BH ساعت	0FH-09H سال (در ارتباط با ۱۹۸۰)
0AH-05H دقیقه	08H-05H ماه
04H-00H ثانیه	04H-00H روز ماه

ثانیه‌ها بطریقی هستند که عدد ۲ ثانیه اضافه می‌شود، ۰-۲۹ در خواست را در AL (۰ = گرفتن یا تنظیم) و دستگیره فایل را در BX قرار دهید. برای درخواست تنظیم، زمان را در CX و تاریخ را در DX قرار دهید. در زیر یک مثال آورده شده است:

```
MOV  AH,57 H      ; درخواست تنظیم
MOV  AL,01        ; تاریخ فایل و زمان
MOV  BX,handle    ; دستگیره فایل
MOV  CX,time      ; زمان جدید
MOV  DX,date      ; تاریخ جدید
INT  21H          ; فراخوانی سرویس وقفه
```

یک عملیات معتبر پرچم نقلی را پاک می‌کند، عملیات گرفتن زمان را در CX و تاریخ را در DX باز می‌گرداند، در حالیکه یک عملیات تنظیم ورودیهای تاریخ و زمان را برای فایل تغییر می‌دهد. یک عملیات نامعتبر پرچم نقلی را تنظیم و کد خطای ۰۱ یا ۰۶ را در AX باز می‌گرداند.

تابع 5AH از وقفه 21H: ایجاد یک فایل کمکی

این سرویس، برای برنامه‌ای که یک فایل کمکی ایجاد می‌کند مفید است، مخصوصاً در شبکه ، جایی که نامهای دیگر فایل‌ها ناشناخته است و برنامه از بازنویسی تصادفی بر روی آنها اجتناب می‌کند. عملیات یک فایل با نام منحصر بفرد داخل مسیر ایجاد می‌کند.

برای استفاده از این سرویس، در CX صفت فایل مورد نیاز را قرار دهید. با هر ترکیبی از فقط خواندنی (بیت ۰) پنهان (بیت ۱)، سیستم (بیت ۲)، برچسب volume (بیت ۳)، شاخه (بیت ۴)، آرشیو (بیت ۵). در DX آدرس یک مسیر ASCIIZ درایو اگر لازم است، زیر شاخه (اگر هست)، یک \ و 00H و بعد از آن 13 بایت جای خالی برای نام فایل جدید، قرار دهید.

```
ASCpath  DB   'd:\path name\' , 00H,13DUP (20H)
. . .
         MOV  AH,5AH       ; درخواست ایجاد فایل
         MOV  CX,attribute ; صفت فایل
         LEA  DX,ASC path  ; مسیر ASCIIZ
         INT  21H          ; فراخوانی سرویس وقفه
```

یک عملیات موفق پرچم نقلی را پاک می‌کند، دستگیره فایل را در AX ارائه می‌دهد، و نام فایل جدید را در رشته ASCIIZ با شروع در بایت 00H می‌افزاید یک عملیات ناموفق پرچم نقلی را تنظیم می‌کند و کد ۰۳، ۰۴، یا ۰۵ را در AX باز می‌گرداند.

تابع 5BH : ایجاد یک فایل جدید

این سرویس فقط در صورتیکه فایل نامبرده وجود نداشته باشد، یک فایل ایجاد می‌کند، و از دیگر جهات مانند تابع

3CH (ایجاد فایل) است. شما می‌توانید از تابع 5BH در هر جا که نمی‌خواهید روی فایل موجود نوشته شود استفاده کنید. یک عملیات معتبر پرچم نقلی را پاک می‌کند و دستگیره فایل را در AX باز می‌گرداند. یک عملیات نامعتبر (شامل یافتن یک نام فایل مساوی) پرچم نقلی را تنظیم می‌کند و کد ۰۳، ۰۴ یا ۰۵ را در AX باز می‌گرداند.

برنامه: حذف انتخابی فایلها

برنامه شکل ۳-۱۸ نحوه استفاده از توابع 4EH و 4FH جهت یافتن همه نام‌های فایل در شاخه و تابع 41H جهت حذف فایل انتخاب شده استفاده می‌کند. این برنامه، درایو را :F فرض می‌کند که می‌توانید آن را تغییر دهید و شامل روال‌های زیر می‌باشد:

- A10MAIN روال B10FIRS، C10NEXT، D10DISP، E10DELET را فرا می‌خواند.
- B10FIRST ، DTA را برای تابع 4EH تنظیم می‌کند و اولین ورودی موافق در شاخه را می‌یابد.
- C10NEXT ، ورودیهای موافق بعدی در شاخه را می‌یابد.
- D10DISPI نام‌ها فایل‌ها را نمایش می‌دهد و می‌پرسد آیا حذف شود.
- EIODELET یک پاسخ Y (بله) برای حذف فایل، N (نه) برای نگه داشتن آن می‌پذیرد، یا <Enter> تا پردازش خاتمه یابد و فایل درخواستی حذف شود.

در طی بررسی جهت احتیاط، در فایل کمکی کپی کنید.

```
TITLE     A18SELDL (COM)  Select and delete files
CODESG    SEGMENT PARA 'Code'
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     A10MAIN
; ---------------------------------------------------------
TAB       EQU     09
LF        EQU     10
CR        EQU     13
CRLF      DB      CR, LF
PATHNAM   DB      'F:\*.*', 00H
DELMSG    DB      TAB, 'Delete '
ENDMSG    DB      CR, LF, 'No more directory entries', CR, LF
ERRMSG1   DB      'Invalid path/file'
ERRMSG2   DB      'Write-protected disk'
PROMPT    DB      'Y = Delete, N = Keep, Ent = Exit', CR, LF
DISKAREA  DB      43 DUP(20H)
; ---------------------------------------------------------
A10MAIN   PROC    NEAR                ;Main procedure
          CALL    Q10SCRN             ;Clear screen
          CALL    Q20CURS             ;Set cursor
          CALL    B10FIRST            ;  directory entry
          CMP     AX,00H              ;If no entries,
          JNE     A90                 ;  exit
          MOV     CX,34               ;Length of prompt
          LEA     DX,PROMPT           ;Display initial prompt
          CALL    Q30LINE
A20:
          CALL    D10DISPL            ;Display filename
          CALL    E10DELET            ;Delete if requested
          CMP     AL,0FFH             ;Request for finish?
          JE      A90                 ;  yes, exit
          MOV     CX,02               ;Length of data
          LEA     DX,CRLF             ;Set cursor on
          CALL    Q30LINE             ;  next line
```

شکل الف ۳-۱۸ حذف انتخابی فایل‌ها

```
            CALL    C10NEXT             ;Get next directory entry
            CMP     AX,00H              ;Any more entries?
            JE      A20                 ;  yes, loop
A90:        MOV     AX,4C00H            ;End processing
            INT     21H
A10MAIN     ENDP
;                   Find first entry in directory:
;                   ------------------------------
B10FIRST    PROC    NEAR
            MOV     AH,1AH              ;Get DTA for function
            LEA     DX,DISKAREA         ;  calls
            INT     21H
            MOV     AH,4EH              ;Locate first directory
            MOV     CX,00               ;  entry
            LEA     DX,PATHNAM          ;Address of ASCIIZ string
            INT     21H
            JNC     B90                 ;Valid operation?
            PUSH    AX                  ;  no,
            MOV     CX,17               ;  display ending
            LEA     DX,ERRMSG1          ;  message
            CALL    Q30LINE             ;
            POP     AX
B90:        RET
B10FIRST    ENDP
;                   Find succeeding entries in directory:
;                   -------------------------------------
C10NEXT     PROC    NEAR                ;Read directory entry
            MOV     AH,4FH              ;Get next
            INT     21H
            CMP     AX,00H              ;More entries?
            JE      C90                 ;  yes, bypass
            PUSH    AX                  ;  no,
            MOV     CX,29               ;  display ending
            LEA     DX,ENDMSG           ;  message
            CALL    Q30LINE             ;
            POP     AX
C90:        RET
C10NEXT     ENDP
;                   Display message and filename:
;                   -----------------------------
D10DISPL    PROC    NEAR
            MOV     CX,08               ;Length of message
            LEA     DX,DELMSG           ;Display delete message
            CALL    Q30LINE
            LEA     SI,DISKAREA+1EH     ;Start of filename
D30:        MOV     DL,[SI]             ;Get char for display
            CALL    Q40CHAR
            INC     SI                  ;Next character
            CMP     BYTE PTR [SI],00H   ;Hex zero stopper?
            JNE     D30                 ;  no, get next char
            MOV     DL,'?'              ;  yes, exit
            CALL    Q40CHAR
            RET
D10DISPL    ENDP
;                   Delete record if requested:
;                   ---------------------------
E10DELET    PROC    NEAR
            MOV     AH,10H              ;Accept 1-character
            INT     16H                 ;  reply (y/n)
            CMP     AL,0DH              ;Enter character?
            JE      E50                 ;  yes, exit
            OR      AL,00100000B        ;Force lowercase
            CMP     AL,'y'              ;Delete requested?
            JNE     E90                 ;  no, bypass
            MOV     AH,41H              ;  yes,
```

شکل ب ۳-۱۸ حذف انتخابی فایل‌ها

```
                LEA     DX,DISKAREA+1EH     ;  address of filename
                INT     21H                 ;  delete entry
                JNC     E90                 ;Valid delete?
                MOV     CX,20               ;  no, display
                LEA     DX,ERRMSG2          ;  warning message
                CALL    Q30LINE             ;
E50:            MOV     AL,0FFH             ;End-of-process indicator
E90:            RET
E10DELET        ENDP
;                       Screen functions:
;                       -----------------
Q10SCRN         PROC    NEAR
                MOV     AX,0600H            ;Request clear screen
                MOV     BH,1EH              ;Set attribute
                MOV     CX,00
                MOV     DX,184FH
                INT     10H
                RET
Q10SCRN         ENDP
Q20CURS         PROC    NEAR
                MOV     AH,02H              ;Request
                MOV     BH,00               ;  set cursor
                MOV     DH,00               ;Row 0
                MOV     DL,10               ;Column 10
                INT     10H
```

شکل پ ۳-۱۸ حذف انتخابی فایل‌ها

نکات کلیدی

- عملیاتی که به منظور دستیابی دیسک درایوها می‌باشند شامل، تنظیم مجدد، انتخاب پیش‌فرض ، گرفتن اطلاعات درایو، گرفتن فضای خالی دیسک، و عملیات سنگینتر کنترل I/O ابزارهاست.
- عملیاتی که به منظور دستیابی شاخه‌ها و FAT می‌باشند شامل ایجاد شاخه، حذف زیر شاخه، تغییر شاخه جاری، و گرفتن شاخه جاری می‌باشد.
- عملیاتی که به منظور دستیابی فایل‌های دیسک می‌باشد (بجز ایجاد، باز کردن، خواندن و نوشتن) شامل نام‌گذاری مجدد فایل، گرفتن / تنظیم صفت، یافتن فایل موافق، و گرفتن / تنظیم، تاریخ / زمان می‌باشد.

پرسش‌ها

برای این پرسش‌ها از DEBUG استفاده کنید. فرمان A 100 را وارد کنید و دستورات اسمبلر مورد نیاز را بنویسید. هر مقدار بازگشتی در ثبات‌ها را بررسی کنید.

۱-۱۸. پرسشهای زیر شامل دیسک درایوها است:

(الف) تابع 19H برای تعیین دیسک درایو پیش فرض.

(ب) تابع 1 BH برای اطلاعات راجع به دیسک درایو پیش فرض.

(پ) تابع 1 FH برای اطلاعات راجع به DPB پیش فرض.

(ت) تابع 36H برای تعیین میزان فضای آزاد دیسک.

(ث) تابع 4400H برای گرفتن اطلاعات ابزار مورد استفاده.

(ج) تابع 4408H برای تعیین اینکه واسطه مورد استفاده قابل انتقال است.

(چ) تابع 440DH کد فرعی 60H برای گرفتن پارامترهای ابزار.

(ح) تابع 440DH کد فرعی 66H برای گرفتن ID واسطه.

۲-۱۸. پرسش‌های زیر مستلزم شاخه‌هاست:

(الف) تابع 39H برای ایجاد یک زیر شاخه برای سرعت بیشتر، می‌توانید آن را روی یک دیسک RAM یا دیسکت ایجاد کنید. از هر نامی استفاده کنید.

(ب) تابع 56H برای نام‌گذاری مجدد زیر شاخه.

(پ) تابع 3AH برای انتقال زیر شاخه.

۱۸-۳. پرسش‌های زیر مستلزم فایل‌های دیسک است (یک فایل کپی برای این تمرین استفاده کنید) :

(الف) تابع 43H برای گرفتن صفت از یک فایل روی دیسکت.

(ب) تابع 56H برای نام‌گذاری مجدد فایل.

(پ) تابع 43H برای تنظیم صفت پنهان.

(ت) تابع 57H برای گرفتن تاریخ و زمان فایل

(ث) تابع 41H برای حذف فایل.

۱۸-۴ یک برنامه کوچک از داخل DEBUG بنویسید که خیلی ساده تابع 29H از وقفه 21H را اجرا کند (تشریح نام فایل) مشخصه فایل در 81H و FCB در 5CH فراهم شده‌اند، اما هر دو در PSP بلافاصله قبل از برنامه هستند. مشخصه‌های فایل مختلف، مانند PROG2 D:PROG1.DOC ، *.PROG3 , ASM . *:C وارد کنید. نتیجه را در افست 5CH بعد از هر اجرا بررسی کنید.

فصل ۱۹

حافظه دیسک ۴ : توابع وسایل چاپ کردن INT 13H

هدف: بررسی موارد ضروری برنامه نویسی پایه برای توابع BIOS INT 13H جهت فرمت، تشخیص، خواندن نوشتن دیسک‌ها.

مقدمه

در فصل ۱۷ و ۱۸ ، ما سرویس‌های INT 21H برای پردازش دیسک را بررسی نمودیم می‌توان عملیات دیسک را مستقیماً در سطح BIOS پردازش نمود، در عملیات دیسک BIOS INT 13H داده‌ها معادل یک سکتور می‌باشند و دستیابی به آدرسهای دیسک بر اساس تراک واقعی و شماره‌های سکتور انجام می‌شود. عملیات دیسک INT 13H مستلزم تنظیم مجدد خواندن، نوشتن، تشخیص و فرمت کردن درایو است. اغلب توابع INT 13H برای توسعه دهندگان با تجربه نرم‌افزار می‌باشد که از خطرات ذاتی استعمال غلط آن آگاهند. همچنین، نسخه‌های BIOS، بر طبق پردازشگر مورد استفاده یا حتی مدل کامپیوتر متفاوت می‌باشند. این فصل توابع INT 13H زیر را معرفی می‌کند:

00H تنظیم مجدد دیسک/ دیسکت سیستم
01H خواندن وضعیت دیسکت
02H خواندن سکتورها
03H نوشتن سکتورها
04H تشخیص سکتورها
05H فرمت تراک‌ها
08H گرفتن پارامترهای درایو
09H مقدار دهی اولیه درایو
0AH خواندن بافر سکتور توسعه یافته
0BH نوشتن بافر سکتور توسعه یافته
0CH گشتن سیلندر
0DH تنظیم بنوبت دیسک
0EH خواندن بافر سکتور
0FH نوشتن بافر سکتور
15H گرفتن نوع دیسک/ دیسکت
16H تغییر وضعیت دیسکت
17H تنظیم نوع دیسکت
18H تنظیم نوع واسطه برای فرمت
19H متوقف کردن هدهای دیسک

بایت وضعیت BIOS

اغلب توابع INT 13H در صورت موفقیت پرچم نقلی را صفر و در صورت عدم موفقیت پرچم نقلی را یک می‌کنند و یک کد وضعیت به ثبات AH باز می‌گردانند. BIOS اطلاعاتی راجع به هر ابزار و وضعیت آن در ناحیه داده خود نگه می‌دارد. ***بایت وضعیت*** در شکل ۱-۱۹ نشان داده شده است که بازتابی از بیت‌های ظاهر شده در ناحیه داده BIOS در 40:41H برای ناحیه داده دیسکت درایو و در 40:74H برای ناحیه داده دیسکت سخت می‌باشد (فصل ۲۵ را برای جزئیات بیشتر ملاحظه نمایید).

در صورت بروز خطا در عملیات دیسک، یک اقدام مشخص برنامه ، تنظیم مجدد دیسک (تابع 00H) و سه مرتبه سعی مجدد در انجام عملیات می‌باشد. اگر هنوز خطائی وجود دارد، برنامه یک پیغام را نمایش می‌دهد و بعنوان یک راه حل مشکل ، برای کاربر امکان تغییر دیسکت را فراهم می‌آورد.

Code	Status
00H	No error
01H	Bad command, not recognized by the controller
02H	Address mark on disk not found
03H	Writing on protected disk attempted
04H	Invalid track/sector
05H	Reset operation failed
06H	Diskette removed since last access
07H	Drive parameters wrong
08H	Direct memory access (DMA) overrun (data accessed too fast to enter)
09H	DMA across a 64K boundary attempted on read/write
10H	Bad CRC on a read encountered (error check indicated corrupted data)
20H	Controller failed (hardware failure)
40H	Seek operation failed (hardware failure)
80H	Device failed to respond (diskette: drive door open or no diskette; hard disk: time out)
AAH	Drive not ready
BBH	Undefined error
CCH	Write fault

شکل ۱-۱۹ کدهای وضعیت *INT 13H*

عملیات دیسک INT 13H پایه

این بخش، عملیات دیسک INT 13H پایه را در بر دارد، که هر کدام به یک کد تابع در ثبات AH نیاز دارد.

تابع 00H از وقفه 13H : تنظیم مجدد دیسک سیستم

از این عملیات وقتی استفاده کنید که عملیات قبلی دیسک، خطای جدی را گزارش داده است. عملیات یک تنظیم مجدد سخت‌افزاری روی کنترلر دیسکت یا دیسک سخت انجام می‌دهد. بدین ترتیب، دفعه بعد که درایو دستیابی می‌شود، عملیات سیستمی در سیلندر 0 تنظیم شده است. برای دیسکت، DL را با شماره درایو (0 = درایو A و غیره) برای دیسک سخت DL را با مقدار 80H یا بالاتر (80H = اولین درایو، 81H = دومین و غیره) تنظیم کنید یک مثال با استفاده از تابع 00H به صورت زیر می‌باشد :

```
MOV   AH,00H    ; درخواست تنظیم دیسکت
MOV   DL,80H    ; دیسک سخت
INT   13H       ; فراخوانی سرویس وقفه
```

یک عملیات مقصد پرچم نقلی را صفر می‌کند و با بروز خطا پرچم نقلی یک می‌شود و یک کد وضعیت 00H در AH باز گردانده می‌شود.

تابع 01H از وقفه 13H : خواندن وضعیت دیسک

این عملیات به شما شانس دیگری جهت بررسی وضعیت آخرین عملیات دیسک انجام شده می‌دهد. (بایت وضعیت در شکل ۱-۱۹ را ببینید). DL را با کد معمول (0 = درایو A و غیره) برای دیسکت و یک مقدار 80H یا بیشتر برای دیسک سخت تنظیم کنید. این عملیات در AL کد وضعیت را باز می‌گرداند که آخرین عملیات دیسک می‌تواند در AH بازگردانده شود. عملیات که همیشه معتبراست پرچم نقلی را صفر می‌کند و کد وضعیت 00H در AL باز می‌گرداند.

تابع 02H از وقفه 13H : خواندن سکتورهای دیسک

این عملیات تعداد خاصی از سکتورها در روی تراک را خوانده و در حافظه قرار می‌دهد. برای استفاده از آن ثبات‌های زیر را مقداردهی کنید:

AL تعداد سکتورها، حداکثر تا تعداد یک تراک
CH تعداد تراک / سیلندر (تعداد با صفر شروع می‌شود)
CL بیت ۶-۷ تعداد تراک/سیلندر (دوبیت بالارتبه) بیت ۰-۵ شماره سکتورشروع (اعداد با یک شروع‌می‌شود)
DH شماره هد / طرف (۰ یا ۱ برای دیسکت)
DL شماره درایو برای دیسکت (A = 0) یا دیسک سخت (80H یا بالاتر)
ES:BX آدرس یک بافر در ناحیه داده که باید برای همه سکتورهایی که خوانده می‌شود بقدر کافی بزرگ باشد. (BX در این حالت با ES می‌باشد)

مثال زیر، یک سکتور را خوانده در یک ناحیه با نام INSECT قرار می‌دهد:

```
INSECT DB    512 DUP (?)          ناحیه ورودی ;
       ...
       MOV   AH,02H       درخواست خواندن سکتور ;
       MOV   AL,01        یک سکتور ;
       LEA   BX,INSECT    بافر ورودی (ES:BX) ;
       MOV   CH,05        تراک ۵ ;
       MOV   CL,03        سکتور ۳ ;
       MOV   DH,00        هد صفر ;
       MOV   DL,03        درایو 03 (D) ;
       INT   13H          فراخوانی سرویس وقفه ;
```

یک عملیات معتبر پرچم نقلی را صفر می‌کند و در AL تعداد سکتورهایی که عملیات باید بخواند قرار می‌دهد. محتویات ثبات‌های DS ، BX ، CX و DX قبلاً مشخص شده‌اند. بروز خطا سبب تنظیم پرچم نقلی و بازگشت کد وضعیت در AH می‌گردد، درایو را تنظیم کنید (تابع 00H) و مجدداً عملیات را انجام دهید.

در اغلب حالات، شما فقط یک سکتور یا همه سکتورهای دیسک را مشخص می‌سازید. CH و CL را مقدار دهی کنید و آنها را برای خواندن ترتیبی سکتورها بیفزایید. وقتی شماره سکتور از حداکثر یک تراک تجاوز نماید، باید آنها را با 01 مجدداً تنظیم کنید و همچنین شماره تراک روی همان طرف از دیسک با شماره هد برای طرف بعدی را بیفزایید.

بررسی آمادگی یک دیسکت

یک برنامه ممکن است درخواست دستیابی به یک دیسکت که هنوز آماده نیست را صادر نماید. یک تجربه، استاندارد سه بار تلاش برای انجام عملیات قبل از نمایش پیغام برای کاربر است. مثالی که در زیر خواهد آمد از تابع 02H وقفهٔ 21H برای خواندن یک سکتور داده استفاده می‌کند. سعی کنید تا در DEBUG برای وارد کردن دستورات و بررسی کد، با و بدون حضور دیسکت در درایو A: استفاده نمایید. برای دیسکت نصب شده عملیات باید محتویات رکورد راه انداز دیسک ۵۱۲ بایت (200H) با شروع از DS:200H را بخواند.

```
0100   MOV CX,03        ;Count for loop
0103   PUSH CX          ;Save count
0104   MOV AX,0201      ;Request read one sector
0107   MOV BX,0200      ;Input address
010A   MOV CX,0001      ;Track and sector numbers
010D   MOV DX,0000      ;Head and drive numbers
0110   INT 13           ;Call interrupt service
0112   POP CX           ;Restore count
0113   JNC 118          ;If no error, exit
0115   CLC              ;If error,
0116   LOOP 103         ;  try 3 times
0118   NOP              ;That's it
```

تابع 03H از وقفه 13H : نوشتن سکتورها

این عملیات مخالف تابع 02H است، یک ناحیه خاص از حافظه (۵۱۲ بایت یا مضربی از ۵۱۲) را بر روی سکتور مشخص فرمت شده می‌نویسد، برای استفاده از آن، ثبات‌ها را بارگذاری کنید و پردازش را مانند تابع 02H دستکاری نمایید. یک عملیات معتبر پرچم نقلی را صفر می‌کند و در AL تعداد سکتورهایی که نوشته شده ارائه می‌دهد، محتویات ثبات‌های DS ، BX ، CX و DX قبلاً مقدار دهی شده‌اند. یک خطا پرچم نقلی را یک می‌کند و یک کد وضعیت در AH باز می‌گرداند، درایو را تنظیم کنید و عملیات را دوباره تکرار کنید.

```
TITLE     A19BIORD (EXE)  Read disk sectors via BIOS
          .MODEL  SMALL
          .STACK  64
; ---------------------------------------------------------
          .DATA
CURADR    DW      0301H             ;Beginning track-sector
ENDADR    DW      0401H             ;Ending track-sector
ENDCDE    DB      00                ;End process indicator
READMSG   DB      '*** Read error ***'
SECTIN    DB      512 DUP(' ')      ;Input area for sector
SIDE      DB      00
; ---------------------------------------------------------
          .CODE
A10MAIN   PROC    FAR
          MOV     AX,@data          ;Initialize
          MOV     DS,AX             ;  segment
          MOV     ES,AX             ;  registers
          MOV     AX,0600H          ;Request scroll
A20LOOP:
          CALL    Q10SCRN           ;Clear screen
          CALL    Q20CURS           ;Set cursor
          CALL    B10ADDR           ;Calculate disk address
          MOV     CX,CURADR
          MOV     DX,ENDADR
          CMP     CX,DX             ;At ending sector?
          JE      A90               ;  yes, exit
          CALL    C10READ           ;Read disk record
          CMP     ENDCDE,00         ;Normal read?
          JNZ     A90               ;  no,  exit
          CALL    D10DISP           ;Display sector
          JMP     A20LOOP           ;Repeat
A90:      MOV     AX,4C00H
          INT     21H               ;End processing
A10MAIN   ENDP
;                   Calculate next disk address:
;                   ----------------------------
B10ADDR   PROC    NEAR
          MOV     CX,CURADR         ;Get track/sector
          CMP     CL,10             ;Past last sector?
          JNE     B90               ;  no, exit
          MOV     CL,01             ;Set sector to 1
          CMP     SIDE,00           ;Bypass if side 0
          JE      B20
          INC     CH                ;Increment track
B20:
          XOR     SIDE,01           ;Change side
          MOV     CURADR,CX
B90:      RET
B10ADDR   ENDP
;                   Read disk sector:
;                   -----------------
C10READ   PROC    NEAR
          MOV     AH,02H            ;Request read
          MOV     AL,01             ;Number of sectors
          LEA     BX,SECTIN         ;Address of buffer
          MOV     CX,CURADR         ;Track/sector
          MOV     DH,SIDE           ;Side
          MOV     DL,00             ;Drive A
          INT     13H
```

شکل الف ۲-۱۹ استفاده از *INT 13H* برای خواندن سکتورهای دیسک

```
            CMP     AH,00           ;Normal read?
            JZ      C90             ;Yes, exit
            MOV     ENDCDE,01       ;No,
            CALL    X10ERR          ;  invalid read
C90:
            INC     CURADR          ;Increment sector
            RET
C10READ ENDP
;                   Display sector:
;                   --------------
D10DISP     PROC    NEAR
            MOV     AH,40H          ;Request display
            MOV     BX,01           ;Handle
            MOV     CX,512          ;Length
            LEA     DX,SECTIN       ;Address of input
            INT     21H
            MOV     AH,10H          ;Wait for keyboard
            INT     16H             ;  entry
            RET
D10DISP     ENDP
;                   Clear screen:
;                   ------------
Q10SCRN     PROC    NEAR
            MOV     AX,0600H        ;Request scroll
            MOV     BH,1EH          ;Set attribute
            MOV     CX,0000         ;Full screen
            MOV     DX,184FH
            INT     10H
            RET
Q10SCRN     ENDP
;                   Set cursor:
;                   ----------
Q20CURS     PROC    NEAR
            MOV     AH,02H          ;Request set
            MOV     BH,00           ;  cursor
            MOV     DX,0000
            INT     10H
            RET
Q20CURS     ENDP
;                   Display disk error message:
;                   --------------------------
X10ERR      PROC    NEAR
            MOV     AH,40H          ;Request display
            MOV     BX,01           ;Handle
            MOV     CX,18           ;Length of message
            LEA     DX,READMSG
            INT     21H
            RET
X10ERR      ENDP
            END     A10MAIN
```

شکل ب ۲-۱۹ استفاده از *INT 13H* برای خواندن سکتورهای دیسک

برنامه: استفاده از INT 13H برای خواندن سکتورها

برنامه شکل الف ۲-۱۹ از INT 13H برای خواندن سکتورها از دیسک به حافظه استفاده می‌کند، توجه کنید که هیچ عملیات باز یا دستگیره فایل وجود ندارد. ناحیه داده اصلی چنین است :

CURADR حاوی تراک شروع (۰۳) و سکتور (۰۱) که برنامه خواهد افزود.

ENDADR حاوی تراک انتها (۰۴) و سکتور (۰۱). مجموع تعداد سکتورها ۹ (برای تراک ۳)×۲ (برای ۲ طرف) = ۱۸. یک راه توسعه برنامه اعلان به کاربر برای وارد کردن تراک و سکتور شروع و انتها است.

SECTIN تعریف ۵۱۲ بایت بعنوان ناحیه برای خواندن یک سکتور. روال‌های اصلی چنین می‌باشد:

A10MAIN B10ADDR ، C10READ و D10DISP را فرا می‌خواند، و بعد از خواندن آخرین سکتور درخواستی خاتمه می‌یابد.

B10ADDR هر آدرس دیسک با طرف، تراک و سکتور محاسبه شود. وقتی سکتور به شماره 10 رسید، روتین سکتور را با 01 تنظیم می‌کند. اگر طرف یک باشد ، برنامه شماره تراک را می‌افزاید، سپس شماره طرف تغییر می‌کند از 0 به 1 یا از 1 به 0. این پردازش فقط برای دیسکت‌ها کار می‌کند (چون دو طرف دارد) که حاوی ۹ سکتور در هر تراک می‌باشد.

C10READ یک سکتور دیسک را از درایو A می‌خواند، (که ممکن است شما آن را تغییر دهید) در SECTIN قرار می‌دهد و شماره سکتور را به یک عملیات خواندن معتبر می‌افزاید.

D10DISP محتویات سکتور خوانده شده جاری را نمایش می‌دهد. نمایش بر روی CR ، Tab و غیره فعال است و روال برای کاربر منتظر می‌شد و تا یک کلید را قبل از ادامه بفشارد.

سعی کنید که این برنامه را تحت DEBUG اجرا کنید. دستوراتی را که ثبات‌های سگمنت را مقدار می‌دهند پیگیری کنید. برای عملیات ورودی، سکتورهای شروع و پایان را با موقعیت FAT از دیسک تنظیم کنید.(فصل ۱۶ را ببینید) با استفاده G (GO) برنامه را اجرا کنید و FAT ورودیهای شاخه در SECTIN را بررسی کنید.

برنامه شما می‌تواند کاراکترهای ASCII در ناحیه ورودی را به معادل شانزدهی آن تبدیل کند و مقادیر شانزدهی را همانطور که DEBUG عمل می‌کند، نمایش می‌دهد. (برنامه شکل ۶-۱۵ را نیز ببینید.) با این روش می‌توانید محتویات هر سکتور را بررسی کنید - حتی فایل‌های پنهان- و می‌توانید و به کاربر امکان ورود تغییرات و نوشتن سکتورهای تغییر یافته را روی دیسک را بدهید.

توجه کنید که وقتی INT 21H یک فایل را ایجاد می‌کند، این رکوردها را در کلاسترهای در دسترس جایگزین می‌سازد، که ممکن در دیسک اتصالی نداشته باشند. به این دلیل نمی‌توانید از INT 13H انتظار داشته باشید که فایل را بطور ترتیبی بخواند، اگر چه می‌توانید ورودیهای FAT را برای موقعیت کلاستر بعدی دستیابی نمایید.

دیگر عملیات دیسک INT 13H

در زیر سرویس‌های دیسک INT 13H اضافی را توصیف می‌کند.

تابع 04H از وقفه 21H : تشخیص سکتورها

این عملیات سکتورهای خاص که می‌تواند خوانده شود را بررسی می‌کند و یک بررسی CRC[1] را انجام می‌دهد. وقتی یک عملیات در یک سکتور می‌نویسد ،کنترلر دیسک محاسبه می‌کند و یک حاصل تطبیق CRC را بلافاصله پس از سکتور می‌نویسد، بر مبنای بیت‌هایی که شما تنظیم کرده‌اید می‌توانید از تابع 04H برای خواندن سکتور، محاسبه مجدد حاصل تطبیق و مقایسه آن با مقادیر ذخیره شده استفاده نمایید. توجه کنید که تشخیص شامل محاسبه مجدد حاصل تطبیق است تا بررسی موافق بودن مقادیر بایت در سکتور با داده خروجی در حافظه، شما می‌توانید از این تابع بعد از نوشتن (تابع 03H) برای اطمینان از قرار گرفتن خروجی استفاده کنید، اگر چه زمان بیشتری برای پردازش نیاز دارد.

ثبات‌ها را مانند تابع 02H بارگذاری نمایید، اما چون عملیات تشخیص صحیحی از داده نوشته شده انجام نمی‌دهد، احتیاجی به تنظیم آدرس در ES:BX نیست. یک عملیات موفق پرچم نقلی را صفر می‌کند و در AL شماره سکتورهایی که واقعاً تشخیص داده شده‌است را بازمی‌گرداند، محتویات DS ، BX ، CX و DX نگهداری می‌شود. بروز خطا سبب تنظیم پرچم نقلی و بازگشت یک کد وضعیت در AH می‌شود، درایو را مجدداً تنظیم کنید و عملیات را تکرار کنید.

تابع 05H از وقفه 13H : فرمت کردن تراک‌ها

عملیات خواندن / نوشتن به اطلاعاتی راجع به فرمت کردن موقعیت و پردازش سکتور خواسته شده نیاز دارد. این

1- cyclical redundency check

عملیات تراک‌ها را بر طبق یکی از ۴ اندازه مختلف فرمت می‌کند. قبل از اجرای عملیات، از تابع 17 H برای تنظیم نوع دیسکت و تابع 18H برای تنظیم نوع واسطه استفاده کنید. برای فرمت کردن دیسکت این ثبات‌ها را مقدار دهی کنید :

AL تعداد سکتورها برای فرمت.

CH شماره سیلندر را تراک (تعداد با صفر آغاز می‌شود).

DH شماره هد/ طرف (0 یا 1 برای دیسکت)

DL شماره درایو برای دیسکت (A = 0) یا درایو سخت (80H یا بالاتر)

ES:BX آدرس سگمنت: افست که به یک گروه از آدرس فیلدها برای یک تراک اشاره می‌کند. برای هر سکتور دیسکت روی یک تراک، باید یک ورودی ۴ بایتی به شکل T/H/S/B باشد، که :

بایت ۰ T = شماره سیلندر / تراک

۱ H = شماره هد / طرف

۲ S = شماره سکتور

۳ B = بایت‌های هر سکتور (00H = 128، 01H = 256، 02H = 512 و 03H = 1029)

برای مثال، اگر شما تراک ۳، هد ۰ و ۵۱۲ بایت در هر سکتور را فرمت کنید، اولین ورودی برای تراک 03000102 شانزدهی می‌باشد، که پس از آن یک ورودی برای هر سکتور باقیمانده وجود دارد.این عملیات پرچم نقلی را صفر می‌کند (اگر معتبر باشد) و یک می‌کند (اگر نامعتبر باشد) و کد وضعیت را در AH باز می‌گرداند.

تابع 08H از وقفه 21H: گرفتن پارامترهای دیسک درایو

این تابع مفید اطلاعاتی راجع به یک دیسک درایو باز می‌گرداند. برای استفاده از آن، شماره درایو را در B DL (A = 0 و B = 1 برای دیسکت و 80H یا بالاتر برای دیسک سخت) قرار دهید. یک عملیات موفق مقادیر زیر را باز می‌گرداند.

BL نوع دیسکت (01H = 360K ، 02H = 1.2M ، 03H = 720K ، 04H = 1.44M)

CH بیشترین شمار، سیلندر / تراک

CL بیت‌های ۰ - ۵ = بیشترین شماره سکتور بیت ۶ - ۷ = ۲ بیت بالا رتبه شماره سیلندر

DH بیشترین شماره هد

DL تعداد درایوهایی که به کنترلر وصل شده‌اند

ES:DI برای دیسکت، آدرس افست، سگمنت از یک جدول پارامترهای درایو دیسکت ۱۱ بایتی. دو فیلد مرتبط چنین هستند: افست ۳ بایت‌های هر سکتور را می‌دهد (00H = ۱۲۸، 01H = ۲۵۶، 02H = ۵۱۲، 03H = ۱۰۲۴) افست ۴ سکتورهای هر تراک را می‌دهد.

می‌توانید از فرمان افست : D ES از DEBUG استفاده کنید (افست در DI باز گردانده شده است) برای نمایش مقادیر استفاده کنید. عملیات پرچم نقلی را در صورتیکه معتبر باشد پاک می‌کند و در صورتیکه معتبر نباشد تنظیم می‌کند و کد وضعیت را در AH باز می‌گرداند.

تابع 09H از وقفه 21H: مقدار دهی درایو

BIOS این تابع را وقتی کامپیوتر را راه‌اندازی می‌کنید ، بر طبق جدول دیسک سخت، انجام می‌دهد. DL حاوی شماره درایو (80H یا بیشتر) می‌باشد. عملیات (اگر معتبر باشد) پرچم نقلی را صفر می‌کند (اگر معتبر نباشد) پرچم نقلی را یک می‌کند و وضعیت را در AH باز می‌گرداند. وقفه‌های 41H و 46H و BIOS مرتبطی هستند.

تابع 0AH از وقفه 13H: خواندن بافر سکتور توسعه یافته

بافر سکتور روی دیسک سخت شامل ۵۱۲ بایت داده بعلاوه ۴ بایت برای یک کد تصحیح خطا است (ECC) که برای بررسی خطا و تصحیح داده می‌باشد. این تابع می‌تواند بیشتر یک بافر سکتور کامل را بخواند، بخشی از داده، برای خواندن یک بافر توسعه داده، این ثبات را بارگذاری کنید :

AL تعداد سکتورها (حداکثر تا مقداری یک درایو)
ES:BX آدرس آفست: سگمنت برای بافر ورودی
CH شماره سیلندر / تراک
CL بیت‌های ۰ - ۵ = بیشترین عدد سکتور بیت‌های ۶ - ۷ = ۲ بیت بالا رتبه عدد سیلندر
DH عدد هد / طرف
DL شماره درایو (80H یا بالاتر)

یک عملیات موفق در AL تعداد سکتورهای انتقال یافته را باز می‌گرداند. عملیات در صورتیکه معتبر باشد پرچم نقلی را پاک می‌کند و در صورتیکه معتبر نباشد پرچم نقلی را تنظیم می‌کند و کد وضعیت را در AH باز می‌گرداند.

تابع 0BH از وقفه 13H: نوشتن بافر سکتور توسعه یافته

این تابع مشابه تابع 0AH است، بجز آنکه بافر سکتور را می‌خواند، و بر روی دیسک (شامل کد ECC) می‌باشد.

تابع 0CH از وقفه 13H: چرخش سیلندر دیسک

این تابع هد خواندن / نوشتن روی دیسک سخت را بر روی سیلندر (تراک) خاص تنظیم می‌کند، اما هیچ داده‌ای را منتقل نمی‌کند. برای چرخش سیلندر این ثبات‌ها را بارگذاری کنید.

CH شماره سیلندر / تراک
CL بیت ۰ - ۵ = شماره سکتور بیت ۶ - ۷ = ۲ بیت بالا رتبه از شماره سیلندر/ تراک
DH شماره هد / طرف
DL درایو (80H یا بالاتر)

این عملیات در صورتیکه معتبر باشد پرچم نقلی را صفر می‌کند و در صورتیکه نامعتبر باشد پرچم نقلی را یک می‌کند و یک کد وضعیت در AH باز می‌گرداند.

تابع 0DH از وقفه 13H: تنظیم متناوب دیسک

این عملیات مشابه تابع 00H است، بجز آنکه فقط به دیسک سخت محدود می‌شود. درایو (80H یا بالاتر) را در DL قرار دهید، عملیات، بازوی دسترسی خواندن/ نوشتن را در سیلندر 0 تنظیم می‌کند. اگر معتبر باشد پرچم نقلی را صفر می‌کند و اگر معتبر نباشد یک کد وضعیت در AH باز می‌گرداند.

تابع 0FH از وقفه 13H: نوشتن بافر سکتور

این عملیات مشابه تابع 0BH است، بجز آنکه فقط بخش ۵۱۲ بایتی داده از داده را روی دیسک می‌نویسد و بایت‌های ECC را نمی‌نویسد.

تابع 10H از وقفه 13H : بررسی آماده‌بودن درایو، 11H : تنظیم مجدد دیسک سخت، 12H : تشخیص ROM ، 13H: تشخیص درایو و 14H: تشخیص کنترلر

این توابع تشخیص داخلی را انجام می‌دهند و اطلاعات خاصی برای BIOS و برنامه‌های سودمند پیشرفته گزارش

می‌دهند این عملیات اگر معتبر باشند پرچم نقلی را صفر می‌کنند یا اگر معتبر نباشند پرچم نقلی را یک می‌کنند و یک کد وضعیت در AH باز می‌گردانند.

تابع 15H از وقفه 13H : گرفتن نوع درایو

این تابع اطلاعاتی راجع به یک دیسک‌درایو بازمی‌گرداند. برای استفاده ازآن، در DL شماره‌درایو را قراردهید (A = 0 وغیره، برای دیسکت 80H یا بالاتر برای دیسک سخت) یک عملیات معتبر یکی از کدهای زیر را در AH بازمی‌گرداند.

00H درایو / دیسک حاضر نیست

01H دیسکت درایو تغییر دیسکت را حس نکرده است

02H دیسکت درایو تغییر دیسکت را حس کرده‌است

03H درایو دیسک سخت

برای کد ۰۳ بازگشتی در AH، جفت CX:DX حاوی مجموع تعداد سکتورهای دیسک روی درایو است. عملیات پرچم نقلی را صفر می‌کند یا یک می‌کند و کدهای خطا را در AH باز می‌گرداند.

تابع 16H از وقفه 13H : تغییر وضعیت دیسکت

این تابع تغییر دیسکت را برای سیستمهایی که تغییر را می‌توانند حس کنند، بررسی می‌کند (تابع 15H را نیز ببینید). برای استفاده از این سرویس، DL را با شماره درایو (A = 0 و غیره) بارگذاری کنید. عملیات یکی از کدهای زیر را در AH باز می‌گرداند:

00H هیچ تغییری برای دیسکت نیست (پرچم نقلی = 0)

01H پارامتر دیسکت نامعتبر (پرچم نقلی = 1)

06H دیسکت تغییر یافته است (پرچم نقلی = 1)

80H درایو دیسکت آماده نیست (پرچم نقلی = 1)

کد وضعیت 01H و 80H خطاهایی هستند که پرچم وضعیت را تنظیم می‌کنند، در حالیکه 06H یک وضعیت معتبر می‌باشد که همچنین پرچم نقلی را تنظیم می‌کند.

تابع 17H از وقفه 13H : تنظیم نوع دیسکت

این عملیات ترکیب درایو و دیسکت را تنظیم می‌کند. این تابع 17H را با تابع 05H برای فرمت کردن دیسک استفاده کنید. برای استفاده از این سرویس، شماره درایو را در DL و نوع دیسکت را در AL قرار دهید. دیسکت‌هاشامل:

01H دیسکت 360K در درایو 360K

02H دیسکت 360K در درایو 1.2M

03H دیسکت 1.2M در درایو 1.2M

04H دیسکت 720K در درایو 720K

05H دیسکت 1.44M در درایو 1.44M

این عملیات در صورتیکه معتبر باشد پرچم نقلی را صفر می‌کند و در صورتیکه معتبر نباشد پرچم نقلی را یک می‌کند و در AH وضعیت را باز می‌گرداند.

تابع 18H از وقفه 13H : تنظیم نوع واسطه برای فرمت

از این عملیات بلافاصله از اجرای تابع 05H استفاده کنید. برای تنظیم نوع واسطه این ثبات‌ها، را بارگذاری کنید:

CH تعداد تراک‌ها (۸ بیت پایین رتبه)

CL تعداد تراک‌ها (۲ بیت بالا ۶-۷)، سکتورهای هر تراک (بیت‌های ۰ - ۵)

DL درایو (0 = A و غیره)

یک عملیات معتبر در ES:DI یک اشاره‌گر به یک جدول پارامتر دیسکت ۱۱ بایتی باز می‌گرداند. (تابع 08H را ببینید). این عملیات اگر معتبر باشند پرچم نقلی را صفر می‌کنند و در صورتیکه معتبر نباشند پرچم نقلی را یک می‌کند و وضعیت را در AH باز می‌گرداند.

تابع 19H از وقفه 13H : متوقف کردن هدهای دیسک

این عملیات به شماره درایو در DL (80H یا بالاتر برای دیسک سخت) نیاز دارد. این عملیات اگر معتبر باشد پرچم نقلی را صفر می‌کنند و اگر معتبر نباشد پرچم نقلی را یک می‌کند و وضعیت را در AH باز می‌گرداند.

نکات کلیدی

- وقفهٔ شمارهٔ 13H از BIOS دستیابی مستقیم به تراک‌ها و سکتورها را فراهم می‌سازد.
- این وقفه (13H) دستکاری شاخه را بطور خودکار، عملیات انتهای فایل بلوکه کردن یا بلوکه نکردن رکوردها را فراهم نمی‌سازد.
- عملیات تشخیص سکتور، بررسی ابتدایی داده‌های نوشته شده را در زمانی مشخص از پردازش انجام می‌دهد.
- یک برنامه باید بایت وضعیت را بعد از هر عملیات دیسک INT 13H بررسی نماید.

پرسش‌ها

۱-۱۹. دو ضرر عمده استفاده از وقفه شماره 13H از BIOS چیست؟ که بر طبق آن استفاده از وقفه 21H همیشه توصیه می‌شود؟

۲-۱۹. تحت چه شرایطی یک برنامه‌نویس باید از وقفه 13H استفاده نماید؟

۳-۱۹. اغلب توابع وقفه 13H یک کد وضعیت باز می‌گردانند. (الف) در کجا کد باز می‌گردد؟ (ب) کد 00H به معنی چیست؟ (ج) کد 04H به معنی چیست؟

۴-۱۹. روال استاندارد برای یک کد بازگشتی توسط وقفه 13H چیست؟ چگونه باید یک خطا بررسی شود و چه عملی باید انجام دهد؟

۵-۱۹. دستوراتی کد نمایید تا کنترلر دیسکت تنظیم شود.

۶-۱۹. دستوراتی کد نمایید تا وضعیت دیسکت را بخواند.

۷-۱۹. از آدرس حافظه DISKIN، درایو B، هد ۰، تراک ۵ و سکتور ۴ استفاده کنید، دستوراتی برای وقفه 13H کد نمایید تا سه سکتور را بخواند.

۸-۱۹. از آدرس حافظه DATAOUT استفاده کنید و درایو را A، هد، تراک و سکتور را به ترتیب ۰، ۷ و ۳ در نظر بگیرید، دستوراتی برای وقفه 13H کد نمایید تا یک سکتور را بنویسد. از خالی بودن دیسکت برای این تمرین اطمینان حاصل کنید.

۹-۱۹. بعد از عملیات نوشتن در پرسش ۸-۱۹، چگونه می‌توانید تلاش برای نوشتن روی یک دیسک محافظت شده را بررسی نمایید.

۱۰-۱۹. بر طبق پرسش ۸-۱۹، دستوراتی کد نمایید تا عملیات نوشتن را تشخیص بدهد.

فصل ۲۰

وسایل چاپ کردن

هدف : توصیف ضروریات چاپ کردن با استفاده از عملیات وقفه‌های مختلف

مقدمه

در مقایسه با صفحه نمایش و دستیابی دیسک، چاپ کردن یک پردازش ساده خواهد بود. تعداد کمی از اعمال وجود دارند که از طریق INT 21H متعلق به DOS یا از طریق INT 17H متعلق به BIOS انجام می‌شوند. فرامین خاص به چاپگر شامل تعویض صفحه، تعویض خط، Tab، رفتن به ابتدای خط می‌باشد. چاپگر برای انجام اعمالی نظیر رفتن به صفحه جدید، رفتن به خط جدید بعدی در یک صفحه یا Tab در یک صفحه، باید علائم ارسالی از چاپگر را درک کند. پردازشگر نیز باید علامت ارسالی از چاپگر، مبنی بر مشغول بودن یا ورق چاپ نداشتن را درک نماید. متأسفانه مختلف چاپگرها بطور متفاوت،

به علائم ارسالی از پردازشگر پاسخ می‌دهند. و یکی از وظائف مشکل متخصصین نرم‌افزار ایجاد رابطه برنامه‌ها با چنین چاپگرهایی است.

این فصل عملیات وقفه زیر را برای کار با چاپگر معرفی می‌نماید.

توابع INT 21H

40H چاپ کاراکترها

05Hچاپ کاراکتر

توابع INT 17H

00H چاپ کاراکتر

01H مقدار دهی درگاه

02H گرفتن وضعیت درگاه چاپگر

DECIMAL	HEX	FUNCTION
09	09H	Horizontal Tab
10	0AH	Line Feed (advance one line)
12	0CH	Form Feed (advance to next page)
13	0DH	Carriage Return (return to left margin)

کاراکترهای عمومی کنترل چاپگر

کاراکترهای استاندارد که عملیات چاپ را در چاپگرهای سازگار، PC را کنترل می‌کنند بشرح زیر می‌باشند :

Tab افقی. (09H) این کاراکتر سبب می‌شود چاپگر به موقعیت چاپ بعدی در Tab بعدی پیش رود (اغلب به میزان ۸ کاراکتر پیش می‌رود) این کاراکتر فقط روی چاپگرهایی کار می‌کند که این خصیصه را دارند و بدین منظور تنظیم شده‌اند. به منظور غلبه بر عدم توانایی چاپگر برای این عمل می‌توانید رشته‌ای از کاراکتر جای خالی را چاپ کنید.

تعویض خط. برای پیش روی یک خط، از تعویض خط (0AH) و دو تعویض خط متوالی، برای فضای دو برابر استفاده کنید.

تعویض صفحه. تنظیم اولیه کاغذ موقع روشن کردن چاپگر موقعیت شروع را برای بالای صفحه تعیین می‌کند. اندازه فرضی یک صفحه یازده اینچ است. پردازشگر و چاپگر هیچکدام بطور خودکار انتهای یک صفحه را بررسی نمی‌کنند. اگر برنامه شما چاپ را تا پایین یک صفحه ادامه دهد، بالاخره بعد از اتمام صفحه، در بالای صفحه بعدی می‌نویسد، برای کنترل صفحه بندی، تعداد خطوطی که چاپ می‌شود را بشمارید و موقع رسیدن به حداکثر خطوط یک صفحه (مثلاً ۵۵ خط) یک تعویض صفحه (OCH) اجرا کنید و سپس شماره خط را مجدداً با صفر یا یک مقدار دهید.

در انتهای چاپ، یک فرمان مثلاً تعویض خط یا تعویض صفحه ارسال دارید تا باعث شود که آخرین خطی که هنوز در بافر چاپگر است ، چاپ گردد. استفاده از تعویض سطر در انتهای اجرا، جدا کردن صفحه آخر را نیز آسانتر می‌کند.

بازگشت به سر سطر. این کاراکتر کنترلی (0DH)، با یک Line Feed ترکیب می‌شود، و چاپگر را در سمت چپ سطر تنظیم می‌کند. این کاراکتر <Enter> یا <Return> در صفحه کلید و CR در صفحه نمایش شناخته می‌شود.

تابع 40H از وقفه 21H : چاپ کاراکترها

تا کنون از دستگیره فایل در فصول دستکاری صفحه نمایش و پردازش استفاده نمودیم. برای چاپ کردن با تابع 40H، این ثبات‌ها را بارگذاری کنید :

AH	تابع 40H	CX	تعداد کاراکترهای چاپ
BX	دستگیره فایل 04	DX	آدرس داده‌هایی که باید چاپ شوند

مثال زیر ۲۷ کاراکتر از ناحیه داده با نام HEADING با شروع از حاشیه سمت چپ، چاپ می‌کند. کاراکترهای (0DH) و (0AH) بلافاصله پس از متن HEADING سبب می‌شود چاپگر به ستون 0 و سطر بعدی برود.

```
HEADING  DB   'Mounting Outfiting Crop', 00H, 0AH
         ...
         MOV  AH,40H        درخواست چاپ کردن ;
         MOV  BX,04         دستگیره 04 برای چاپگر است ;
         MOV  CX,27         فرستادن ۲۷ کاراکتر ;
         LEA  DX,HEADING    آدرس ناحیه چاپ ;
         INT  21H           فراخوانی سرویس وقفه ;
```

یک عملیات موفق متن را چاپ می‌کند، پرچم نقلی را صفر می‌کند و در AX تعداد کاراکترهایی که چاپ می‌شود باز می‌گرداند. یک عملیات ناموفق پرچم نقلی را یک می‌کند و در AX کد خطای ۰۵ (دستیابی لغو شد) یا ۰۶ (دستگیره نامعتبر) باز می‌گرداند. علامت انتهای فایل (Ctrl+Z یا 0AH) داده‌ها را منتقل می‌کند و سبب خاتمه عملیات می‌شود.

دو حالتی که سبب قطع چاپ می‌شود :

۱) چاپگر روشن نیست. سیستم پیغام می‌دهد : "Write Fault Error Writing Device PRN" "Abort Retry Ignore Fail"

۲) ورق چاپ نیست یا جمع شده است. سیستم پیغام می‌دهد : "Printer out of paper error writing device PRN"

برنامه : چاپ کردن بیشتر از یک صفحه و عناوین

برنامه شکل ۲-۹ اسامی را از صفحه کلید اخذ می‌نماید و آنها را روی صفحه تصویر نمایش می‌دهد. برنامه شکل ۱-۲۰ با آن مشابه است لیکن این برنامه اسامی را به چاپگر ارسال می‌کند. هر صفحه چاپ شونده حاوی یک عنوان و به دنبال آن یک فضای مضاعف و اسامی وارد شده به شکل زیر می‌باشد :

برنامه، هر خط چاپ شده را می‌شمارد و بعد از رسیدن به انتهای صفحه، کنترل چاپ را به بالای صفحه بعدی منتقل می‌کند، روال‌های اصلی بشرح زیرند :

- A10MAIN : B10INPT و C10PRINT را فرا می‌خواند و پردازش را وقتی کاربر فقط <Enter> را وارد کند خاتمه می‌دهد.

```
List of Employee Names    Page 01
Annie Hall
Fanny Hill
Danny Rose
...
```

```
TITLE     A20PRTNM (EXE)  Accept entered names and print
          .MODEL  SMALL
          .STACK  64
          .DATA
NAMEPAR   LABEL   BYTE              ;Keyboard parameter list:
MAXNLEN   DB      20                ;  maximum length of name
NAMELEN   DB      ?                 ;  actual length entered
NAMEFLD   DB      20 DUP(' ')       ;  name entered
                                    ;Heading line:
HEADG     DB      'List of Employee Names    Page
PAGECTR   DB      '01', 0DH, 0AH, 0AH

FFEED     DB      0CH               ;Form feed
LFEED     DB      0DH, 0AH          ;CR, line feed
LINECTR   DB      01
PROMPT    DB      'Name? '
; ---------------------------------------------------------
          .CODE
A10MAIN   PROC    FAR
          MOV     AX,@data          ;Initialize
          MOV     DS,AX             ;  segment
          MOV     ES,AX             ;  registers
          CALL    Q10CLR            ;Clear screen
          CALL    D10PAGE           ;Page heading
A20LOOP:
          MOV     DX,0000           ;Set cursor to 00,00
          CALL    Q20CURS
          CALL    B10INPT           ;Accept input of name
          CALL    Q10CLR
          CMP     NAMELEN,00        ;Name entered?
          JE      A30               ;  no, exit
          CALL    C10PRINT          ;  yes, prepare printing
          JMP     A20LOOP
A30:      MOV     CX,01             ;End of processing:
          LEA     DX,FFEED          ;  one character
          CALL    P10OUT            ;  for form feed,
          MOV     AX,4C00H          ;  exit
          INT     21H
A10MAIN   ENDP
;                 Accept input of name:
;                 --------------------
B10INPT   PROC    NEAR
          MOV     AH,40H            ;Request display
          MOV     BX,01             ;
          MOV     CX,05             ;  5 characters,
          LEA     DX,PROMPT         ;  prompt message
          INT     21H
          MOV     AH,0AH            ;Request keyboard
          LEA     DX,NAMEPAR        ;  input
          INT     21H
          RET
B10INPT   ENDP
;                 Prepare for printing:
;                 --------------------
C10PRINT  PROC    NEAR
          CMP     LINECTR,60        ;End of page?
          JB      C20               ;  no,  bypass
          CALL    D10PAGE           ;  yes, print heading
```

شکل الف ۱-۲۰ چاپ بیشتر از یک صفحه و عناوین

```
C20:        MOV     CH,00
            MOV     CL,NAMELEN          ;Set no. of characters
            LEA     DX,NAMEFLD          ;Set address of name
            CALL    P10OUT              ;Print name
            MOV     CX,02               ;Request CR,
            LEA     DX,LFEED            ;  line feed
            CALL    P10OUT
            INC     LINECTR             ;Add to line count
            RET
C10PRINT    ENDP
;                   Page heading routine:
;                   --------------------
D10PAGE     PROC    NEAR
            CMP     WORD PTR PAGECTR,3130H  ;First page?
            JE      D30                 ;  yes, bypass
            MOV     CX,01               ;
            LEA     DX,FFEED            ;  no,
            CALL    P10OUT              ;  form feed,
            MOV     LINECTR,03          ;  reset line count
D30:
            MOV     CX,37               ;Length heading, page
            LEA     DX,HEADG            ;Address of heading
            CALL    P10OUT
            INC     PAGECTR+1           ;Add to page count
            CMP     PAGECTR+1,3AH       ;Page no. = hex 3A?
            JNE     D90                 ;  no,  bypass
            MOV     PAGECTR+1,30H       ;  yes, set to ASCII
            INC     PAGECTR             ;
D90:        RET
D10PAGE     ENDP
;                   Print routine:
;                   -------------
P10OUT      PROC    NEAR                ;CX and DX set on entry
            MOV     AH,40H              ;Request print
            MOV     BX,04               ;Handle
            INT     21H
            RET
P10OUT      ENDP
;                   Clear screen:
;                   ------------
Q10CLR      PROC    NEAR
            MOV     AX,0600H            ;Request scroll
            MOV     BH,60H              ;Attribute
            MOV     CX,0000             ;From 00,00
            MOV     DX,184FH            ;  to 24,79
            INT     10H
            RET
Q10CLR      ENDP
;                   Set cursor row/col:
;                   ------------------
Q20CURS     PROC    NEAR                ;DX set on entry
            MOV     AH,02H              ;Request set cursor
            MOV     BH,00               ;Page number 0
            INT     10H
            RET
Q20CURS     ENDP
            END     A10MAIN
```

شکل ب ۱-۲۰ چاپ بیشتر از یک صفحه و عناوین

- B10INPT : برای پذیرش نامی از صفحه کلید اعلان می‌کند.
- C10PRINT : در انتهای صفحه (۶ خط) M10PAGE را فرا می‌خواند، نام را چاپ می‌کند (طول آن بر مبنای طول واقعی در لیست پارامترهای ورودی صفحه کلید است).
- D10PAGE : به یک صفحه بعدی پیش می‌رود، عنوان را چاپ می‌کند، شماره خط را صفر می‌کند و به شماره صفحه می‌افزاید.

- PIOOUT : روتین عمومی که درخواست چاپ را عمل می‌کند.

در شروع اجرا، لازم است که یک عنوان چاپ شود ولی به صفحه جدید نمی‌رود. بدین منظور، D10PAGE، در صورتی که PAGECTR حاوی ۰۱ باشد تعویض صفحه را کنار می‌گذارد، این مقدار اولیه PAGECTR است. PAGECTR بصورت DB '0' تعریف شده است که یک عدد ASCII ، 3031H را تولید می‌کند. روال PAGECTR را یک واحد می‌افزاید که به ترتیب 3032 ، 3033 و الی آخر می‌شود. حداکثر تعداد 3039 است و آنگاه 303A می‌شود که به صورت یک صفر و یک (:) چاپ می‌شود، در اینجا، روتین 3AH را به 30H تنظیم می‌کند و یک را به بایت سمت چپ می‌افزاید، بنابراین 303AH تبدیل به 3130H یا مقدار دهدهی 10 خواهد شد. جایگذاری یک بررسی انتهای صفحه قبل از (نه بعد از) چاپ یک نام این اطمینان را ایجاد می‌کند که صفحه بعد یک نام قبل از عنوان دارد .

برنامه : لیست‌گیری فایل‌های ASCII و دستکاری جدول‌بندیها

یک روال عمومی، که مثلاً توسط یک تطبیق دهنده ویدئو اجرا شده است، به منظور توسعه یک کاراکتر جدول‌بندی (09) با جاهای خالی به مکان بعدی قابل قسمت بر ۸، می‌باشد. بنابراین، مکث‌های جدول‌بندی در مکانهای ۸، ۱۶، ۲۴ و الی آخر می‌باشد. بنابراین همه موقعیت‌های بین ۰ تا ۷ در ۸ مکث می‌کنند و هم موقعیت‌های بین ۸ تا ۱۵ در ۱۶ مکث دارند. اما بسیاری از چاپگرها کاراکترهای جدول‌بندی را نادیده می‌گیرند. یک برنامه مثل PRINT متعلق به DOS که فایل‌های ASCII را چاپ می‌کند (مثل برنامه‌های منبع اسمبلی) بایستی هر کاراکتری را که به چاپگر ارسال می‌کند، بررسی کند. اگر یک کاراکتر Tab باشد، برنامه، جاهای خالی را به مکان ۸ بعدی توسعه می‌دهد.

برنامه شکل ۲-۲۰ از کاربر می‌خواهد تا نام یک فایل را وارد کند و محتویات آن را چاپ نماید. برنامه مشابه شکل ۳-۱۷ می‌باشد که رکوردها را نمایش می‌دهد لیکن برنامه باید مکث‌های Tab را برای چاپگر با جای خالی جایگزین نمایید. در زیر سه مثال از مکث‌های Tab برای چاپ موقعیت ۱، ۹ و ۲۱ و منطق تنظیم موقعیت Tab بعدی می‌باشد.

موقعیت چاپ جاری :	1	9	21
ارزش دودویی :	00000001	00001001	00010101
پاک کردن ۳ بیت سمت راست :	00000000	00001000	00010000
جمع با ۸ :	00001000	00010000	00011000
موقعیت Tab جدید:	8	16	24

این برنامه به صورت زیر سازماندهی شده است :

- A10MAIN روال‌های B10PROMP ، C10OPEN ، D10READ و E10XFER را فرا می‌خواند.
- B10PROMP از کاربر می‌خواهد تا یک نام فایل را وارد کند، فشردن فقط کلید <Enter> مبنی بر خاتمه کار کاربر می‌باشد.
- C10OPEN فایل ورودی مورد نظر را باز می‌کند. اگر عملیات معتبر باشد، روال از تابع 24H واقع در وقفه 21H جهت تعیین اندازه فایل استفاده می‌کند (فقط از قسمت پایین رتبه استفاده نمایید، با حداکثر ۶۵۵۳۵ بایت)
- D10READ یک سکتور از فایل را می‌خواند.
- E10XFER داده ورودی را برای انتهای سکتور، انتهای فایل، انتهای ناحیه نمایش، Line Feed و Tab بررسی می‌کند. اصولاً، روال کاراکترهای عادی را به ناحیه نمایش ارسال می‌کند و منطق دستکاری مکث‌های Tab را نیز انجام می‌دهد. همچنین، روال انتهای فایل را نیز تعیین می‌کند، با این روش که اندازه فایل ذخیره شده را با پردازش هر کاراکتر یک واحد می‌کاهد.
- P10OPRINT خط خروجی را چاپ می‌کند و با جای خالی آن را پاک می‌کند .

می‌توانید برنامه را با شمارش خطوط چاپ شده و رفتن به صفحه بعدی وقتی نزدیک انتهای صفحه در خط ۶۰

است، تغییر دهید. همچنین می‌توانید با استفاده از یک برنامه ویرایشگر کاراکترهای Form Feed را مستقیماً در فایل ASCII جایگذاری کنید ، در موقعیت‌هایی که می‌خواهید یک صفحه شکسته شود، مانند انتهای یک روال، یک رویهٔ معمول نگه داشتن، کلید Alt و فشردن اعداد روی صفحه کلید مانند 012 برای From feed می‌باشد.

```
          page 60,132
TITLE     A20PRTAS (EXE)  Read and print disk records
          .MODEL SMALL
          .STACK 64
; ------------------------------------------------------
          .DATA
PATHPAR   LABEL  BYTE              ;Parameter list for
MAXLEN    DB     32                ;  input of
NAMELEN   DB     ?                 ;  filename
FILENAME  DB     32 DUP(' ')

FILEDTA   LABEL  BYTE              ;File DTA
          DB     26 DUP(20H)       ;Reserved
FILESIZE  DW     0                 ;File size (low-order)
          DW     0                 ;File size (high-order
          DB     13 DUP(20H)       ;Rest of file DTA

COUNT     DW     00
ENDCDE    DW     00                ;End process indicator
FFEED     DB     0CH
HANDLE    DW     0
OPENMSG   DB     '*** Open error ***', 0DH, 0FH
PRTAREA   DB     120 DUP(' ')      ;Print area
PROMPT    DB     'Name of file? '
ROW       DB     0                 ;Screen row
SECTOR    DB     512 DUP(' ')      ;Input area for file
; ------------------------------------------------------
          .CODE
.386
A10MAIN   PROC   FAR               ;Main procedure
          MOV    AX,@data          ;Initialize
          MOV    DS,AX             ;  segment
          MOV    ES,AX             ;  registers
          CALL   Q10SCR            ;Clear screen
A20LOOP:
          MOV    ENDCDE,00         ;Initialize
          CALL   B10PROMP          ;Request filename
          CMP    NAMELEN,00        ;Any request?
          JE     A90               ;  no, exit
          CALL   C10OPEN           ;Open file, get handle
          CMP    ENDCDE,00         ;Valid open?
          JNE    A20LOOP           ;  no, request again
          CALL   D10READ           ;Read 1st disk sector
          CMP    ENDCDE,00         ;End-of-file, no data?
          JE     A80               ;  yes, request next
          CALL   E10XFER           ;Print/read
A80:
          MOV    AH,3EH            ;Close file
          MOV    BX,HANDLE
          INT    21H
          JMP    A20LOOP           ;Repeat processing
A90:
          MOV    AX,4C00H          ;End processing
          INT    21H
A10MAIN   ENDP
 ;                 Request file name:
 ;                 ------------------
 B10PROMP  PROC   NEAR
           CALL   Q20CURS           ;Set cursor
           MOV    AH,40H            ;Prompt for filename
           MOV    BX,01             ;Handle for screen
           MOV    CX,13             ;No. of characters
```

برنامه الف ۲-۲۰ چاپ یک فایل *ASCII*

```
            LEA     DX,PROMPT
            INT     21H
            MOV     AH,0AH              ;Accept filename
            LEA     DX,PATHPAR
            INT     21H
            MOVZX   BX,NAMELEN          ;Insert zero at end
            MOV     FILENAME[BX],0      ;  of filespec
            RET
B10PROMP    ENDP
;                   Open disk file:
;                   ---------------
C10OPEN     PROC    NEAR
            MOV     AH,3DH              ;Request open
            MOV     AL,00               ;Read only
            LEA     DX,FILENAME
            INT     21H
            JNC     C20                 ;Test carry flag,
            CALL    X10ERR              ;  error if set
            JMP     C90
C20:
            MOV     HANDLE,AX           ;Save handle
            MOV     AH,1AH              ;Set DTA
            LEA     DX,FILEDTA          ;
            INT     21H                 ;
            MOV     AH,4EH              ;Find file
            MOV     CX,0                ;  and get
            LEA     DX,FILENAME         ;  file size
            INT     21H
C90:        RET
C10OPEN     ENDP
;                   Read disk sector:
;                   -----------------
D10READ     PROC    NEAR
            MOV     AH,3FH              ;Request read
            MOV     BX,HANDLE           ;Device
            MOV     CX,512              ;Length
            LEA     DX,SECTOR           ;Buffer
            INT     21H
            MOV     ENDCDE,AX
            RET
D10READ     ENDP
;                   Transfer data to print line:
;                   ----------------------------
E10XFER     PROC    NEAR
            CLD                         ;Set left-to-right
            LEA     SI,SECTOR           ;Initialize
E20:
            LEA     DI,PRTAREA
            MOV     COUNT,00
E30:        LEA     DX,SECTOR+512
            CMP     SI,DX               ;End of sector?
            JNE     E40                 ;  no,  bypass
            CALL    D10READ             ;  yes, read next
            CMP     ENDCDE,00           ;End of file?
            JE      E90                 ;  yes, exit
            LEA     SI,SECTOR
E40:
            MOV     BX,COUNT
            CMP     BX,120              ;At end of print area?
            JB      E50                 ;  no,  bypass
            MOV     [DI+BX],0D0AH       ;  yes, set CR/LF
            CALL    P10PRNT
            LEA     DI,PRTAREA          ;Reinitialize
            MOV     COUNT,00
E50:
            LODSB                       ;[SI] to AL, INC SI
            MOV     BX,COUNT
            MOV     [DI+BX],AL          ;Character to print line
            INC     BX
            DEC     FILESIZE            ;All chars processed?
```

برنامه ب ۲-۲۰ چاپ یک فایل ASCII

```
                JZ      E90             ;  yes, exit
                CMP     AL,0AH          ;Line feed?
                JNE     E60             ;  no,  bypass
                CALL    P10PRNT         ;  yes, call print
                JMP     E20
E60:
                CMP     AL,09H          ;Tab character?
                JNE     E70             ;  no, bypass
                DEC     BX              ;  yes, reset BX
                MOV     BYTE PTR [DI+BX],20H ;Clear tab to blank
                AND     BX,0FFF8H       ;Clear rightmost 3 bits,
                ADD     BX,08           ;  add 8 for tab stop
E70:
                MOV     COUNT,BX
                JMP     E30
E90:
                MOV     BX,COUNT        ;End of file
                MOV     BYTE PTR [DI+BX],0CH   ;Form feed
                CALL    P10PRNT         ;Print last line
                RET
E10XFER         ENDP
;                       Print line:
;                       ----------
P10PRNT         PROC    NEAR
                MOV     AH,40H          ;Request print
                MOV     BX,04
                MOV     CX,COUNT        ;Length
                INC     CX
                LEA     DX,PRTAREA
                INT     21H
                MOV     AX,2020H        ;Clear print line
                MOV     CX,60
                LEA     DI,PRTAREA
                REP STOSW
                RET
P10PRNT         ENDP
;                       Scroll screen:
;                       ------------
Q10SCR          PROC    NEAR
                MOV     AX,0600H        ;Request scroll
                MOV     BH,1EH          ;Set attribute
                MOV     CX,0000
                MOV     DX,184FH
                INT     10H
                RET
Q10SCR          ENDP
;                       Set cursor:
;                       ----------
Q20CURS         PROC    NEAR
                MOV     AH,02H          ;Request set
                MOV     BH,00           ;  cursor
                MOV     DH,ROW          ;Row
                MOV     DL,10           ;Column 10
                INT     10H
                INC     ROW             ;Next screen row
                RET
Q20CURS         ENDP
;                       Display disk error message:
;                       --------------------------
X10ERR          PROC    NEAR
                CALL    Q20CURS         ;Set cursor
                MOV     AH,40H          ;Request display
                MOV     BX,01           ;Handle
                MOV     CX,20           ;Length
                LEA     DX,OPENMSG      ;Error message
                INT     21H
                MOV     ENDCDE,01       ;Error indicator
                RET
X10ERR          ENDP
                END     A10MAIN
```

برنامه پ ۲-۲۰ چاپ یک فایل ASCII

می‌توانید برنامه را با تابع 05H از وقفه 21H با فرستادن هر کاراکتر مستقیماً به چاپگر بازنویسی کنید، بدینوسیله تعریف و استفاده از ناحیه چاپ حذف خواهد شد.

تابع 05H از وقفه 21H : چاپ کاراکتر

تابع 05H نیز وسایل چاپ را فراهم می‌سازد. برای استفاده از آن تابع 05H را در ثبات AH و کاراکتری را که می‌خواهید چاپ شود در DL قرار دهید، بصورت زیر :

```
MOV   AH,05H      ; درخواست چاپ کاراکتر
MOV   DL,char     ; کاراکتر جهت چاپ
INT   21H         ; فراخوانی سرویس وقفه
```

این دستورات برای فرستادن یک کاراکتر تک به چاپگر در نظر گرفته شده است. اما معمولاً چاپ کردن مستلزم یک بخش از خط متن است و برای مراحل یک خط قالب بندی شده در ناحیه داده به برنامه نیاز دارد. مثال زیر چاپ یک خط کامل را مشخص می‌سازد. ابتدا آدرس HEADING را در ثبات SI مقدار دهی نموده و CX را با طول HEADING تنظیم می‌کند. حلقه در P20 بطور مرتب کاراکترهای HEADING را اقتباس نموده و به چاپگر ارسال می‌کند. چون اولین کاراکتر HEADING یک From Feed است و دو کاراکتر بعدی Line Feed می‌باشد، عنوان در بالای صفحه جدید چاپ می‌شود و پس از آن دو جای خالی خواهد بود که بصورت زیر می‌باشد :

```
HEADING  DB    0CH,'Mountain Oufitting Corp',0DH,0AH, 0AH
         ...
         MOV   CX,29          ; مقدار دهی طول و
         LEA   SI,HEADING     ; آدرس عنوان
P20 :
         MOV   AH,05H         ; درخواست چاپ
         MOV   DL,[SI]        ; کاراکتری از عنوان
         INT   21H            ; فراخوانی سرویس وقفه
         INC   SI             ; کاراکتر بعدی عنوان
         LOOP  P20            ; تکرار ۲۹ بار حلقه
```

اگر چاپگر روشن نباشد، سیستم یک پیغام 'out of papre' نمایش می‌دهد. اگر آن را روشن کنید برنامه چاپ را به نحو صحیحی آغاز خواهد کرد. همچنین می‌توانید Ctrl + Break را برای لغو اجرای عملیات چاپ بفشارید .

کاراکترهای ویژه کنترل چاپگر

حال استفاده از کاراکترهای اصلی کنترل چاپگر، Tab ، Line Feed ، Form Feed و CarrigeReturn را بررسی می‌کنیم. دیگر فرمان‌های مناسب برای اغلب چاپگرها بشرح زیر می‌باشند.

دهدهی	شانزدهی	عمل
08	08	Back space
11	0B	Tab عمودی
14	0E	روشن کردن حالت کشیده
15	0F	روشن کردن حالت فشرده
18	12	خاموش کردن حالت فشرده
20	14	خاموش کردن حالت کشیده

برخی فرمان‌های چاپ به یک کاراکتر ESC (1BH) نیاز دارند.

30	1B	تنظیم فاصله خطوط با ۸ خط در هر اینچ
32	1B	تنظیم فاصله خطوط با ۶ خط در هر اینچ
45	1B	روشن کردن تنظیم حالت چاپ پر رنگ
46	1B	خاموش کردن تنظیم حالت چاپ پر رنگ

می‌توانید کاراکترهای کنترلی را با دو روش به چاپگر ارسال نمایید :

۱) کاراکترهای کنترلی را در ناحیه داده تعریف کنید. در زیر حالت فشرده را تنظیم می‌کند، با ۸ خط در هر اینچ یک عنوان را چاپ می‌کند و سبب یک Line Feed , CR می‌شود.

```
HEADING DB 0FH,1BH , 30H ,"Mountain Outfiting Corp" ,0DH , 0AH
```

۲) از تابع 05H برای ارسال کاراکترها به چاپگر استفاده کنید :

```
MOV  AH,05H      ; درخواست چاپ
MOV  DL,0FH      ; درخواست حالت فشرده
INT  21H         ; فراخوانی سرویس وقفه
```

همه کاراکترهای بعدی در حالت فشرده چاپ خواهند شد تا برنامه فرمان دیگری ارسال کند که حالت را مجدداً تنظیم نماید.

فرمانهایی که گفته شده الزاماً با همه چاپگرها کار نمی‌کند، لذا راهنمای فرمانهای خاص چاپگرتان را بررسی نمایید .

INT 17H توابع چاپ کردن

INT 17H درسطح BIOS تسهیلاتی رابرای چاپ‌کردن فراهم‌می‌سازد. درگاه چاپگرمعتبر، LPT1، LPF2 و LPT3 با شماره‌های 0 (پیش فرض)، ۱، ۲ می‌باشند. INT 17H سه تابع فراهم می‌سازد که در ثبات AH مشخص می‌شود.

۱) تابع 02H را برای تعیین وضعیت چاپگر ، بر طبق یک شماره درگاه انتخابی صادر نمایید، این بررسی وضعیت را قبل از هر تلاش برای چاپ کردن انجام دهید.

۲) اگر چاپگر در دسترس می‌باشد، سپس تابعی 01H را برای مقدار دهی درگاه چاپگر استفاده کنید.

۳) تابع 02H را برای فرستادن یک کاراکتر به چاپگر صادر کنید.

این عملیات وضعیت چاپگر را در AH، با یک یا چند بیت یک، باز می‌گرداند.

بیت	سبب	بیت	سبب
۰	اتمام زمان	۵	بدون ورق چاپ
۳	خطای ورودی/ خروجی	۶	تصدیق چاپگر
۴	انتخاب شده	۷	مشغول نیست

اگر چاپگر روشن و آماده باشد ، عملیات 90H (10010000 دودویی) را باز می‌گرداند. چاپگر آماده نیست، اما انتخاب شده‌است، یک حالت معتبر می‌باشد. خطای چاپگر بیت ۵ (بدون ورق چاپ) و بیت ۳ (خطای خروجی) است. اگر چاپگر روشن نباشد ، عملیات B0H را باز می‌گرداند یا 10110000 دودویی که به معنی 'out ofpaper' می‌باشد.

تابع 00H از وقفه 17H : چاپ یک کاراکتر

این عملیات سبب چاپ یک کاراکتر می‌شود و درگاه‌های چاپگر ۰، ۱، یا ۲ را مجاز می‌داند. برای استفاده از آن، کاراکتر را در AL قرار دهید و شماره درگاه چاپگر را در DX قرار دهید :

```
MOV   AH,00H        ; درخواست چاپ
MOV   AL,char       ; کاراکتری که باید چاپ شود
MOV   DX,00         ; انتخاب درگاه صفر، چاپگر
INT   17H           ; فراخوانی سرویس وقفه
```

این عملیات وضعیت را در ثبات AH باز می‌گرداند. بر اساس تجربه توصیه می‌شود ابتدا از تابع 02H برای بررسی وضعیت چاپگر استفاده کنید.

تابع 01H از وقفه 17H : مقدار دهی درگاه چاپگر

این عملیات یک درگاه را انتخاب می‌کند، چاپگر را تنظیم می‌کند، و آن را برای داده‌ها مقدار دهی می‌کند. مثال زیر درگاه صفر را انتخاب می‌کند.

```
MOV   AH,01H        ; درخواست مقدار دهی درگاه
MOV   DX,00         ; انتخاب درگاه صفر، چاپگر
INT   17H           ; فراخوانی سرویس وقفه
```

چون عملیات یک کاراکتر Form Feed را به چاپگر ارسال می‌دارد، می‌توانید با استفاده از آن کاغذ چاپ را در موقعیت بالای صفحه تنظیم کنید ، اگر چه اغلب چاپگرها در هنگام روشن شدن این عمل را بطور خودکار انجام می‌دهند. عملیات کد وضعیت را در AH باز می‌گرداند.

تابع 02H از وقفه 17H : گرفتن وضعیت درگاه چاپگر

هدف از این عملیات تعیین وضعیت چاپگر می‌باشد. مثال زیر درگاه صفر را انتخاب می‌کند :

```
MOV   AH,02H        ; درخواست وضعیت درگاه
MOV   DX,00         ; انتخاب درگاه صفر
INT   17H           ; فراخوانی سرویس وقفه
TEST  AH,00101001B  ; آماده است؟
JNZ   error msg     ; نه، نمایش پیغام
```

عملیات وضعیت درگاه چاپگر را همانند تابع 01H باز می‌گرداند. وقتی برنامه اجرا شود، اگر چاپگر روشن نباشد، BIOS قادر به بازگرداندن یک پیغام بطور خودکار نخواهد بود - برنامه شما باید این مورد را بررسی نماید و بر اساس آن عمل کند. اگر برنامهٔ شما وضعیت را بررسی نکند، تنها مشخصه شما چشمک زدن مکان نما است. اگر در این حالت چاپگر را روشن کنید مقداری از داده‌های خروجی از بین می‌رود. در نتیجه، قبل از اعمال چاپ با استفاده از BIOS، وضعیت گذرگاه را بررسی کنید و اگر خطایی وجود دارد. یک پیغام نمایش دهید. (INT 21H این بررسی را بطور خودکار انجام می‌دهد، اگر چه پیغام 'Out Of Paper' در بسیاری از حالات ظاهر می‌شود). وقتی چاپگر روشن می‌شود، پیغام نمایش داده می‌شود و چاپ، به طور طبیعی بدون از دست دادن داده‌ای شروع می‌شود.

در هر حال، یک چاپگر ممکن است در حالت نبودن کاغذ به کارش ادامه دهد یا سهواً خاموش شود. در نتیجه اگر برنامه‌ای بنویسید که سایرین آن را اجرامی‌کنند، یک تست وضعیت قبل‌از هر اقدامی برای چاپ در برنامه‌تان قراردهید.

نکات کلیدی

- بعد از تکمیل چاپ، از یک Line Feed یا Form Feed برای پاک کردن بافر چاپگر استفاده کنید .
- تابع 40H از وقفه 21H: رشته‌ای از کارکترها را چاپ می‌کند در حالیکه تابع 05H از وقفه 21H و BIOS INT17H یک کاراکتر را در هر بار چاپ می‌کند.

سیستم در صورت خطای چاپگر یک پیغام را نمایش می‌دهد ، اگر چه BIOS فقط یک کد وضعیت باز می‌گرداند. وقتی از BIOS INT 17H استفاده‌می‌کنید، قبل‌از چاپ‌کردن ازتابع 02H برای بررسی‌وضعیت چاپگراستفاده‌نمایید.

پرسش‌ها

۲۰-۱. کاراکترهای کنترلی چاپگر در موارد زیر چیست (الف) Carraige Return (ب) Line Feed (ج) Form Feed (د) Horizental Tab

۲۰-۲. برنامه‌ای با استفاده از تابع 40H واقع در وقفه 21H برای موارد زیر بنویسید : (الف) رد کردن کاغذ به صفحه بعد (ب) چاپ نام شما (ج) یک CR و Line Feed انجام دهد و آدرس خیابان شما را چاپ کند (د) یک CR و Line Feed انجام دهد و کشور و شهر شما را چاپ کند (ه) کاغذ را رد کند.

۲۰-۳. پرسش ۲۰-۲ را با استفاده از تابع 05H از وقفه 21H بازنویسی کنید.

۲۰-۴. عنوانی را تعریف کنید که عملیات سر سطر و سطر بعدی، تنظیم حالت فشرده، تعریف یک تیتر (با هر نام) و غیر فعال نمودن حالت فشرده را انجام دهد.

۲۰-۵. پرسش ۲۰-۳ را طوری بازنویسی کنید که نام با حالت کشیده خیابان و آدرس با حالت فشرده و کشور - شهر با اندازه معمولی و پر رنگ چاپ شود.

۲۰-۶. INT 17H بعد از چاپ کردن، یک کد خطا در AH باز می‌گرداند معنی کدهای زیر چیست؟ (الف) 08H، (ب) 10H، (ج) 90H.

۲۰-۷. پرسش ۲۰-۲ را با استفاده از INT 17H بازنویسی کنید. یک بررسی برای وضعیت چاپگر نیز قرار دهید.

۲۰-۸. پرسش ۲۰-۲ را بازنویسی کنید بطوریکه برنامه بخش‌های (ب)، (ج) و (د)، ۵ مرتبه انجام دهد.

۲۰-۹. شکل ۲۰-۱ را برای اجرا تحت تابع 05H از وقفه 21H بازنویسی کنید.

۲۰-۱۰. شکل ۲۰-۲ را بازنویسی کنید بطوریکه خطوط چاپ شده را نمایش دهد.

فصل ۲۱

دیگر وسایل ورودی / خروجی

هدف : توضیح موارد ضروری برنامه نویسی برای موس و استفاده از درگاه‌ها

مقدمه

این فصل استفاده از موس، دستیابی به درگاه‌های PC، دستورات IN ، OUT و تولید صدا در بلندگو کامپیوتر را توضیح می‌دهد. دستوراتی که معرفی می‌شوند.

- INT 33H برای دستیابی موس
- IN/INS و OUT/OUTS برای دستیابی درگاه‌ها

ویژگی‌های موس

موس یک ابزار اشاره‌گر عمومی است که با یک رابط نرم‌افزاری بعنوان راه‌انداز کنترل می‌شود و معمولاً توسط یک دستور در فایل CONFIG.SYS یا AUTOEXEC.BAT نصب می‌شود. راه انداز باید نصب شود تا برنامه بتواند فعالیت‌های موس را تشخیص و پاسخ دهد. برخی واژه‌های تخصصی در رابطه با موس بشرح زیر می‌باشند.

- ***پیکسل*** : کوچکترین عنصر قابل آدرسدهی در یک صفحه را گویند. مثلاً برای حالت متن ۰۳ ، ۸ پیکسل در هر بایت وجود دارد.
- ***نشانه موس*** : در حالت متن یک بلوک کاراکتری با رنگ معکوس می‌باشد و در حالت گرافیک نشانه به صورت یک فلش می‌باشد.
- ***میکی*** : واحد اندازه‌گیری حرکت موس، می‌باشد. هر میکی تقریباً ۱/۲۰۰ اینچ می‌باشد.
- ***تعداد میکی*** :تعداد میکی‌هایی که توپی موس در جهت عمودی یا افقی حرکت می‌کند. راه‌انداز موس از تعداد میکی برای حرکت نشانه روی صفحه با تعداد خاصی پیکسل استفاده می‌کند.
- ***سرعت آستانه*** : حداکثر سرعت در واحد میکی در هر ثانیه برای موس را گویند. سرعت حرکت نشانه موس روی صفحه نمایش معمولاً دو برابر سرعت خود موس می‌باشد. پیش فرض ۶۴ میکی در هر ثانیه است. اعمال موس از طریق توابع استاندارد INT 33H و بصورت زیر انجام می‌شود.

```
MOV   AX,function      درخواست تابع موس ;
...                    در صورت وجود پارامترها ;
INT   33H              فراخوانی راه‌انداز موس ;
```

توجه کنید که بر خلاف دیگر عملیات INT که از ثبات AH استفاده می‌کنند، توابع INT 33H در *ثبات کامل* **AX** قرار می‌گیرند.

اولین دستور موس که یک برنامه باید صادر نماید تابع 00H است که رابط بین راه‌انداز موس و برنامه را مقدار دهی می‌نماید. معمولاً لازم است که این فرمان یکبار در ابتدای برنامه صادر گردد. بعد از تابع 00H، برنامه تابع 01H را اجرا می‌کند، که سبب می‌شود تا نشانهٔ موس در صفحه ظاهر گردد. بعد از آن، شما محدوده وسیعی از عملیات موس را می‌توانید اجرا نمایید.

توابع موس

در زیر، توابع INT 33H موس را که نسبتاً کاربرد کمی دارند مشاهده می‌کنید.

00H مقدار دهی اولیه موس	14H تعویض وقفه حوادث موس
01H نمایش نشانهٔ موس	15H گرفتن اندازه بافر برای وضعیت راه انداز موس
02H پنهان کردن نشانه موس	16H ذخیره وضعیت راه انداز موس
03H گرفتن وضعیت کلید موس و موقعیت نشانه	17H بازیابی وضعیت راه‌انداز موس
04H تنظیم موقعیت نشانه	18H نصب دستیاب متناوب برای حوادث موس
05H گرفتن اطلاعات کلید فشرده شده موس	19H گرفتن آدرس دستیاب متناوب
06H گرفتن اطلاعات کلید رها شده موس	1AH تنظیم حساسیت موس
07H تنظیم محدوده افقی نشانه	1BH گرفتن حساسیت موس
08H تنظیم محدوده عمودی نشانه	1CH تنظیم درجه وقفه موس
09H تنظیم نوع نشانه گرافیکی	1DH انتخاب صفحه نمایش برای نشانه
0AH تنظیم نوع نشانه متن	1EH گرفتن صفحه نمایش برای نشانه
0BH خواندن شمارشگرهای حرکت - موس	1FH ناتوان سازی راه انداز موس
0CH نصب دستیاب وقفه برای حوادث موس	20H فعال ساختن راه انداز موس
0DH روشن کردن قلم نوری	21H تنظیم مجدد راه انداز موس
0EH خاموش کردن قلم نوری	22H تنظیم زبان برای پیغام‌های راه انداز موس
0FH تنظیم نسبت میکی به پیکسل	23H گرفتن شماره زبان
10H تنظیم ناحیه منحصر نشانه	24H گرفتن اطلاعات موس
13H تنظیم آستانه دو برابر سرعت	

عملیات معمول INT 33H

در این بخش، بیشتر عملیات معمول INT 33H را بررسی می‌کنیم، که برای اغلب برنامه‌هایی که از موس استفاده می‌کنند لازم است.

تابع 00H : مقدار دهی اولیه موس

این اولین فرمانی است که برنامه جهت دستیابی یک موس صادر می‌کند، و فقط یک بار لازم است اجراشود. در AX تابع 00H قرار دهید و INT 33H را صادر کنید. این عملیات به پارامترهای ورودی نیاز ندارد ولی این مقادیر را باز می‌گرداند :

- AX = 0000H اگر هیچ موسی در دسترس نباشد یا FFFFH اگر پشتیبان موس در دسترس باشد.
- BX = تعداد کلیدهای موس (در صورت وجود موس)

اگر پشتیبان موس در دسترس باشد، عملیات راه انداز موس را به صورت زیر مقدار دهی می‌کند :

- تنظیم نشانه موس به مرکز صفحه
- پنهان کردن نشانه موس اگر دیده می‌شود
- تنظیم صفحه نمایش نشانه موس با صفر
- تنظیم نشانه موس بر طبق حالت صفحه : مربع مستطیل و رنگ معکوس در حالت متن یا به صورت فلش در حالت گرافیک
- تنظیم نسبت میکی به پیکسل، در جایی که درجه افقی = ۸ به ۸ و درجه عمودی = ۱۶ به ۸ می‌باشد.
- تنظیم محدوده افقی و عمودی برای نشانه با حداقل و حداکثر مقدار
- فعال ساختن شبیه ساز قلم فوری
- تنظیم سرعت آستانه تا ۶۴ میکی در هر ثانیه، که می‌توانید آن را تغییر دهید.

تابع 01H : نمایش نشانه موس

بعد از صدور تابع 00H از این عملیات برای نمایش نشانه موس در صفحه استفاده نمایید. عملیات به پارامتر ورودی نیاز ندارد و مقداری باز نمی‌گرداند.

راه انداز موس یک پرچم نشانه دارد که نمایش یا عدم نمایش نشانه را تعیین می‌کند، اگر این پرچم صفر باشد، نشانه نمایش نشانه داده می‌شود و اگر هر مقدار دیگری داشته باشد، نشان پنهان خواهد شد. در ابتدا مقدار آن ۱- است، تابع ۰۱ این پرچم را خواهد افزود، بنابراین نشانه ظاهر خواهد شد (تابع 02H را نیز ببینید)

تابع 02H، پنهان کردن نشانه موس

از جمله جنبه‌های کاربردهای این تابع، صدور فرمان در انتهای اجرای برنامه است، تا نشانه پنهان گردد. این عملیات هیچ پارامتر ورودی نیاز ندارد و هیچ مقداری باز نمی‌گرداند.

نشانه در صورتیکه پرچم آن حاوی صفر باشد، نمایش داده خواهد شد و اگر هر مقدار دیگری داشته باشد، پنهان می‌شود. این تابع پرچم را می‌افزاید تا آن را پنهان نماید.

تابع 03H : گرفتن وضعیت کلید موس و موقعیت نشانه

این تابع اطلاعات مفیدی راجع به موس باز می‌گرداند. به هیچ پارامتر ورودی نیاز ندارد ولی این مقادیر را باز می‌گرداند.

- BX = وضعیت کلید موس، بر طبق موقعیت بیت، بشرح زیر :

  بیت ۰ کلید چپ (۰ = بالا، ۱ = فشرده شد) بیت ۱ کلید راست (۰ = بالا، ۱ = فشرده شد)

  بیت ۲ کلید وسط (۰ = بالا، ۱ = فشرده شد) بیت ۳ - ۱۵ برای استفاده داخلی در نظر گرفته شده

- CX = موقعیت افقی (X)
- DX = موقعیت عمودی (Y)

موقعیت افقی و عمودی با عبارت ***پیکسل*** بیان می‌شود. حتی در حالت متن (در حالت ۰۳ ویدئو در هر بایت ۸) این مقادیر همیشه بین محدوده حداقل و حداکثر برای نشانه می‌باشد.

تابع 04H : تنظیم موقعیت نشانه

برای تنظیم موقعیت عمودی و افقی نشانه موس روی صفحه از این عملیات استفاده نمایید (مقدار موقعیت با

عبارت پیکسل - در حالت ۰۳ ویدئو را در هر بایت)

```
MOV   AX,04H          ; درخواست تنظیم اشاره‌گر موس
MOV   CX,Horiz_locn   ; موقعیت عمودی
MOV   DX,Vertl_locn   ; موقعیت افقی
INT   33H             ; فراخوانی سرویس موس
```

عملیات نشانه را در موقعیت جدید تنظیم می‌کند ، در صورتی که خارج از محدوده حداقل و حداکثر باشد آن را تصحیح می‌کند.

مثال : عملیات موس اصلی

مثال زیر ، نحوه استفاده از دستورات موس را که تا اینجا گفته شد، مشخص می‌سازد.

```
MOV   AX,00H          ; درخواست مقدار دهی موس
INT   33              ; فراخوانی راه انداز موس
CMP   AX,00H          ; موس در دسترس است؟
JE    exit            ; نه خارج شو
MOV   AX,01H          ; درخواست نمایش نشانه موس
INT   33H             ; فراخوانی راه انداز موس
MOV   AX,04H          ; درخواست تنظیم نشانه موس
MOV   CX,24H          ; موقعیت افقی
MOV   DX,16           ; موقعیت عمودی
INT   33H             ; فراخوانی راه‌انداز موس
...
MOV   AX,02H          ; درخواست پنهان کردن موس
INT   33H             ; فراخوانی راه انداز موس
```

تابع 05H : گرفتن اطلاعات فشردن کلید موس

از این تابع وقتی استفاده کنید که اطلاعاتی راجع به کلیدهای فشرده شده، می‌خواهید BX، را با شماره کلید تنظیم کنید، بطوریکه ۰ = چپ، ۱ = راست ۲ = مرکز :

```
MOV   AX,05H          ; درخواست اطلاعات فشردن
MOV   BX,button - no  ; شماره کلید
INT   33H             ; فراخوانی راه انداز موس
```

عملیات وضعیت بالا / پایین همه کلیدها و شمارش فشرده و موقعیت کلید درخواستی را باز می‌گرداند.

- AX = وضعیت کلیدها، بر طبق موقعیت بیت، بشرح زیر :

  بیت ۰ کلید چپ (۰ = بالا، ۱ = فشرده شده) بیت ۱ کلید راست (۰ = بالا، ۱ = فشرده شده)

  بیت ۲ کلید مرکز (۰ = بالا و ۱ = فشرده شده) بیت۳-۱۵ برای استفاده داخلی رزرو شده

- BX = شمارشگر کلیدهای فشرده
- CX = موقعیت افقی (X) آخرین کلید فشرده
- DX = موقعیت عمودی (Y) آخرین کلید فشرده

عملیات شمارشگر کلید فشرده شده را صفر می‌کند

تابع 06H : گرفتن اطلاعات کلیدهای رها شده

برای استفاده از این تابع جهت باز گرداندن اطلاعاتی راجع به کلیدهای رها شده، BX را با شماره کلید تنظیم کنید

(۰ = چپ، ۱ = راست و ۲ = مرکز)

```
MOV   AX,06H          ; درخواست اطلاعات کلید رها شده
MOV   BX,button - no  ; شماره کلید
INT   33H             ; فراخوانی راه انداز موس
```

عملیات وضعیت بالا / پایین همه کلیدها و شمارش رها شدن آنها و موقعیت کلید مورد نظر را بشرح زیر باز می‌گرداند.

- AX = وضعیت کلیدها، بر طبق موقعیت بیت، بشرح زیر :

بیت ۰ کلید چپ (۰ = بالا، ۱ = فشرده شده) بیت ۱ کلید راست (۰ = بالا، ۱ = فشرده شده)

بیت ۲ کلید وسط (۰ = بالا، ۱ = فشرده شده) بیت‌های ۳-۱۵ برای استفاده‌های داخلی رزرو شده

- BX = شمارشگر کلید رها شده
- CX = موقعیت افقی (X) آخرین کلید رها شده
- DX = موقعیت عمودی (Y) آخرین کلید رها شده

این عملیات، شمارشگر رها شدن کلید را با صفر مجدداً تنظیم می‌کند.

تابع 07H : تنظیم محدوده افقی برای نشانه

می‌توانید از این عملیات برای تنظیم حداقل و حداکثر محدوده افقی نشانه استفاده کنید.

```
MOV   AX,07H          ; درخواست تنظیم محدوده افقی
MOV   CX,min_locn     ; حداقل
MOV   DX,max_locn     ; حداکثر
INT   33H             ; فراخوانی راه انداز موس
```

اگر مقدار حداقل بیشتر از حداکثر باشد، عملیات این دو مقدار را تعویض می‌کند. در صورت لزوم، عملیات نشانه را در محدوده جدید حرکت می‌دهد. توابع 08H و 10H را نیز ببینید.

تابع 08H : تنظیم محدوده عمودی نشانه

می‌توانید از این عملیات جهت تنظیم حداقل و حداکثر محدوده عمودی نشانه استفاده کنید.

```
MOV   AX,08H          ; درخواست تنظیم محدوده عمودی
MOV   CX,min_locn     ; حداقل
MOV   DX,max_locn     ; حداکثر
INT   33H             ; فراخوانی راه انداز موس
```

اگر حداقل مقدار از حداکثر بیشتر باشد، عملیات این مقادیر را تعویض می‌کند. در صورت لزوم، عملیات نشانه را داخل ناحیه جدید حرکت می‌دهد. تابع 07H و 10H را نیز ببینید.

تابع 0BH : خواندن شمارشگرهای حرکت موس

این عملیات شمارش عمودی و افقی را در زمان آخرین فراخوانی تابع، باز می‌گرداند (داخل محدوده ۳۲۷۶۸- تا ۳۲۷۶۷+). مقادیر بازگشتی چنین هستند.

CX = شمارش افقی (یک مقدار مثبت، یعنی حرکت به راست ، مقدار منفی یعنی حرکت به چپ)

DX = شمارش عمودی(یک مقدار مثبت، یعنی حرکت به پایین، مقدار منفی یعنی حرکت به بالا)

تابع 0CH : نصب دستیاب وقفه برای حوادث موس

برنامه شما ممکن است، لازم باشد وقتی یک فعالیت مرتبط با موس رخ می‌دهد بطور خودکار تعیین نماید. هدف از تابع OCH فراهم ساختن ***دستیاب حوادث*** است بطریقی که نرم‌افزار موس برنامه شما وقفه بدهد و دستیاب حوادث را فراخوانی کند، که تابع مورد نیاز را انجام دهد و با تکمیل کار به نقطه اجرای برنامه شما باز گردد.

در CX یک پوشش حوادث مبنی بر فعالیت‌هایی که دستیاب باید پاسخگو باشد قرار دهید و در ES:DX آدرس آفست سگمنت از روتین دستیاب وقفه را قرار دهید :

```
MOV   AX,0CH      ; درخواست دستیاب وقفه
LEA   CX,mask     ; آدرس ماسک حوادث
LEA   DX,handler  ; آدرس دستیاب (ES:DX)
INT   33H         ; فراخوانی راه انداز موس
```

ماسک حوادث با تنظیم بیت‌هایی که لازم است را تعریف کنید :

۰ = نشانه موس حرکت کرد	۴ = کلید سمت راست رها شد
۱ = کلید سمت چپ فشرده شد	۵ = کلید وسط فشرده شد
۲ = کلید سمت چپ رها شد	۶ = کلید وسط رها شد
۳ = کلید سمت راست فشرده شد	۷ = رزرو شده، بصورت ۰ تعریف می‌شود

دستیاب وقفه را بصورت یک روال FAR تعریف کنید. راه‌انداز موس از یک فراخوانی دور برای ورود به دستیاب وقفه با این تنظیم ثبات‌ها استفاده می‌کند.

- AX = ماسک حوادث بطوریکه تعریف شد، بجز آنکه بیت‌ها فقط وقتی شرایط رخ دهد تنظیم می‌شوند.
- BX = وضعیت کلیدها (اگر تنظیم باشد، بیت ۰ یعنی کلید سمت چپ فشرده شده، بیت ۱ یعنی کلید راست فشرده شده، و بیت ۲ یعنی کلید وسط فشرده شده)
- CX = موقعیت افقی (X)
- DX = موقعیت عمودی (Y)
- SI = آخرین شماره عمودی میکی
- DI = آخرین شماره افقی میکی
- DS = سگمنت داده برای راه انداز موس

در ورودی برنامه به دستیاب وقفه، همه ثبات‌ها را در پشته قرار دهید و ثبات DS را با آدرس سگمنت داده خود مقدار دهی نمایید. داخل دستیاب، فقط از BIOS استفاده کنید نه وقفه‌های DOS در هنگام خروج، همه ثبات‌ها را از روی پشته بردارید.

تابع 10H : تنظیم ناحیه منحصری نشانه

این عملیات یک ناحیه صفحه را طوری تعریف می‌کند که نشانه در آن دیده نمی‌شود :

```
MOV   AX,10H          ; درخواست تنظیم ناحیه انحصاری
MOV   CX,Upleft_X     ; موقعیت X بالای سمت چپ
MOV   DX,upleft_Y     ; موقعیت Y در بالای سمت چپ
MOV   SI,lowrgt_X     ; موقعیت X پایین سمت راست
MOV   DI,lowrgt_Y     ; موقعیت Y پایین سمت راست
INT   33H             ; فراخوانی راه انداز موس
```

برای جایگزینی ناحیه انحصاری، این تابع را با پارامترهای متفاوت فراخوانی کنید، یا تابع 00H یا 01H را مجدداً صادر نمایید.

تابع 13H : تنظیم آستانه سرعت دو برابر

این تابع سرعت آستانه را تنظیم می‌کند بطوریکه حرکت نشانه روی صفحه دو برابر خواهد شد. در DX مقدار جدید را قرار دهید (پیش فرض ۶۴ میکی در ثانیه است). (تابع 1AH را نیز ببینید)

تابع 1AH : تنظیم حساسیت موس

حساسیت یعنی تعداد میکی‌هایی که موس نیز دارد تا حرکت کند قبل از آنکه نشانه حرکت کند. تابع 1AH حرکت افقی و عمودی موس را با عبارت تعداد میکی در هر ۸ پیکسل تنظیم می‌کند، بطوریکه سرعت آستانه که نشانه روی صفحه حرکت می‌کند دو برابر می‌شود (توابع 0FH ، 13H و 1BH را ببینید)

```
MOV   AX,1AH          درخواست تنظیم حساسیت موس ;
MOV   BX,horzon       میکی افقی (پیش فرض ۸) ;
MOV   CX,vertic       میکی عمودی (پیش فرض ۱۶) ;
MOV   CX,threshold    آستانه سرعت (پیش فرض ۶۴) ;
INT   33H             فراخوانی راه انداز موس ;
```

تابع 1BH : گرفتن حساسیت موس

این عملیات حرکت افقی و عمودی موس را با عبارت تعداد میکی در هر ۸ پیکسل باز می‌گرداند، بطوریکه سرعت آستانه حرکت نشانه صفحه دو برابر خواهد بود، (تابع 1AH را برای ثبات‌ها و مقادیر بازگشتی ببینید).

تابعی 1DH : انتخاب صفحه نمایش برای نشانه

صفحه نمایش ویدئو با INT 10H تابع 05H تنظیم می‌شود. برای عملیات موس، شماره صفحه را در BX تنظیم کنید و INT 33H تابع 1DH صادر نمایید.

تابع 1EH : گرفتن صفحه نمایش برای نشانه

این تابع صفحه جاری نمایش ویدئو را در BX باز می‌گرداند.

تابع 24H : گرفتن اطلاعات موس

این عملیات اطلاعاتی راجع به نسخه و نوع موس نصب شده باز می‌گرداند.

- BH = شمار نسخه اصلی
- BL = شماره نسخه فرعی
- CH = نوع موس (۱ = موس گذرگاه، ۲ = موس سریال، ۳ = موس درگاه ورودی، ۴ = موس PS/2 و ۵ = موس HP)

برنامه : استفاده از موس

برنامه شکل ۱-۲۱ موقعیت‌های افقی و عمودی نشانه را با حرکت دادن موس توسط کاربر نمایش می‌دهد روال‌های اصلی چنین هستند :

- A10MAIN برنامه را شروع می‌کند، B10INIT ، C10PTK ، D10CONV و Q30DISP را فرا می‌خواند و پردازش را وقتی کاربر کلید سمت چپ را بفشارید، خاتمه می‌دهد.
- B10INIT یک INT 33H تابع 00H صادر می‌کند تا موس را مقدار دهی اولیه نماید (یا دلالت می‌کند بر اینکه هیچ

راه انداز موس حاضر نیست) و یک تابع 01H صادر می‌کند تا نشانه موس نمایش داده شود.

- C10PTR اگر کاربر کلید سمت چپ را فشرده باشد، تابع 03H را برای بررسی و خروج صادر می‌کند اگر فشرده نباشد، برنامه موقعیت افقی و عمودی را از مقادیر پیکسل به اعداد دودویی تبدیل می‌کند (با شیفت دادن ۳ بیت به راست ، در حقیقت تقسیم بر ۸). اگر موقعیت با همان موقعیت بررسی شده قبلی یکسان باشد ، روتین مجدداً تابع 03H را صادر می‌نماید، اگر موقعیت تغییر کرده باشد، کنترل به روال فرا خواننده باز می‌گردد.

```
            page 60,132
TITLE       A21MOUSE (EXE)  Handling the mouse
            .MODEL SMALL
            .STACK 64
            .DATA
XBINARY     DW      0                   ;Binary X coordinate
YBINARY     DW      0                   ;Binary Y coordinate
ASCVAL      DW      ?                   ;ASCII field

;           Screen display fields:
DISPDATA    LABEL   BYTE
XMSG        DB      'X = '              ;X message
XASCII      DW      ?                   ;X ASCII value
            DB      '  '                ;
YMSG        DB      'Y = '              ;Y message
YASCII      DW      ?                   ;Y ASCII value
.286
            .CODE
A10MAIN     PROC    FAR
            MOV     AX,@data            ;Initialize
            MOV     DS,AX               ;  DS register
            CALL    Q10CLEAR            ;Clear screen
            CALL    B10INIT             ;Initialize mouse
            CMP     AX,00               ;Mouse installed?
            JE      A90                 ;  no, exit
A20:
            CALL    C10PTR              ;Get mouse pointer
            CMP     BX,01               ;Button pressed?
            JE      A80                 ;  yes, exit
            CALL    Q20CURS             ;Set cursor
            MOV     AX,XBINARY          ;
            CALL    D10CONV             ;X to ASCII
            MOV     AX,ASCVAL           ;
            MOV     XASCII,AX           ;
            MOV     AX,YBINARY          ;
            CALL    D10CONV             ;Y to  ASCII
            MOV     AX,ASCVAL           ;
            MOV     YASCII,AX           ;
            CALL    Q30DISP             ;Display X and Y values
            JMP     A20                 ;Repeat
A80:
            CALL    E10HIDE             ;Hide mouse pointer
A90:
            CALL    Q10CLEAR            ;Clear screen
            MOV     AX,4C00H            ;End processing
            INT     21H
A10MAIN     ENDP
;           Initialize mouse pointer:
;           -------------------------
B10INIT     PROC    NEAR
            MOV     AX,00H              ;Request initialize
            INT     33H                 ;  mouse
            CMP     AX,00               ;Mouse installed?
            JE      B90                 ;  no, exit
            MOV     AX,01H              ;Show pointer
            INT     33H
B90:
            RET                         ;Return to caller
B10INIT     ENDP
```

شکل الف ۱-۲۱ استفاده از موس

```
;                       Get mouse pointer location:
;                       ---------------------------
C10PTR      PROC        NEAR
C20:        MOV         AX,03H              ;Get pointer location
            INT         33H
            CMP         BX,01               ;Right button pressed?
            JE          C90                 ;  yes, means exit
            SHR         CX,03               ;Divide pixel value
            SHR         DX,03               ;  by 8
            CMP         CX,XBINARY          ;Has pointer location
            JNE         C30                 ;  changed?
            CMP         DX,YBINARY          ;
            JE          C20                 ;  no, repeat operation
C30:        MOV         XBINARY,CX          ;  yes, save new location
            MOV         YBINARY,DX          ;
C90:        RET                             ;Return to caller
C10PTR      ENDP
;                       Convert binary to ASCII:
;                       ------------------------
D10CONV     PROC        NEAR                ;AX = binary X or Y
            MOV         ASCVAL,2020H        ;Clear ASCII field
            MOV         CX,10               ;Set divide factor
            LEA         SI,ASCVAL+1         ;Load ASCVAL address
            CMP         AX,CX               ;Compare location to 10
            JB          D20                 ;  lower, bypass
            DIV         CL                  ;  higher, divide by 10
            OR          AH,30H              ;Insert ASCII 3s
            MOV         [SI],AH             ;Store in rightmost byte
            DEC         SI                  ;Decr address of ASCVAL
D20:        OR          AL,30H              ;Insert ASCII 3s
            MOV         [SI],AL             ;Store in leftmost byte
            RET                             ;Return to caller
D10CONV     ENDP
;                       Hide mouse pointer before ending:
;                       ---------------------------------
E10HIDE     PROC        NEAR
            MOV         AX,02H              ;Request hide pointer
            INT         33H
            RET                             ;Return to caller
E10HIDE     ENDP
;                       Screen operations:
;                       ------------------
Q10CLEAR    PROC        NEAR
            MOV         AX,0600H            ;Request clear screen
            MOV         BH,30H              ;Colors
            MOV         CX,00               ;Full
            MOV         DX,184FH            ;  screen
            INT         10H
            RET                             ;Return to caller
Q10CLEAR    ENDP

Q20CURS     PROC        NEAR
            MOV         AH,02H              ;Set cursor
            MOV         BH,0                ;Page 0
            MOV         DH,0                ;Row
            MOV         DL,25               ;Column
            INT         10H
            RET                             ;Return to caller
Q20CURS     ENDP
Q30DISP     PROC        NEAR
            MOV         AH,40H              ;Request display
            MOV         BX,01               ;Screen handle
            MOV         CX,14               ;Number of characters
            LEA         DX,DISPDATA         ;Display area
            INT         21H
            RET                             ;Return to caller
Q30DISP     ENDP
            END         A10MAIN
```

شکل ب ۱-۲۱ استفاده از موس

- D10CONV مقادیر دودویی را برای موقعیت صفحه افقی و عمودی به کاراکترهای ASCII قابل نمایش تبدیل می‌کند. توجه کنید که با ۸ پیکسل در هر بایت ، مقدار افقی بازگشتی در صفحه ستون ۷۹ (موقعیت سمت راست صفحه) خواهدبود ۶۳۲ = ۸ × ۷۹، روال این‌مقدار را بر ۸ تقسیم‌می‌کند تا آن را بدست آورد. دراین حالت ۷۹ حالت حداکثر می‌باشد. در نتیجه، تبدیل وقتی می‌تواند صحیح در نظر گرفته شود که مقادیر بازگشتی بین ۰ تا ۷۹ باشد.
- E10HDE نشانه را بلافاصله قبل از خاتمه پردازش برنامه پنهان می‌سازد.
- Q30DISP مقادیر افقی و عمودی را در مرکز صفحه بصورت ستون = X و سطر = Y نمایش می‌دهد. یک روش گسترش برنامه صدور تابع 0CH برای تنظیم دستیاب وقفه است. با این روش، دستورات مورد نیاز، بطور خودکار هر جا که موس فعال باشد ، احضار خواهند شد.

درگاه‌ها

یک ***درگاه*** ابزاری است که پردازشگر را با دنیای خارج مرتبط می‌سازد. از داخل یک درگاه، پردازشگر علائم را از ابزار ورودی دریافت می‌کند و یک علامت به ابزار خروجی می‌فرستد. درگاه‌ها با آدرس آنها مشخص می‌شوند، محدودهٔ 0H تا 3FFH یا ۱۰۲۴ درگاه در کل وجود دارد. توجه کنید که این آدرسها، آدرس حافظه مرسوم نیستند. می‌توانید از دستورات IN و OUT مستقیماً از سطح درگاه I/0 را دستیابی نمایید.

IN داده را از یک درگاه ورودی به AL منتقل می‌کند اگر یک بایت باشد و در AX اگر یک کلمه باشد، در حالیکه OUT داده‌ها از AL به درگاه خروجی منتقل می‌کند اگر یک بایت باشد و اگر یک کلمه باشد از AX منتقل می‌کند. شکل کلی چنین است :

[label:]	IN	accum-reg,port
[label:]	OUT	port , accum-reg

شما می‌توانید آدرس پورت بصورت ثابت و یا پویا مشخص کنید :

ثابت. یک عملوند از ۰ تا ۲۵۵ مستقیماً بصورت زیر استفاده نمایید

```
Input   IN    AL,port #   ; وارد کردن یک بایت از درگاه
Output  OUT   port #,ax   ; خارج کردن یک کلمه به درگاه
```

پویا. از محتویات ثبات DX بطور غیر مستقیم از ۰ تا ۶۵۵۳۵ استفاده کنید. می‌توانید از این روش برای پردازش درگاه‌های متوالی با افزایش آدرس در DXاستفاده نمایید.

مثال زیر از درگاه 60H استفاده می‌کند :

```
MOV   DX,60H   ; درگاه 60H (صفحه کلید)
INT   AL,DX    ; گرفتن بایت از درگاه
```

برخی از آدرس‌های درگاه اصلی چنین هستند :

020H-023H ثبات‌های ماسک وقفه	378H-37FH تطبیق دهنده درگاه موازی LPT2
040H-043H زمان سنج / شمارشگر	3B0H-3BBH تطبیق دهنده نمایش تک رنگ
060H ورودی از صفحه کلید	3BCH-3BFH تطبیق دهنده درگاه موازی LPT1
061H سخنگو (بیت‌های ۰ و ۱)	VGA/EGA 3C0H-3CFH
200H-20FH کنترلر بازی	3D0H-3DFH تطبیق دهنده گرافیکی رنگی (CGA)
278H-27FH تطبیق دهنده درگاه موازی LPT3	3F0H-3F7H کنترلر دیسک
2F8H-2FFH درگاه سریال COM2	3F8H-3FFH درگاه سریال COM1

اگر چه تجربه استاندارد استفاده از وقفه‌های DOS و BIOS می‌باشد، ممکن است، وقتی درگاه‌های 21H ، 40-42H ، 60H ، 61H و 201H دستیابی می‌کند BIOS را کنار بگذارید برای مثال، زمان راه‌اندازی، یک روتین

ROM BIOS سیستم را برای آدرس تطبیق دهنده‌های درگاه سریال و موازی پیمایش می‌کند. اگر آدرسهای درگاه سریال پیدا شود، BIOS آن را در ناحیه داده خود با شروع از موقعیت حافظه 40.00H قرار می‌دهد. هر موقعیت، فضایی برای چهار ورودی یک کلمه‌ای دارد. جدول BIOS برای سیستمی با دو درگاه سریال و دو درگاه موازی مشابه این خواهد بود.

40:00	F803	COM1	40:08	7803	LPT1
40:02	F802	COM2	40:0A	7802	LPT2
40:04	0000	unused	40:0C	0000	Unused
40:06	0000	unused	40:0E	0000	unused

برای مثال، برای استفاده از BIOS INT 17H جهت چاپ یک کاراکتر ، شماره درگاه چاپگر را در ثبات DX جایگزین کنید :

```
MOV  AX,00H      ; درخواست چاپ
MOV  AL,char     ; کاراکتر چاپ
MOV  DX,0        ; LPT1 = 0 درگاه چاپگر
INT  17H         ; فراخوانی سرویس وقفه
```

برخی برنامه‌ها، چاپ کردن را فقط از طریق درگاه LPT1 مجاز می‌دانند. اگر شما دو چاپگر نصب شده بصورت LPT1 و LPT2 داشته باشید، می‌توانید از برنامه شکل ۲-۲۱ جهت معکوس نکردن آدرس آنها در جدول BIOS استفاده نمایید. در برنامه، BIOSDAT ناحیه داده BIOS را تعریف می‌کند و PARLPRT اولین چهار آدرس درگاه با اندازه کلمه را تعریف می‌کند.

رشته ورودی / خروجی

شما می‌توانید داده‌ها را توسط دستورات رشته‌ای (در ۸۰۲۸۶ و ما بعد) INSn و outn منتقل می‌کند. این کار بسیار شبیه دستورات رشته‌ای مطرح شده در فصل ۱۲ می‌باشد.

```
            page 60,132
TITLE       A21PORT (COM)  Switch printer ports LPT1 & 2
BIOSDAT     SEGMENT AT 40H          ;BIOS data area
            ORG     8H              ;Printer port addresses
PARLPRT     DW      4 DUP(?)        ;4 words
BIOSDAT     ENDS

CODESG      SEGMENT PARA 'code'
            ASSUME DS:BIOSDAT,CS:CODESG
            ORG     100H
BEGIN:
            MOV     AX,BIOSDAT      ;Init address of BIOS
            MOV     DS,AX           ;  data area in DS

            MOV     AX,PARLPRT(0)   ;LPT1 address to AX
            MOV     BX,PARLPRT(2)   ;LPT2 address to BX
            MOV     PARLPRT(0),BX   ;Exchange
            MOV     PARLPRT(2),AX   ;  addresses
            MOV     AX,4C00H        ;End processing
            INT     21H
CODESG      ENDS
            END     BEGIN
```

شکل ۲-۲۱ راه انداز درگاه‌های چاپگر

دستورات INSn

دستورات برای عملیات INSn چنین است :

دستورات	مثال
INS ES:destination,DX	INS
REP INSB	INSB
REP INSW	INSW
REP INSD	INSD (80386+)

داده‌های دریافتی (یا مقصد) رشته‌ای هستند که با ES:DI آدرسدهی می‌شوند و DX حاوی آدرس درگاه ورودی است. تمرین معمولی استفاده از INSn با پیشوند REP است و CX حاوی تعداد عناصر دریافتی است (بایت‌ها، کلمات، یا دو کلمه‌ای‌ها). اگر پرچم جهت (DF) صفر باشد، DI به اندازه هر عنصر دریافتی افزوده خواهد شد، اگر DF یک باشد، DI کاسته خواهد شد.

مثال زیر، عملیات INSn را مشخص می‌سازد :

```
MOV   CX,no_bytes          ; تعداد بایت‌ها
LEA   DI,destination       ; رشته مقصد (ES :DI)
MOV   DX,port -no          ; شماره درگاه
REP   INSB                 ; بایت‌های دریافتی
```

دستور OUTsn

دستورات عملیات OUTsn چنین هستند :

دستورات	مثال
OUTS	OUTS DX, DS:source
OUTSB	REP OUTSB
OUTSW	REP OUTSW
OUTSD (80386+)	REP OUTSD

داده‌های ارسالی (یا منبع) رشته‌ای آدرسدهی شده با DS:SI و DX حاوی آدرس درگاه خروجی است. تجربه معمولی برای استفاده از OUTsn با پیشوند REP می‌باشد و CX حاوی تعداد عناصری است که باید فرستاده شود (بایت‌ها، کلمات، یا دو کلمه‌ای‌ها). اگر پرچم جهت (DF) صفر باشد، SI با اندازه هر عنصر دریافتی افزوده خواهد شد، اگر DF یک باشد، SI کاسته خواهد شد مثال زیر عملیات OUTSn را مشخص می‌سازد :

```
MOV   CX,no _bytes         ; تعداد بایت‌ها
LEA   SI,source            ; رشته مقصد (DS:SI)
MOV   DX,port_no           ; شماره درگاه
REP   OUTSB                ; ارسال بایت‌ها
```

تولید صدا

کامپیوترهای شخصی صدا را به وسیله یک بلندگوی مغناطیسی دائمی درونی تولید می‌کنند. یکی از دو راه یا ترکیبی از هر دو راه را می‌توانید انتخاب کنید : (۱) استفاده از بیت ۱ از درگاه 61H برای فعال کردن تراشه رابط جانبی قبل برنامه‌ریزی 5 - 8255A اینتل به کار ببرید یا (۲) از زمان سنج قابل برنامه‌ریزی 5 - 8353 اینتل استفاده کنید. ساعت، یک سیگنال ۱.۱۹۳۱۸ مگاهرتزی (MHZ) تولید می‌کند.

برنامهٔ شکل ۳-۲۱ دنباله از نت‌ها در فرکانس صعودی تولید می‌کند. DURTION طول هرنت را مهیا می‌سازد و TONE تعیین کننده فرکانس آن است. برنامه ابتدا گذرگاه 61H را دستیابی می‌کند و ارزشی را که عمل ارائه می‌دهد، ذخیره می‌سازد. دستور CLI پرچم وقفه را برای فعال کردن یک آهنگ ثابت پاک می‌کند. زمان سنج مدت ۱۸۰۲ تیک ساعت در ثانیه تولید می‌کند (مگر آنکه CLI را بنویسید) که اجرای برنامه را معوق می‌گذارد و سبب می‌گردد تا آهنگ نامنظم گردد. محتویات TONE فرکانس را تعیین می‌کند. مقادیر بزرگ سبب فرکانس‌های کم و مقادیر پایین سبب فرکانس‌ها زیاد می‌شود. بعد از آنکه روال B10SPKB هر آهنگی را می‌نوازد فرکانس TONE را با انجام یک عمل یک بیت شیفت به راست (تقسیم بر دو) می‌افزاید. چون افزودن TONE در این مثال مدتی را که نواخته می‌شود، می‌کاهد، روال نیز DURTION را به وسیله انجام تغییر مکان یک بیت به طرف چپ (ضرب در دو) می‌افزاید. وقتی TONE

صفر شود برنامه خاتمه می‌پذیرد. مقادیر اولیه DURTION و TONE دارای هیچ مفهوم تکنیکی نیستند. می‌توانید با سایر مقادیر امتحان کنید و اجرای برنامه را بدون CLI پیگیری کنید. می‌توانید هر نوع منطقی را برای نواختن دنباله‌ای از نت‌ها، مثلاً برای جلب توجه کاربر، بکار ببرید. همچنین می‌توانید برنامه را مانند پرسش ۷ - ۲۱ بازنویسی کنید.

```
            page 60,132
TITLE       A21SOUND (COM)  Produce sound from speaker
SOUNSG      SEGMENT PARA 'Code'
            ASSUME CS:SOUNSG,DS:SOUNSG,SS:SOUNSG
            ORG    100H
BEGIN:      JMP    SHORT A10MAIN
; ----------------------------------------------
DURTION     DW     10000            ;Length of tone
TONE        DW     512H             ;Frequency
; ----------------------------------------------
A10MAIN     PROC.  NEAR
            IN     AL,61H           ;Get port data
            PUSH   AX               ;  and save
            CLI                     ;Clear interrupts
            CALL   B10SPKR          ;Produce sound
            POP    AX               ;Reset
            OUT    61H,AL           ;  port value
            STI                     ;Reset interrupts
            MOV    AX,4C00H         ;End
            INT    21H              ;  processing
A10MAIN     ENDP

B10SPKR     PROC   NEAR
B20:        MOV    DX,DURTION       ;Set duration of sound
B30:
            AND    AL,11111100B     ;Clear bits 0 & 1
            OUT    61H,AL           ;Transmit to speaker
            MOV    CX,TONE          ;Set length
B40:
            LOOP   B40              ;Time delay
            OR     AL,00000010B     ;Set bit 1 on
            OUT    61H,AL           ;Transmit to speaker
            MOV    CX,TONE          ;Set length
B50:
            LOOP   B50              ;Time delay
            DEC    DX               ;Reduce duration
            JNZ    B30              ;Continue?
            SHL    DURTION,1        ;  no, increase length
            SHR    TONE,1           ;Reduce frequency
            JNZ    B20              ;Now zero?
            RET                     ;  yes, return
B10SPKR     ENDP

SOUNSG      ENDS
            END    BEGIN
```

شکل ۳-۲۱ تولید صدا

نکات کلیدی

- در حالت متن، نشانه موس یک بلوک نورانی، با نمایش معکوس صفحه است و در حالت گرافیکی نشانه یک سر فلش است.
- عملیات موس از INT 33H استفاده می‌کنند، با کدهای تابعی که در AX قرار می‌گیرند.
- اولین عملیات موس که باید اجرا شود تابع 00H است، که راه انداز موس را مقدار دهی اولیه می‌نماید.
- تابع 01H برای نمایش نشانه موس لازم است، 03H برای گرفتن وضعیت کلید و 04H برای گرفتن موقعیت نشانه می‌باشد.

- از یک درگاه ،پردازشگر یک علامت از ابزار ورودی دریافت می‌کند و یک علامت به ابزار خروجی ارسال می‌دارد. درگاه‌ها با آدرس مشخص می‌شوند، که این آدرس در محدوده 3FFH0H یا در مجموع 1024 می‌باشد.
- کامپیوترهای شخصی صدا را توسط یک بلندگوی مغناطیسی دائمی درونی تولید می‌کنند، می‌توانید یکی از دو روش راه‌اندازی بلندگو یا ترکیبی از هر دو راه را، استفاده نمایید.

پرسش‌ها

۱-۲۱. این عبارات را توضیح دهید :

(الف) میکی

(ب) شمارش میکی

(ج) نشانهٔ موس.

۲-۲۱. تابع INT 33H برای هر یک از عملیات موس زیر را فراهم سازید :

(الف) پنهان کردن نشانه موس

(ب) گرفتن اطلاعات کلید فشرده شده

(ج) تنظیم موقعیت نشانه

(د) نصب دستیاب وقفه برای حوادث موس

(ه) گرفتن اطلاعات کلید فشرده شده

(و) خواندن شمارشگرهای حرکت موس

۳-۲۱. هدف از پرچم نشانه موس را توضیح دهید.

۴-۲۱. دستورات لازم برای موارد زیر را بنویسید :

(الف) مقدار دهی اولیه موس

(ب) نمایش اشاره‌گر موس

(ج) گرفتن اطلاعات موس

(د) تنظیم اشاره‌گر موس در سطر ۱۶، ستون مرکزی

(ه) گرفتن حساسیت موس

(و) گرفتن وضعیت کلید و موقعیت اشاره‌گر

(ز) پنهان ساختن نشانه موس

۵-۲۱. موارد ذکر شده در پرسش ۴-۲۱ را در یک برنامه کامل ترکیب کنید. می‌توانید برنامه را تحت DEBUG اجرا کنید، اگر چه در برنامه DEBUG نشانه روی صفحهٔ پنهان خواهد بود.

۶-۲۱. به شکل ۲-۲۱ توجه کنید و دستورات را طوری بازنویسی کنید که برنامه آدرس COM1 و COM2 را معکوس کند.

۷-۲۱. برنامه شکل ۳-۲۱ را برای موارد زیر بازنویسی کنید : نت‌هایی تولید کنید که فرکانس آن‌ها کاهش یابد، TONE را با ۰۱ و PURTION را یک مقدار زیاد، مقدار دهی کنید. در هر حلقه، مقدار TONE را بیفزایید مقدار DURTION را بکاهید تا وقتی DOURTION مساوی صفر شود، برنامه خاتمه پذیرد.

فصل ۲۲

تعریف و استفاده از ماکروها

هدف: توضیح نحوه تعریف استفاده از دستورات ماکروها

مقدمه

برای هر دستور سمبولیک، اسمبلر یک دستور زبان ماشین تولید می‌کند. به عبارت دیگر برای هر دستور نوشته شده به یک زبان سطح بالا مانند C یا BASIC، کمپایلر ممکن است، تعداد زیادی دستورات زبان ماشین تولید کند. بدین ترتیب می‌توان یک زبان سطح بالا را، شامل یک مجموعه از جملات ***ماکرو*** در نظر بگیرید.

اسمبلر نیز وسائلی برای تعریف ماکروها دارد. یک نام منحصر بفرد برای ماکرو، با مجموعه‌ای از دستورات زبان اسمبلی که ماکرو تولید می‌نماید تعریف می‌شود. در هر جا که نیاز به مجموعه دستورات مربوطه داشتید، کافی است نام ماکرو را بنویسید و اسمبلر بطور خودکار دستوراتی که تعریف شده، تولید می‌کند. ماکروها جهت اهداف زیر مفید می‌باشند.

- ساده سازی و کاهش میزان دستورات تکراری.
- کاستن خطاهایی که ممکن است در دستورات تکراری پدید آیند.
- افزایش خوانایی برنامه اسمبلی.

مثالهایی از توابعی که ممکن است با ماکروها انجام داد، عملیات ورودی خروجی است که ثبات‌ها را بارگذاری نموده و وقفه‌ها را انجام می‌دهد، تبدیل داده‌های ASCII به دودویی، عملیات محاسباتی چند کلمه‌ای و روتین‌های دستکاری رشته‌ها.

در اینجا شکل کلی تعریف ماکرو آورده شده است :

mocroname	MACRO [parameter List]	; Define macro
	[instrction]	; Body of macro
	ENDM	; End of macro

پیش‌پردازنده MACRO در خط اول به اسمبلر می‌فهماند دستوراتی‌که تا ENDM خواهندبود، بخشی از تعریف یک ماکرو هستند. پیش‌پردازنده ENDM (انتهای ماکرو) تعریف ماکرو را خاتمه می‌دهد. دستورات بین MACRO و ENDM شامل بدنه تعریف ماکرو هستند.

برای آنکه برنامه شما یک ماکرو داشته باشد، ابتدا آن را تعریف کنید یا از کتابخانه ماکرو آن را کپی نمایید تعریف ماکرو قبل از دستورات هر سگمنت باید قرار گیرد.

دو تعریف ماکروی ساده

ابتدا یک تعریف ماکرو ساده که ثبات‌های سگمنت را برای یک برنامه EXE. مقدار دهی می‌کند بررسی کنیم :

```
INITZ  MACRO            ; تعریف ماکرو
       MOV   AX,@data   ; بدنه
       MOV   DS,AX      ; تعریف
       MOV   ES,AX      ; ماکرو
       ENDM             ; انتهای ماکرو
```

نام این ماکرو INTZ است، اما هر نام معتبر منحصر بفرد دیگری نیز قابل پذیرش خواهد بود. نامی که در تعریف ماکرو ارجاع می‌شود - ES ، DS ، AX و data @ - باید در جایی از برنامه تعریف شوند یا برای اسمبلر شناخته شده باشند.

ممکن است بعداً از دستور ماکرو INITZ در سگمنت کد هر جا که ثبات‌ها را مقداردهی می‌کنید، استفاده نمایید. وقتی اسمبلر به دستور ماکرو INITZ می‌رسد، جدول دستورات سمبولیک را پیمایش می‌کند و در صورت پیدا نکردن یک ورودی، دستورات ماکرو را بررسی می‌کند. چون برنامه شامل یک تعریف از ماکرو INITZ است، اسمبلر بدنهٔ تعریف را جایگزین کرده دستورات مورد نظر تولید می‌شوند - **بسط ماکرو** یک برنامه ممکن است از دستور ماکرو INITZ فقط یکبار استفاده کند، اگر چه ماکروها طوری طراحی شده‌اند که هر تعداد دفعه قابل استفاده‌اند و هر بار اسمبلر بسط ماکرو را تولید می‌کند.

بیایید ماکرو دوم با نام FINISH را تعریف کنیم که خروج طبیعیٔ از یک برنامه را پیاده سازی می‌کند :

```
FINISH MACRO            ; تعریف ماکرو
       MOV   AX,4C00H   ; درخواست
       INT   21H        ; انتهای پردازش
       ENDM             ; انتهای ماکرو
```

شکل ۱-۲۲ یک لیست از برنامه اسمبل شده‌ای که ماکروهای INITZ و FINISH را تعریف و استفاده می‌کند، فراهم می‌سازد. این نسخه اسمبلر خاص درکنار بسط ماکرو عدد یک را در سمت چپ هر دستور قرار می‌دهد تا مبنی بر تولید دستور ماکرو باشد.

```
                        page 60,132
                TITLE   A22MACR1 (EXE)  Simple macros
                ; ------------------------------------
                INITZ   MACRO               ;Define macro
                        MOV   AX,@data      ;Initialize segment
                        MOV   DS,AX         ;  registers
                        MOV   ES,AX
                        ENDM                ;End macro

                FINISH  MACRO               ;Define macro
                        MOV   AX,4C00H      ;End processing
                        INT   21H
                        ENDM                ;End macro
                ; ------------------------------------------
                        .MODEL SMALL
                        .STACK 64
                        .DATA
0000 54 65 73 74 20 6D  MESSGE DB    'Test macro instruction',13,10,'$'
     61 63 72 6F 20 69
     6E 73 74 72 75 63
     74 69 6F 6E 0D 0A
     24
                ; ------------------------------------------
                        .CODE
0000                    BEGIN   PROC   FAR
                        INITZ               ;Macro instruction
0000 B8 ---- R    1             MOV   AX,@data      ;Initialize segment
0003 8E D8        1             MOV   DS,AX         ;  registers
0005 8E C0        1             MOV   ES,AX
0007 B4 09                      MOV   AH,09H        ;Request display
0009 8D 16 0000 R               LEA   DX,MESSGE     ;Message
000D CD 21                      INT   21H
                                FINISH
000F B8 4C00      1             MOV   AX,4C00H      ;End processing
0012 CD 21        1             INT   21H
0014                    BEGIN   ENDP
                                END   BEGIN
```

شکل ۱-۲۲ دستورات ماکرو اسمبل شده ساده

(TASM در سمت چپ لیست '1' نمایش می‌دهد) یک بسط ماکرو مبنی بر این است که فقط دستورات برای کد مقصد تولید می‌شوند، بنابراین پیش‌پردازنده‌هایی مانند ASSUME یا PAGE در تعریف ماکرو دیده نخواهد شد.

تعریف ماکرویی که قرار است فقط یک بار از آن استفاده شود، کار خوبی نیست اما می‌توانید ماکروها را در یک کتابخانه جمع کنید و برای همه برنامه‌ها استفاده کنید. بخش بعدی توضیح می‌دهد که چگونه ماکروها را در یک کتابخانه گرد آورید و آن را در یک برنامه بطور خودکار اضافه کنید.

استفاده از پارامترها در ماکروها

برای انعطاف بیشتر یک ماکرو، می‌توانید پارامترهایی در عملوندها بصورت ***آرگومان‌های فرضی*** تعریف کنید. تعریف ماکروی زیر با نام PROMPT برای استفاده از INT 21H تابع 09H جهت نمایش پیغام می‌باشد :

```
PROMPT  MACRO   MESSGE      ; آرگومان فرضی
        MOV     AH,09H
        LEA     DX,MESSGE
        INT     21H
        ENDM                ; انتهای ماکرو
```

وقتی از این دستور ماکرو استفاده می‌کنید، می‌توانید نام پیغام را طوری قرار دهید که به یک ناحیه داده که با یک علامت دلار خاتمه می‌یابد، ارجاع کنید.

یک آرگومان فرضی در تعریف ماکرو به اسمبلر می‌گوید که این نام را هر نام مشابه در بدنه ماکرو مرتبط کند. برای مثال، آرگومان فرضی MESSGE در دستور LEA نیز وجود دارد. برنامه یک اعلان با نام MESSAGE2 تعریف می‌کند.

```
MESSAGE2  DB  'Enter thd date as mm/dd/yy'
```

حال اگر بخواهیم برای نمایش MESSAGE2 از ماکروی PROMPT استفاده کنیم به شکل زیر عمل خواهیم کرد.

```
PROMPT  MESSAGE2
```

پارامتر (MESSAGE2) در دستور ماکرو با آرگومان فرضی (MESSGE) در تعریف ماکرو اصلی مرتبط می‌شود.

```
Macro definition :   PROMPT MACRO MESSGE   (آرگومان)
                                  |
Macro instrution :   PROMPT MESSAGE2       (پارامتر)
```

حال اسمبلر آرگومان تعریف ماکروی اصلی را با عملوند جمله LEA جور می‌کند. حال پارامترهای دستور ماکرو (MESSAGE2) را بجای آرگومان فرضی MESSGE، در تعریف ماکرو، جانشین می‌کند. اسمبلر MESSAGE2 را جانشین MESSGE در دستور LEA، جانشین هر یک از رویدادهای دیگر MESSGE می‌نماید.

تعریف ماکرو و بسط ماکرو در شکل ۲-۲۲ بطور کامل نشان داده شده است. همچنین برنامه ماکروهای INITZ و FINISH در شروع تعریف می‌کند و در سگمنت کد از آنها تعریف می‌کند.

یک آرگومان فرضی ممکن است شامل هر نام معتبر و یک نام ثبات مانند CX باشد. ممکن است یک ماکرو را با هر تعداد از آرگومان‌های فرضی که با کاما جدا می‌شوند و حداکثر تا ستون 120 یک خط هستند، تعریف کنید (به نسخه اسمبلر بستگی دارد). اسمبلر پارامترهای دستور ماکرو را از چپ به راست و یکی یکی جانشین آرگومانهای فرضی در تعریف ماکرو می‌کند.

توضیحات ماکرو

برای توصیفی‌تر کردن ماکروها می‌توان توضیحاتی را در آن به کار برد یک ; یا پیش پردازنده COMMENT مبنی بر یک خط توضیح است، مثال زیر برای توضیح از ; استفاده می‌کند :

```
                        PROMPT MACRO  MESSGE
                        ;      This macro permits a display of messages
                               MOV    AH,09H      درخواست نمایش :
                               LEA    DX,MESSGE           اعلان ;
                               INT    21H
                               ENDM

                               page  60,132
                        TITLE  A22MACR2 (EXE)  Use of parameters
                        ; ----------------------------------------
                        INITZ  MACRO                ;Define macro
                               MOV    AX,@data      ;Initialize segment
                               MOV    DS,AX         ;  registers
                               MOV    ES,AX
                               ENDM                 ;End macro

                        PROMPT MACRO MESSGE         ;Define macro
                               MOV    AH,09H        ;Request display
                               LEA    DX,MESSGE     ;  prompt
                               INT    21H
                               ENDM                 ;End macro

                        FINISH MACRO                ;Define macro
                               MOV    AX,4C00H      ;End processing
                               INT    21H
                               ENDM                 ;End macro
                        ; ----------------------------------------
                               .MODEL SMALL
                               .STACK 64
                               .DATA
0000 43 75 73 74 6F 6D  MESSG1 DB     'Customer name?', '$'
     65 72 20 6E 61 6D
     65 3F 24
000F 43 75 73 74 6F 6D  MESSG2 DB     'Customer address?', '$'
     65 72 20 61 64 64
     72 65 73 73 3F 24
                        ; ----------------------------------------
                               .CODE
0000                    BEGIN  PROC   FAR
                               INITZ
0000 B8 ---- R     1           MOV    AX,@data      ;Initialize segment
0003 8E D8         1           MOV    DS,AX         ;  registers
0005 8E C0         1           MOV    ES,AX
                               PROMPT MESSG2
0007 B4 09         1           MOV    AH,09H        ;Request display
0009 8D 16 000F R  1           LEA    DX,MESSG2     ;  prompt
000D CD 21         1           INT    21H
                               FINISH
000F B8 4C00       1           MOV    AX,4C00H      ;End processing
0012 CD 21         1           INT    21H
0014                    BEGIN  ENDP
                               END    BEGIN
```

شکل ۲-۲۲ استفاده از پارامترهای ماکرو

چون پیش فرض فقط لیست کردن دستوراتی است که کد مقصد تولید می‌کند، اسمبلر بطور خودکار وقتی تعریف ماکرو را بسط می‌دهد، توضیحات را نمایش نمی‌دهد. اگر می‌خواهید توضیحات ماکرو نیز ظاهر شوند شبه عمل لیست LALL. (لیست کردن همه شامل نقطه نام) را قبل از تقاضای دستورالعمل ماکرو بکار ببرید :

```
.LALL
PROMPT MESSAGE 1
```

یک تعریف ماکرو می‌تواند دارای تعدادی توضیح باشد که ممکن است بخواهید برخی از آنها در بسط ماکرو دیده

شوند و برخی ظاهر نشوند. این کار نیز با دستور LALL. عملی است ولی قبل از توضیحاتی که نباید ظاهر شوند یک جفت علامت (; ;) قرار دهید. پیش فرض اسمبلر XALL. است که سبب می‌شود فقط دستوراتی که کد مقصد تولید می‌کنند لیست شوند و در نهایت ممکن است نخواهید هیچ کد اسمبلی لیست شود بخصوص اگر دستور ماکرو چندین بار در برنامه استفاده شده باشد. بدین منظور از پیش‌پردازنده SALL. (موقوف گذارندن همه) استفاده کنید که اندازه برنامه چاپ شده را کاهش می‌دهد لیکن هیچ تاثیری بر اندازه برنامه مقصد ندارد.

یک پیش پردازندهٔ لیست‌گیری تا زمانی که پیش پردازنده لیست‌دیگری ظاهر نشده است، تاثیرش را برنامه حفظ می‌کند. می‌توانید آنها را در یک برنامه قرار دهید تا برخی ماکروها توضیحات را نمایش دهند. برخی بسط ماکرو را نمایش دهند و برخی دیگر نیز نمایش ندهند.

برنامه شکل ۳-۲۲ خصوصیات فوق را تشریح می‌کند. برنامه، ماکروهای INITZ ، FINISH و PROMPT که قبلاً توضیح داده شد دارد. سگمنت کد حاوی پیش‌پردازنده لیست‌گیری SALL. است که از لیست‌گیری بسط INITZ و FINISH و اولین بسط PROMPT جلوگیری می‌کند. برای دومین PROMPT، پیش پردازنده LALL سبب می‌شود تا اسمبلر همه توضیحات و بسط ماکرو را نمایش دهد. اما توجه کنید که در تعریف ماکروی PROMPT، توضیح بسط ماکرو که علامت (;;) دارد لیست نشده است.

استفاده از یک ماکرو داخل یک تعریف ماکرو

یک تعریف ماکرو ممکن است حاوی ارجاع به ماکروی تعریف شده دیگری باشد. یک ماکروی ساده با نام INT 21 که یک تابع را در ثبات AH قرار می‌دهد و INT 21H را صادر می‌کند :

```
INT21   MACRO   FUNCTN
        MOV     AH,FUNCTN
        INT     21H
        ENDM
```

برای آنکه ماکرو INT 21 را استفاده کنید و ورودی را از صفحه کلید بپذیرید، دستورات زیر را به کار ببرید :

```
        LEA     DX,NAMEPAR
        INT21   0AH
```

برای تولید کد برای INT 21 تابع 0AH را در AH قرار دهید و INT 21H را برای ورودی صفحه کلید صادر کنید. حال فرض کنید ماکرو دیگری دارید با نام DISP، که INT 21H تابع 02H را در ثبات AH برای نمایش یک کاراکتر قرار دهد :

```
DISP    MACRO   CHAR
        MOV     AH,02H
        MOV     DL,CHAR
        INT     21H
                ENDM
```

برای نمایش یک علامت سئوال، ماکرو را چنین بنویسید '?' DISP. اما می‌توانید برای اخذ مزیت ماکرو INT21 با اشاره به INT 21 در تعریف DISP تغییرات ایجاد کنید.

```
DISP    MACRO   CHAR
        MOV     DL,CHAR
        INT 21  02H
ENDM
```

حال اگر ماکرو DISP را بصورت '?' DISP کد نمایید، اسمبلر کد زیر را تولید می‌کند

```
        MOV     D,'?'
        MOV     AH,02H
        INT     21H
```

پیش پردازنده LOCAL

تعدادی از ماکروها به تعریف یک عنصر داده یا برچسب یک دستور نیاز دارند. اگر ماکرو را پیش از یک بار در برنامه استفاده کنید، اسمبلر عنصر داده یا برچسب را برای هر رخداد را تعریف می‌کند و یک پیغام خطا به سبب نسخه‌های متعدد نام‌ها تولید می‌شود. پیش‌پردازنده LOCAL بلافاصله بعد از جمله MACRO حتی قبل از توضیح کد می‌شود. قالب کلی آن چنین است LOCAL dummy1, dummy2, . . . ; یک یا چند آرگومان فرضی

```
                        page  60,132
                TITLE   A22MACR3 (EXE)  Use of .LALL & .SALL
                ; ----------------------------------------
                INITZ   MACRO               ;Define macro
                        MOV   AX,@data      ;Initialize segment
                        MOV   DS,AX         ;  registers
                        MOV   ES,AX
                        ENDM  ;End macro

                PROMPT MACRO  MESSGE
                ;This macro displays any message
                ;;Generates code that requests display
                        MOV   AH,09H        ;Request display
                        LEA   DX,MESSGE     ;  prompt
                        INT   21H
                        ENDM

                FINISH MACRO                ;Define macro
                        MOV   AX,4C00H      ;End processing
                        INT   21H
                        ENDM  ;End macro
                ; ----------------------------------------
                        .MODEL  SMALL
                        .STACK  64
                        .DATA
0000 43 75 73 74 6F 6D  MESSG1 DB     'Customer name?', 13, 10, '$'
     65 72 20 6E 61 6D
     65 3F 0D 0A 24
0011 43 75 73 74 6F 6D  MESSG2 DB     'Customer address?', 13, 10, '$'
     65 72 20 61 64 64
     72 65 73 73 3F 0D
     0A 24
                ; ----------------------------------------
                        .CODE
0000                    BEGIN   PROC  FAR
                        .SALL
                        INITZ
                PROMPT MESSG1
                        .LALL
                PROMPT MESSG2
                ;This macro displays any message
   1                            MOV   AH,09H        ;Request display
000F B4 09              1
0011 8D 16 0011 R       1            LEA   DX,MESSG2     ;  prompt
0015 CD 21              1            INT   21H
                        .SALL
                        FINISH
001C                    BEGIN   ENDP
                        END     BEGIN
```

شکل ۳-۲۲ لیست کردن و جلوگیری از بسط ماکرو

شکل ۴-۲۲ نحوه استفاده از پیش پردازندهٔ LOCAL را تشریح می‌کند. هدف از برنامه انجام تقسیم بوسیله تفریق متوالی است. روال مقسوم علیه را از مقسوم کم می‌کند و تا زمانیکه مقسوم کمتر از مقسوم علیه نشده یکی به خارج قسمت می‌افزاید. روال به دو برچسب نیاز دارد. COMP برای آدرس حلقه و OUT برای خارج شدن از روال در هنگام تکمیل برنامه، هم COMP و هم OUT بصورت LOCAL تعریف شده‌اند و ممکن است هر نام معتبری باشند.

در بسط ماکرو بر چسب سمبولیک تولید شده برای COM عبارت از 0000?? است و برای OUT 0001?? می‌باشد. اگر دستور ماکرو DIVIDE مجدداً در یک برنامه استفاده شده باشد، برچسب‌های سمبولیک برای بسط ماکروهای بعدی به

ترتیب عبارت از 0002?? و 0003?? خواهند بود. به این طریق این خصیصه تضمین می‌کند که برچسب‌های تولید شده، منحصر بفردند.

```
                                page  60,132
                        TITLE   A22MACR4 (EXE)  Use of LOCAL
                        ; ----------------------------------------
                        INITZ   MACRO               ;Define macro
                                MOV   AX,@data      ;Initialize segment
                                MOV   DS,AX         ;  registers
                                MOV   ES,AX
                                ENDM                ;End macro
                        DIVIDE  MACRO  DIVIDEND,DIVISOR,QUOTIENT
                                LOCAL COMP
                                LOCAL OUT
                        ;       AX = div'd, BX = divisor, CX = quotient
                                MOV   AX,DIVIDEND ;Set dividend
                                MOV   BX,DIVISOR  ;Set divisor
                                SUB   CX,CX       ;Clear quotient
                        COMP:
                                CMP   AX,BX       ;Dividend < divisor?
                                JB    OUT         ;  yes, exit
                                SUB   AX,BX       ;Dividend - divisor
                                INC   CX          ;Add to quotient
                                JMP   COMP
                        OUT:
                                MOV   QUOTIENT,CX ;Store quotient
                                ENDM              ;End macro
                        FINISH  MACRO             ;Define macro
                                MOV   AX,4C00H    ;End processing
                                INT   21H
                                ENDM              ;End macro
                        ; ----------------------------------------
                                .MODEL SMALL
                                .STACK 64
                                .DATA
0000 0096               DIVDND  DW     150        ;Dividend
0002 001B               DIVSOR  DW    27          ;Divisor
0004 0000               QUOTNT  DW    ?           ;Quotient
                        ; ----------------------------------------
                                .CODE
0000                    BEGIN   PROC  FAR
                                .SALL
                                INITZ
                                .LALL
                        DIVIDE  DIVDND,DIVSOR,QUOTNT
                    1   ;       AX = div'd, BX = divisor, CX = quotient
0007 A1 0000 R      1           MOV   AX,DIVDND   ;Set dividend
000A  8B 1E 0002 R  1           MOV   BX,DIVSOR   ;Set divisor
000E  2B C9         1           SUB   CX,CX       ;Clear quotient
0010                1   ??0000:
0010  3B C3         1           CMP   AX,BX       ;Dividend < divisor?
0012  72 05         1           JB    ??0001      ;  yes, exit
0014  2B C3         1           SUB   AX,BX       ;Dividend - divisor
0016  41            1           INC   CX          ;Add to quotient
0017  EB F7         1           JMP   ??0000
0019                1   ??0001:
0019  89 0E 0004 R  1           MOV   QUOTNT,CX   ;Store quotient
                                .SALL
                                FINISH
0022                    BEGIN   ENDP
                                END   BEGIN
```

شکل ۴-۲۲ استفاده از پیش پردازنده *LOCAL*

مشمول نمودن ماکروها از یک کتابخانه

تعریف ماکروهایی مانند INITZ ، FINISH و PROMPT و استفاده از آن‌ها فقط یک بار در برنامه ممکن است حاصلی نداشته باشد. یک روش بهتر، ثبت نام تمام ماکروها در یک کتابخانه دیسک تحت یک نام مانند

MACRO.LBY است. باید براحتی همه تعاریف ماکروها را در یک یا چند فایل روی دیسک ذخیره کنید.

```
INITZ    MACRO
         ...
         ENDM
         ...
PROMPT   MESSGE   MACRO
         ...
         ENDM
```

می‌توانید از یک ویرایشگر یا پردازشگر کلمات برای نوشتن در فایل استفاده کنید، اما باید یک فایل ASCII قالب بندی نشده باشد. در مثال زیر، فرض شده است که فایل با نام MACRO.LBY. روی درایو :F ذخیره شده است. برنامه شما می‌تواند از هر یک از ماکروهای ثبت شده استفاده کنید، اما بجای درج دستورات MACRO در شروع برنامه، از پیش‌پردازنده INCLUDE استفاده کنید: INCLUDE F:\MACRO.LBY

اسمبلر فایل MACRO.LBY. را روی درایو :F دستیابی می‌کند و **تمام** تعاریف ماکروی ثبت شده در آن شامل برنامه خواهند شد، اگر چه برنامه شما ممکن است به برخی از آنها نیاز داشته باشد. لیست اسمبل شده حاوی یک کپی از تعاریف ماکرو است، که با حرف C در ستون ۳۰ فایل LST. در برخی از نسخه‌هایی اسمبلر مشخص می‌شوند.

چون برخی اسمبلرها مستلزم عملیات دو مرحله‌ای هستند، می‌توانید با استفاده از دستورات زیر کاری کنید که از INCLUDE فقط در مرحله اول استفاده شود (بجای هر دو مرحله)

```
IF1
        F:\MACRO.LBY INCLUDE
ENDIF
```

IF1 و ENDIF پیش پردازنده‌های شرطی‌اند. IF1 به اسمبلر می‌گوید تا کتابخانه را فقط در گذر اول فرآیند اسمبلی دستیابی کند. پیش پردازنده ENDIF منطق IF را خاتمه می‌دهد. اینک کپی تعریف ماکرو روی لیست ظاهر نمی‌شود و از نظر زمان و مکان صرفه‌جویی می‌شود. (MASM نسخه 6.0 نیازی به پیش پردازنده که به دو مرحله اشاره کند ندارند، در حالیکه TASM در صورت استفاده از سوئیچ /m در خط فرمان بیش از یک مرحله خواهیم داشت، گرچه اسمبلر، فقط ENDIF را در فایل LST نشان می‌دهد ولی برنامه شکل ۵-۲۲ دارای جملات IF1 ، INCLUDE و ENDIF است که قبلاً توضیح داده شدند. دستورات ماکرویی که در سگمنت کد INITZ ، FINISH و PROMPT وجود دارند به صورت فایل MACRO.LBY توسط برنامه ویرایشگر ذخیره می‌شود.

```
                page 60,132
TITLE           A22MACR5 (EXE)  Test of INCLUDE
        IF1
                INCLUDE F:A22MACRO.LBY
        ENDIF
; ------------------------------------
                .MODEL SMALL
                .STACK 64
                .DATA
MESSGE          DB      'Test of macro','$'
; ------------------------------------
                .CODE
BEGIN           PROC    FAR
                INITZ
                PROMPT MESSGE
                FINISH
BEGIN           ENDP
                END     BEGIN
```

شکل ۵-۲۲ استفاده از کتابخانه *INCLUDE*

مکان INCLUDE از حساسیت زیادی برخوردار نیست، لیکن باید قبل از یک دستور ماکرو که به یک ورودی کتابخانه مراجعه دارد، ظاهر شود.

پیش پردازنده PURGE

اجرای یک دستور INCLUDE سبب می‌شود که اسمبلر تمام تعاریف ماکروی موجود در کتابخانه را در برنامه قرار دهد. فرض کنید که یک کتابخانه حاوی ماکروهای INITZ ، FINISH و PROMPT و DIVIDE است، اما برنامه فقط INITZ و FINISH را نیاز دارد. پیش‌پردازنده PURGE به شما توان حذف ماکروهای ناخواسته PROMPT و DIVIDE را می‌دهد :

```
IF1
INCLUDE   D:\MACRO.LBY          ; کتابخانه کامل
ENDIF
PURGE     PROMPT,DIVIDE         ; حذف ماکروهای ناخواسته
...
INITZ     ...                   ; استفاده از ماکروهای باقیمانده
```

عملیات PURGE فقط مرحله اسمبلی را تسهیل می‌بخشد و هیچ تاثیری بر روی ماکروهای ذخیره‌شده در کتابخانه ماکرو ندارد.

الحاق

کاراکتر & به اسمبلر می‌گوید تا متن یا سمبول‌ها را با یکدیگر الحاق نماید. در ماکروی زیر، یک & تولید دستورات MOVSB ، MOVSW یا MOVSD را سهولت می‌بخشد.

```
STRMOVE  MACRO  TAG
         REP    MOVS & TAG
         ENDM
```

کاربر می‌تواند دستور ماکرو را به صورت STR MOVE B و STR MOVE W یا STR MOVE D بنویسد. سپس اسمبلر پارامترهای B ، W یا D را به دستور MOVS الحاق می‌کند، تا REP MOVSB، REP MOVSW یا REP MOVSD ایجاد شود. (این مثال، برای اهداف تشریحی می‌باشد).

پیش پردازنده تکرار

پیش پردازنده‌های تکرار، سبب می‌شوند تا اسمبلر بلوکی از احکام را که به وسیله جمله ENDM خاتمه می‌یابد، تکرار کند. MASM6.0 عبارت‌های REPEAT ، FORC و FOR را برای REPT ، IRP و IRPC معرفی می‌کند). این پیش‌پردازندها در یک تعریف MACRO نباید باشد، ولی اگر وجود داشته باشد، باید یک ENDM برای خاتمه هر پیش‌پردازنده، تکرار قرار دهید و یک ENDM دیگر برای خاتمه تعریف MACRO بگذارید.

REPT : پیش پردازنده، تکرار

پیش پردازنده REPT سبب می‌شود تا بلوکی از جملات، حداکثر تا ENDM بر اساس تعداد دفعات ورودی عبارت، تکرار شود :

```
REPT expression
```

اولین مثال دستور DEC را چهار بار تولید می‌کند :

```
REPT  4
      DEC SI
EVDM
```

مثال دوم N را با صفر مقدار می‌دهد و تولید DB N را با ۵ بار تکرار می‌کند:

```
N = 0
REPT 5
    N = N + 1
    DB N
ENDM
```

عملیات ۵ جمله DB تولید می‌کند، DB1 تا DB5. یک مورد استفاده از REPT می‌تواند تعریف یک جدول یا بخشی از جدول باشد. مثال بعد یک ماکرو تعریف می‌کند که از REPT برای ۵ بار بوق زدن بلند گو استفاده می‌کند :

```
BEEPSPKR  MACRO
          MOV     AH,02H      ; درخواست خروجی
          MOV     DL,07       ; کاراکتر بوق
          REPT 5              ; ۵ بار
          INT 21H             ; فراخوانی سرویس وقفه
          ENDM                ; پایان تکرار
          ENDM                ; پایان ماکرو
```

IRP : پیش‌پردازنده تکرار نامحدود

پیش پردازنده IRP سبب می‌شود اسمبلر یک بلوک از دستورات تا ENDM را تکرار کند. شکل کلی آن چنین است :

```
IRP dummy,<arguments>
```

آرگومانهای دستور، می‌تواند هر تعداد نمادهای معتبر، رشته‌ای، عددی یا ثابت حسابی باشد، اسمبلر بلوکی از کد را برای هر آرگومان تولید می‌کند. در مثال زیر اسمبلر DB 3 ، DB 9 ، DB 17 ، DB 25 و DB 28 را تولید می‌کند.

```
IRP   N (3,9,17,25,28)
 DB   N
ENDM
```

در مثال دوم، اسمبلر یک جمله PUSH برای هر ثبات مشخص تولید می‌کند :

```
IRP    REG <AX,BX,CX,DX>
PUSH  REG
ENDM
```

IRPC : پیش پردازنده تکرار نامحدود کاراکتری

پیش پردازنده IRPC (یا FORC) سبب تکرار بلوکی از احکام تا ENDM می‌شود. شکل کلی آن در زیر نشان داده شده است :

```
IRPC  dummy,string
```

اسمبلربلوکی از کد را برای هر کاراکتر دررشته string تولیدمی‌کند. درمثال زیر، اسمبلر DW3 تا DW8 را تولیدمی‌کند.

```
IRPC    N,345678
DW      N
ENDM
```

پیش‌پردازنده‌های شرطی

اسمبلر تعدادی پیش پردازنده‌های شرطی را ارائه می‌کند. قبلاً از پیش‌پردازنده IF1 برای قرار دادن یک ورودی کتابخانه فقط در گذر اول پردازش اسمبلی استفاده کردیم. پیش‌پردازنده‌های اعمال شرطی اکثراً برای استفاده در داخل تعریف یک ماکرو مفیدند لیکن برای منظور محدود نمی‌شوند. هر پیش‌پردازنده IF باید دارای یک ENDIF متناظر باشد تا شرط تست شونده را خاتمه دهد. یک ELSE اختیاری یک عمل متفاوتی را مهیا می‌سازد. در اینجا شکل کلی خانواده IF از پیش‌پردازنده‌های شرطی آورده شده است :

```
IFxx    (شرط)
...             شرطی
ELSE    (انتخابی)
...             بلوک
ENDIF   (انتهای IF)
```

با حذف ENDIF یک پیغام خطای "شرط محدود نشده" را ایجاد می‌کند. اگر شرط بررسی شده درست باشد ، اسمبلر بلوک شرطی را تا ELSE و اگر ELSE نباشد حداکثر تا ENDIF اجرا می‌کند. اگر شرط غلط باشد اسمبلر بلوک شرطی بعد از ELSE را اجرا می‌کند، اگر ELSE نباشد، هیچ بلوک شرطی را تولید نمی‌کند.

موارد زیر، چگونگی دستیابی پیش‌پردازنده‌های شرطی توسط اسمبلر را توضیح می‌دهد:

- عبارت IF : اگر اسمبلر عبارت را غیر صفر ارزیابی کند، دستورات داخل بلوک شرطی را اسمبل می‌کند.
- عبارت IFE : اگر اسمبلر عبارت را صفر ارزیابی کند دستورات داخل بلوک شرطی را اسمبل می‌کند.
- (عبارتی نیست) IF1 : اگر اسمبلر گذر اول را پردازش می‌کند روی دستورات داخل بلوک شرطی را عمل می‌کند.
- (عبارتی نیست) IF2 : اگر اسمبلر عبارت دوم را پردازش کند، روی دستورات داخل بلوک شرطی را عمل می‌کند.
- نماد IFDEF : اگر نماد در برنامه یا به عنوان EXTRN تعریف شده باشد، اسمبلر دستورات داخل بلوک شرطی را پردازش می‌کند.
- نماد IFNDEF : اگر نماد در برنامه یا به عنوان EXTRN تعریف نشده باشد، اسمبلر دستورات داخل بلوک شرطی را پردازش می‌کند.
- <آرگومان> IFB : اگر آرگومان، جای خالی باشد، اسمبلر دستورات داخل بلوک شرطی را پردازش می‌کند. آرگومان باید در داخل یک جفت پرانتز زاویه‌ای باشد.
- <آرگومان> IFNB : اگر آرگومان، جای خالی نباشد. اسمبلر دستورات داخل بلوک شرطی را پردازش می‌کند.آرگومان باید در داخل یک جفت پرانتز زاویه‌ای باشد.
- <arg-1>,<arg-2> IFIDN : اگر آرگومان رشته‌ای اول با آرگومان رشته دوم مساوی باشد، اسمبلر دستورات بلوک شرطی را پردازش می‌کند. آرگومان‌ها باید در پرانتزهای زاویه‌ای باشند.
- <arg-1>,<arg-2> IFIDN : اگر آرگومان رشته‌ای اول با آرگومان رشته‌ای دوم متفاوت باشد، اسمبلر دستورات بلوک شرطی را پردازش می‌کند. آرگومان‌ها باید در پرانتزهای زاویه‌ای باشند.
- IFE,IF : می‌تواند از عملگرهای رابطه‌ای EQ (مساوی)، NE (نامساوی)، LT (کمتر از)، LE (کمتر یا مساوی)، GT (بزرگتر از) و GE (بزرگتر یا مساوی) استفاده کند، برای مثال در جمله :

```
IF  expression1  EQ  expression2
```

در اینجا یک مثال ساده از استفاده IFNB (اگر خالی نیست) آورده شده است. INT 21H تابع 3CH یک برنامه را قادر می‌سازد تا پردازش را خاتمه دهد و یک کد بازگشت در AL ارائه دهد. مثال زیر ، ماکرو FINISH را بازنویسی می‌کند که اخیراً برای فراهم ساختن یک کد بازگشت استفاده شد :

```
FINISH  MACRO   RETCODE
        MOV     AH,3CH          درخواست خاتمه پردازش ;
IFNB    <RETCODE>
MOV     AL,RETCODE
ENDIF
INT     21H                     فراخوانی سرویس وقفه ;
ENDM
```

در اینجا مثال دیگری از استفاده IFNB آورده شده است. همه درخواست‌های INT 21H به یک تابع در ثبات AH نیاز دارند و برخی درخواست‌ها مقداری نیز در DX می‌خواهند. ماکرو INT 21 با استفاده از IFNB آرگومان DX را بررسی می‌کند تا خالی نباشد ، اگر حاصل صحیح باشد (آرگومان غیر خالی باشد) اسمبلر دستور MOV برای قرار دادن مقدار در DX تولید می‌کند:

```
INT21   MACRO   FUNCTN,DXADDRES
        MOV     AH,FUNCTN
        IFNB    <DXADDRES>
        MOV     DX,OFFSET DXADDRES
        ENDIF
INT     21H
        ENDM
```

استفاده از INT 21H برای ورودی صفحه کلید ساده برای یک کاراکتر فقط به مقداری در AH نیاز دارد در این حالت:

```
INT 21   01
```

اسمبلر MOV AH,01 و INT 21H را تولید می‌کند. ورودی صفحه کلید یک رشته کاراکتر به تابع 0AH در ثبات AH و آدرس ورودی در DX نیاز دارد. می‌توانید ماکرو INT 21 را چنین کد نمایید:

```
INT 21   0AH,IPFIELD
```

سپس اسمبلر دو جمله MOV AH,0AH و MOV DX,OFFSET و دستور INT 21H را تولید می‌کند.

پیش پردازنده EXITM

تعریف یک ماکرو ممکن است شامل یک پیش پردازنده شرطی باشد که یک شرط مهم را بررسی نماید. اگر شرط صحیح باشد، اسمبلر از بسط ماکرو خارج می‌شود. پیش‌پردازنده EXITM بدین منظور در نظر گرفته شده است :

```
IF××     [شرط]
...      [شرط نامعتبر]
EXITM
...
ENDIF
```

اگر اسمبلر در توسعه‌اش با EXITM مواجه شود، بسط ماکرو را ادامه نمی‌دهد و دستورات بعد از پیش پردازنده ENDM را از سر می‌گیرد. از EXITM برای خاتمه REPT ، IRP و IRPC استفاده کنید حتی اگر آنها در داخل تعریف یک ماکرو قرار داشته باشند.

ماکرو با استفاده از شرایط IF و IFNDEF

ساختار برنامه ۶-۲۲ حاوی تعریف یک ماکرو به نام DIVIDE است که یک روال برای انجام تقسیم بوسیله تفریق متوالی تولید می‌کند. کاربرد باید دستور ماکرو را با پارامترهایی برای مقسوم، مقسوم‌علیه و خارج قسمت، به ترتیب کد نویسی کند. ماکرو از IFNDEF برای بررسی اینکه آیا عناصر داده واقعاً در برنامه تعریف شده‌اند، استفاده می‌کند. برای هر عنصر تعریف‌نشده ماکرو یک فیلد اختیاری با نام CNTR تعریف‌می‌کند. (CNTR می‌تواند هرنام معتبر دیگری باشد و استفاده‌کمکی درتعریف ماکرودارد) پس‌از بررسی تمام سه پارامتر ، ماکرو CNTR را ازنظر صفر نبودن بررسی‌می‌کند.

```
IF      CNTR
بسط ماکرو خاتمه یافت ;
EXITM
ENDIF
```

اگر CNTR دارای هر ارزش غیر صفر باشد، اسمبلر توضیح را تولید می‌کند و از توسعه اضافی ماکرو خارج می‌شود. توجه داشته باشید که یک دستور اولیه، CNTR را با 0 مقدار می‌دهد و همچنین بلوکهای IFNDEF نیاز دارند فقط CNTR را با کمیت 1 بجای افزایش آن مقدار دهند.

اگر اسمبلر تمام تست‌ها را به طور صحیح بگذارند، بسط ماکرو را تولید می‌کند. در قطعهٔ کد، اولین دستور ماکروی DIVIDE حاوی یک مقسوم و مقسوم علیه نامعتبر است و فقط توضیحات را تولید می‌کند. یک راه برای اصلاح ماکرو می‌تواند تست غیرصفربودن مقسوم‌علیه و داشتن علامت یکسان مقسوم و مقسوم‌علیه باشد. بدین منظور از کد اسمبلی بجای پیش‌پردازنده‌های شرطی استفاده‌کنید، چون شرایط در هنگام اجرای برنامه رخ می‌دهند، نه وقتی اسمبل می‌شوند.

ماکرو با استفاده از شرط IFIDN

ساختار برنامه شکل ۷-۲۲ حاوی تعریف یک ماکرو با نام MOVIF است که بسته به پارامتر داده شده، MOVSB

یا MOVSW تولید می‌کند. کاربر باید دستور ماکرو را با پارامتر B (بایت) یا W (کلمه) مبنی براینکه آیا MOVS یک MOVSB است یا MOVSW است، مقدار دهی نماید.

دو رخداد IFIDN در تعریف ماکرو چنین هستند :

```
IFIDN <&TAG>,<B>          IFIDN <&TAG>,<W
REP MOVB                  REP MOVSW
...                       ...
```

```
                page  60,132
        TITLE   A22MACR6 (EXE)  Test of IF and IFNDEF
        ; ---------------------------------------
        INITZ   MACRO                   ;Define macro
                MOV   AX,@data          ;Initialize segment
                MOV   DS,AX             ;  registers
                MOV   ES,AX
                ENDM                    ;End macro
        FINISH  MACRO                   ;Define macro
                MOV   AX,4C00H          ;End processing
                INT   21H
                ENDM                    ;End macro
        DIVIDE  MACRO  DIVIDEND,DIVISOR,QUOTIENT
                LOCAL COMP
                LOCAL OUT
                CNTR   = 0
        ;       AX = div'nd, BX = div'r, CX = quot't
                IFNDEF DIVIDEND
        ;               Dividend not defined
                CNTR   = CNTR +1
                ENDIF
                IFNDEF DIVISOR
        ;               Divisor not defined
                CNTR   = CNTR +1
                ENDIF
                IFNDEF QUOTIENT
        ;               Quotient not defined
                CNTR   = CNTR + 1
                ENDIF
                IF     CNTR
        ;               Macro expansion terminated
                EXITM
                ENDIF
                MOV    AX,DIVIDEND ;Set dividend
                MOV    BX,DIVISOR  ;Set divisor
                SUB    CX,CX       ;Clear quotient
        COMP:
                CMP    AX,BX       ;Dividend < divisor?
                JB     OUT         ;  yes, exit
                SUB    AX,BX       ;Dividend - divisor
                INC    CX          ;Add to quotient
                JMP    COMP
        OUT:
                MOV    QUOTIENT,CX ;Store quotient
                ENDM
        ; ---------------------------------------
                .MODEL SMALL
                .STACK 64
                .DATA
0000  0096      DIVDND DW     150          ;Dividend
0002  001B      DIVSOR DW     27           ;Divisor
0004  0000      QUOTNT DW     ?            ;Quotient
                ; ---------------------------------------
                .CODE
0000            BEGIN  PROC   FAR
                .SALL
                INITZ
                .LALL
```

شکل الف ۶-۲۲ استفاده از پیش‌پردازنده‌های *IF* و *IFNDEF*

```
                                        DIVIDE DIVDND,DIVSOR,QUOTNT
= 0000                          1       CNTR  = 0
                                1   ;   AX = div'nd, BX = div'r, CX = quot't
0007  A1 0000 R                 1       MOV   AX,DIVDND    ;Set dividend
000A  8B 1E 0002 R              1       MOV   BX,DIVSOR    ;Set divisor
000E  2B C9                     1       SUB   CX,CX        ;Clear quotient
0010                            1   ??0000:
0010  3B C3                     1       CMP   AX,BX        ;Dividend < divisor?
0012  72 05                     1       JB    ??0001       ;  yes, exit
0014  2B C3                     1       SUB   AX,BX        ;Dividend - divisor
0016  41                        1       INC   CX           ;Add to quotient
0017  EB F7                     1       JMP   ??0000
0019                            1   ??0001:
0019  89 0E 0004 R              1       MOV   QUOTNT,CX    ;Store quotient
                                        DIVIDE DIDND,DIVSOR,QUOT
= 0000                          1       CNTR   = 0
                                1   ;   AX = div'nd, BX = div'r, CX = quot't
                                1       IFNDEF DIDND
                                1   ;         Dividend not defined
= 0001                          1       CNTR   = CNTR +1
                                1       ENDIF
                                1       IFNDEF QUOT
                                1   ;         Quotient not defined
= 0002                          1       CNTR  = CNTR + 1
                                1       ENDIF
                                1       IF    CNTR
                                1   ;         Macro expansion terminated
                                1       EXITM
                                        .SALL
                                        FINISH
0022                            BEGIN   ENDP
                                        END   BEGIN
```

شکل ب ۶-۲۲ استفاده از پیش‌پردازنده‌های *IF* و *IFNDEF*

اولین IFIDN، دستور REP MOVSB را در صورتیکه دستور ماکرو بصورت MOVIF B نوشته شده باشد تولید می‌کند، و اگر MOVIF W نوشته شده باشد،دومین IFIDN دستور REP MOVSW را تولید می‌کند. اگر یک کاربر B یا W نداده باشد، اسمبلر یک توضیح تولید می‌کند و پیش‌فرض آن MOVSB خواهد بود (استفاده معمول عملگر (&) برای الحاق می‌باشد)، سه مثال MOVIF در سگمنت کد، حالت B حالت W، یک حالت نامعتبر را بررسی می‌کنند. سعی نکنید این برنامه را به همان صورت اجرا کنید زیرا در بایت‌های SI و DI باید مقادیر درستی وجود داشته باشند. مسلماً این ماکرو خیلی مفید نیست زیرا هدف آن تشریح استفاده از پیش پردازنده‌های شرطی با روشی ساده است. لیکن تاکنون، قابلیت لازم برای نوشتن ماکروهای بسیار مفید را بدست آورده‌اید.

نکات کلیدی

- یک تعریف ماکرو به پیش پردازنده MACRO، یک بلوک از یک یا چند جمله شناخته شده بعنوان بدنه که تعریف ماکرو را تولید می‌کند و یک پیش‌پردازنده ENDM برای خاتمه تعریف، نیاز دارد.
- یک دستور ماکرو، استفاده از ماکرو در یک برنامه است. کدی که دستورات ماکرو تولید می‌کنند. بسط ماکرو است.
- پیش‌پردازنده‌های SALL، LALL، و XALL. لیست‌توضیحات و کدمقصدتولیدشده دربسط ماکرو راکنترل‌می‌کنند.
- پیش پردازنده LOCAL. استفاده از اسامی داخل تعریف ماکرو را تسهیل می‌بخشد و باید بلافاصله بعد از جمله ماکرو ظاهر شود.
- استفاده از آرگومان‌های فرضی در تعریف ماکرو به یک کاربر امکان کدنویسی پارامتر با انعطاف بیشتر را می‌دهد.
- یک کتابخانه ماکرو، ماکروهای ثبت شده را برای دیگر برنامه‌ها در اختیار قرار می‌دهد.
- با پیش‌پردازنده‌های شرطی، می‌توان اعتبار پارامترهای ماکرو را تعیین نمود.

```
                        page  60,132
                TITLE   A22MACR7 (EXE)  Tests of IFIDN
                ; ----------------------------------------
                INITZ   MACRO               ;Define macro
                        MOV   AX,@data      ;Initialize segment
                        MOV   DS,AX         ;  registers
                        MOV   ES,AX
                        ENDM                ;End macro

                MOVIF   MACRO TAG           ;Define macro
                        IFIDN <&TAG>,<B>
                        REP MOVSB           ;Move bytes
                        EXITM
                        ENDIF
                        IFIDN <&TAG>,<W>
                        REP MOVSW           ;Move words
                        ELSE
                ;       No B or W tag, default to B
                        REP MOVSB           ;Move bytes
                        ENDIF
                        ENDM                ;End of macro

                FINISH  MACRO               ;Define macro
                        MOV   AX,4C00H      ;End processing
                        INT   21H
                        ENDM                ;End macro
                ; ----------------------------------------
                        .MODEL SMALL
                        .STACK 64
                        .CODE
0000                    BEGIN   PROC  FAR
                        .LALL
                        INITZ
0000  B8 ---- R  1              MOV   AX,@data      ;Initialize segment
0003  8E D8      1              MOV   DS,AX         ;  registers
0005  8E C0      1              MOV   ES,AX
                                MOVIF B
                 1              IFIDN  <B>,<B>
0007  F3/ A4     1              REP MOVSB           ;Move bytes
                 1              EXITM
                                MOVIF W
                 1              IFIDN  <W>,<W>
0009  F3/ A5     1              REP MOVSW           ;Move words
                 1              ENDIF
                                MOVIF
                 1              ELSE
                 1      ;       No B or W tag, default to B
000B  F3/ A4     1              REP MOVSB           ;Move bytes
   1                            ENDIF
                                .LALL
                                FINISH
0012                    BEGIN   ENDP
                                END   BEGIN
```

شکل ۷-۲۲ استفاده از پیش‌پردازنده *IFIDN*

پرسش‌ها

۲۲-۱. تحت چه شرایطی استفاده از ماکروها توصیه می‌شود؟

۲۲-۲. اولین و دومین خط یک ماکرو ساده با نام **BIGMARCO** را بنویسید.

۲۲-۳. بین بدنه تعریف ماکرو و بسط ماکرو چه فرقی است.

۲۲-۴. آرگومان فرضی چیست ؟

۲۲-۵. پیش پردازنده‌ای برای مقاصد زیر کد کنید:

(الف) فقط دستوراتی که کد مقصد تولید می‌کنند، لیست کند،

(ب) از لیست همه دستوراتی که یک ماکرو تولید می‌کند، جلوگیری کند.

۶-۲۲. دو تعریف ماکرو که ضرب را انجام می‌دهند بنویسید:

(الف) MULTBYTE کدی تولید می‌کند که بایت در بایت ضرب می‌کند،

(ب) MULTWORD کدی تولید می‌کند که کلمه در کلمه ضرب می‌کند. مضروب و مضروب فیلد بصورت آرگومان‌های فرضی در تعریف ماکرو هستند. اجرای ماکروها را با یک برنامه کوچک که داده‌های مورد نیاز را تعریف می‌کند، بررسی کنید.

۷-۲۲. ماکروهایی که در پرسش ۶-۲۲ تعریف شد در یک کتابخانه ماکرو ذخیره کنید. برنامه را طوری بازنویسی کنید که ورودیهای کتابخانه را در طی گذر اول اسمبل شدن INCLUDE نماید.

۸-۲۲. یک ماکرو با نام PRINT17 که از INT 17H برای چاپ استفاده می‌شود، بنویسید. ماکرو باید شامل بررسی وضعیت چاپگر باشد و برای هر خط چاپ تعریف شده با هر طول، فراهم باشد.

۹-۲۲. ماکرو شکل ۶-۲۲ را بطوری بازنویسی کنید که اگر مقسوم‌علیه صفر باشد ، در زمان اجرای برنامه، از تقسیم منصرف شود.

۱۰-۲۲. برنامه‌ای بنویسید، اسمبل نموده و بررسی کنید که از ماکروهایی با نام PRINT17 ، MULTWORD و MULTBYTE استفاده کند.

(الف) دو فیلد یک بایتی با نام BYTE1 و BYTE2 و دو فیلد یک کلمه‌ای با نام WORD1 و WORD2 که همه حاوی مقادیر عددی‌اند، تعریف کنید.

(ب) از MULTBYTE برای ضرب دو فیلد یک بایتی و MULTWORD برای ضرب دو فیلد یک کلمه‌ای استفاده کنید.

(ج) حاصل را به قالب ASCII تبدیل و با استفاده از PRINT 17 آن را چاپ کنید.

فصل ۲۳

پیوند با زیر برنامه‌ها

هدف : ارائه تکنیک‌های برنامه‌نویسی که مستلزم پیوند زدن و اجرای برنامه‌های اسمبل شده مجزا است.

مقدمه

تا این فصل، همه برنامه‌های ما شامل یک ماجول اسمبل شده مستقل بود. اما می‌توان برنامه‌ای را که شامل یک برنامه اصلی است با یک یا چند زیر برنامه اسمبل شده مجزا پیوند داد. دلایل زیر، علت اصلی سازماندهی یک برنامه با زیر برنامه‌ها است :

- برای پیوندزدن زبانها - برای مثال، ترکیب سهولت کدنویسی به یک زبان سطح‌بالا با کارآیی پردازش در زبان اسمبلی.
- سهولت توسعه پروژه‌های بزرگ، بطریقی که تیم‌های مختلف ماجول خود را جداگانه تولید می‌کنند.
- چون اندازه برنامه بزرگ‌می‌شود. ممکن‌است ضروری باشد تا بخش‌هایی از آن در طی اجرا به جای یکدیگر قرار دارد.

هر برنامه بطور مجزا اسمبل می‌شود و ماجول مقصد (OBJ.) منحصر بفرد خود را تولید می‌کند. سپس برنامه پیونده دهنده (LINKER) ماجولهای مقصد را به یک ماجول اجرایی (EXE) ترکیب شده پیوند می‌دهد. معمولاً برنامه اصلی، برنامه‌ای است که اجرا را آغاز می‌کند و یک یا چند زیر برنامه را فراخوانی می‌کند. زیر برنامه‌ها نیز به نوبه خود ممکن است زیر برنامه‌های دیگری را فراخوانی کنند.

شکل ۱-۲۳ در مثال از تسلسل برنامه اصلی و سه زیر برنامه را نشان می‌دهد. در بخش (a)، برنامه اصلی، زیر برنامه ۱، ۲ و ۳ را فرا می‌خواند. در بخش (b)، برنامه اصلی زیر برنامه‌های ۱ و ۲ را فرا می‌خواند و فقط زیر برنامه ۱ زیر برنامه ۳ را فرامی خواند.

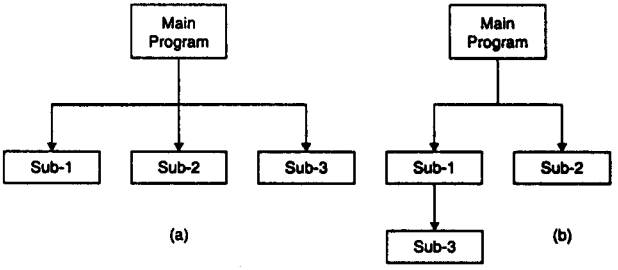
انواع متعددی از سازماندهی زیر برنامه وجود دارد اما سازمان باید حس تشخیص را برای اسمبلر و لینکر ایجاد کند. همچنین باید مراقب حالت‌هایی که مثلاً زیر برنامه ۱، زیربرنامه ۲ را فراخوانی می‌کند، ۲، ۳ را صدا می‌زند و ۳، ۱ را اجرا می‌کند باشید. استفاده از این روش که با عنوان بازگشتی نامیده می‌شود. بلامانع است ولی در صورت استفاده نادرست از آن با اشکالات عجیبی روبرو خواهیم شد.

پیش پردازنده SEGMENT

این بخش حاوی تعدادی انتخاب‌های مورد استفاده برای کدنویسی پیش‌پردازندهٔ SEGMENT است. قالب کلی برای پیش‌پردازنده SEGMENT کامل چنین است.

seg-name	SEGMENT	[align] [combine] [class]

انواع class ، vombine ، align توضیح داده خواهد شد.



شکل ۱-۲۳ سلسله مراتب برنامه

نوع مرزبندی

عملگر Align (اگر کد شود) به اسمبلر می‌گوید تا سگمنت نامبرده را با شروع از مرز یک حافظه خاص آغاز نماید :

- BYTE مرز بایت، برای یک سگمنت از یک زیر برنامه که با دیگر برنامه‌ها ترکیب خواهد شد. قرار دادن در مرز بایت برای برنامه‌هایی که در پردازشگر 8088 اجرا می‌شوند، مناسب است.
- WORD مرز کلمه، برای یک سگمنت از یک زیر برنامه که با دیگر برنامه‌ها ترکیب می‌شود. قرار دادن از مرز کلمه برای برنامه‌هایی که در پردازشگرهای 8086/80286 اجرا می‌شوند، مناسب است.
- DWORD مرز دو کلمه‌ای، معمولاً برای ۸۰۳۸۶ و پردازشگرهای ما بعد استفاده می‌شود.
- PARA مرز پاراگراف و (قابل تقسیم بر ۱۶، یا 10H)، قرارگیری پیش فرض و اغلب مورد استفاده برای برنامه‌های اصلی و زیربرنامه‌ها.
- PAGE مرز صفحه (قابل تقسیم بر ۲۵۶ یا 100H)

حذف عملگر aling از سگمنت اول سبب می‌شود تا پیش‌فرض PARA در نظر گرفته شود. در صورتیکه منحصر بفرد باشد، حذف آن از سگمنت‌های بعدی نیز سبب انتخاب پیش فرض PARA خواهد شد. اگر نام سگمنت منحصر بفرد نباشد، حالت پیش‌فرض، روش مرزبندی سگمنت هم نام قبلی خواهد بود.

نوع ترکیب

عملگر ترکیب در مورد ادغام کردن یا مجزا نگه داشتن سگمنت‌ها، اطلاعاتی را به اسمبلر و لینکر می‌دهد. (شما همیشه از نوع ترکیب STACK برای برنامه‌های EXE. استفاده کنید). دیگر انواع ترکیب مربوط، به این فصل NONE، PUBLIC و COMMON می‌باشد :

- NONE سگمنت بطور منطقی جدا از دیگر سگمنت‌ها است، اگر چه ممکن است بطور فیزیکی مجاور باشند. این نوع پیش‌فرض برای همه پیش‌پردازنده‌های سگمنت کامل است.
- PUBLIC لینکر، سگمنت را با همهٔ دیگر سگمنت‌هایی که بصورت PUBLIC تعریف شده‌اند و نام سگمنت و کلاس یکسانی دارند ترکیب می‌کند. اسمبلر افست را از شروع اولین سگمنت محاسبه می‌کند. در حقیقت، سگمنت ترکیبی شامل تعدادی بخش است، که هر یک با پیش‌پردازنده SEGMENT آغاز و با END خاتمه می‌دهد. این نوع، پیش‌فرض برای پیش‌پردازنده‌های سگمنت ساده شده است.
- COMMON اگر سگمنت‌های COMMON نام و کلاس یکسانی داشته باشند، لینکر به آنها آدرس پایه یکسانی می‌دهد. در طی اجرا، دومین سگمنت بر روی اولین سگمنت قرار می‌گیرد. بزرگترین سگمنت طول ناحیه مشترک را

تعیین می‌کند.

نوع کلاس

شما همیشه از نام‌های کلاس 'stack' ، 'Data' و 'Code' استفاده می‌کنید. می‌توانید نام کلاس مشابهی را برای سگمنت‌های مرتبط مشخص کنید بنابراین اسمبلر و لینکر آنها را با هم در یک گروه قرار می‌دهد. به عبارت بهتر، در مورد سگمنت‌هایی که به صورت متوالی قرار می‌گیرند، تا وقتی گزینهٔ PUBLIC انتخاب نشده است در قالب یک سگمنت با هم ترکیب نمی‌شوند. هر نام معتبری می‌تواند باشد، که در داخل یک جفت علائم نقل قول تکی قرار می‌گیرد با این حال نام code برای سگمنت کد توصیه می‌شود.

دو جمله SEGMENT غیر مرتبط زیر حاصل یکسانی دارند، یک سگمنت کد غیر وابسته که از مرز یک پاراگراف آغاز می‌شود.

```
CODESG1   SEGMENT  PARA NONE 'Code'
CODESG2   SEGMENT  'Code'  (پیش فرض NONE,PARA می‌باشد)
```

ما پیش پردازنده‌های سگمنت تعریف شده کامل را در فصل ۴ توضیح دادیم اما در فصل‌های قبلی از پیش‌پردازنده‌های سگمنت ساده استفاده کردیم. چون پیش‌پردازنده سگمنت کامل کنترل دقیق‌تری را هنگام اسمبل و لینک شدن برنامه‌ها فراهم می‌سازد، اغلب مثالهای این فصل از آن استفاده می‌کنند. مثال‌های برنامه‌نویسی این فصل و فصل‌های بعدی بسیاری از گزینه‌های کلاس، ترکیب و مرزبندی را مشخص می‌سازند.

فراخوانی‌های درون سگمنتی

دستورات CALL، تا اینجا، برای فراخوانی‌های **درون سگمنتی** استفاده می‌شود، که در این روش روال فراخوانده شده در همان سگمنت کدی است که روال فراخوانی قرار دارد. یک فراخوانی **درون سگمنت**، نزدیک است اگر روال فراخوانده شده بصورت NEAR تعریف شده باشد (که داخل 32K±است). فراخوانی نزدیک ثبات IP را روی پشته قرار می‌دهد و IP را با افست آدرس مقصد جایگزین می‌سازد. بنابراین فراخوانی نزدیک به یک روال نزدیک داخل همان سگمنت مراجعه می‌کند.

```
            CALL  nearproc    ;Near call: push IP,
            ....              ;  link to nearproc
nearproc    PROC  NEAR
            ...
            RET/RETN          ;Near return: pop IP,
nearproc    ENDP              ;  return to caller
```

حال فرض کنید یک جمله CALL درون سگمنت حاوی کد مقصد E82000 است. E8کد عملیاتی برای CALL و 2000 (0020) افست یک روال فراخوانده شده می‌باشد. عملیات IP را روی پشته قرار می‌دهد و 2000 را بصورت 0020 در IP ذخیره می‌سازد. سپس پردازشگر، آدرس سگمنت جاری در CS را با افست IP ترکیب می‌کند (CS:IP) تا دستور بعدی برای اجرا مشخص شود. زمان خروج از روال فراخوانده شده، یک RET (نزدیک) IP را از روی پشته برمی دارد و در IP می‌گذارد که بصورت افست : سگمنت ترکیب می‌شود و آدرس دستور بعد از CALL تولید می‌شود.

همان طور که گفته شد، یک فراخوانی درون سگمنت ممکن است نزدیک باشد، یا فراخوانی روالی که در همان سگمنت بصورت دور تعریف شده است. فراخوانی از نوع دور است. RET نزدیک است اگر در یک روال NEAR ظاهر شده باشد، دور است اگر در یک روال FAR ظاهر شود. می‌توانید این دستورات را بصورت RETN یا RETF تعریف کنید.

فراخوانی‌های میان سگمنتی

یک CALL بصورت **دور** کلاس بندی می‌شود اگر روال فراخوانده شده بصورت FAR یا EXTRN تعریف شده باشد اما اغلب لازم نیست که در سگمنت دیگری باشد. CALL دور ابتدا محتویات ثبات CS را روی پشته قرار می‌دهد و یک آدرس افست جدید در IP می‌گذارد. (مقادیر CS:IP روی پشته آدرس دستور بلافاصله بعد از CALL را فراهم می‌سازد). با این روش، هر دو آدرس سگمنت کد و افست برای بازگشت از روال فراخوانده شده ذخیره می‌شوند.

```
        CALL  farproc     ;Far call: push CS and IP,
        ...               ;  link to farproc
farproc PROC  FAR
        ...
        RET/RETF          ;Far return: pop IP and CS,
farproc ENDP              ;  return to caller
```

یک دستور CALL میان سگمنتی که حاوی کد مقصد 9A 0002 AF04 است در نظر بگیرید. 9Aکد عملیاتی برای یک CALL دور، 0002 (یا 0200) آدس افست و AF04 (یا 04AF) آدرس سگمنت جدید می‌باشد. عملیات IP جاری را روی پشته می‌گذارد و افست جدید 0002 بصورت 0200 را در IP ذخیره می‌کند. سپس CS جاری را روی پشته می‌گذارد و آدرس سگمنت جدید را AF04 بصورت 04AF در CS قرار می‌دهد. پردازشگر آدرس سگمنت جاری در CS را با افست IP (CS:IP) ترکیب می‌کند تا آدرس اولین دستور اجرایی در زیر برنامه فراخوانده شده فراهم شود.

آدرس در سگمنت کد : 04AF0H
افست در IP : +0200H
آدرس مؤثر : 04CF0H

با خروج از روال فراخوانده شده، یک RET (دور) بین قطعه، عکس عملیات CALL را انجام می‌دهد. آدرسهای اصلی IP و CS را از روی پشته برمی دارد و در ثبات‌ها قرار می‌دهد. حال جفت ثبات CS:IP به آدرس دستور بعد از CALL اصلی اشاره می‌کنند، جائیکه اجرا از سر گرفته می‌شود.

صفات EXTRN و PUBLIC

در شکل ۲-۲۳، برنامه اصلی (MAINPROG) یک زیر برنامه SUBPROG را فرا می‌خواند، در اینجا یک CALL میان سگمنتی نیاز است.

```
            EXTRN   SUBPROG:FAR
MAINPROG    PROC    FAR
            ...
            CALL    SUBPROG
            ...
MAINPROG    ENDP
-------------------------------
            PUBLIC  SUBPROG
SUBPROG     PROC    FAR
            ...
            RETF
SUBPROG     ENDP
```

شکل ۲-۲۳ فراخوانی بین سگمنت

فراخوانی در MAINPROG باید بداند که SUBPROG خارج از این سگمنت قرار دارد (در غیر اینصورت اسمبلر یک یک پیغام خطا که SUBPROG یک نماد تعریف نشده است تولید می‌کند). پیش‌پردازنده EXTRN SUBPROG:FAR به اسمبلر یادآوری می‌کند که هر ارجاع به SUBPROG به یک برچسب FAR است که در این حالت بصورت خارجی در دیگر مراحل اسمبل تعریف شده است. از آنجائیکه اسمبلر روشی برای دانستن اینکه چه آدرس در زمان اجرا خواهد داشت، ندارد در فراخوانی دور عملوندهای کد مقصد را "خالی" تولید می‌کند (لیست برای افست صفر و برای سگمنت خط فاصله نشان می‌دهد)، که بعداً لینکر آن را پر خواهد کرد :

MASM : 9A 0000 ---- E ; E = خارجی
TASM : 9A 0000 0000se ; se= سگمنت خارجی

SUBPROG دارای یک پیش پردازنده PUBLIC است که به اسمبلر و لینکر می‌گوید که ماجول دیگری باید آدرس SUBPROG را بداند. در مرحله بعد، وقتی MAINPROG و SUBPROG با موفقیت در ماجولهای مقصد جداگانه‌ای اسمبل شدند، ممکن است بصورت زیر با هم پیوند زده شوند :

```
LINK/TLINK  D:MAINPROG+D:SUBPROG,D:,CON
```

لینکر، EXTRNهای یک ماجول مقصد را با PUBLICهای دیگر جور می‌کند و هر آدرس افست ضروری را جایگزین می‌سازد. سپس لینکر در ماجول مقصد را در یک ماجول قابل اجرا تلفیق می‌کند. اگر لینکر قادر نباشد ارجاعات را با یکدیگر تطبیق دهد، پیغام‌های خطا صادر می‌کند. قبل از اجرا به خطاها توجه کنید.

پیش پردازنده EXTERN/EXTRN

پیش پردازنده EXTERN به اسمبلر می‌گوید که عنصر نامبرده - عنصر داده روال، یا برچسب - در مقطع دیگری از اسمبل تعریف و می‌شود. (MASM 6.9 با عبارت EXETRN آن را معرفی می‌کند). شکل کلی آن بصورت زیر است :

```
EXTRN/EXTERN name : type [ , . . . ]
```

می‌توانید بیش از یک نام تا انتهای یک خط تعریف کنید یا جمله EXTRNدیگری کد نمایید. دیگر ماجول اسمبلی بنوبه خودباید نام را بصورت PUBLIC تعریف و مشخص‌نماید. نوع ورودی باید برحسب تعریف واقعی یک نام‌معتبرباشد :

- ABS یک مقدار ثابت را مشخص می‌کند.
- BYTE ، WORD و DWORD عناصر داده‌ای را مشخص می‌سازند که یک ماجول ارجاع می‌کند ولی ماجول دیگر تعریف نموده است.
- NEAR و FAR یک روال یا برچسب دستورات را مشخص می‌کند که یک ماجول ارجاع می‌کند ولی ماجول دیگری تعریف می‌کند.
- یک نام توسط EQU تعریف شده است.

پیش پردازنده PUBLIC

پیش پردازنده PUBLIC به اسمبلر و لینکر می‌گوید که آدرس یک سمبول تعریف شده خاص در اسمبل فعلی در ماجول دیگری قرار دارد. قالب کلی PUBLIC چنین است :

```
PUBLIC symbol [, . . .]
```

می‌توانید چندین سمبول تا انتهای خط تعریف کنید یا یک جمله PUBLIC دیگری قرار دهید. سمبول ورودی می‌تواند یک برچسب (شامل برچسب PROC)، یک متغیر، یا یک عدد باشد. ورودیهای نامعتبر شامل نام ثبات و سمبول‌های EQU است که مقادیر تعریف شده آنها بزرگتر از ۲ بایت است.

فراخوانی روال‌های دور و استفاده از EXTRN و PUBLIC مشکل نیستند، اگر چه برای گرفتن داده تعریف شده در یک ماجول، شناخته شده در ماجول دیگر، وقت لازم دارد.

حال سه روش مختلف اخذ دادهٔ شناخته شدهٔ بین برنامه‌ها را بررسی می‌کنیم با استفاده از PUBLIC و EXTRN، تعریف داده مشترک در زیر برنامه، و ارسال پارامترها.

استفاده از EXTRN و PUBLIC برای یک نقطه ورودی

برنامه شکل ۳-۲۳ شامل یک برنامه اصلی است A23MAIN1 و یک زیر برنامه A23SUB1 و هر دو از پیش پردازنده‌های سگمنت کامل استفاده می‌کنند. برنامه اصلی سگمنت‌ها را برای پشته، داده، و کد تعریف می‌کند. سگمنت

داده QTY و PRICE را تعریف می‌کند.

```
                        TITLE     A23MAIN1 (EXE)  Call subprogram
                                  EXTRN  A23SUB1:FAR
                        ; ----------------------------------------
0000                    STACKSG   SEGMENT PARA STACK 'Stack'
0000 0040[????]                   DW      64 DUP(?)
0080                    STACKSG   ENDS
                        ; ----------------------------------------
0000                    DATASG    SEGMENT PARA 'Data'
0000 0140               QTY       DW      0140H
0002 2500               PRICE     DW      2500H
0004                    DATASG    ENDS
                        ; ----------------------------------------
0000                    CODESG    SEGMENT PARA 'Code'
0000                    BEGIN     PROC    FAR
                                  ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                    MOV     AX,DATASG
0003 8E D8                        MOV     DS,AX
0005 A1 0002 R                    MOV     AX,PRICE      ;Set up price
0008 8B 1E 0000 R                 MOV     BX,QTY        ;  and quantity
000C 9A 0000 ---- E               CALL    A23SUB1       ;Call subprogram
0011 B8 4C00                      MOV     AX,4C00H      ;End processing
0014 CD 21                        INT     21H
0016                    BEGIN     ENDP
0016                    CODESG    ENDS
                                  END     BEGIN

Segments and Groups:
    N a m e             Length    Align     Combine    Class
CODESG . . . . . .      0016      PARA      NONE       'CODE'
DATASG . . . . . .      0004      PARA      NONE       'DATA'
STACKSG  . . . . .      0080      PARA      STACK      'STACK'
Symbols:
    N a m e             Type      Value      Attr
A23SUB1  . . . . .      L FAR     0000       External
BEGIN  . . . . . .      F PROC    0000       CODESG     Length = 0016
PRICE  . . . . . .      L WORD    0002       DATASG
QTY  . . . . . . .      L WORD    0000       DATASG
-------------------------------------------------------------------------
                        TITLE     A23SUB1 Called subprogram
0000                    CODESG    SEGMENT PARA 'Code'
0000                    A23SUB1   PROC    FAR
                                  ASSUME CS:CODESG
                                  PUBLIC A23SUB1
0000 F7 E3                        MUL     BX        ;AX = price, BX = qty
0002 CB                           RETF              ;DX:AX = product
0003                    A23SUB1   ENDP
0003                    CODESG    ENDS
                                  END     A23SUB1

Segments and Groups:
    N a m e             Length    Align     Combine    Class
CODESG . . . . . .      0003      PARA      NONE       'CODE'
Symbols:
    N a m e             Type      Value      Attr
A23SUB1  . . . . .      F PROC    0000       CODESG  Global Length = 0003
-------------------------------------------------------------------------

Link Map
Object Modules: A23MAIN1+A23SUB1

 Start  Stop  Length Name          Class
00000H 0007FH 00080H STACKSG       STACK
00080H 00083H 00004H DATASG        DATA
00090H 000A5H 00016H CODESG        CODE <-- Note 2 code
000B0H 000B2H 00003H CODESG        CODE <--   segments

Program entry point at 0009:0000
```

شکل ۳-۲۳ استفاده از *EXTRN* و *PUBLIC*

سگمنت کد AX را با PRICE و BX را با QTY مقدار دهی می‌کند و سپس زیر برنامه را فرامی‌خواند.

یک EXTRN در برنامه اصلی نقطه ورودی به زیر برنامه را بصورت A23SUB1:FAR مشخص می‌سازد. زیر برنامه حاوی یک جمله PUBLIC (بعد از ASSUME) است که A23SUBI را برای لینکر بصورت نقطه ورود جهت اجرا می‌شناساند. این زیر برنامه محتویات AX (قیمت) را در BX (میزان فروش) ضرب می‌کند و حاصل را در DX:AX بصورت 002E 4000H ذخیره می‌سازد. چون زیربرنامه هیچ داده‌ای تعریف نمی‌کند، نیازی به یک سگمنت داده ندارد، می‌تواند داده تعریف کند، ولی فقط زیر برنامه داده‌ها را تشخیص می‌دهد.

ثبات‌های SS و SP در زیر برنامه حاوی همان آدرس‌هایی هستند که در برنامه اصلی داشتند. در نتیجه زیر برنامه، به پشته تعریف شده در برنامه اصلی مراجعه می‌کند و خود اقدام به تعریف پشته نمی‌کند. چون لینکر به تعریف پشته در یک برنامه EXE نیاز دارد، پشته تعریف شده در برنامه اصلی بدین منظور در نظر گرفته می‌شود.

حال بیایید جدول سمبول را پس از هر مرحله اسمبل بررسی کنیم. توجه کنید که جدول سمبول بُرای برنامه اصلی A23SUB1 را بصورت دورو خارجی نشان می‌دهد. جدول سمبول برای زیر برنامه A23SUB1 را بصورت F (برای دور) و محلی نشان می‌دهد. عبارت global یعنی نام در دیگر زیر برنامه‌های خارج از A23SUB1 شناخته شده است.

نقشه پیوند در انتهای لیست، سازماندهی برنامه در حافظه را نشان می‌دهد. توجه کنید که یک پشته و یک سگمنت داده اما ۲ سگمنت کد وجود دارد (یکی برای هر مرحله اسمبل) با دو آدرس شروع مختلف ، چون نوع ترکیب آنها NONE می‌باشد. این سگمنت‌ها به ترتیبی که در فرمان LINK وارد کرده‌اید ظاهر می‌شوند. با این مثال ، سگمنت کد برای برنامه اصلی (معمولاً اولی) از افست 00090H و سگمنت کد زیر برنامه از افست 000B0H کد زیر را تولید می‌کند.

9A 0000 220F (مقدار سگمنت شما متفاوت است)

کد ماشین برای یک فراخوانی بین سگمنت 9AH است. عملیات ثبات IP را روی پشته قرار می‌دهد و 0000 را در IP می‌گذارد (اولین عملوند CALL). سپس CS حاوی 0F20[0] را روی پشته و 0F22[0] دومین عملوند را در CS قرار می‌دهد. (دیدیم که محتویات ثبات به ترتیب بایت معکوس نیست).

جفت CS:IP دستور بعدی جهت اجرا را در 0F22[0] بعلاوه 0000 مشخص می‌سازد. در 0F220 چیست؟ نقطه ورود به A23SUB1 از اولین دستور اجرایی ، که می‌توانید محاسبه کنید. برنامه اصلی باثبات CS که حاوی 0F20[0] آغاز شده است. بر طبق نقشه افست سگمنت کد اصلی از 00090H و افست زیر برنامه از افست 000B0H آغاز می‌شود، 20H بایت بعد ، جمع مقدار CS از برنامه اصلی با 20H آدرس موثر سگمنت کد زیر برنامه را فراهم می‌سازد :

```
آدرس CS برای A23MAIN1 ;   0F200H
اندازه A23MAIN1 ;         +00020H
آدرس CS برای A23SUB1 ;    0F220H
```

بارگذار برنامه این آدرس را دقیقاً مانند آنچه ما داریم تعیین می‌کند و در عملوند CALL جایگزین می‌سازد. A23SUB1 دو مقدار AX و BX را در هم ضرب می‌کند و حاصل در DX:AX تولید می‌کند و یک بازگشت دور (RETF) به A23MAIN1 خواهد داشت (چون بازگشت به روال FAR می‌باشد).

تعریف سگمنت کد بصورت PUBLIC

شکل ۴-۲۳ حالت دیگری از شکل ۳-۲۳ است. یک تغییر در برنامه اصلی A23MAIN2 و زیر برنامه A23SUB2 داده شده است ، که در هر دو از PUBLIC در پیش پردازنده SEGMENT برای سگمنت کد استفاده شده است :

```
CODESG SEGMENT PARA PUBLIC 'code'
```

نتیجه جالبی در نقشه پیوند و کد مقصد CALL ظاهر می‌شود. در جدول سمبولیک که بعد از هر مرحله اسمبل

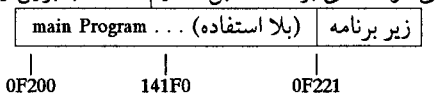
ایجاد می‌شود ، نوع ترکیب CODESG ، PUBLIC است در حالیکه در شکل ۳-۲۳ NONE می‌باشد. همچنین نقشه پیوند، در انتها فقط یک سگمنت کد را نمایش می‌دهد. این واقعیت که هر دو سگمنت نام یکسان (CODESG) ، نوع ('code') و صفت PUBLIC دارند سبب شده تا لینکر دو سگمنت کد منطقی را در یک سگمنت کد فیزیکی ترکیب کند. پیگیری اجرا نشان می‌دهد که CALL از نوع دور است، که حتی فراخوانی در داخل همان سگمنت به یک روال دور است ، بطوریکه تعریف شده است.

(توجه کنید که آدرس سگمنت شما متفاوت است) 9A 2000 200F

این فراخوانی دور 2000H را در IP بصورت 0020H و 200FH را در CS بصورت 0F20[0] ذخیره می‌سازد. چون زیر برنامه یک سگمنت کد عمومی با برنامه اصلی به اشتراک می‌گذارد ، ثبات CS با همان آدرس شروع 0F20H تنظیم می‌شود. اما CS:IP برای A23SUB2 آدرس موثر زیر را فراهم می‌سازد :

```
0F200H    ; آدرس CS برای A23SUB2 , A23MAIN2
+0020H    ; افست IP برای A23SUB2
0F220H    ; آدرس موثر برای A23SUB2
```

بنابراین احتمالاً سگمنت کد زیر برنامه از 0F220H آغاز می‌شود آیا این موضوع درست است؟ نقشه پیوند این نقطه را آشکار نمی‌سازد ، اما می‌توانید این آدرس را ، از لیست برنامه اصلی که در افست 0015H خاتمه می‌یابد ، استنباط کنید. (نقطه 16H را نشان می‌دهد ، که موقعیت بعدی است). چون سگمنت کد برای زیر برنامه بصورت PARA تعریف شده است ، از مرز یک پاراگراف آغاز می‌شود (حتی بر 10H قابل تقسیم است ، بنابراین رقم سمت راست 0 می‌باشد.)



لینکر ، زیر برنامه را از اولین مرز پاراگراف بلافاصله بعد از برنامه اصلی از افست 00020H تنظیم می‌کند. بنابراین ، دقیقاً همانطوری که محاسبه شد ، سگمنت کد در زیر برنامه از 0F200H بعلاوه 0020H ، یا 0F220H آغاز می‌شود.

حال همین برنامه را با تعریف پیش پردازنده‌های سگمنت ساده شده بررسی می‌کنیم.

استفاده از پیش‌پردازنده‌های سگمنت ساده شده

شکل ۵-۲۳ برنامه قبلی را با پیش پردازنده‌های سگمنت ساده شده نشان می‌دهد. شکل ۴-۲۳ سگمنت کد را بصورت PUBLIC تعریف می‌کند. در حالیکه در شکل ۵-۲۳ پیش فرض PUBLIC است ، بنابراین هر دو مثال یک سگمنت کد تولید می‌کنند. اما استفاده از پیش پردازنده‌های سگمنت ساده شده سبب برخی تفاوت‌های علمی خواهد شد. ابتدا ، لینکر سگمنت‌ها را (همانطور که نقشه نشان داده) بترتیب کد ، داده و پشته مرتب می‌کند. اگر چه تاثیری بر اجرای برنامه ندارد. ثانیاً کد سگمنت زیر برنامه (-TEXT) از مرز یک کلمه (نه یک پاراگراف) آغاز می‌کند. پیگیری اجرای برنامه که مقصد زیر را برای CALL نمایش می‌دهد.

(توجه کنید که آدرس سگمنت شما تفاوت دارد) 9A 1600 170F

هم اکنون ، مقدار افست جدید 16H است و آدرس سگمنت 0F17H می‌باشد. چون زیر برنامه یک سگمنت کد عمومی با برنامه اصلی به اشتراک می‌گذارد ، ثبات CS با همان آدرس شروع 0F17(0) برای هر دو تنظیم می‌شود. ممکن است آدرس موثر A23SUB3 را بصورت زیر محاسبه کنید :

```
F170H     ; آدرس CS برای A23MAIN3 و A23SUB3
+ 016H    ; افست IP برای A23SUB3
F186H     ; آدرس موثر برای A23SUB3
```

```
                        TITLE     A23MAIN2 (EXE)  Call subprogram
                                  EXTRN  A23SUB2:FAR
                        ; ----------------------------------------
0000                    STACKSG   SEGMENT PARA STACK 'Stack'
0000 0040[????]                   DW     64 DUP(?)
0080                    STACKSG   ENDS
                        ; ----------------------------------------
0000                    DATASG    SEGMENT PARA 'Data'
0000 0140               QTY       DW     0140H
0002 2500               PRICE     DW     2500H
0004                    DATASG    ENDS
                        ; ----------------------------------------
0000                    CODESG    SEGMENT PARA PUBLIC 'Code'
0000                    BEGIN     PROC   FAR
                                  ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                    MOV    AX,DATASG
0003 8E D8                        MOV    DS,AX
0005 A1 0002 R                    MOV    AX,PRICE      ;Set up price
0008 8B 1E 0000 R                 MOV    BX,QTY        ; and quantity
000C 9A 0000 ---- E               CALL   A23SUB2       ;Call subprogram
0011 B8 4C00                      MOV    AX,4C00H      ;End processing
0014 CD 21                        INT    21H
0016                    BEGIN     ENDP
0016                    CODESG    ENDS
                                  END    BEGIN

Segments and Groups:
    N a m e         Length     Align      Combine    Class
CODESG . . . . . .  0016       PARA       PUBLIC     'CODE'
DATASG . . . . . .  0004       PARA       NONE       'DATA'
STACKSG  . . . . .  0080       PARA       STACK      'STACK'
Symbols:
    N a m e          Type      Value       Attr
A23SUB2  . . . . .  L FAR      0000       External
BEGIN  . . . . . .  F PROC     0000       CODESG  Length = 0016
PRICE  . . . . . .  L WORD     0002       DATASG
QTY  . . . . . . .  L WORD     0000       DATASG
-------------------------------------------------------------------

                        TITLE     A23SUB2 Called subprogram
                        ; -------------------------------
0000                    CODESG    SEGMENT PARA PUBLIC 'Code'
0000                    A23SUB2   PROC   FAR
                                  ASSUME CS:CODESG
                                  PUBLIC A23SUB2
0000 F7 E3                        MUL    BX        ;AX = price, BX = qty
0002 CB                           RETF             ;DX:AX = product
0003                    A23SUB2   ENDP
0003                    CODESG    ENDS
                                  END    A23SUB2

Segments and Groups:
    N a m e         Length     Align   Combine  Class
CODESG . . . . . .  0003       PARA    PUBLIC   'CODE'
Symbols:
    N a m e          Type      Value    Attr
A23SUB2  . . . . .  F PROC     0000    CODESG  Global  Length = 0003
```

```
Link Map
Object Modules: A23MAIN2+A23SUB2

 Start  Stop   Length Name       Class
 00000H 0007FH 00080H STACKSG   STACK
 00080H 00083H 00004H DATASG    DATA
 00090H 000B2H 00023H CODESG    CODE <-- 1 code
                                         segment
 Program entry point at 0009:0000
```

شکل ۴-۲۳ سگمنت کد تعریف شده بصورت *PUBLIC*

```
                        TITLE     A23MAIN3 (EXE)  Call subprogram
                                  .MODEL  SMALL
                                  .STACK  64
                                  EXTRN   A23SUB3:FAR
                        ; ----------------------------------------
                                  .DATA
0000 0140               QTY       DW      0140H
0002 2500               PRICE     DW      2500H
                        ; ----------------------------------------
                                  .CODE
0000                    BEGIN     PROC    FAR
0000 B8 ---- R                    MOV     AX,@data
0003 8E D8                        MOV     DS,AX
0005 A1 0002 R                    MOV     AX,PRICE      ;Set up price
0008 8B 1E 0000 R                 MOV     BX,QTY        ;  and quantity
000C 9A 0000 ---- E               CALL    A23SUB3       ;Call subprogram
0011 B8 4C00                      MOV     AX,4C00H      ;End processing
0014 CD 21                        INT     21H
0016                    BEGIN     ENDP
                                  END     BEGIN

Segments and Groups:
    N a m e             Length    Align     Combine   Class
DGROUP . . . . . .      GROUP
 _DATA . . . . . .      0004      WORD      PUBLIC    'DATA'
 STACK . . . . . .      0040      PARA      STACK     'STACK'
_TEXT  . . . . . .      0016      WORD      PUBLIC    'CODE'
Symbols:
    N a m e             Type      Value      Attr
A23SUB3  . . . . . .    L FAR     0000       External
BEGIN  . . . . . .      F PROC    0000       _TEXT  Length = 0016
PRICE  . . . . . .      L WORD    0002       _DATA
QTY  . . . . . .        L WORD    0000       _DATA
```

```
                        TITLE     A23SUB3 Called subprogram
                                  .MODEL SMALL
                                  .CODE
0000                    A23SUB3   PROC    FAR
                                  PUBLIC A23SUB3
0000 F7 E3                        MUL     BX        ;AX = price, BX = qty
0002 CB                           RETF              ;DX:AX = product
0003                    A23SUB3   ENDP
                                  END     A23SUB3

Segments and Groups:
    N a m e             Length    Align     Combine   Class
DGROUP . . . . . .      GROUP
 _DATA . . . . . .      0000      WORD      PUBLIC    'DATA'
_TEXT  . . . . . .      0003      WORD      PUBLIC    'CODE'

Symbols:
    N a m e             Type      Value      Attr
A23SUB3  . . . . . .    F PROC    0000       _TEXT  Global  Length=0003
```

```
Link Map
Object Modules: A23MAIN3+A23SUB3

 Start  Stop   Length Name        Class
 00000H 00018H 00019H _TEXT      CODE <-- code segment
 0001AH 0001DH 00004H _DATA      DATA          first
 00020H 0005FH 00040H STACK      STACK

 Program entry point at 0000:0000
```

شکل ۵-۲۳ استفاده از پیش پردازنده‌های سگمنت ساده شده.

می‌توانید آدرس را از لیست برنامه اصلی که در افست 0015H خاتمه می‌پذیرد ، استنباط نمایید. (نقشه 16Hرا نشان می‌دهد که موقعیت در دسترس بعدی است). چون نقشه ، سگمنت کد اصلی را با شروع از 00000H نشان می‌دهد ، مرز کلمه بعدی پس از 0015H در 00016H است که A23SUB3 شروع می‌شود.

تعریف داده‌های مشترک بصورت PUBLIC

یک ضرورت عمومی برنامه نویسی پردازش داده‌هایی است که در یک ماجول هستند و در ماجول دیگری تعریف می‌شوند.

بیایید مثال قبل را طوری تغییر دهیم ، که اگر چه برنامه اصلی هنوز QTY و PRICE را تعریف می‌کند ، زیربرنامه این مقادیر را در BX و AX قرار می‌دهد. شکل ۶-۲۳ کد بازنویسی شده با تغییرات زیر را نشان می‌دهد :

- برنامه اصلی ، A23MAIN4 ، QTY و PRICE را بصورت PUBLIC تعریف می‌کند. سگمنت داده نیز با صفت PUBLIC تعریف می‌شوند. توجه کنید که در جدول سمبول QTY و PRICE صفت سراسری دارند.
- زیر برنامه A23SUB4 ، QTY و PRICE را بصورت EXTRN و WORD تعریف می‌کند این تعریف اسمبلر را از طول دو فیلد آگاه می‌سازد. اسمبلر می‌تواند کد عملیاتی صحیح برای دستورات MOV را تولید کند ، اما لینکر باید عملوندها را تکمیل سازد. (در جدول سمبول زیربرنامه ، PRICE و QTY بصورت خارجی هستند).

اسمبلر دستورات MOV در زیربرنامه را بصورت زیر لیست می‌کند.

```
A1 0000 E          MOV AX,PRICE
8B 1E 0000 E       MOV BX,QTY
```

حال کد مقصدی که تولید شده برای هر سه برنامه قبلی یکسان است در جائیکه دستورات MOV در برنامه فراخوان قرار دارند. این یک نتیجه منطقی است چون عملوندها در هر سه برنامه آدرس سگمنت داده را در ثبات DS و مقادیر افست یکسانی را ارجاع می‌کنند.

برنامه اصلی و زیربرنامه ممکن است دیگر عناصر داده را تعریف کنند ، اما فقط آنهایی که بصورت PUBLIC و EXTRN تعریف می‌شوند بصورت مشترک شناخته می‌شوند.

کد A1 به مفهوم انتقال یک داده دو بایتی از حافظه به ثبات AX و کد 8B به منزله انتقال یک داده دو بایتی به ثبات BX می‌باشد. (عملیاتی که روی AX انجام می‌شوند، غالباً به بایت‌های کنترلی نیاز دارند.) در مورد A23SUB4 اسمبلر هیچ ایده‌ای برای تعیین موقعیت QTY و PRICE ندارد، در نتیجه در عملوندهای هر دو دستورالعمل MOV ، مقدار صفر را ذخیره می‌کند. با پیمایش مرحله‌ای اجرای برنامه به این نتیجه می‌رسیم که لینکر برای کد مربوط به عملوندها مقادیر زیر را قرار می‌دهد :

```
A1 0200
8B 1E 0000
```

تعریف داده در هر دو برنامه

در مثال بعد A23MAIN4 ، QTY و PRICE را تعریف می‌کند ، در حالیکه A23SUB4 هیچ داده‌ای تعریف نمی‌کند. دلیل اینکه A23SUB4 می‌تواند به داده‌های A23MAIN4 مراجعه کند این است که آدرس سگمنت داده در ثبات DS، هنوز به سگمنت داده در A23MAIN4 اشاره‌می‌کند. (انتها آدرس سگمنت تغییریافته سگمنت کد می‌باشد.) اما برنامه‌ها همیشه اینقدر ساده نیستند و زیربرنامه‌ها اغلب نه فقط داده‌های خود را تعریف می‌کنند بلکه به داده‌های برنامه فراخوان نیز مراجعه می‌کنند ، همانطوری که مثال بعد نشان می‌دهد در تغییرات برنامه قبل، شکل ۷-۲۳ QTY را در A23MAIN5 تعریف می‌کند، اما PRICE در A23SUB5 تعریف می‌شود. از داخل A23MAIN5 ، PRICE خارج

نمی‌شود ، اگر چه A23SUB5 باید موقعیت هر دو عنصر را بداند. سگمنت کد A23SUB5 باید QTY را بازیابی کند ، هنگامیکه ثبات DS هنوز حاوی آدرس سگمنت داده A23MAIN5 است. سپس A23SUB5 ثبات DS را بر روی پشته می‌گذارد و DS را با آدرس سگمنت داده خود بارگذاری می‌کند. A23SUB5 هم اکنون می‌تواند PRICE را بگیرد و ضرب QTY در PRICE را انجام دهد.

قبل از بازگشت به A23MAIN5 و A23SUB5 باید DS را از روی پشته بردارد بنابراین A23MAIN می‌تواند سگمنت داده خودش را دستیابی کند. (بطور تکنیکی ، در این مثال ، لازم نیست ، چون A23MAIN5 بلافاصله پردازش را خاتمه می‌دهد، (ما بعنوان تجربه استاندارد آن را انجام می‌دهیم.)

```
                        TITLE   A23MAIN4 (EXE)  Call subprogram
                                EXTRN  A23SUB4:FAR
                                PUBLIC QTY,PRICE
                        ; ----------------------------------------
0000                    STACKSG SEGMENT PARA STACK 'Stack'
0000 0040[????]                 DW      64 DUP(?)
0080                    STACKSG ENDS
                        ; ----------------------------------------
0000                    DATASG  SEGMENT PARA PUBLIC 'Data'
0000 0140               QTY     DW      0140H
0002 2500               PRICE   DW      2500H
0004                    DATASG  ENDS
                        ; ----------------------------------------
0000                    CODESG  SEGMENT PARA PUBLIC 'Code'
0000                    BEGIN   PROC    FAR
                                ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                  MOV     AX,DATASG
0003 8E D8                      MOV     DS,AX
0005 9A 0000 ---- E             CALL    A23SUB4         ;Call subprogram
000A B8 4C00                    MOV     AX,4C00H        ;End processing
000D CD 21                      INT     21H
000F                    BEGIN   ENDP
000F                    CODESG  ENDS
                                END     BEGIN

Segments and Groups:
    N a m e             Length   Align   Combine   Class
CODESG . . . . . .      000F     PARA    PUBLIC    'CODE'
DATASG . . . . . .       0004    PARA    PUBLIC    'DATA'
STACKSG  . . . . .       0080    PARA    STACK     'STACK'
Symbols:
    N a m e              Type    Value    Attr
A23SUB4  . . . . .      L FAR    0000     External
BEGIN  . . . . . .      F PROC   0000     CODESG    Length = 000F
PRICE  . . . . . .      L WORD   0002     DATASG    Global
QTY  . . . . . . .      L WORD   0000     DATASG    Global
-------------------------------------------------------------------

                        TITLE   A23SUB4 Called subprogram
                                EXTRN  QTY:WORD, PRICE:WORD
                        ; --------------------------------
0000                    CODESG  SEGMENT PARA PUBLIC 'CODE'
0000                    A23SUB4 PROC    FAR
                                ASSUME CS:CODESG
                                PUBLIC A23SUB4
0000 A1 0000 E                  MOV     AX,PRICE
0003 8B 1E 0000 E               MOV     BX,QTY
0007 F7 E3                      MUL     BX          ;DX:AX = product
0009 CB                         RETF                ;Return to caller
000A                    A23SUB4 ENDP
000A                    CODESG  ENDS
                                END     A23SUB4
```

شکل الف ٦-٢٣ داده‌های مشترک در زیر برنامه‌ها

```
Segments and Groups:
   N a m e          Length   Align  Combine    Class
CODESG . . . . . .  000A     PARA    PUBLIC     'CODE'
Symbols:
   N a m e          Type     Value    Attr
A23SUB4  . . . . .  F PROC   0000     CODESG  Global  Length = 000A
PRICE  . . . . . .  V WORD   0000     External
QTY  . . . . . .    V WORD   0000     External
```

```
Link Map
Object Modules: A23MAIN4+A23SUB4

 Start  Stop   Length Name          Class
 00000H 0007FH 00080H STACKSG      STACK
 00080H 00083H 00004H DATASG       DATA
 00090H 000A9H 0001AH CODESG       CODE

 Program entry point at 0009:0000
```

شکل ب ٦-٢٣ داده‌های مشترک در زیر برنامه‌ها

بعنوان نکته نهایی ، می‌توانید هر دو سگمنت داده را PUBLIC و با یک نام و کلاس قرار دهید. در آن حالت ، لینکر آنها را با هم ترکیب می‌کند و A23SUB5 نباید DS را PUSH و POP کند ، چون برنامه از یک سگمنت داده و آدرس DS استفاده خواهد کرد. ما این تغییرات را بصورت تمرینی برای شما خواهیم گذاشت تا آن را بازنویسی کرده و با DEBUG پیگیری نمایید ، سگمنت کد A23SUB5 می‌تواند چنین باشد.

```
EXTRN   QTY:WORD
ASSUME  CS:CODESG,DS:DATASG
PUBLIC  A23SUB5
MOV     AX,PRICE    ; PRICE در سگمنت داده خود
MOV     BX,QTY      ; QTY در A23MAIN
MUL     BX          ; حاصل در DX:AX
RETF                بار سگمنت دور
```

ارسال پارامترها به یک زیربرنامه

روش دیگر شناساندن داده به یک زیربرنامه فراخوانده شده ارسال پارامترهاست ، بطوریکه برنامه داده را بطور فیزیکی بر طبق پشته ارسال می‌دارد. در این حالت ، از اینکه هر PUSH یک کلمه (یا دو کلمه در سیستم‌های پیشرفته) در حافظه یا ثبات ارجاع می‌کند ، اطمینان حاصل کنید.

قاب پشته

بخشی از پشته است که برنامه فراخوان برای ارسال پارامترها و برنامه فراخوانده شده برای دستیابی به پارامترها استفاده می‌کند. زیر برنامه فراخوانده شده از قاب پشته ذخیره موقت داده‌های محلی استفاده می‌کند.

ثبات BP مانند یک اشاره‌گر قاب عمل می‌کند. برای ارسال پارامترها از ثبات‌های BP و SP استفاده می‌کنیم.

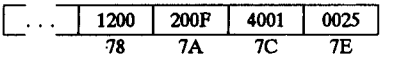
در شکل ۸-۲۳ ، برنامه فراخوان A23MAIN6 هر دو PRICE و QTY قبلی را روی پشته می‌گذارد تا زیر برنامه A23SUB6 را فراخوانی کنند. در ابتدا SP حاوی اندازه پشته، 80H است. هر کلمه که بر روی پشته گذاشته می‌شود SP را ۲ واحد می‌کاهد. بعد از CALL قاب پشته به صورت زیر ظاهر می‌شود :

```
                        TITLE   A23MAIN5 (EXE)  Call subprogram
                                EXTRN  A23SUB5:FAR
                                PUBLIC QTY
                        ; ----------------------------------------
0000                    STACKSG SEGMENT PARA STACK 'Stack'
0000 0040[????]                 DW      64 DUP(?)
0080                    STACKSG ENDS
                        ; ----------------------------------------
0000                    DATASG  SEGMENT PARA 'Data'
0000 0140               QTY     DW      0140H
0002                    DATASG  ENDS
                        ; ----------------------------------------
0000                    CODESG  SEGMENT PARA 'Code'
0000                    BEGIN   PROC    FAR
                                ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                  MOV     AX,DATASG
0003 8E D8                      MOV     DS,AX
0005 9A 0000 ---- E             CALL    A23SUB5         ;Call subprogram
000A B8 4C00                    MOV     AX,4C00H        ;End processing
000D CD 21                      INT     21H
000F                    BEGIN   ENDP
000F                    CODESG  ENDS
                                END     BEGIN

Segments and Groups:
    N a m e         Length     Align   Combine  Class
CODESG . . . . . .  000F       PARA     NONE    'CODE'
DATASG . . . . . .  0002       PARA     NONE    'DATA'
STACKSG  . . . . .  0080       PARA     STACK   'STACK'
Symbols:
    N a m e         Type       Value     Attr
A23SUB5  . . . . .  L FAR      0000     External
BEGIN  . . . . . .  F PROC     0000     CODESG  Length = 000F
QTY  . . . . . . .  L WORD     0000     DATASG  Global
-------------------------------------------------------------

                        TITLE   A23SUB5 Called subprogram
                                EXTRN   QTY:WORD
                        ; ----------------------------------------
0000                    DATASG  SEGMENT PARA 'Data'
0000 2500               PRICE   DW      2500H
0002                    DATASG  ENDS
                        ; ----------------------------------------
0000                    CODESG  SEGMENT PARA 'CODE'
0000                    A23SUB5 PROC    FAR
                                ASSUME CS:CODESG
                                PUBLIC A23SUB5
0000 8B 1E 0000 E               MOV     BX,QTY          ;Get QTY from CALLMUL
0004 1E                         PUSH    DS              ;Save CALLMUL's DS
                                ASSUME DS:DATASG
0005 B8 ---- R                  MOV     AX,DATASG       ;Set up own DS
0008 8E D8                      MOV     DS,AX           ;Price from own
000A A1 0000 R                  MOV     AX,PRICE        ;  data segment
000D F7 E3                      MUL     BX              ;DX:AX = product
000F 1F                         POP     DS              ;Restore CALLMUL's DS
0010 CB                         RETF                    ;Return to caller
0011                    A23SUB5 ENDP
0011                    CODESG  ENDS
                                END     A23SUB5

Segments and Groups:
    N a m e         Length  Align   Combine  Class
CODESG . . . . . .  0011     PARA   NONE     'CODE'
DATASG . . . . . .  0002     PARA   NONE     'DATA'
Symbols:
    N a m e         Type    Value    Attr
A23SUB5  . . . . .  F PROC   0000   CODESG  Global  Length = 0011
PRICE  . . . . . .  L WORD   0000   DATASG
QTY  . . . . . . .  V WORD   0000   External
```

شکل ۷-۲۳ تعریف داده در هر دو برنامه

```
Link Map
Object Modules: A23MAIN5+A23SUB5

 Start  Stop   Length Name               Class
 00000H 0007FH 00080H STACKSG           STACK
 00080H 00081H 00002H DATASG            DATA
 00090H 00091H 00002H DATASG            DATA
 000A0H 000AEH 0000FH CODESG            CODE
 000B0H 000C0H 00011H CODESG            CODE

 Program entry point at 000A:0000
```

شکل ۷-۲۳ تعریف داده در هر دو برنامه



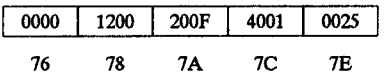
۱) یک PUSH ، در قاب پشته در افست PRICE,7EH (2500H) را قرار می‌دهد.

۲) یک PUSH در قاب پشته در افست QTY,7CH (0140H) را قرار می‌دهد.

۳) CALL محتویات CS را در این اجرا (0F20H) روی قاب پشته در 7AH قرار می‌دهد. چون زیر برنامه PUBLIC است ، دیگر دو سگمنت کد را ترکیب می‌کند و آدرس CS برای هر دو یکی است.

۴) همچنین CALL محتویات ثبات IP (0012H) روی قاب پشته در 78H قرار می‌دهد.

برنامه فراخوانده شده به استفاده از BP برای دستیابی پارامترهای قاب پشته نیاز دارد. اولین عمل ، ذخیره سازی محتویات BP برای برنامه فراخوان است ، بنابراین BP روی پشته گذاشته می‌شود. در این مثال ، BP حاوی صفر است که PUSH روی پشته در افست 76H ذخیره کرده است :




```
                        TITLE     A23MAIN6 (EXE)  Passing parameters
                                  EXTRN  A23SUB6:FAR
                        ; ----------------------------------------
0000                    STACKSG   SEGMENT PARA STACK 'Stack'
0000 0040[????]                   DW      64 DUP(?)
0080                    STACKSG   ENDS
                        ; ----------------------------------------
0000                    DATASG    SEGMENT PARA 'Data'
0000 0140               QTY       DW      0140H
0002 2500               PRICE     DW      2500H
0004                    DATASG    ENDS
                        ; ----------------------------------------
0000                    CODESG    SEGMENT PARA PUBLIC 'Code'
0000                    BEGIN     PROC    FAR
                                  ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                    MOV     AX,DATASG
0003 8E D8                        MOV     DS,AX
0005 FF 36 0002 R                 PUSH    PRICE           ;Save price
0009 FF 36 0000 R                 PUSH    QTY             ;  and quantity
000D 9A 0000 ---- E               CALL    A23SUB6         ;Call subprogram
0012 B8 4C00                      MOV     AX,4C00H        ;End processing
0015 CD 21                        INT     21H
0017                    BEGIN     ENDP
0017                    CODESG    ENDS
                                  END     BEGIN
```

شکل الف ۸-۲۳ ارسال پارامترها

```
Segments and Groups:
   N a m e          Length   Align   Combine   Class
CODESG . . . . . . . 0017     PARA    PUBLIC    'CODE'
DATASG . . . . . . . 0004     PARA    NONE      'DATA'
STACKSG  . . . . . . 0080     PARA    STACK     'STACK'
Symbols:
   N a m e          Type     Value    Attr
A23SUB6  . . . . . . L FAR    0000     External
BEGIN  . . . . . . . F PROC   0000     CODESG  Length = 0017
PRICE  . . . . . . . L WORD   0002     DATASG
QTY  . . . . . . . . L WORD   0000     DATASG
------------------------------------------------------------------

                     TITLE    A23SUB6 Called subprogram
0000                 CODESG   SEGMENT PARA PUBLIC 'Code'
0000                 A23SUB6  PROC    FAR
                              ASSUME CS:CODESG
                              PUBLIC A23SUB6
0000 55                       PUSH    BP
0001 8B EC                    MOV     BP,SP
0003 8B 46 08                 MOV     AX,[BP+8]     ;Get price
0006 8B 5E 06                 MOV     BX,[BP+6]     ;Get quantity
0009 F7 E3                    MUL     BX            ;DX:AX = product
000B 5D                       POP     BP
000C CA 0004                  RETF    4             ;Return to caller
000F                 A23SUB6  ENDP
000F                 CODESG   ENDS
                              END

Segments and Groups:
   N a m e          Length    Align    Combine Class
CODESG . . . . . . . 000F     PARA     PUBLIC    'CODE'
Symbols:
   N a m e          Type     Value     Attr
A23SUB6  . . . . . . F PROC   0000     CODESG  Global  Length=000F
------------------------------------------------------------------

        Link Map
        Object Modules: A23MAIN6+A23SUB6

        Start  Stop   Length Name              Class
        00000H 0007FH 00080H STACKSG          STACK
        00080H 00083H 00004H DATASG           DATA
        00090H 000BEH 0002FH CODESG           CODE

        Program entry point at 0009:0000
```

شکل ب ۸-۲۳ ارسال پارامترها

سپس برنامه محتویات SP (0076H) را در BP قرار می‌دهد چون BP (نه SP) برای یک ثبات شاخص قابل استفاده است. چون BP هم اکنون حاوی 0076H است ، PRICE در پشته در BP + 8 (افست 7EH) و QTY در BP + b (افست 7CH) قرار دارد. می‌دانیم این موقعیت‌ها در ارتباط هستند چون ۳ کلمه (۶ بایت) را پس از QTY روی پشته قرار دادیم. روتین PRICE و QTY را از پشته بترتیب به AX و BX منتقل نموده و ضرب را انجام می‌دهد.

قبل از بازگشت به برنامه فراخوان ، زیر برنامه BP (بازگشت آدرس صفر در BP) را از پشته برمی‌دارد ، SP را ۲ واحد می‌افزاید و از 76H و 78H خواهد شد.

دستور بعدی RETF ، یک بازگشت دور به برنامه فراخوان است ، که اعمال را انجام می‌دهد :

- کلمه بالای قاب پشته (1200H) را در IP گذاشته و SP را ۲ واحد می‌افزاید، از 78H ، 7AH خواهد شد.
- کلمه بالای قاب پشته (0F20) را در CS گذاشته و SP را ۲ واحد، می‌افزاید، از 7AH ، 7CH خواهد شد. چون دو پارامتر ارسالی در افست 7CH و 7EH هستند ، دستور بازگشت بصورت RETF 4 کد می‌شود.

بعنوان ***مقدار برداشتی*** شناخته می‌شود که حاوی تعداد بایت‌های پارامتر ارسالی است. (در این حالت ۲ پارامتر یک کلمه‌ای). RETF مقدار برداشتنی را با SP جمع می‌کند ، مقدار آن را 80H تصحیح می‌کند. در حقیقت چون پارامترهای پشته دیگر نیازی نیست ، عملیات آنها را دور انداخته و به برنامه خوان باز می‌گردد. توجه کنید که اگر چه عملیات POP و RET و SP را می‌افزایند، محتویات پشته را پاک نمی‌کنند.

اگر قانون کلی مورد بحث این فصل را دنبال کنید ، قادر خواهید بود تا یک برنامه که شامل چندین ماجول اسمبلی است با هم پیوند بزنید و داده شناخته شده در همه ماجول‌ها، طراحی کنید، اما مراقب اندازه پشته باشید ، در برنامه‌های بزرگ با عملیات PUSH و CALL زیاد ، تعریف ۶۴ کلمه می‌تواند برآورد معقولی باشد. فصل ۲۴ برخی مفاهیم مهم مدیریت حافظه و اجرای برنامه‌های مجاور هم قرار گرفته را دربر دارد. فصل ۲۶ خصیصه‌های اضافی از سگمنت ، شامل تعریف چندین سگمنت کد یا داده در یک ماجول اسمبلی و استفاده از GROUP برای ترکیب آنها در یک سگمنت مشترک را فراهم می‌سازد.

پیوند زدن پاسکال با یک برنامه زبان اسمبلی

این بخش نحوه پیوند یک برنامه پاسکال با یک زیر برنامه اسمبلی را توضیح می‌دهد. برنامه ساده پاسکال در شکل ۹-۲۳ به یک زیر برنامه اسمبلی که منظور از آن تنظیم مکان نما است ، پیوند می‌زند. برنامه پاسکال برای تولید یک ماجول OBJ. کامپایل می‌شود و برنامه اسمبلی نیز برای تولید ماجول OBJ. اسمبل می‌شود. سپس لینکر این دو ماجول OBJ. را در یک ماجول اجرایی EXE. ترکیب می‌کند.

برنامه پاسکال دو عنصر، temp_row و temp_col را تعریف می‌کند و ورودیهایی برای سطر و ستون از صفحه کلید می‌پذیرد. در این دو متغیر قرار می‌دهد. برنامه نام زیر برنامه اسمبلی را set - curs تعیین کرده و دو پارامتر را بعنوان خارجی تعریف می‌کند. آدرس temp_row و temp_col را بصورت پارامتر به زیر برنامه می‌فرستد تا مکان نما در آن موقعیت تنظیم شود. جمله پاسکال که نام زیربرنامه را فراخوانی کرده و پارامترها را ارسال می‌دارد چنین است :

```
set_curs (temp_row,temp_col) ;
```

مقادیری که روی پشته گذاشته می‌شود ، عبارتند از : اشاره‌گر پشته برنامه فراخوان ، اشاره‌گر سگمنت بازگشتی، افست بازگشتی و آدرس دو پارامتر ارسالی. در زیر افست هر ورودی به پشته را نشان می‌دهد :

00 اشاره‌گر پشته فراخواننده. 06 آدرس دومین پارامتر.

02 اشاره‌گر سگمنت بازگشت فراخواننده. 08 آدرس اولین پارامتر.

04 افست بازگشت فراخواننده.

چون زیر برنامه اسمبلی از ثبات BP استفاده می‌کند ، باید BP را در پشته قرار دهید تا آدرس آن در بازگشت برای برنامهٔ فراخوان پاسکال ، ذخیره شود. توجه کنید که مراحل در زیربرنامه فراخواننده شده مشابه برنامه شکل ۷-۲۳ می‌باشد.

ثبات SP معمولاً ورودیهای پشته را آدرسدهی می‌کند. اما چون نمی‌توانید از SP بعنوان یک ثبات شاخص استفاده کنید ، مرحلهٔ بعد از قرار دادن BP روی پشته، انتقال آدرس SP به BP است. این مرحله شما را قادر می‌سازد تا از BP بعنوان یک ثبات شاخص برای دستیابی ورودی در قاب پشته ، استفاده کنید. مرحله بعد دستیابی به آدرسهای دو پارامتر در قاب پشته می‌باشد. اولین پارامتر ارسالی ، سطر ، در افست 80H در قاب پشته است و توسط BP + 06H قابل دستیابی است. دومین پارامتر ارسالی، ستون، در افست 06H است و توسط BP + 06H قابل دستیابی است.

هر یک از دو آدرس قاب پشته باید به یکی از ثبات‌های شاخص : BX ، DI یا SI منتقل شوند. این مثال از [BP + 08] برای انتقال آدرس سطر به SI استفاده می‌کند و از [SI] برای انتقال محتویات پارامتر ارسالی به ثبات DH استفاده می‌کند.

ستون با روشی مشابه به ثبات DL منتقل می‌شود. سپس زیر برنامه از سطر و ستون در ثبات DX برای INT 10H جهت تنظیم مکان نما استفاده می‌کند. زمان خروج ، زیر برنامه BP را از روی پشته بر می‌دارد. دستور RET بـه یک مقدار عملوند نیاز دارد که دو برابر تعداد پارامترها باشد - در این حالت ، ۲×۲ یا ۴. مقادیر بطور اتوماتیک از روی پشته برداشته می‌شوند و کنترل به برنامه فراخوان باز می‌گردد.

اگر یک ثبات سگمنت را تغییر می‌دهید ، حتماً آن را به هنگام ورود PUSH و در هنگام خروج از زیر برنامه POP کنید. تجربه توصیه شده برای فراخوانی در پاسکال نگهداری ثبات‌های DI ، SI ، BP ، DS و SS می‌باشد. همچنین می‌توانید از پشته برای ارسال مقادیر از یک زیر برنامه به یک برنامه فراخوان استفاده کنید اگر چه زیر برنامه شکل ۹-۲۳ مقداری بازنمی‌گرداند، پاسکال انتظار دارد که یک زیر برنامه ، یک کلمه تنها در AX یا دو کلمه‌ای در DX:AX باز گرداند.

```
program p23pascl ( input, output );

  procedure set_curs( const row: integer;
                      const col: integer ); extern;
  var
       temp_row:      integer;
       temp_col:      integer;

  begin
       write( 'Enter cursor row: ' );
       readln( temp_row );

       write( 'Enter cursor column: ' );
       readln( temp_col );

       set_curs( temp_row, temp_col );
       write( 'New cursor location' );
  end.
```

```
TITLE     A23SETCR  Assembler subprogram called by Pascal
          PUBLIC SET_CURS
;
;     SET_CURS: Set cursor on screen at passed location
;     Passed:   const row        Row and column where
;               const col        cursor is to be set
;     Returned: Nothing
;
CODESEG   SEGMENT PARA PUBLIC 'CODE'
SET_CURS  PROC    FAR
          ASSUME CS:CODESEG
          PUSH    BP          ;Caller's BP register
          MOV     BP,SP       ;Point to parameters passed

          MOV     SI,[BP+8]   ;SI points to row
          MOV     DH,[SI]     ;Move row to DH

          MOV     SI,[BP+6]   ;SI points to column
          MOV     DL,[SI]     ;Move column to DL

          MOV     AH,02H      ;Request set cursor
          MOV     BH,0        ;Video page
          INT     10H

          POP     BP          ;Return to caller
          RETF    4
SET_CURS  ENDP
CODESEG   ENDS
          END
```

شکل ۹-۲۳ پیوند زدن پاسکال با اسمبلر

این برنامه کوچک یک ماجول بزرگتر از 20K بایت را تولید می‌کند. یک زبان کامپایلری ، عنوان فایلی بزرگی بدون توجه به اندازه برنامه منبع تولید می‌کند. دیگر نسخه‌های پاسکال نیازی به پیگیری مرسومات استفاده شده در اینجا را ندارند. استاندارد مناسب در راهنمای کامپایلر توضیح داده شده ، همیشه در یک بخش با تیتر '. . . Inter facing' یا 'Mixed languayes' می‌باشد.

پیوند زدن C با یک برنامه زبان اسمبلی

مشکلی که در پیوند C با برنامه اسمبلی وجود دارد این است که نسخه‌های C تفاوت عمده‌ای با هم دارند. (به ضروریات درست در راهنمای C شما اشاره شده است). برخی نکات قابل توجه بشرح زیرند :

- برای نسخه‌هایی از C که به حروف بزرگ و کوچک حساسند ، نام ماجول اسمبلی باید به همان حالت باشد که برنامه C مراجعه می‌کند.
- برخی نسخه‌های C پارامترها را به ترتیبی به پشته ارسال می‌دارند که معکوس دیگر زبانها است. برای مثال جمله C زیر را در نظر بگیرید. `Add (m,n) ;`

جمله n و سپس m را روی پشته گذارده و Add را فراخوانی می‌کند. در بازگشت از ماجول فراخوانده شده ماجول C (نه ماجول اسمبلی) ۴ را با SP جمع می‌کند تا پارامترهای ارسالی کنار گذاشته شوند. روال در ماجول اسمبلی فراخوانده شده برای دستیابی به دو پارامترهای ارسالی چنین هستند :

```
POSH    BP
MOV     BP,SP
MOV     DH,[BP+4]
MOV     DL,[BP+6]
. . .
POP     BP
RET
```

- در برخی نسخه‌های C لازم است که یک ماجول اسمبلی که ثبات‌های DI و SI را تغییر می‌دهد در موقع ورود روی پشته گذاشته و هنگام خروج از روی پشته بردارد.
- ماجول اسمبلی باید مقادیر را در صورت لزوم اگر لازم باشد ، بصورت یک کلمه در AX یا دو کلمه در جفت DX:AX باز گرداند.
- در برخی نسخه‌های C ، برنامه اسمبلی که ثبات DF را تنظیم می‌کند قبل از بازگشت آن را پاک کند (CLD).

پیوند C مایکروسافت با اسمبلر مایکروسافت

نامگذاری معمول. در C مایکروسافت و اسمبلر ، ماجولهای اسمبلی باید از نامگذاری معمول و برای سگمنت‌ها و متغیرها استفاده کنند بطوریک با C قابل تطبیق باشد. همه ارجاعات اسمبلر به توابع و متغیرهای ماجول C باید با یک خط فاصله آغاز شود (_). علاوه بر آن ، چون C ، نسبت به ***حالت حروف حساسیت دارد*** ، ماجول اسمبلی ، باید همان حالتی (بزرگ یا کوچک) باشد که برای هر نام متغیر مشترک با ماجول C می‌باشد.

ثبات‌ها. ماجول اسمبلی باید مقادیر اصلی در ثبات‌های BP ، SP ، CS ، DS ، SS ، DI و SI را نگهداری کند.

ارسال پارامترها. دو روش برای ارسال پارامترها وجود دارد :

۱) **توسط ارجاع.** بصورت نزدیک (یک افست در سگمنت پیش فرض) یا بصورت دور (یک افست در سگمنت دیگر). ماجول اسمبلی فراخوانده شده ، می‌تواند مستقیماً مقدار تعریف شده در ماجول C را تغییر دهد.

۲) **توسط مقدار.** بطوریکه فراخواننده C یک کپی از متغییر را روی پشته ارسال می‌دارد ، ماجول اسمبلی فراخوانده شده می‌تواند مقدار ارسال شده را تغییر دهد ، اما هیچ دستیابی به مقدار C اصلی ندارد. اگر بیشتر از یک پارامتر باشد، C آنها را بترتیب از پارامتر سمت راست روی پشته قرار می‌دهد.

قابلیت تطبیق انواع داده. لیست زیر انواع متغییرهای C و انواع اسمبلر معادل آن را نشان می‌دهد :

نوع داده C	نوع MASM 5.X	نوع MASM 6.X
char	DB	BYTE
unsigned short/int	DW	WORD
int,short	DW	SWORD
Unsigned long	DD	DWORD
long	DD	SDWORD

مقادیر بازگشتی. ماجول اسمبلی فراخوانده شده از ثبات‌های زیر برای هر مقدار بازگشتی استفاده می‌کند :

نوع داده C	ثبات
Char	AS
Short,near, int (۱۶ بیت)	AX
Short, near, int (۳۲ بیت)	EAX
long, far (۱۶ بیت)	DX:AX
long, far (۳۲ بیت)	EDX:EAX

در بازگشت از ماجول فراخوانده شده RET را بدون مقدار برداشتی صادر کنید.

کامپایل کردن و اسمبل کردن. از یک مدل حافظه در هر دو زبان استفاده کنید. جمله MODEL. اسمبلی مبنی بر C معمول‌است ، مانند MODEL SMALLC. همچنین از سوئیچ اسمبلی مناسب برای نگه‌داری حالت نام‌ها استفاده کنید.

پیوند توربو C و توربو اسمبلر.

رابط های زبان. توربو C دو روش برای رابطه با توربو اسمبلر دارد – با ماجولهای مجزا و کدهای درون خطی.

۱) **ماجولهای مجزا.** برای این روش معمول ، برنامه‌های C و اسمبلی را بطور جداگانه کدنویسی می‌کنید. از TCC برای کامپایل کردن ماجول C و از TASM برای اسمبل کردن ماجول اسمبلی استفاده کنید. از TLINK برای پیوند زدن آنها استفاده کنید.

۲) **کد اسمبلی درون خطی.** برای کامپایل کردن ماجول C ، باید از TCC.EXE استفاده کنید (نسخه فرمان از توربو C). جملات اسمبلی را جایگزین کنید. در ابتدا آن کلمه کلیدی asm را در کد منبع قرار دهید ، مانند :

```
asm INC WORD PTR FLDX
```

سگمنت‌ها. سگمنت کد باید با TEXT- نامگذاری شود. سگمنت داده (اگر دو تا لازم است) DATA- برای داده‌هایی که در ورود مقدار دهی می‌شوند و BSS- برای داده‌های مقدار دهی نشده نامگذاری شود.

نام گذاری معمول. ماجولهای توربو اسمبلر باید از نامگذاری مرسوم برای سگمنت‌ها و متغیرهایی که با توربو C قابل تطبیق باشد ، استفاده کنند. همه اسمبلرها به توابع و متغیرهای ماجول C که با یک خط فاصله آغاز می‌شود مراجعه می‌کنند علاوه بر آن چون C نسبت به حالت حساس است ماجول اسمبلی باید از همان حالت برای هر نام متغیر مشترک با ماجول C استفاده کند (بزرگ یا کوچک).

ثبات‌ها. ماجول اسمبلی ممکن است بطور آزادانه از ثبات‌های AX ، BX ، CX ، DX ، ES و پرچم استفاده کند. همچنین ممکن است از ثبات‌های BP ، SP ، CS ، DS ، SS ، DI و SI استفاده کند که آنها ذخیره و بازیابی می‌شوند. (در پشته)

ارسال پارامترها. توربو C پارامترها را با مقدار ارسال می‌کند اگر بیشتر از یک پارامتر وجود داشته باشد توربو آنها را از راست به چپ روی پشته می‌گذارد.

بازگشت. برنامه اسمبل از RET برای بازگشت به ماجول C استفاده می‌کند (مقداری برداشته نمی‌شود). ماجول C هنگام ورود به پشته مقدار آنرا برمی دارد.

مثال یک برنامه C.

برنامه شکل ۱۰-۲۳ پیوند یک برنامه توربو C با یک ماجول اسمبلی را مشخص می‌سازد. برنامه همان عمل برنامه پاسکال در بخش قبل را انجام می‌دهد. برنامه C مقادیر سطر و ستون را از صفحه کلید می‌پذیرد و آنها را به زیر برنامه اسمبل ارسال می‌دارد. زیر برنامه اسمبلی به نوبه خود مکان نما را تنظیم نموده و به ماجول C باز می‌گردد.

```
#include <stdio.h>

int main (void)
{
   int temp_row, temp_col;

   printf ("Enter cursor row: ");
   scanf ("%d", &temp_row);

   printf ("Enter cursor column: ");
   scanf ("%d", &temp_col);

   set_curs (temp_row, temp_col);
   printf ("New cursor location\n");
}
```

```
;
; Use small memory model for C: near code, near data
; Use 'standard' segment names, and group directive

_DATA     segment word 'DATA'
row       equ     [bp+4]          ;Parameters
col       equ     [bp+6]          ;  (arguments)
_DATA     ends


_TEXT     SEGMENT BYTE PUBLIC 'CODE'
DGROUP    GROUP   _DATA
          ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP

          PUBLIC  _set_curs
_set_curs PROC    NEAR
          PUSH    BP              ;Caller's BP register
          MOV     BP,SP           ;Point to parameters

          MOV     AH,02H          ;Request set cursor
          MOV     BX,0            ;Video page 0
          MOV     DH,ROW          ;Row from BP+4
          MOV     DL,COL          ;Column from BP+6
          INT     10H             ;Call interrupt

          POP     BP              ;Restore BP
          RETF                    ;Return to caller
_set_curs ENDP
_TEXT     ENDS
          END
```

شکل ۱۰-۲۳ پیوند زدن C با اسمبلر

نکات کلیدی

- عملگر قرارگیری به اسمبلر می‌گوید تا سگمنت نامبرده با شروع از یک مرز حافظه خاص قرار گیرد.
- عملگر ترکیب به اسمبلر و لینکر می‌گوید که آیا سگمنت‌ها را ترکیب کند یا آنها را جدا نگه دارد.
- می‌توانید نام کلاس یکسانی را برای سگمنت‌های مرتبط معین کنید ، بنابراین ، اسمبلر و لینکر آنها را با هم در یک

گروه قرار می‌دهد.

- یک CALL درون سگمنت از نوع نزدیک است اگر روال فراخوانده شده بصورت NEAR تعریف شده باشد (داخل 32K) یک فراخوانی درون سگمنت ممکن‌است ازنوع دورباشد اگر فراخوانی‌به یک روال دور درهمان سگمنت‌باشد.
- یک CALL بین سگمنت یک‌روال درسگمنت دیگر را فرامی‌خواند که بصورت FAR یا EXTRN تعریف‌شده باشد.
- در یک برنامه اصلی که یک زیربرنامه را فرا می‌خواند نقطه ورود بصورت EXTRN تعریف می‌شود. در زیر برنامه نقطه ورود بصورت PUBLIC تعریف می‌شود.
- برای پیوند دو سگمنت کد در یک سگمنت آنها را با یک نام ، یک کلاس و نوع ترکیب PUBLIC تعریف کنید.
- غالباً راحت‌تر است (اما نه الزاماً) داده‌های مشترک در برنامه اصلی تعریف شود. برنامه اصلی داده‌های مشترک را بصورت PUBLIC تعریف می‌کند و زیر برنامه (زیر برنامه‌ها) داده‌های مشترک را بصورت EXTRN تعریف می‌کند.

پرسش‌ها.

۱-۲۳. برای سازماندهی برنامه‌ها در زیر برنامه‌ها چهار دلیل بیاورید :

سه پرسش بعدی به قالب کلی برای پیش پردازنده SEGMENT اشاره دارند :

```
Seg_name SEGMENT [align] [combine] ['Class']
```

۲-۲۳. (الف) پیش فرض انتخاب قرارگیری برای پیش پردازنده SEGMENT چیست؟ (ب) تاثیر انتخاب بایت چیست؟ (با این اسمبلر چه کاری انجام می‌دهد).

۳-۲۳. (الف) پیش فرض انتخاب ترکیب پیش پردازنده سگمنت چیست؟ (ب) چرا شما از انتخاب PUBLIC استفاده می‌کنید؟ (ج) چرا از انتخاب COMMON استفاده می‌کنید؟

۴-۲۳. (الف) معمولاً انتخاب کلاس سگمنت کد در پیش پردازنده SEGMENT چیست؟ (ب) اگر دو سگمنت، کلاس یکسانی داشته باشند اما انتخاب ترکیب آنها PUBLIC نباشد چه تاثیری دارد؟ (ج) اگر دو سگمنت ، کلاس یکسانی داشته باشند و انتخاب ترکیب PUBLIC هم داشته باشند چه اثری دارد؟

۵-۲۳. تفاوت بین فراخوانی درون سگمنت و فراخوانی بین سگمنت را توضیح دهید.

۶-۲۳. یک برنامه با نام MPROG23 یک زیر برنامه با نام SCALC23 را فرا می‌خواند (الف) چه جمله‌ای در EMPROG23 اسمبلر را آگاه می‌سازد که نام CALC23 خارج از این مرحله اسمبل تعریف شده است؟ (ب) چه جمله‌ای در SCALS23 لازم است تا این نام EMPROG23 شناخته شده باشد ؟

۷-۲۳. فرض کنیم EMPROG23 در پرسش ۶-۲۳ سه عنصر داده بصورت DW تعریف می‌کند : QUANTITY (مقدار موجودی در دسترس) UNITCOST (ارزش واحد) و STVALUE (ارزش موجودی). SCALC23 و STVALUE را بر QUANTITY تقسیم می‌کند و خارج قسمت را در UNITCOST ذخیره می‌سازد. (الف) چگونه MPROG23 اسمبلر را آگاه می‌سازد که سه عنصر داده در خارج این مرحله اسمبل شناخته شده هستند؟ (ب) SCALC23 چگونه اسمبلر را آگاه‌می‌سازد که سه عنصرداده در ماجول دیگری تعریف‌می‌شوند؟

۸-۲۳. پرسش ۶-۲۳ و ۷-۲۳ را در یک برنامه کاری ترکیب کنید و آن را بررسی نمایید.

۹-۲۳. پرسش ۸-۲۳ را طوری بازنویسی کنید که MPROG23 همه سه عنصر داده را بصورت پارامتر ارسال کند. اما توجه کنید که SCALC23 قیمت محاسبه شده را دست نخورده در پارامتر خود باز می‌گرداند.

۱۰-۲۳. پرسش ۹-۲۳ را توسعه دهید بطوریکه MPROG23 مقدار موجودی و ارزش را از صفحه کلید بپذیرد ، زیر برنامه SBINRY23 میزان ASCII را به دودویی تبدیل کند ، زیر برنامه SCALC23 قیمت را محاسبه کند ، و زیر برنامه SASCII23 قیمت دودویی را به ASCII تبدیل کند و حاصل را نمایش دهد.

فصل ۳۷

مدیریت حافظه

هدف : توضیح نحوه مدیریت سیستم و بارگذاری برنامه‌ها برای اجرا.

برنامه

این فصل مراحل راه‌اندازی سیستم، مقدار دهی سیستم، پیشوند سگمنت برنامه، محیط، کنترل حافظه، بارگذار برنامه و برنامه‌های مقیم را توضیح می‌دهد. از جمله توابعی که در این فصل مورد استفاده قرار می‌گیرند تابع وقفه تسهیم[1] یا 4A 01H از INT 2FH و مجموعه توابع زیر از INT 21H می‌باشند.

25H تنظیم بردار وقفه	49H آزاد سازی حافظه تخصیص یافته
31H نگه داشتن برنامه	4AH تغییر بلوک حافظه تخصیص یافته
3306H گرفتن نسخه DOS	4BH بارگذاری یا اجرای یک برنامه
34H گرفتن آدرس پرچم مشغولیت DOS	51H گرفتن آدرس PSP جاری
35H گرفتن بردار وقفه	52H گرفتن آدرس لیست DOS داخلی
48H تخصیص حافظه	58H روش گرفتن/ تنظیم تخصیص حافظه

برنامه‌های اصلی DOS

چهار برنامه اصلی DOS ، رکورد راه انداز، IO.SYS ، MSDOS.SYS و COMMAND.COM است:

۱. **رکورد راه انداز** در تراک ۰ ، سکتور ۱ از هر دیسکی که با /S FORMAT فرمت نموده‌اید، قرار دارد وقتی شما کامپیوتر را روشن می‌کنید، سیستم به طور اتوماتیک رکورد راه‌انداز را اجرا می‌کند، که بنوبه خود IO.SYS را در دیسک در حافظه قرار می‌دهد.

۲. **IO.SYS** یک رابط سطح پایین با روالهای BIOS در ROM است. در زمان شروع، وضعیت ابزارهای کامپیوتر و تجهیزات را تعیین می‌کند، آدرس‌های جدول بردار وقفه را تا 20H تنظیم می‌کند و ورودی/خروجی این حافظه و ابزارهای خارجی را دستکاری می‌کند.

۳. **MSDOS.SYS** یک رابط سطح بالا برای برنامه‌هایی است که آدرس‌های جدول بردار وقفه، را برای وقفه‌های 20H تا 3FH تنظیم می‌کنند و سرویس‌های آن شامل مدیریت شاخه و فایل‌های روی دیسک، بلوکه کردن از بلاک در آوردن رکوردهای دیسک و توابع INT 21H است. همچنین COMMAND.COM را بارگذاری می‌کند.

۴. **COMMAND.COM** شامل سه بخش است که در حافظه هم بطور دائمی یا موقت بارگذاری می‌شوند.

1- Moltiplex Interupt

الف) توابع **بخش مقیم** با وقفه‌های زیرکارمی‌کنند. INT 22H (آدرس خاتمه)، INT 23H (دستیاب Ctrl+Break) INT 24H (تشخیص خطا در خواندن/ نوشتن دیسک) و INT 27H (خاتمه اما مقیم ماندن (TSR)).

ب) وقتی سیستم روشن می‌شود، ***بخش اولیه*** فایل AUTOEXE را پردازش نموده و آدرس سگمنت جایی که برنامه‌ها باید برای اجرا بارگذاری شوند را مشخص می‌سازند. چون هیچ یک از روتینهای اولیه در طی مراحل بعدی لازم نیستند. اولین برنامه برگذاری شده از روی دیسک روی این بخش قرار می‌گیرند.

پ) ***بخش ناپایدار*** در ناحیه بالایی حافظه قرار می‌گیرد، که ممکن است دیگر برنامه‌های درخواستی نیز بر روی آن قرار گیرند. این بخش اعلان صفحه را نشان می‌دهد و اجرایی مانند درخواست DIR و COPY را نیز می‌پذیرد. همچنین شامل وسیله بارگذاراست که برنامه‌های COM. و EXE. از روی دیسک جهت‌اجرا درحافظه قرارمی‌دهد. خاتمه معمولی برنامه سبب بازگشت به بخش مقیم COMMAND.COM می‌شود. اگر برنامهٔ اجرا شده بر روی بخش ناپایدار قرار گیرد، بخش مقیم آن را مجدداً در حافظه قرار می‌دهد.

شکل ۱-۲۴ نقشه‌ای از حافظه پس از قرارگیری برنامه‌های سیستم را نشان می‌دهد. جزئیات با سیستم فرق می‌کند.

Beginning Address	Contents
F0000H	System ROM area
E0000H	ROM BIOS
D0000H	ROM BIOS
C0000H	ROM BIOS
B0000H	Video buffers
A0000H	Video buffers
xxxx0H	Transient portion of COMMAND.COM, at top of RAM
	...
	User programs
	Resident programs (if any)
xxxx0H	Resident portion of COMMAND.COM
xxxx0H	MSDOS.SYS and IO.SYS
00500H	DOS communication area
00400H	BIOS data area
00000H	Interrupt vector table

توجه : حافظهٔ متعارف در آدرس بین A0000H - 00000H قرار دارد.(640 k)
حافظهٔ فوقانی در آدرس بین FFFF0H - A0000H قرار دارد. (1 Meg)
حافظهٔ (HMA) در آدرس بین FFFFFH - FFFF0H قرار دارد. (64 k)
حافظهٔ توسعه یافته بعد از حافظهٔ HMA قرار می‌گیرد.

شکل ۱-۲۴ نقشه حافظه اصلی

ناحیه فوقانی حافظه

پردازشگر از تعدادی خطوط آدرس برای دستیابی به حافظه کمک می‌گیرد. برای 80286 و پردازشگرهای بعد از آن خط آدرس A20 می‌تواند 64K فضای شناخته شده بعنوان ***ناحیه فوقانی حافظه*** (HMA) را آدرسدهی کند، از FFFF:10H تا FFFF:FFFFH، دقیقاً بالای آدرس یک مگابایت می‌باشد. وقتی پردازشگر در حالت واقعی (8086)کار می‌کند، بطور معمول خط A20 را ناتوان می‌سازد بنابراین آدرسهایی که فراتر از این محدوده است، در یک چرخش به ابتدای حافظه باز می‌گردند. فعال کردن خط A20 امکان آدرسدهی موقعیت‌های HMA را فراهم می‌سازد. در DOS 5.0 می‌توانید در CONFIG.SYS فایل‌های سیستم را از حافظه پایینی در HMA قرار دهید، که بدینوسیله فضایی برای برنامه‌های کاربر آزاد می‌شود. می‌توانید با استفاده از INT 21H تابع 3306H (گرفتن نسخه DOS) فایل‌های سیستم حاضر در HMA را تعیین کنید.

```
MOV  AX,3306H   ; در خواست نسخه DOS
INT  21H        ; فراخوانی سرویس وقفه
```

عملیات مقادیر زیر را در ثبات‌ها باز می‌گرداند:

- BL = شماره نسخه اصلی (مانند n در نسخه n.2)
- BH = شماره نسخه فرعی (مانند 2 در نسخه n.2)
- DL = شماره بازنویسی در ۳ بیت پایین (۰ - ۲)
- DH = پرچم نسخهٔ DOS، که اگر بیت ۴ یک باشد یعنی در HMA قرار دارد.

INT 2FH (وقفه‌مخلوط)، درمیان سرویس‌های فراوانش،یک بررسی‌فضای دردسترس در HMA دارد (تابع 4A01)

```
MOV   AX,4A01H   ; در خواست فضای HMA
INT   2FH        ; فراخوانی وقفه مخلوط
```

عملیات مقادیر زیر را در ثبات‌ها باز می‌گرداند :

- BX = تعداد بایت‌های آزاد در HMA اگر COMMAND.COM در حافظه فوقانی قرار نگرفته باشد صفر است).
- ES:DI = آدرس اولین بایت آزاد در HMA (FFFF:FFFF اگر DOS در حافظه فوقانی قرار نگرفته باشد.

پیشوند سگمنت برنامه

بارگذار برنامه، برنامه‌های COM. و EXE. را بر اجرای در یک سگمنت برنامه قرار می‌دهد و یک PSP در افست 00H ایجاد می‌کند و برنامه خودش در افست 100H از سگمنت قرار می‌گیرد. PSP حاوی فیلدهای زیر است، بر طبق موقعیت رابطه‌ای :

00-01H یک دستور INT 20H (CD20H) برای سهولت بازگشت به سیستم

03-02H آدرس سگمنت در آخرین پاراگراف حافظه تخصیص داده شده برنامه، بصورت XXXX0. برای مثال، 640K مبنی بر 00A0H به معنی [0]A0000 است.

09-04H توسط سیستم رزرو شده است.

0D-0AH آدرس خاتمه (آدرس سگمنت برای INT 22H)

11-0EH آدرس خروج Ctrl + Break (آدرس سگمنت برای INT 23H)

15-12H آدرس خروج خطای بحرانی (آدرس سگمنت برای INT 24H)

17-16H توسط سیستم رزرو شده است

2B-18H جدول دستگیره فایل پیش فرض

2D-2CH آدرس سگمنت از محیط برنامه

31-2EH توسط سیستم رزرو شده است

33-32H طول جدول دستگیره فایل

37-34H اشاره‌گر دور به جدول دستگیره

4F-38H توسط سیستم رزرو شده است

51-50H فراخوانی تابع INT 21H (INT 21H,RETF)

5B-52H توسط سیستم رزرو شده است

6B-5CH ناحیه پارامتر ۱، بصورت غیر باز استاندارد فرمت شده است (#1) FCB

7F-6CH ناحیه پارامتر ۲، بصورت غیر باز استاندارد فرمت شده است (#2) FCB اگر FCB در 5CH باز شود، روی این قرار می‌گیرد

FF-80H بافر برای DTA پیش فرض

18-28H PSP جدول دستگیره فایل پیش فرض

PSP 18-2BH :

هر بایت در جدول دستگیره فایل پیش فرض به یک ورودی در جدول سیستم اشاره دارد که راه انداز یا ابزار مرتبط را تعریف می‌کند. در ابتدا جدول حاوی 0101010002FF...FF است، که اولین 01 ، به صفحه کلید دومین 01 به صفحه نمایش و الی آخر اشاره دارد.

جدولی با ۲۰ دستگیره مشخص می‌سازد به همین دلیل سیستم حداکثر ۲۰ فایل باز در یک زمان را مجاز می‌شمارد. معمولاً، کلمه‌ای در PSP افست 32H حاوی طول جدول است (20 یا 14H) و 34H حاوی آدرس سگمنت به صورت IP:CS است، که IP , 18H است (افست در PSP) و CS آدرس سگمنت PSP است.

TABLE	DEVICE	HANDLE	DEVICE
01	Console	0	Keyboard (standard input)
01	Console	1	Screen (standard output)
01	Console	2	Screen (standard error)
00	COM1 (serial port)	3	Auxiliary
02	Printer	4	Standard printer
FF	Unassigned	5	Unassigned

برنامه‌هایی که بیشتر از ۲۰ فایل باز نیاز دارند، باید حافظه آزاد سازند (INT 21H تابع 4AH) و از تابع 67H استفاده کنند (تنظیم حداکثر تعداد دستگیره) :

```
در خواست دستگیره بیشتر ;   MOV   AH,67H
شماره جدید (۲۰ تا ۶۵۵۳۵) ;   MOV   BX,count
فراخوانی سرویس وقفه ;   INT   21H
```

میزان حافظه مورد نیاز برای هر دستگیره یک بایت است، که در بایت بعدی پاراگراف بعلاوه ۱۶ بایت جمع‌آوری می‌شود. عملیات یک جدول دستگیره جدید خارج از PSP ایجاد می‌کند و موقعیت‌های 32H و 34H از PSP بروزرسانی می‌شوند. یک عملیات نامعتبر پرچم نقلی را یک کرده و یک کد خطا در AX تنظیم می‌کند.

PSP 2C-2DH : آدرس سگمنت محیط

هر برنامه که جهت اجرا بارگذاری می‌شود یک محیط مرتبط دارد که سیستم در حافظه ذخیره می‌سازد، که از مرز یک پاراگراف قبل از سگمنت برنامه آغاز می‌شود. اندازه پیش فرض، ۱۶۰ بایت است و تا حداکثر 32K را می‌توان به کار برد. محیط شاملِ برخی فرمان‌های سیستم مانند PATH COMSPEC ، PROMPT و SET می‌باشد که برای برنامه‌های کاربردی است.

PSP 5C-6BH : FCB#1 غیر بار استاندارد

ممکن است درخواست اجرای برنامه، شامل یک نام فایل باشد مانند MASM E:PROG1.ASM. بارگذار برنامه این ناحیه را بصورت FCB#1 **فرمت** می‌کند، که 05H (برای درایو E) و سپس نام فایل (۸کاراکتر) و توسعه (۳کاراکتر) قرار می‌گیرد، حذف درایو و نام فایل سبب می‌شود بارگذار اولین بایت را با 00H (پیش فرض) و انتهای FCB را با جای خالی (20H) تنظیم کند.

PSP 6C-7FH : FCB#2 غیر باز استاندارد

ممکن است اجرای برنامه‌ای با دو نام فایل درخواست کنید مانند COPY C:FILEA.DOC ، D:FILEB.DOC

برنامه بارگذار این ناحیه را بصورت FCB#2 برای **دومین** نام فایل وارد شده فرمت می‌کند، که 04H (برای درایو D) و پس از آن (۸ کاراکتر) نام فایل و (۳ کاراکتر) توسعه آن قرار دارد.

PSP 80-FFH : بافر DTA پیش فرض

بارگذار برنامه بافر **پیش فرض** برای DTA را با متن کاملی که بعد از نام برنامه درخواستی وارد کرده‌اید، مقدار دهی می‌کند مانند MASM یا COPY اولین بایت حاوی تعداد کلیدهای فشرده شده (اگر باشد) پس از نام برنامه است. پس از تعداد، کاراکترهای وارد شده (اگر باشد) و سپس هر مقدار بی‌مصرفی در حافظه از برنامه قبلی قرار دارد.
چهار مثال زیر محتویات و هدف از FCB#1 و FCB#2 ، DTA را مشخص می‌سازد.

مثال ۱ : فرمانهای بدون عملوند

فرض کنید می‌خواهید یک برنامه با نام CALCIT.EXE را اجراکنید، برای این کار فرمان CALCIT <Enter>. را وارد می‌کنید. وقتی بارگذار برنامه PSP را بسازد FCB#1 و FCB#2 ، DTA پیش فرض را چنین تنظیم می‌کند :

```
5CH   FCB#1 :  00  20  20  20  20  20  20  20  20  20  20  20 . . .
6CH   FCB#2 :  00  20  20  20  20  20  20  20  20  20  20  20 . . .
80    DTA :    00  0D . . .
```

FCB#1 و FCB#2 : دو FCB مجازی وجود دارد. اولین بایت هر دو، 00H، به شماره درایو پیش فرض اشاره دارد. بایت‌های بعدی برای نام فایل و توسعه آن خالی هستند، چون متن تایپ شده‌ای پس از نام برنامه وجود ندارد.
DTA : اولین بایت حاوی تعداد بایت‌های وارد شده، پس از نام CALCIT بدون <Enter> می‌باشد. چون هیچ کلیدی بجز <Enter> فشرده نشده، تعداد صفر است. دومین بایت بخاطر <Entet> حاوی 0DH است.

مثال ۲ : فرمان با عملوند متن

فرض کنید می‌خواهید یک برنامه با نام COLOR را اجراکنید و یک پارامتر 'BY' ارسال نمایید که به برنامه می‌گوید رنگ را با آبی (B) روی زرد (Y) زمینه تنظیم نماید. شما نام برنامه و سپس پارامتر را وارد می‌کنید :
COLOR BY. بارگذار برنامه PSP بصورت زیر شکل می‌دهد:

```
5CH   FCB#1 :  00  42  59  20  20  20  20  20  20  20  20  20 . . .
6CH   FCB#2 :  00  20  20  20  20  20  20  20  20  20  20  20
80H   DTA :    03  20  42  59  0D . . .
```

FCB#1 : این فیلد حاوی 00H برای درایو پیش فرض و 4259H (BY) بعنوان نام فایل می‌باشد. توجه کنید که بارگذار برنامه نمی‌داند که نام فایل معتبر است.
DTA : بایت‌ها طول ۳ و سپس یک فاصله 'BY' , 0DH برای <Enter> می‌باشد. بجز طول، این فیلد دقیقاً حاوی آنچه که پس از نام برنامه COLOR وارد شده می‌باشد.

مثال ۳ : فرمانی با یک عملوند نام فایل

بسیاری از برنامه‌ها اجازه ورود یک نام فایل پس از نام برنامه را، می‌دهند، برای مثال اگر وارد کنید :
CALCIT.OBJ <Enter> : DEL D , PSP بصورت زیر خواهد بود.

```
5CH   FCB#1 :  04  43  41  4C  43  49  54  20  20  4F  42  4A . . .
                   C   A   L   C   I   T           0   B   J
6CH   FCB#2 :  20  20  20  20  20  20  20  20  20  20  20  20 . . .
80H   DTA :    0D  20  44  3A  43  41  4C  43  49  54  2E  4F  42  4A  0D . . .
                       D   :   C   A   L   C   I   T       0   B   J
```

FCB#1 : اولین بایت یعنی شماره درایو (D = 04) و سپس نام فایل، CALCIT که برنامه به آن ارجاع می‌کند. بعد از دو جای خالی که ۸ کاراکتر نام فایل را کامل می‌کند و در نهایت توسعه، OBJ که نقطه ندارد.
DTA : طول ۱۳ (0DH) و سپس دقیقاً آنچه که وارد کردید، حتی شامل 0DH برای <Enter>.

مثال ۴ : فرمان با دو عملوند نام فایل

فرض کنید یک فرمان و دو عملوند بصورت زیر وارد کرده‌اید :

```
COPY    A: FILEA.ASM D:FILEB.ASM
```

بارگذار برنامه FCB و DTA را بصورت زیر تنظیم می‌کند:

```
5CH   FCB#1 :  01  46  49  4C  45  41  20  20  20  41  53  40 . . .
                   F   I   L   E   A                   A   S   M
6CH   FCB#2 :  04  46  49  4C  45  42  20  20  20  41  53  4D . . .
                   F   I   L   E   B                   A   S   M
80H   DTA :    10  20  41  3A  46  49  4C  45  41  2E  41  53  4D  20  etc . . .
                       A   :   F   I   L   E   A   .   A   S   M      ete . . .
```

FCB#1. اولین بایت، ۰۱ ، به درایو A و سپس نام اولین فایل اشاره دارد.
FCB#2. اولین بایت، ۰۴ ، به درایو D و سپس نام دومین فایل اشاره دارد.
DTA. DTA حاوی تعداد کاراکترهای وارد شده (10H) و یک فاصله (20H) و سپس A:FILEA.ASM D:FILEB.A,0DH می‌باشد.

دستیابی به PSP

با تعیین آدرس PSP می‌توانید به منظور پردازش فایل‌های خاص یا انجام فعالیت خاص به آن دستیابی داشته باشید. از آنجائیکه همیشه نمی‌توان فرض کرد که سگمنت کد در یک برنامه EXE. بلافاصله پس از PSP است، می‌توانید با INT 21H تابع 51H آدرس سگمنت از PSP جاری را به BX ارائه دهید. جملات زیر آدرس PSP را گرفته و در ثبات ES ذخیره می‌سازد.

```
MOV   AH,51H    ; درخواست آدرس PSP
INT   21H       ; فراخوانی سرویس وقفه
MOV   ES,BX     ; ذخیره آدرس PSP در ES
```

حال ممکن است از ES برای دستیابی داده‌ها در PSP استفاده کنید :

```
CMP   ES:BYTE   PTR [80],0    ; بررسی بافر PSP
JE    EXIT                    ; صفر داده‌ای نیست
```

برای قرار دادن DTA برای یک برنامه COM. ، 80H را در نسبت‌های BX ، DI و SI تنظیم کنید و محتویات را دستیابی کنید :

```
MOV   SI,80H                  ; آدرس DTA
CMP   BYTE PTR [SI],0         ; بررسی بافر (DS:SI)
JE    EXIT                    ; صفر داده‌ای نیست
```

برنامه مثال : دستیابی PSP

بخشی از برنامه COM. در شکل ۲-۲۴ صفت یک فایل خواسته شده را با حالت معمولی (00H) تنظیم می‌کند کاربر باید نام برنامه و سپس نام فایل را وارد کند مانند : A 24ATTRB d:filename.ext. برنامه DTA را برای کاراکتر <Enter> پیمایش می‌کند و آن را با یک بایت صفر شانزدهی جایگزین می‌سازد. بدین ترتیب یک رشته ASCIIZ ایجاد می‌شود. کاربر همچنین باید مسیر شاخه را وارد کند.

```
TITLE   A24ATTRB (.COM)  Set file attribute to normal
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN   PROC    NEAR
        MOV     AL,0DH          ;Search character <Enter>
        MOV     CX,21           ;Number of bytes
        MOV     DI,82H          ;Start address in PSP
        REPNZ SCASB             ;Scan for <Enter>
        JNZ     A90             ;Not found, error
        DEC     DI              ;Found:
        MOV     BYTE PTR [DI],0 ;Replace with 00H
        MOV     AH,43H          ;Request
        MOV     AL,01           ;  set attribute
        MOV     CX,00           ;  to normal
        MOV     DX,82H          ;ASCIIZ string in PSP
        INT     21H             ;Call interrupt service
        JC      A90             ;Write error?...
;       ...
A90:            ...             ;Error processing
BEGIN   ENDP
        END     BEGIN
```

شکل ۲-۲۴ تنظیم صفت فایل

بلوک‌های حافظه

وقتی که دیگر برنامه‌ها اجرا می‌شوند سیستم امکان بارگذاری هر تعداد از برنامه‌هایی مانند RAMDISK و MOUSE و مقیم باقی ماندن آنها را می‌دهد. سیستم یک یا دو بلوک حافظه، برای هر برنامه بارگذاری شده تنظیم می‌کند. بلافاصله قبل از هر بلوک حافظه یک بازه عنوان (یا رکورد کنترل حافظه) با شروع از یک مرز پاراگراف وجود دارد که شامل فیلدهای زیر می‌باشد :

00-00H کد، که 4DH ('M') یعنی در ادامه بلوک‌های دیگری نیز وجود دارند و 5AH ('Z') یعنی دیگر بلوکی موجود نیست (آخرین بلوک). و این مقادیر صرفاً جنبه توصیفی دارند و وجود آنها ضروری نیست.

01-02H آدرس سگمنت از مالک PSP .0800H یعنی که سگمنت به MSDOS.SYS متعلق است 0000H یعنی که آزاد و در دسترس می‌باشد.

03-04H طول بلوک حافظه، در پاراگراف.

05-07H رزرو شده.

08-0FH نام فایل مالک، در قالب ASCIIZ از DOS4.0

یک لیست پیوندی بلوک‌های حافظه را مرتبط می‌کند *اولین* بلوک حافظه، توسط MSDOS.SYS تنظیم و تسخیر می‌شود و حاوی بافرهای فایل DOS ، FCBهای استفاده شده توسط توابع دستگیره فایل و ژاه اندازهای ابزار بارگذاری شده توسط فرمانهای DEVICE در CONFIG.SYS می‌باشد.

بلوک حافظه **دوم** بخش مقیم COMMAND.COM با PSP خودش می‌باشد. یک برنامهٔ خاص کوچک‌تر مانند FASTOPEN و SHARE ممکن است قبل از COMMAND.COM قرار گیرد.

سومین بلوک حافظه، محیط کارفرما شامل فرمان، COMSPE فرامین PROMPT ، فرامین PATH و هر رشته تنظیم شده با SET می‌باشد.

بلوک‌های بعدی شامل هر برنامه مقیم (TSR) و برنامه اجرایی جاری می‌باشد. هر یک از این برنامه‌ها دو بلوک دارد، اولی یک کپی در محیط و دومی سگمنت برنامه با PSP و ماجول اجرایی می‌باشد.

INT 21H تابع 52H: گرفتن آدرس لیست DOS داخلی

میدان عنوان برای اولین بلوک حافظه که به MSDOSSYS متعلق است می‌تواند توسط یک خصیصه بدون مدرک قرار بگیرد. INT 21H تابع 52H جدول سیستم از آدرسهایی با این ورودی که در قالب دوکلمه‌ای، آغاز می‌شود :

```
DD 00H   آدرس اولین بلوک پارامتر درایو
DD 04H   آدرس لیست جدول‌های فایل سیستم
DD 08H   آدرس راه انداز ابزار CLOCK$
DD 0CH   آدرس راه انداز ابزار CON
. . .
```

تابع 52H آدرس سگمنت از جدولهای فایل (دومین ورودی) در ES و یک افست در BX را باز می‌گرداند. بنابراین ES:[BX-4] به ورودی قبلی اشاره می‌کند، یک دو کلمه‌ای در قالب IP:CS که حاوی آدرس اولین میدان عنوان می‌باشد.

برای یافتن بلوک‌های پیاپی در زنجیره :

۱) از آدرس میدان عنوان برای بلوک حافظ استفاده کنید.

۲) یک را با آدرس سگمنت میدان عنوان برای گرفتن شروع بلوک حافظه جمع کنید. میدان عنوان 10H بایت طول دارد.

۳) طول بلوک حافظه از افست 03-04H میدان عنوان را جمع کنید. حال آدرس سگمنت میدان عنوان بعدی را دارید.

برای تعیین پاراگراف حافظه در دسترس سیستم برای آخرین برنامه، میدان عنوان حاوی 'Z' در بایت صفر را بیابید و محاسبات قبل را انجام دهید. آخرین بلوک تا همه حافظه بالاتر باقیمانده در دسترس می‌باشد.

مثال : پیگیری بلوک‌های حافظه

اگر از DEBUG برای پیگیری بلوک‌های حافظه روی سیستم استفاده می‌کنید، می‌توانید از فرمان E برای محاسبات شانزدهی استفاده نمایید. نحوه استفاده از این دستور به این شکل است :

مقدار شانزدهی دوم و مقدار شانزدهی اول H. فرمان H مجموع و اختلاف در مقدار را باز می‌گرداند. مثال زیر، DEBUG محتویات حافظه را نشان می‌دهد (ترتیب معکوس بایت‌ها را در نظر داشته باشید) پیگیری چنین است :

۱) تابع 52H در ES مقدار [0]CC 02 باز می‌گرداند و در BX مقدار 0026H را باز می‌گرداند. از آنجائیکه چهار بایت سمت چپ در 0022H را می‌خواهید، از D 02 CC:22 برای نمایش آدرس میدان عنوان برای اولین بلوک در قالب IP:CS استفاده کنید که بصورت 560B 0000 و آدرس بصورت [0]0B56 خواهد بود.

۲) از D B56:0 برای نمایش اولین میدان عنوان استفاده کنید : . . . 4D 08 00 AE 05

4D ('M') یعنی 'بلوک‌های حافظه بیشتری خواهد بود 0800 (0008H) به ما می‌گوید که بلوک حافظه به MSDOS.SYS متعلق است و AE05 طول بلوک حافظه است.

۳) در موقعیت دومین میدان عنوان قرار گیرید (COMMAND.COM)

```
موقعیت اولین میدان عنوان :  B56[0]
جمع یک پاراگراف :           +1[0]
جمع طول بلوک حافظه :        + 5AE[0]
موقعیت میدان عنوان بعدی :   1105[0]
```

با استفاده از D nos:0 میدان عنوان بعدی را نمایش دهید :

```
46 06 11 64 01 . . .
```

همچنین می‌توانید در اینجا محتویات COMMAND.COM را بررسی کنید.

۴) در موقعیت میدان عنوان سوم، محیط کارفرما قرار گیرید :

موقعیت میدان عنوان قبلی : 1105[0]
جمع یک پاراگراف : +1[0]
جمع طول بلوک حافظه : +164[0]
موقعیت میدان عنوان بعدی : 1264[0]

می‌توانید به همین روال ادامه دهید تا محتویات محیط کارفرما را بررسی کنید و در موقعیت بلوک‌های باقیمانده قرار گیرید، توجه کنید که برنامه‌های پیاپی دو بلوک حافظه دارند : یکی برای محیط و دیگری برای سگمنت برنامه خود. آخرین میدان عنوان 5AH ('Z') در اولین بایت خود دارد. اگر شما از روال DEBUG نمایش می‌دهید، بلوک حافظه شما است، چون DEBUG آخرین برنامهٔ قرار گرفته در حافظه است.

کار با بلوک‌های حافظه بالایی

از DOS 5.0 به بعد می‌توان از دستور DOS = UMB (بلوک حافظه فوقانی) برای تخصیص حافظه بالای حافظه متعارف بین 640K و مرز یک مگابایت استفاده کرد. این دستور سبب می‌شود که سیستم یک میدان عنوان فرضی ۱۶ بایتی قبل از مرز 640K برقرار سازد و بعنوان خودش علامت بزند. اندازه فیلد آن شامل یک مقدار بزرگ کافی برای کنار گذاشتن بافرهای ویدئو و روتینهای ROM می‌باشد.

با این روش، می‌توانید از آخرین میدان عنوان در حافظه مرسوم برای قرار گرفتن در بلوک‌های حافظه ناحیه فوقانی استفاده کنید. داخل حافظه فوقانی از دیگر میدان‌های عنوان که بصورت مارک دار علامت زده شده‌اند برای کنار گذاشتن هر ناحیه که هم اکنون توسط ROM یا ویدئو اشغال شده نیز استفاده می‌شود.

روش تخصیص حافظه

INT 21H تابع 58H چندین روش تعیین مکان حافظه برای بارگذاری برنامه دارد.

تابع 5800H : گرفتن روش تخصیص حافظه

این تابع امکان درخواست روش تخصیص حافظه را دارا است :

```
MOV   AX,5800H  ; درخواست اخذ روش
INT   21H       ; فراخوانی سرویس وقفه
```

این تابع پرچم نقلی را پاک می‌کند و روش را در AX باز می‌گرداند.

- 00H = اولین قرارگیری (پیش فرض) : جستجو از پایین‌ترین آدرس در حافظهٔ مرسوم تا اولین بلوک مورد دسترس که بقدر کافی برای برنامه فضا دارد.
- 01H = بهترین قرارگیری : جستجو برای یافتن کوچکترین بلوک مورد دسترس در حافظه مرسوم که بقدر کافی برای برنامه فضا دارد.
- 02H = آخرین قرارگیری : جستجو از بالاترین آدرس حافظه مرسوم تا اولین بلوک مورد دسترس.
- 40H = اولین قرارگیری، فقط بالا : جستجو از پایین‌ترین آدرس در حافظه فوقانی برای اولین بلوک مورد دسترسی.
- 41H = بهترین قرارگیری، فقط بالا : جستجو برای کوچکترین بلوک مورد دسترس در حافظه فوقانی.
- 42H = آخرین قرارگیری، فقط بالا : جستجو از بالاترین آدرس در حافظه فوقانی برای اولین بلوک مورد دسترسی.
- 80H = اولین قرارگیری، بالا : جستجو کمترین آدرس در حافظه فوقانی برای اولین بلوک مورد دسترسی و اگر پیدا نشد، جستجوی حافظه مرسوم.
- 81H = بهترین قرارگیری، بالا : جستجو از پایین‌ترین آدرس در حافظه فوقانی برای کوچکترین بلوک مورد دسترسی

و اگر پیدا نشد، جستجوی حافظه مرسوم.

- 82H = آخرین قرارگیری، بالا : جستجو از بالاترین آدرس در حافظه فوقانی برای اولین بلوک مورد دسترسی و اگر پیدا نشد، جستجوی حافظه مرسوم.

بهترین‌قرارگیری و آخرین قرارگیری برای سیستم‌های چندکاره مناسب‌هستند، که می‌توانند حافظه را قطعه‌بندی‌کنند چون برنامه‌هابصورت همزمان‌اجرامی‌شوند. وقتی یک‌برنامه، پردازش را خاتمه‌می‌دهد، حافظه برای سیستم‌آزادمی‌شود.

تابع 5801H : تنظیم روش تخصیص حافظه

این تابع امکان تغییر روش تخصیص حافظه را دارد. برای تنظیم یک روش، AL را با کد ۰۱ و BX را با کد روش تنظیم کنید. یک خطا پرچم نقلی را تنظیم می‌کند و ۰۱ را در AX باز می‌گرداند (تابع نامعتبر).

تابع 5802H : گرفتن پیوند حافظه فوقانی

این تابع مبنی بر این است که آیا یک برنامه می‌تواند حافظه‌ای از ناحیه حافظه فوقانی (بالای 640K) تخصیص دهد. عملیات پرچم نقلی را صفر می‌کند و یکی از کدهای زیر را در AL باز می‌گرداند. 00H یعنی ناحیهٔ پیوند زده نیست و نمی‌توانید آن را تخصیص دهید و 01H یعنی ناحیه پیوند زده است و می‌توانید آن را تخصیص دهید.

تابع 5803H : تنظیم پیوند حافظه فوقانی

این تابع می‌تواند ناحیه حافظه فوقانی را پیوند بزند یا ناحیه پیوند زده شده را تفکیک کند. اگر ناحیه پیوند زده است می‌توان حافظه‌ای از آن تخصیص داد :

```
MOV   AX,5803H        ; درخواست
MOV   BX,linkflag     ; link / unlink
INT   21H             ; ناحیه بالایی حافظه
```

پارامتر پرچم پیوند، معنای زیر را دارد : 00H = پیوند نزدن ناحیه و 01H = پیوند زدن حافظه، یک عملیات موفق پرچم نقلی را پاک می‌کند و به برنامه امکان تخصیص حافظه از آن را می‌دهد. بروز خطا سبب تنظیم پرچم نقلی و بازگشت کد ۰۱ در AX (یعنی CONFIG.SYS حاوی DOS = UMB) یا 07 (پیوند حافظه شکست خورد) می‌شود.

بارگذار برنامه

در هنگام بارگذاری برنامه‌های COM. و EXE. بارگذار برنامه مراحل زیر را انجام می‌دهد :

۱) تنظیم اولیه بلوک‌های حافظه برای محیط برنامه و سگمنت برنامه.

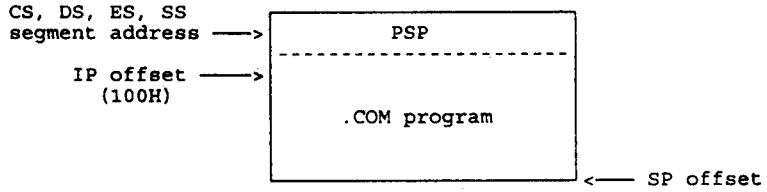
۲) ایجاد پیشوند سگمنت برنامه در موقعیت 00H از سگمنت برنامه و بارگذاری برنامه در 100H.

بارگذاری و اجرای یک برنامه COM.

چون سازماندهی یک برنامه COM. نسبتاً ساده است، بارگذار برنامه کافی است بداند توسعه فایل COM. است. همانطور که قبلاً توصیف شد، یک پیشوند سگمنت برنامه قبل از بارگذاری برنامه‌های COM. و EXE. در حافظه قرار می‌گیرد.اولین دو بایت PSP حاوی دستور INT 20H (بازگشت به DOS) است. در زمان بارگذاری برنامه COM بارگذار برنامه :

- چهار ثبات سگمنت را با آدرس اولین بایت از PSP تنظیم می‌کند.
- اشاره‌گر پشته را (SP) با انتهای سگمنت 64K افست FFFEH تنظیم می‌کند (یا اگر سگمنت بقدر کافی بزرگ نیست با انتهای حافظه) و یک کلمه صفر روی پشته قرار می‌دهد.

- اشاره‌گر دستور را با 100H (اندازهٔ PSP) تنظیم می‌کند و کنترل را به قبل از آدرس تولید شده توسط CS:IP منتقل می‌کند، یعنی اولین موقعیت پس از PSP. این اولین بایت از برنامه شماست و ممکن است شامل یک دستور اجرایی باشد. شکل ۳-۲۴ این مقدار دهی را مشخص می‌سازد.



شکل ۳-۲۴ مقدار دهی یک برنامه COM.

بارگذاری و اجرای یک برنامه EXE.

فایل (EXE.)ای که توسط لینکر روی دیسک ذخیره می‌شود، شامل دو بخش است : یک رکورد ***عنوان*** که شامل اطلاعات کنترل و تخصیص دهی مجدد است و ***ماجول بارگذار*** واقعی.

عنوان حداقل ۵۱۲ بایت است و اگر عناصر جایگیری بیشتری وجود داشته باشد، ممکن است طولانی‌تر باشد. بارگذار شامل اطلاعاتی راجع به اندازه ماجول اجرایی است که باید به حافظه بارگذاری شود، آدرس پشته، افست‌های جایگیری که باید در آدرسهای ماشین تکمیل نشده جایگزین شود. در لیست زیر، عبارت ***بلوک*** به یک ناحیه ۵۱۲ بایتی از حافظه اشاره دارد :

00-01H 4D5A شانزدهی ('MZ') یک فایل EXE. را مشخص می‌سازد.

02-03H تعداد بایت‌های موجود در آخرین بلوک فایل EXE.

04-05H اندازه فایل شامل عنوان، در یک بلوک ۵۱۲ بایتی. برای مثال، اگر اندازه آن ۱۰۲۵ باشد، این فیلد حاوی ۲ و 02-03H حاوی یک می‌باشد.

06-07H تعداد عناصر جدول تخصیص‌دهی (1CH) را ببینید.

08-02H اندازه عنوان، در ۱۶ بایت (پاراگراف) افزایشی، برای کمک به بارگذار برنامه تا در شروع ماجول اجرایی بعد از عنوان قرار گیرد. حداقل تعداد 20H (32) است (بایت 16=512x32).

0A-0BH حداقل تعداد پاراگراف‌هایی که باید بعد از انتهای برنامه در هنگام بارگذاری برنامه قرار گیرد.

0C-0DH سوئیچ بارگذار بالا / پایین. وقتی پیوند زده می‌شود، باید تصمیم بگیرید که برنامه جهت اجرا در یک ناحیه پایین یا آدرس حافظه فوقانی قرار گیرد. مقدار 0000H مبنی بر بالا است. در غیراینصورت، این موقعیت شامل حداکثر تعداد پاراگراف‌هایی است که باید بعد از برنامه در هنگام بارگذاری قرار گیرند.

0E-0FH افست موقعیت از سگمنت پشته در ماجولهای اجرایی.

10-11H اندازه تعریف شده پشته بصورت افست که بارگذار در ثبات SP قرار می‌دهد وقتی کنترل به ماجول اجرایی منتقل می‌شود.

12-13H مقدار مجموع بررسی - مجموع همه کلمات فایل (بدون در نظر گرفتن سرریز)، بعنوان بررسی اعتبار داده از دست رفته احتمالی.

14-15H افستی (همیشه، امانه الزاماً 00H) که بارگذار درثبات IP، هنگام‌انتقال‌کنترل یک ماجول‌اجرایی قرارمی‌دهد.

16-17H افست سگمنت کد موجود در ماجول اجرایی در این قسمت قرار دارد. این آدرس توسط بارگذار ثبات CS قرار می‌گیرد. افست با دیگر سگمنت‌ها مرتبط است، بنابراین اگر سگمنت کد درابتدا باشد، افست صفر خواهد بود.

18-19H افست جدول تخصیص دهی مجدد (عنصری که در 1CH است ببینید).

1A-1BH شماره روکش، وقتی صفر است یعنی که فایل EXE. حاوی برنامه اصلی است.

1CH-end جدول تخصیص دهی مجدد شامل تعداد متغیرهای عناصر تخصیص دهی که در افست 06-07H مشخص شده‌اند. موقعیت 06-07H از عنوان مبنی برتعداد عناصر ماجول اجرایی که تخصیص داده‌شده‌اند. هرعنصر تخصیص دهی، با شروع از عنوان 1CH ، شامل ۲ بایت مقدار افست و ۲ بایت مقدار سگمنت می‌باشند.

سیستم بلوک‌های حافظه را برای محیط و سگمنت برنامه مهیا می‌سازد.

در زیر، مراحلی که بارگذار برنامه در هنگام بارگذاری و مقدار دهی یک برنامه EXE. انجام می‌دهد، آورده شده است :

- خواندن بخش‌های فرمت شده از عنوان در حافظه.
- محاسبه اندازه ماجول اجرایی (حاصل اندازه فایل در موقعیت 04H مدلهای اندازه عنوان در موقعیت 08H) و خواندن ماجول در حافظه در سگمنت شروع.
- خواندن عناصر جدول تخصیص دهی در یک ناحیه کاری و جمع هر مقدار عنصر با مقدار سگمنت شروع
- تنظیم ثبات‌های DS و ES با آدرس سگمنت PSP.
- تنظیم ثبات SS با آدرس PSP ، بعلاوه 100H (اندازه PSP)، بعلاوه مقدار افست SS (در 0EH). همچنین، تنظیم ثبات SP ، با مقداری که در 10H است، یعنی اندازه پشته.
- تنظیم CS با آدرس PSP بعلاوه 100H (اندازه PSP)، بعلاوه افست CS در عنوان (در 16H) همچنین تنظیم IP با افست در 14H. جفت ثبات CS:IP آدرس شروع سگمنت کد و در حقیقت اجرای برنامه را فراهم می‌سازند. شکل ۴-۲۴ این مقدار دهی را مشخص می‌سازد.

بعد از مراحل قبلی، بارگذار با عنوان EXE. خاتمه می‌یابد و آن را کنار می‌گذارد. ثبات‌های CS و SS به نحوه صحیحی تنظیم می‌شوند. اما برنامه شما باید DS و ES را برای سگمنت داده خودش تنظیم کند.

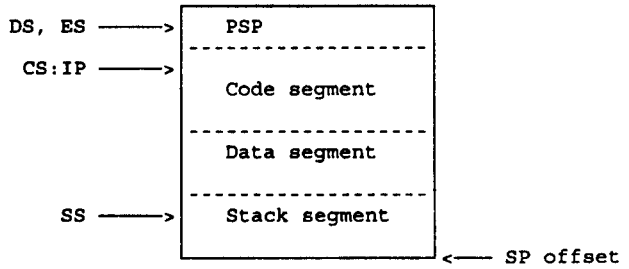
```
MOV   AX,data segname    ; تنظیم ثبات‌های DS و ES
MOV   DS,AX              ; با آدرس
MOV   ES,AX              ; سگمنت داده
```

مثال : بارگذاری یک برنامه EXE.

نقشه پیوند زیر که لینکر برای یک برنامه EXE. تولید می‌کند در نظر بگیرید:

نقشه موقعیت مرتبط (غیر واقعی) از هر سه سگمنت را فراهم می‌سازد. توجه کنید که برخی سیستمها این سگمنت‌ها را به ترتیب حروف الفبائی نام مرتب می‌کنند بر طبق نقشه سگمنت کد (CSEG) از 00000H آغاز می‌شود - این موقعیت مرتبط با شروع ماجول اجرائی است و طول آن 003BH بایت می‌باشد. سگمنت داده DSEG از 00040H شروع می‌شود و طول آن 001BH می‌باشد. 00040H اولین آدرس پس CSEG است که در مرز یک پاراگراف قرار می‌گیرد (یک مرز قابل تقسیم بر 10H است). سگمنت پشته STACK که از 00060H آغاز می‌شود، اولین آدرس پس از DSEG است که در مرز پاراگراف قرار می‌گیرد.

```
Start     Stop   Length  Name          Class

00000H  0003AH  003BH  CSEG          Code
00040H  0005AH  001BH  DSEG          Data
00060H  0007FH  0020H  STACK         Stack

Program entry point at 0000:0000
```



شکل ۴-۲۴ مقدار دهی یک برنامه *EXE*.

DEBUG رکورد عنوان را پس از بارگذاری برنامه نمی‌تواند نشان دهد چون بارگذار رکورد عنوان را با PSP جایگزین می‌کند. اما می‌توانید از فرمان L برای بارگذاری یک سکتور از دیسک و از فرمان D برای نمایش آن استفاده کنید. بعنوان مثال بارگذاری CS:100 از درایو :A (0)، سکتور رابطه‌ای 3 و یک سکتور 512 بایتی وارد کنید : L 100 03 1. عنوان برنامه‌ای که ما بررسی می‌کنیم شامل اطلاعات زیر است، به ترتیب موقعیت شانزدهی (داده‌های عددی به ترتیب بایت معکوس هستند).

00H 4D5A شانزدهی ('MZ')

02H تعداد بایت‌های آخرین بلوک :5B00H (یا 005BH)

04H اندازه فایل شامل عنوان در بلوک‌های 512 بایتی :0200H (یا 1024 = 512 × 0002)

06H تعداد عناصر جدول تخصیص دهی بعد از بخش فرمت شده عنوان 0100H (0001)

08H اندازه عنوان در شانزده بایت افزایشی :2000H (0020H = 32 و بایت 512 = 16×32)

0CH بارگذاری در حافظهٔ پائینی :FFFFH.

0EH موقعیت افست سگمنت پشته :6000H ، یا 0060H.

10H افست جایگذاری در SP که : 2000H ، یا 0020H.

14H افست برای IP:0000H.

16H افست برای CS:0000H.

18H افست جدول تخصیص دهی 1E00H ، یا 001EH.

وقتی DEBUG این برنامه را بارگذاری می‌کند ثبات‌ها حاوی مقادیر زیرند:

```
SP=0020  05=138F  ES=138F
SS=13A5  CS=139F  IP=0000
```

برای ماجولهای EXE. بارگذار ES,DS را با آدرس PSP تنظیم می‌کند و CS ، IP ، SS و SP با مقادیری از رکورد عنوان تنظیم می‌شوند. بیایید ببینیم چگونه بارگذار این ثبات‌ها را مقدار می‌دهد.

ثبات‌های CS:IP

بر طبق ثبات DS ، وقتی برنامه بارگذاری می‌شود، آدرس PSP ، 13BF[0]H خواهد بود. چون PSP ، 100H بایت طول دارد و سگمنت کد در افست 0 است، سگمنت کد پس از PSP در 139F[0]H قرار دارد. می‌توانید افست موقعیت 16H عنوان را ببینید. بارگذار این مقدار برای مقدار دهی ثبات CS استفاده می‌کند.

آدرس شروع PSP(DS) :	138F0H
طول PSP :	+ 100H
افست سگمنت کد :	0H
آدرس سگمنت کد	139F0

حال CS آدرس شروع بخش کد برنامه (CSEG) را فراهم می‌سازد. می‌توانید از فرمان نمایش D DEBUG CS:0000 برای دیدن کد ماشین یک برنامه در حافظه استفاده کنید. کد باقسمت شانزدهی لیست خروجی اسمبلر یکسان‌است، بجز اینکه عملوندهادر LST. با R برچسب زده‌شوند. همچنین بارگذار IP را با 0000H، افستی‌از 14H عنوان، تنظیم‌می‌کند.

ثبات‌های SS:SP

بارگذار از مقدار 60H در عنوان (در 0EH) برای تنظیم آدرس پشته در ثبات SS استفاده می‌کند.

آدرس شروع PSP(DS را ببینید) :	138F0H
طول PSP :	+ 100H
افست پشته (موقعیت 0EH در عنوان را ببینید):	60H
آدرس پشته	13A50H

بارگذار از 20H عنوان (در 10H) برای مقدار دهی اشاره‌گر پشته با طول پشته استفاده می‌کند در این مثال، پشته بصورت DW 16DUP (?) تعریف شده است، که ۱۶ فیلد ۲ بایتی = ۳۲ یا 20H است. SP بالای پشته اشاره می‌کند.

ثبات DS

بارگذار از ثبات DS برای برقراری نقطه شروع PSP در 138F[0] استفاده می‌کند. چون عنوان شامل آدرس شروع DS نیست، برنامه شما باید آن را مقدار دهی نماید :

```
0004   B8 . . . R     MOV AX,DSEG
0007   8E D8          MOV DS,AX
```

آدرس ماشین پر نشده سمت چپ از DSEG است که یک ورودی در جدول تخصیص عنوان می‌باشد (که در 1EH آغاز می‌شود) بارگذار آدرس DS را چنین محاسبه می‌کند :

آدرس CS :	139F0H
بعلاوه افست برای DS :	+ 40H
آدرس DS :	13A30H

DEBUG دستور کامل را بصورت B8A313 نمایش می‌دهد. بارگذار A313 را در DS بصورت 13A3 ذخیره می‌کند.

حال ما این مقادیر را در شروع اجرا داریم :

ثبات	آدرس	افست نقشه
CS	139F[0]H	00H
DS	13A3[0]H	40H
SS	13AS[0]H	60H

بعنوان تمرین، هر برنامه EXE. لینک شده را با DEBUG پیگیری نمایید و به مقادیر تغییر یافته در ثبات‌ها توجه کنید:

دستور	ثبات‌های تغییر یافته
MOV AX,DSEG	IP,AX
MOV DS,AX	IP,DS
MOV ES,AX	IP,ES

حال DS حاوی آدرس صحیح سگمنت داده است. می‌توانید از D DS:00 برای دیدن محتویات سگمنت داده استفاده کنید و با دستور D SS:00 محتویات پشته را ببینید.

تخصیص دهی و آزاد سازی حافظه

سرویس‌های INT 21H به شما امکان تخصیص دهی، آزادسازی و تغییر اندازه یک ناحیه حافظه را می‌دهند. احتمالاً از این سرویس برای برنامه‌های مقیم و برنامه‌هایی که دیگر برنامه‌ها را جهت اجرا بارگذاری می‌کنند استفاده

خواهید کرد. چون DOS بصورت یک محیط تک کاربره طراحی شده است، یک برنامه که به بارگذاری برنامه دیگری نیاز دارد باید برخی از فضای حافظه خود را آزاد سازد.

INT 21H تابع 48H : تخصیص حافظه

تخصیص حافظه برای یک برنامه مستلزم استفاده از تابع 48H و تنظیم BX با تعداد پاراگراف‌های مورد نیاز است :

```
MOV  AH,48H        ; درخواست تخصیص حافظه
MOV  BX,paragraphs ; تعداد پاراگراف
INT  21H           ; فراخوانی سرویس وقفه
```

این تابع کار را از اولین بلوک حافظه آغاز می‌کند و هر بلوک مرحله‌ای پیش می‌رود تا در یک فضای کافی بزرگ قرار گیرد، این ناحیه غالباً در انتهای حافظه قرار دارد.

یک عملیات موفق پرچم نقلی را پاک می‌کند و در AX آدرس سگمنت از بلوک حافظه تخصیص داده شده باز می‌گرداند. یک عملیات ناموفق پرچم نقلی را تنظیم می‌کند و در AX یک کد خطا (۰۷ = بلوک‌های حافظه تخریب شده یا ۰۷ = حافظه کافی نیست) و در BX اندازه، به پاراگراف، از بزرگترین بلوک در دسترس باز می‌گرداند. یک بلوک حافظه تخریب شده یعنی عملیات یک بلوک پیدا کرده که اولین بایت آن 'M' یا 'Z' نیست.

INT 21H تابع 49H : آزاد سازی حافظه تخصیص یافته

تابع 49H حافظه تخصیص یافته را آزاد می‌سازد، عموماً برای پاک کردن برنامه مقیم بکار می‌رود، در ES آدرس سگمنت از بلوکی که بازگردانده می‌شود قرار دهید.

```
MOV  AH,49H          ; درخواست آزاد سازی حافظه تخصیص یافته
LEA  ES,seg-address  ; آدرس بلوک برای پاراگراف
INT  21H             ; فراخوانی سرویس وقفه
```

یک عملیات موفق پرچم نقلی را پاک می‌کند و 00H را در دومین و سومین بایت بلوک حافظه ذخیره می‌سازد به معنی آنکه بعد از این مورد استفاده نیست. یک عملیات ناموفق پرچم نقلی را یک می‌کند و در AX یک کد خطا (۰۷ = بلوک حافظه تخریب شده و ۰۹ = آدرس بلوک حافظه نامعتبر) باز می‌گرداند.

INT 21H تابع 4AH : تغییر بلوک حافظه تخصیص یافته

تابع 4AH می‌تواند اندازه بلوک حافظه را افزایش یا کاهش دهد. BX را با تعداد پاراگراف‌هایی که برای برنامه نگهداری می‌شود و ES را با آدرس PSP مقدار دهی کنید:

```
MOV  AH,4AH          ; درخواست تغییر حافظهٔ تخصیص یافته :
MOV  BX,paragraphs   ; تعداد پاراگراف
LEA  ES,PSP-address; ; آدرس PSP
INT  21H             ; فراخوانی سرویس وقفه
```

یک برنامه می‌تواند اندازه خودش را با تفریق انتهای آخرین سگمنت از آدرس PSP محاسبه کند. اگر لینکر سگمنت‌ها را بترتیب الفبایی مرتب نموده است از اینکه آخرین سگمنت را استفاده می‌کنید اطمینان حاصل کنید.

یک عملیات موفق پرچم نقلی را پاک می‌کند یک عملیات ناموفق پرچم نقلی را یک می‌کند و در AX یک کد خطا باز می‌گرداند (۰۷ = بلوک حافظه تخریب شده، ۰۷ = حافظه کافی نیست، ۰۹ = آدرس بلوک حافظه نامتعبر) و در BX حداکثر اندازه ممکن را باز می‌گرداند (اگر تلاش برای افزایش اندازه باشد) یک آدرس غلط در ES سبب خطای ۰۷ خواهد شد.

بارگذاری یا اجرای یک تابع برنامه

حال بیایید چگونگی اخذ یک برنامه اجرایی برای بارگذاری و اجرای یک زیر برنامه را بررسی کنیم. تابع 4BH برنامه را قادر می‌سازد تا یک زیر برنامه را در حافظه جهت اجرا بارگذاری کند. این ثبات‌ها را مقدار دهی کنید :

- AL = کد تابعی برای یکی از موارد زیر : 00H = بارگذار و اجرا، 01H = بارگذاری برنامه، 03H = بارگذاری رویهم، 05H = تنظیم حالت اجرا (در این متن نیست).
- ES:BX = آدرس یک بلوک پارامتر
- DS:DX = آدرس نام برای زیر برنامه فراخوانده شده، یک رشته ASCIIZ با حروف بزرگ در اینجا دستورات جهت بارگذاری زیر برنامه آورده شده است :

```
MOV   AH,4BH        ; درخواست بارگذاری زیر برنامه
MOV   AL,code       ; کد تابع (فقط بارگذاری)
LEA   BX,para-block ; آدرس بلوک پارامتر
LEA   DX,path       ; آدرس نام مسیر
INT   21H           : فراخوانی سرویس وقفه
```

یک عملیات نامعتبر پرچم نقلی را تنظیم می‌کند و در AX کد خطا باز می‌گرداند.

AL = 00H : بارگذاری و اجرا

این عملیات یک برنامه EXE. یا COM. را در حافظه قرار داده، پیشوند سگمنت برنامه را برقرار می‌سازد و کنترل را به آن جهت اجرا منتقل می‌کند. چون همه ثبات‌ها حتی SS تغییر می‌کند، این عملیات برای مبتدیان نیست. بلوک پارامتر آدرسدهی شده توسط ES:BX قالب زیر را دارد :

افست	هدف
00H	آدرس سگمنت بلوک - محیطی ارسالی به PSP + 2CH. یک آدرس صفر یعنی برنامه بارگذاری شده وارد محیط ماقبل خود شده است.
02H	اشاره‌گر دو کلمه‌ای به یک خط فرمان برای قرارگیری در PSP+80H.
06H	اشاره‌گر دو کلمه‌ای به یک FCB#1 پیش فرض برای ارسال در PSP+5CH.
0AH	اشاره‌گر دو کلمه‌ای به FCB#2 پیش فرض برای ارسال در PSP+6CH اشاره‌گرهای دو کلمه‌ای بصورت آدرس سگمنت: افست می‌باشند.

AL = 01H : بارگذاری برنامه

عملیات یک برنامه EXE. یا COM. را در حافظه قرارمی‌دهد و یک پیشوند سگمنت برنامه برای آن برقرارمی‌کند، اما کنترل را برای اجرا به آن منتقل نمی‌کند. بلوک پارامتر که توسط ES:BX آدرسدهی شده قالب زیر دارد :

افست	هدف
00H	آدرس سگمنت بلوک محیطی برای ارسال در PSP + 2CH. اگر آدرس صفر باشد، برنامه بارگذاری شده محیط را از ماقبل خود ارث می‌برد.
02H	اشاره‌گر دو کلمه‌ای به فرمان خطی برای جایگذاری در PSP+80H.
06H	اشاره‌گر دو کلمه‌ای به FCB#1 پیش فرض برای ارسال به PSP+5CH.
0AH	اشاره‌گر دو کلمه‌ای به FCB#2 پیش فرض برای ارسال در PSP+6CH
0EH	آدرس شروع پشته
12H	آدرس شروع سگمنت کد

اشاره‌گرهای دو کلمه‌ای بصورت سگمنت : افست آدرسدهی می‌شوند.

AL = 03H بارگذاری رویهم

این عملیات یک برنامه یا بلوک از کد را بارگذاری می‌کند، اما یک PSP برقرار نمی‌سازد یا اجرای برنامه یا بلوک را آغاز نمی‌کند. بنابراین درخواستی می‌تواند روکش شود. بلوک پارامتر آدرسدهی شده توسط ES:BX قالب زیر را دارد:
افست 00H آدرس کلمه‌ای سگمنت جائیکه فایل قرار می‌گیرد.
افست 02H عامل کلمه‌ای تخصیص دهی مجدد برای فراهم سازی تصاویر
یک خطا پرچم نقلی را تنظیم می‌کند و یک کد خطا در AX باز می‌گرداند، در فصل ۱-۱۸ توضیح داده شد.

برنامه : بارگذاری و اجرا

برنامه شکل ۵-۲۴ از سیستم تقاضا می‌کند تا فرمان DIR را برای درایو D اجرا نماید. ابتدا برنامه از تابع 4DH برای کاهش تقاضای حافظه به اندازه، واقعی استفاده می‌کند اختلاف بین آخرین سگمنت (فرضی) ZNDSEG و شروع PSP خودش. توجه کنید که در اینجا، ES هنوز حاوی آدرس PSP است، همانطوری که هنگام ورود بارگذاری شده است. (جملات ASSUME قبل و بعد از MOV BX ، SEG ZNDSEG ظاهر می‌شود برای MASM5.1 لازم است ولی برای دیگر اسمبلرها نیازی نیست). اندازه ماجول ۸۰ بایت است، بنابراین PSP (10H پاراگراف) و برنامه (۸ پاراگراف) مجموعاً 18H پاراگراف می‌باشد.

تابع 4BH با کد 00 در AL بارگذاری و اجرای COMMAND.COM را دستکاری می‌کند. برنامه ورودیهای شاخه برای درایو D را نمایش می‌دهد.

INT 21H تابع 4DH : گرفتن مقدار بازگشتی زیر برنامه

این تابع، مقدار بازگشتی را بازیابی می‌کند که آخرین زیر برنامه وقتی توسط تابع 4CH یا 31H خاتمه می‌یابد، ارائه می‌دهد. مقدار بازگشتی چنین است :

- AH حاوی روش خاتمه زیر برنامه است که 00H = خاتمه معمول، 01H = خاتمه با Ctrl + C
- 02H خطای بحرانی ابزار و 03H = خاتمه توسط تابع 31H (گرفتن برنامه)
- AL حاوی مقدار بازگشتی از زیر برنامه است.

روکش گذاری برنامه

برنامه شکل ۶-۲۴ از همان سرویس (4BH) برنامه ۵-۲۴ استفاده می‌کند، اما این بار یک برنامه در حافظه بدون آنکه اجرا شود، قرار می‌گیرد. پردازش شامل برنامه اصلی، A24CALLV و دو زیر برنامه، A24SUB1 و A24SUB2 است. A24CALLV شامل سگمنت‌های زیر می‌باشد :

```
STACKSG   SEGMENT   PARA   STACK 'stack
DATASG    SEGMENT   PARA   'DATA'
CODESG    SEGMENT   PARA   'code'
ZENDSG    SEGMENT          ; سگمنت فرضی(خالی)
```

A24SUB1 با A24CALLV پیوند زده شده و فراخوانی می‌شود. سگمنت‌های آن چنین هستند.

```
DATASG    SEGMENT   PARA   'Data2'
CODESG    SEGMENT   PARA   'Code2'
```

```
TITLE     A24EXDIR (EXE) INT 21H function 4BH to execute DIR
; ---------------------------------------------------------------
SSEG      SEGMENT PARA STACK 'Stack'
          DW      32(?)
SSEG      ENDS
; ---------------------------------------------------------------
DSEG      SEGMENT PARA 'Data'
PARAREA   LABEL   BYTE              ;Parameter block for load/exe
          DW      0                 ;  address of envir. string
          DW      OFFSET DIRCOM     ;  pointer to command line
          DW      DSEG
          DW      OFFSET FCB1       ;  pointer to default FCB1
          DW      DSEG
          DW      OFFSET FCB2       ;  pointer to default FCB2
          DW      DSEG
DIRCOM    DB      17,'/C DIR D:',13,0   ;Command line
FCB1      DB      16 DUP(0)
FCB2      DB      16 DUP(0)
PROGNAM   DB      'C:\COMMAND.COM',0    ;Location of COMMAND.COM
DSEG      ENDS
; ---------------------------------------------------------------
CSEG      SEGMENT PARA 'Code'
          ASSUME  CS:CSEG,DS:DSEG,SS:SSEG,ES:DSEG
A10MAIN   PROC    FAR
          MOV     AH,4AH            ;Reduce allocated memory spac
          ASSUME  CS:ZNDSEG
          MOV     BX,SEG ZNDSEG     ;Ending segment
          ASSUME  CS:CSEG
          MOV     CX,ES             ;  minus start of
          SUB     BX,CX             ;  program segment
          INT     21H
          JC      A20ERR            ;Not enough space?
          MOV     AX,DSEG           ;Yes,
          MOV     DS,AX             ;  set DS and ES
          MOV     ES,AX
          MOV     AH,4BH            ;Request load
          MOV     AL,00             ;  and execute
          LEA     BX,PARAREA        ;  COMMAND.COM
          LEA     DX,PROGNAM
          INT     21H
          JC      A30ERR            ;Execute error?
          MOV     AL,00             ;OK, no error code
          JMP     A90XIT
A20ERR:
          MOV     AL,01             ;Error code 1
          JMP     A90XIT
A30ERR:
          MOV     AL,02             ;Error code 2
          JMP     A90XIT
A90XIT:
          MOV     AH,4CH            ;Request
          INT     21H               ;  end processing
A10MAIN   ENDP
CSEG      ENDS

ZNDSEG    SEGMENT                   ;Dummy segment
ZNDSEG    ENDS
          END     A10MAIN
```

شکل ۵-۲۴ اجرای *DIR* از داخل برنامه

سگمنت A24CALLV ابتدا پیوند زده می‌شود - بهمین دلیل نام کلاس‌ها فرق می‌کند : 'Data1'، 'Data2'، 'Code1'، 'Code2' و الی آخر، در اینجا نقشه پیوند A24CALLV + A24SUB1 آورده شده است.

```
TITLE     A24CALLV (EXE)  Call subprogram and overlay
          EXTRN,  A24SUB1:FAR
; ------------------------------------------------
STACKSG   SEGMENT PARA STACK 'Stack1'
          DW      64 DUP(?)
STACKSG   ENDS
; ------------------------------------------------
DATASG    SEGMENT PARA 'Data1'
PARABLK   LABEL   WORD              ;Parameter block
          DW      0                 ;
          DW      0                 ;
FILENAM   DB      'F:\A24SUB2.EXE',0
ERRMSG1   DB      'Modify mem error'
ERRMSG2   DB      'Allocate error  '
ERRMSG3   DB      'Seg call error  '
DATASG    ENDS
; ------------------------------------------------
CODESG    SEGMENT PARA 'Code1'
A10MAIN   PROC    FAR
          ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
          MOV     AX,DATASG
          MOV     DS,AX
          CALL    Q10SCR            ;Scroll screen
          CALL    A24SUB1           ;Call subprogram 1

          MOV     AH,4AH            ;Shrink memory
          ASSUME  CS:ZENDSG
          MOV     BX,SEG ZENDSG     ;Address of end program
          ASSUME  CS:CODESG
          MOV     CX,ES             ;Address of PSP
          SUB     BX,CX             ;Size of this program
          INT     21H
          JC      A20ERR            ;If error, exit

          MOV     AX,DS             ;Initialize ES for
          MOV     ES,AX             ;  this service
          MOV     AH,48H            ;Allocate memory for overlay
          MOV     BX,40             ;40 paragraphs
          INT     21H
          JC      A30ERR            ;If error, exit
          MOV     PARABLK,AX        ;Save segment address

          MOV     AH,4BH            ;Load subprogram 2
          MOV     AL,03             ;  with no execute
          LEA     BX,PARABLK
          LEA     DX,FILENAM
          INT     21H
          JC      A50ERR            ;If error, exit
          MOV     AX,PARABLK        ;Exchange two words
          MOV     PARABLK+2,AX      ;  of PARABLK
          MOV     PARABLK,20H       ;Set CS offset to 20H
          LEA     BX,PARABLK
          CALL    DWORD PTR [BX]    ;Call subprogram 2
          JMP     A90
A20ERR:
          CALL    Q20SET            ;Set cursor
          LEA     DX,ERRMSG1
          CALL    Q30DISP           ;Display message
          JMP     A90
A30ERR:
          CALL    Q20SET            ;Set cursor
          LEA     DX,ERRMSG2
          CALL    Q30DISP           ;Display message
          JMP     A90
A50ERR:
          CALL    Q20SET            ;Set cursor
          LEA     DX,ERRMSG3
          CALL    Q30DISP           ;Display message
          JMP     A90
```

شکل الف ٦-٢۴ فراخوانی یک زیر برنامه و روکش گذاری

```
A90:
            MOV     AX,4C00H          ;End processing
            INT     21H
A10MAIN     ENDP
;                   Video screen services:
;                   ----------------------
Q10SCR      PROC    NEAR
            MOV     AX,0600H          ;Request scroll
            MOV     BH,1EH            ;Set attribute
            MOV     CX,0000
            MOV     DX,184FH
            INT     10H
            RET
Q10SCR      ENDP

Q20SET      PROC    NEAR
            MOV     AH,02H            ;Request set
            MOV     BH,00             ;  cursor
            MOV     DH,12
            MOV     DL,00
            INT     10H
            RET
Q20SET      ENDP

Q30DISP     PROC    NEAR              ;DX set on entry
            MOV     AH,40H            ;Request display
            MOV     BX,01             ;Handle
            MOV     CX,16             ;Length
            INT     21H
            RET
Q30DISP     ENDP
CODESG      ENDS

ZENDSG      SEGMENT                   ;Dummy (empty) segment
ZENDSG      ENDS
            END     A10MAIN
-------------------------------------------------------------

TITLE       A24SUB1  Called subprogram
DATASG      SEGMENT PARA 'Data2'
SUBMSG      DB      'Subprogram 1 reporting'
DATASG      ENDS

CODESG      SEGMENT PARA 'Code2'
A24SUB1     PROC    FAR
            ASSUME  CS:CODESG,DS:DATASG
            PUBLIC  A24SUB1
            PUSH    DS                ;Save caller's DS
            MOV     AX,DATASG         ;Initialize DS
            MOV     DS,AX
            MOV     AH,02H            ;Request set
            MOV     BH,00             ;  cursor
            MOV     DH,05
            MOV     DL,00
            INT     10H
            MOV     AH,40H            ;Request display
            MOV     BX,01             ;Handle
            MOV     CX,22             ;Length
            LEA     DX,SUBMSG         ;Message
            INT     21H
            POP     DS                ;Restore DS for caller
            RET
A24SUB1     ENDP
CODESG      ENDS
            END
-------------------------------------------------------------

TITLE       A24SUB2 Called overlay subprogram
DATASG      SEGMENT PARA 'Data'
SUBMSG      DB      'Subprogram 2 reporting'
```

شکل ب ۶-۲۴ فراخوانی یک زیر برنامه و روکش گذاری

```
DATASG    ENDS

CODESG    SEGMENT PARA 'Code'
A24SUB2   PROC    FAR
          ASSUME  CS:CODESG,DS:DATASG
          PUSH    DS                ;Save caller's DS
          MOV     AX,CS             ;Set address of first
          MOV     DS,AX             ;  segment in DS
          MOV     AH,02H            ;Request set
          MOV     BH,00             ;  cursor
          MOV     DH,10
          MOV     DL,00
          INT     10H
          MOV     AH,40H            ;Request display
          MOV     BX,01             ;Handle
          MOV     CX,22             ;Length
          LEA     DX,SUBMSG         ;Message
          INT     21H
          POP     DS                ;Restore caller's DS
          RET
A24SUB2   ENDP
CODESG    ENDS
          END
```

Start	Stop	Length	Name	Class
00000H	0007FH	00080H	STACKSG	Stack1
00080H	000C2H	00043H	DATASG	Data1
000D0H	0016DH	0009FH	CODESG	Code1
00170H	00170H	00000H	ZENDSG	
00170H	00185H	00016H	DATASG	Data2
00190H	001AFH	00020H	CODESG	Code2

شکل پ ۶-۲۴ فراخوانی یک زیر برنامه و روکش گذاری

A24SUB2 توسط A24CALLV فراخوانی می‌شود، اما جداگانه لینک می‌شود. سگمنت‌های آن چنین است.

```
DATASG    SEGMENT    PARA    'Data'

CODESG    SEGMENT    PARA    'Code'
```

نقشهٔ لینک A24SUB2 چنین می‌باشد (یک پیغام مبنی بر نبودن سگمنت پشته اعلان می‌شود) :

start	stop	length	name	Class
00000H	00015H	00016H	DATASG	Data
00020H	0003EH	0001FH	CODESG	Code

وقتی بارگذار برنامه، A24SUB1 + A24CALLV را به حافظه جهت اجرای منتقل می‌کند، A24SUB1، A24CALLV را بروش معمول فراخوانی و اجرا می‌کند. CALL نزدیک، IP را به نحو صحیحی ارزشدهی می‌کند، اما چون A24SUB1 سگمنت داده خودش را دارد، باید DS از A24CALLV را روی پشته گذاشته و آدرس DS خودش را برقرار سازد. A24SUB1 مکان نما را تنظیم می‌کند، یک پیغام را نمایش می‌دهد، DS را برمی‌دارد و به A24CALLV باز می‌گردد. برای روکش گذاری A24SUB2 روی A24SUB1 , A24CALLV باید فضای حافظه خود را کاهش دهد چون سیستم همه حافظه‌اش را به آن داده است. بالاترین سگمنت A24CALLV , ZENDSG است که خالی است، A24CALLV آدرس PSP (هنوز در ES است) خودش را از آدرس ZENDSG کم می‌کند. اختلاف 270H (27H پاراگراف)، بصورت اندازه PSP (100H) بعلاوه افست ZENDSG (170H) محاسبه می‌شود که توسط تابع 4AH به سیستم ارائه می‌شود.

سپس INT 21H تابع 48H حافظه را تخصیص می‌دهد تا فضایی برای قرار گرفتن A24SUB2 در بالای A24SUB1 (روکش شده) بطور دلخواه با 40H پاراگراف تنظیم شده، فراهم سازد. عملیات آدرس بارگذاری را در ثبات AX باز

می‌گرداند، که A24CALLV در PARABLK ذخیره می‌سازد. این اولین کلمه از بلوک پارامتر است که توسط تابع 4BH استفاده می‌شود.

تابع 4BH با کد ۰۳ در AL ، A24SUB2 را در حافظه قرار می‌دهد. به تعریفی که در سگمنت داده است توجه کنید : F:\A24SUB2.EXE , 0. تابع 4BH به CS و PARABLK مراجعه می‌کند اولین کلمه حاوی آدرس سگمنتی است که روکش گذاری می‌شود و دومین کلمه یک افست است که در این حالت صفر می‌باشد. یک دیاگرام ممکن است به روشن شدن این مراحل کمک نماید :

بعد از سرویس 48H
حافظه تخصیص می‌یابد

000	PSP
100	A24CALLV
270	A24SUB2

بعد از سرویس 4AH
حافظه کاهش می‌یابد

000	PSP
100	A24CALLV

بعد از بارگذاری اولیه

000	PSP
100	A24CALLV
270	A24SUB1

یک CALL دور به A24SUB2 به ارجاع تعریف شده بصورت IP:CS نیاز دارد، اما PARABLK بصورت CS:IP است. مقدار CS به دومین کلمه منتقل می‌شود و 20H در اولین کلمه برای IP ذخیره می‌شود، زیرا نقشه پیوند نشان می‌دهد که مقدار بصورت افست سگمنت کد A24SUB2 می‌باشد. دستور بعدی آدرس PARABLK را در BX قرار می‌دهد و A24SUB را فراخوانی می‌کند:

```
LEA   BX,PARABLK          ; آدرس PARABLK
CALL  DWORD PTR[BX]       ; فراخوانی A24SUB2
```

توجه کنید که A24CALLV به A24SUB2 توسط نامش در سگمنت کد خودش مراجعه نمی‌کند و بنابراین به جمله EXTRN که A24SUB2 را مشخص سازد، نیاز ندارد. چون A24SUB2 سگمنت داده خود را دارد، آن ابتدا DS را روی پشته می‌گذارد و آدرس خود را ارزشدهی می‌کند. اما A24SUB2 با A24CALLV پیوند زده نمی‌شود. بعنوان نتیجه، دستور MOV AX,DATASG , AX را فقط با افست آدرس DATASG[0]H,0، تنظیم می‌کند، نه با آدرس سگمنت خودش. ما می‌دانیم که CALL ، CS را با آدرس اولین سگمنت تنظیم می‌کند، که (بر طبق نقشه پیوند) آدرس سگمنت داده خواهد بود. کپی کردن CS در DS آدرس صحیح را در DS خواهد داد. اگر کد A24SUB2 و سگمنت داده به یک ترتیب مختلف مرتب می‌شوند، کد نویسی باید بر طبق آن بازنویسی شود.

A24SUB2 مکان نما را تنظیم می‌کند، یک پیغام را نمایش می‌دهد، DS را از روی پشته بر می‌دارد و به A24CALLV باز می‌گردد.

برنامه‌های مقیم

تعدادی از برنامه طوری طراحی می‌شوند که در حافظه مقیم می‌شوند و وقتی دیگر برنامه‌ها اجرا می‌شوند، می‌توانید با فشردن کلیدهای خاصی سرویس‌های آنها را فعال کنید. برنامه مقیم را قبل از فعال کردن دیگر برنامه‌های پردازشی معمول، بارگذاری کنید. آنها اغلب برنامه‌هایی از نوع COM. هستند و بعنوان برنامه‌های خاتمه یافته اما مقیم مانده (TSR) شناخته می‌شوند.

ساده‌ترین بخش نوشتن یک برنامه مقیم، مقیم کردن آن است، بجای خاتمه معمول، توسط INT 21H تابع 31H (نگه داری برنامه) از آن خارج شوید. عملیات به اندازه برنامه در ثبات DX نیاز دارد:

```
MOV   AH,31H              ; درخواست TSR
MOV   DX,prog-size        ; اندازه برنامه
INT   21H                 ; فراخوانی سرویس وقفه
```

وقتی روتین مقدار دهی، را اجرا می‌کنید، در جائیکه برنامه باید مقیم شود سیستم بلوک حافظه را رزرو می‌کند و

برنامه‌های بعدی در موقعیت بالاتری در حافظه قرار می‌گیرند.

بخش نه چندان ساده نوشتن یک برنامهٔ مقیم، فعال کردن آن پس از مقیم نمودن آن می‌باشد، چون یک برنامه داخلی سیستم مانند CLS ، COPY و DIR نیست. یک روش معمول، تغییر جدول بردار وقفه است که برنامه مقیم همه کلیدهای فشرده را متوقف می‌سازد، با کلیدهای خاص یا ترکیبی از آنها فعال می‌شود و دیگر کلیدهای فشرده شده را ارسال می‌دارد. با این روش، یک برنامه مقیم نوعاً اما نه الزاماً، شامل بخش‌های زیر خواهد بود :

۱) یک بخش که موقعیت‌های جدول بردار وقفه را مجدداً تعریف می‌کند.

۲) یک روال مقدار دهی که فقط بار اول اجرای برنامه، انجام می‌شود و به موارد زیر عمل خواهد کرد :

- جایگزینی آدرس در جدول بردار وقفه با آدرس خودش
- برقراری اندازه بخشی از برنامه که مقیم باقی می‌ماند و
- استفاده از یک وقفه که به سیستم می‌گوید اجرای برنامه جاری را خاتمه داده و به بخش خاصی از برنامه در حافظه دسترسی پیدا کند.

۳) یک روال که مقیم باقی می‌ماند و فعال است، برای مثال با برخی فعالیت‌ها مانند ورودی صفحه کلید خاص یا زمان‌سنج ساعت فعال خواهد شد.

در حقیقت روال مقدار دهی همه شرایط را تنظیم می‌کند تا برنامه مقیم کار کند و به خودش امکان پاک شدن را نیز بدهد. حال سازماندهی برنامه به شکل زیر ظاهر خواهد شد :

- سکون حافظه در دسترس
- بخش مقدار دهی اولیه برنامه (با برنامه دیگری پوشانده می‌شود)
- بخش مقیم برنامه (در حافظه باقی می‌ماند)
- COMMAND.COM
- MSDOS.SYS و IO.SYS
- جدول بردار وقفه

یک برنامه مقیم ممکن است از دو تابع INT 21H برای دستیابی جدول بردار وقفه استفاده نماید، چون اطمینانی نیست که بیشتر کامپیوترهای پیشرفته جدول وقفه در موقعیت 0000H داشته باشند.

INT 21H تابع 35H : گرفتن بردار وقفه

برای ارزیابی آدرس یک وقفه خاص از جدول بردار وقفه، AL را با شماره وقفه مورد نظر مقدار دهید:

```
MOV  AH,35H    ; درخواست بردار وقفه
MOV  AL,int#   ; شماره وقفه
INT  21H       ; فراخوانی سرویس وقفه
```

عملیات آدرس وقفه را در ES:BX بصورت افست : سگمنت باز می‌گرداند. برای حافظه معمولی، برای مثال، درخواست آدرس INT 09H در ES ، 00H و در BX ، 24H (36) باز می‌گرداند.

INT 21H تابع 25H: به تنظیم بردار وقفه

برای تنظیم یک وقفه جدید، شماره وقفه مورد نظر را در AL و آدرس جدید را در DX قرار دهید :

```
MOV  AH,25H      ; درخواست تنظیم بردار وقفه
MOV  AL,int#     ; شماره وقفه
IEA  DX,newaddr  ; آدرس جدید وقفه
INT  21H         ; فراخوانی سرویس وقفه
```

عملیات آدرس فعلی را با آدرس جدید وقفه جایگزین می‌کند. در حقیقت، وقتی وقفه خاصی رخ دهد، پردازش به برنامه مقیم شما پیوند زده می‌شود بجای اینکه به آدرس وقفه معمول پیوند زده شود.

```
TITLE    A24TSTNM (COM)  Resident program
BIODATA  SEGMENT AT 40H          ;BIOS data area
         ORG    17H
KBSTAT   DB     ?                ;Keyboard status byte
BIODATA  ENDS
;        ------------------------------------------------
CODESG   SEGMENT PARA
         ASSUME  CS:CODESG,DS:BIODATA
         ORG    100H
BEGIN:
         JMP    B10INIT          ;Jump to initialization
SAVINT9  DD     ?                ;INT 09H address
A10TEST:
         PUSH   AX               ;Save registers
         PUSH   CX
         PUSH   DS

         MOV    AX,BIODATA       ;Segment address of
         MOV    DS,AX            ;  BIOS data area
         MOV    AL,KBSTAT        ;Get keyboard flag
         TEST   AL,00100000B     ;NumLock state?
         JZ     A30EXIT          ;No, exit

         IN     AL,60H           ;Get keystroke from port
         CMP    AL,71            ;Scan code < 71?
         JL     A30EXIT          ;  yes, exit
         CMP    AL,83            ;Scan code > 83?
         JG     A30EXIT          ;  yes, exit
                                 ;Must be from numeric keypad
         MOV    AL,10110110B     ;Set frequency
         OUT    43H,AL
         MOV    AX,1000
         OUT    42H,AL
         MOV    AL,AH
         OUT    42H,AL
         IN     AL,61H           ;Turn on speaker
         MOV    AH,AL
         OR     AL,03
         OUT    61H,AL
         MOV    CX,9000          ;Set duration
A20PAUSE:
         LOOP   A20PAUSE
         MOV    AL,AH            ;Turn off speaker
         OUT    61H,AL
A30EXIT:
         POP    DS               ;Restore registers
         POP    CX
         POP    AX
         JMP    CS:SAVINT9       ;Resume INT 09H

;                Initialization routine:
B10INIT:         ;----------------------
         CLI                     ;Prevent further interrupts
         MOV    AH,35H           ;Get address of INT 09H
         MOV    AL,09H           ;  in ES:BX
         INT    21H
         MOV    WORD PTR SAVINT9,BX ;  and save it
         MOV    WORD PTR SAVINT9+2,ES
         MOV    AH,25H
         MOV    AL,09H           ;Set new address for INT 09H
         MOV    DX,OFFSET A10TEST ;  in A10TEST
         INT    21H
```

شکل الف ۷-۲۴ برنامهٔ مقیم

```
        MOV     AH,31H          ;Request stay resident
        MOV     DX,OFFSET B10INIT ;Set size of resident portion
        STI                     ;Restore interrupts
        INT     21H
CODESG  ENDS
        END     BEGIN
```

شکل ب ۷-۲۴ برنامهٔ مقیم

مثالی از یک برنامه مقیم

برنامه مقیم در شکل ۷-۲۴ با نام A24TSTNM اگر وقتی NUM LOCK روشن است از کلیدهای عددی سمت راست استفاده کنید بوق می‌زند. هدف این است که شما را آگاه کند که یک کلید عددی فشرده‌اید نه یک کلید جهت مکان نما. این برنامه باید INT 09H را در نیمه قطع کند (ورودی صفحه کلید) تا کلید فشرده شده را بررسی نماید.

نکات زیر راجع به برنامه‌های مقیم جالب توجه هستند :

BIODATA سگمنت داده BIOS را با شروع از [0]40 تعریف می‌کند - بویژه بایت پرچم صفحه کلید، که در اینجا KBSTAT نامیده می‌شود، وضعیت صفحه کلید را باز می‌گرداند. بیت ۵ که یک باشد یعنی NUMLOCK فعال است.

CODESG سگمنت کد A24TSTNM را آغاز می‌کند. اولین دستور اجرایی JMP B10INIT اجرا را به بعد از بخش مقیم به روال B10INIT نزدیک انتها منتقل می‌کند. این روتین ابتدا با استفاده از CLI از وقوع وقفه‌های دیگر که ممکن است هر لحظه اتفاق بیفتد، جلوگیری می‌کند. سپس از INT 21H تابع 35H جهت فراگرفتن در آدرس INT 09H در جدول بردار وقفه استفاده می‌کند. عملیات آدرس را در ES:BX بار می‌گرداند، که روتین B10INIT در INT9SAV ذخیره می‌سازد. بعد تابع 25H، آدرس برنامه خود را برای INT 09H در جدول وقفه ذخیره می‌سازد، A10TEST نقطه ورود به برنامه مقیم. در حقیقت، برنامه آدرس INT 09H را ذخیره می‌سازد و آن را با آدرس خودش جایگزین می‌کند. در مرحله بعدی اندازه بخش مقیم (همه کدها تا B10INT) را در DX قرار می‌دهد و با استفاده از INT 21H تابع 31H خارج می‌شود (خاتمه اما مقیم ماندن). کد B10INIT در انتهای برنامه توسط برنامه‌های دیگری که جهت اجرا بارگذاری می‌شوند، پوشیده شود.

A10TEST نام روال مقیم است که وقتی یک کاربر کلید را بفشارد فعال خواهد شد. سیستم اجرا را به آدرس INT 09H در جدول بردار وقفه منتقل می‌کند که با آدرس A10TEST تعویض شده است. چون وقفه ممکن است رخ دهد، برای مثال وقتی کاربر در DOS یا یک برنامه ویرایشگر یا پردازشگر کلمات است، A24TSTNM باید ثبات‌هایی که استفاده می‌کند، ذخیره نماید. برنامه به پرچم صفحه کلید دستیابی دارد تا تعیین کند که آیا NUMLOCK روشن است و آیا کلیدهای عددی سمت راست صفحه کلید فشرده است (شامل کد پیمایش بین ۷۱ و ۸۳). در اینصورت برنامه بلندگو را به صدا در می‌آورد. (استفاده از بلندگو در فصل ۲۱ توضیح داده شد، در بخش "تولید صوت") دستورات نهایی شامل بازیابی ثبات‌هایی است که بر روی پشته هستند - به ترتیب معکوس - و پرش به INT9SAL است، که حاوی آدرس INT 09H اصلی است. حال کنترل به وقفه باز می‌گردد.

مثال بعد، کمک می‌کند تا روال پاک شود. ابتدا، در اینجا توضیحی از عملیات مرسوم TSR بدون قطع وقفه است:

۱) کاربر یک کلید را می‌فشارد و صفحه کلید INT 09H را به BIOS می‌فرستد.

۲) BIOS از آدرس INT 09H در جدول بردار وقفه استفاده می‌کند تا در موقعیت روتین BIOS قرار گیرد.

۳) سپس کنترل به روتین BIOS منتقل می‌شود.

۴) روتین کاراکتر را می‌گیرد و اگر یک کاراکتر استاندارد باشد به بافر صفحه کلید ارائه می‌دهد.

روال بعد برای برنامه مقیم است :

۱) کاربر یک کلید را می‌فشارد و صفحه کلید INT 09H را به BIOS می‌فرستد.

۲) BIOS از آدرس INT 09H در جدول بردار وقفه استفاده می‌کند و در موقعیت روتین BIOS مربوطه قرار می‌گیرد.
۳) اما هم اکنون جدول حاوی آدرس A10TEST از برنامه مقیم است، که کنترل منتقل می‌شود.
۴) اگر NumLock فعال باشد و کاراکتر یک کلید عددی سمت راست صفحه کلید باشد، A10TEST بلندگو را به صدا در می‌آورد.
۵) A10TEST باپرش به آدرس INT 09H ذخیره شده اصلی خارج‌می‌شود، که کنترل به روتین BIOS منتقل می‌شود.
۶) روتین BIOS کاراکتر را می‌گیرد و اگر یک کاراکتر استاندارد باشد، آن را با بافر صفحه کلید ارائه می‌دهد. سعی کنید با استفاده از DEBUG حاصل اجرای این برنامه را بررسی کنید. از D 0:20 برای نمایش محتویات جدول وقفه در 20H(36) استفاده کنید، جائیکه آدرس وقفه INT 09H ذخیره شده است. اولین کلمه افست و دومین کلمه آدرس سگمنت است، که هر دو به ترتیب بایت معکوس هستند، برای مثال، اگر آدرس ذخیره شده 0701:EF05 باشد، برای دیدن محتویات آدرس ذخیره شده از فرمان D 107:05EF استفاده کنید. نمایش باید با 50511EB8 آغاز شود، که شروع کد ماشین A10TEST در برنامه مقیم است.

می‌توانید این برنامه را برای اهداف خودتان تغییر یا توسعه دهید. برنامه‌هایی که آدرس جدول INT 09H را جایگزین می‌کنند، استفاده همزمان از برنامه مقیم مانند برنامه فوق الذکر را اجازه نخواهد داد.

INT 21H تابع 34H : گرفتن آدرس پرچم مشغولیت DOS

اگر چه این وقفه توسط DOS بطور داخلی استفاده می‌شود، برخی TSRها نیز وقتی یک وقفه DOS را تقاضا می‌کنند تا بررسی کنند که آیا وقفهٔ دیگری هم اکنون فعال است و از آن استفاده می‌کنند. وقتی DOS مشغول است (هنگامیکه DOS فعال است نمی‌توانید DOS را وارد کنید) TSR باید منتظر شود تا DOS دیگر مشغول نباشد، که توسط پرچم مشغولیت inDOS مشخص می‌شود.

```
MOV   AH,34H                ; درخواست مشغولیت
INT   21H                   ; فراخوانی سرویس وقفه
CMP   ES:BYTE PTR [BX],0    ; بررسی اینکه پرچم صفر است
JE    . . .
```

این سرویس آدرس inDOS را در ES:BX باز می‌گرداند. پرچم حاوی تعداد توابع DOS فعال جاری است، که صفر یعنی هیچ یک فعال نیستند. فقط وقتی ممکن است DOS را وارد کنید که inDOS صفر باشد.

نکات کلیدی

- رکورد راه انداز در تراک 0، سکتور 1 هر دیسکی است که شما از فرمان FORMAT/S برای فرمت کردن آن استفاده کرده‌اید، وقتی سیستم را روشن کنید بطور اتوماتیک رکورد راه انداز را از دیسک در حافظه قرار می‌دهد. سپس رکورد راه انداز IO.SYS را از دیسک روی حافظه قرار می‌دهد.
- IO.SYS یک رابطه سطح پایین برای روتین‌های BIOS در ROM است. در ابتدا، IO.SYS وضعیت همه ابزارها و تجهیزات متصل به کامپیوتر را تعیین می‌کند و آدرسهای جدول بردار وقفه را برای وقفه‌ها تا 20H تنظیم می‌کند. IO.SYS همچنین I/O را بین حافظه و ابزارهای خارجی دستکاری می‌کند.
- MSDOS.SYS یک رابط سطح بالای برنامه‌هاست که پس از IO.SYS در حافظه قرار می‌گیرد. عملیات آن شامل، تنظیم آدرسهای جدول‌بردار وقفه برای وقفه‌های 20H تا 3FH، مدیریت شاخه‌ها و فایل‌های روی دیسک، دستکاری بلوکه کردن و بلوکه نکردن رکوردهای دیسک و دستکاری توابع INT 21H است.
- COMMAND.COM فرامین مختلف سیستم را دستکاری می‌کند و فایلهای درخواستی EXE. و COM. و

BAT. را اجرا می‌کند. این فایل شامل یک بخش مقیم کوچک، یک بخش آغازین، یک بخش فانی است.

- COMMAND.COM مسئول بارگذاری برنامه‌های اجرایی از دیسک روی حافظه است.
- ماجول EXE. که لینکر ایجاد می‌کند شامل یک رکورد عنوان حاوی اطلاعات کنترل و جایگیری و ماجول بارگذاری واقعی است.
- با بارگذاری یک برنامه COM. یا EXE. سیستم بلوک‌های حافظه را برای محیط برنامه و سگمنت برنامه تنظیم می‌کند. هر بلوک حافظه فوق‌الذکر یک میدان عنوان ۱۶ بایتی با شروع از مرز یک پاراگراف دارد. همچنین برنامه بارگذار یک PSP در موقعیت 00H سگمنت برنامه ایجاد می‌کند و برنامه را در 100H قرار می‌دهد.
- در بارگذاری یک برنامه COM. بارگذار ثبات‌های سگمنت را با آدرس PSP تنظیم می‌کند، اشاره‌گر پشته را با انتهای سگمنت تنظیم می‌کند و یک کلمه صفر روی پشته می‌گذارد و اشاره‌گر دستور را با 100H مقدار می‌دهد (اندازهٔ PSP). سپس کنترل به آدرس تولید شده توسط CS:IP یعنی اولین موقعیت بلافاصله پس از PSP می‌رود.
- در بارگذاری یک برنامه EXE. بارگذار رکورد عنوان را خوانده و در حافظه قرار می‌دهد، اندازه ماجول اجرایی را محاسبه می‌کند و ماجول را خوانده و در حافظه در شروع سگمنت قرار می‌دهد. سپس مقدار هر عنصر جدول تخصیص دهی را با مقدار شروع سگمنت جمع می‌کند. بارگذار DS و ES را با آدرس سگمنت PSP ، SS را با آدرس PSP بعلاوه 100H بعلاوه مقدار افست SS ، SP را با اندازه پشته و CS را با افستی که در 14H است تنظیم می‌کند. جفت ثبات CS:IP آدرس شروع سگمنت کد در برنامه اجرایی را فراهم می‌سازد.
- فیلد مفید در PSP شامل، ناحیه پارامتر 1 در 5CH ، ناحیه پارامتر ۲ در 6CH و ناحیه انتقال دیسک پیش‌فرض در 80H است.
- یک برنامه مقیم را قبل از فعال نمودن دیگر برنامه‌های پردازشی بارگذاری کنید. توسط INT 21H تابع 31H که به اندازه زیر برنامه در DX نیاز دارد، خارج شوید.

پرسش‌ها

۲۴-۱. (الف) موقعیت رکورد راه انداز در کجاست؟ (ب) هدف از آن چیست؟

۲۴-۲. هدف از IO.SYS را شرح دهید.

۲۴-۳. هدف از MSDOS.SYS را شرح دهید.

۲۴-۴. بخش‌های زیر متعلق به COMMAND.COM در کدام قسمت حافظه قرار می‌گیرد و هدف از آن چیست؟ (الف) مقیم (ب) ناپایدار.

۲۴-۵. (الف) سیستم، پیشوند سگمنت برنامه را کجا ذخیره می‌سازد؟ (ب) چه اندازه است؟

۲۴-۶. یک کاربر جهت اجرای برنامه FORGE دستور FORGE E:SLIM.ASM را وارد کرده است. محتویات PSP برنامه در موقعیت‌های زیر را نشان دهید (الف) 5CH ، ناحیه پارامتر 1(FCB#1) (ب) 80H و DTA پیش‌فرض.

۲۴-۷. برنامهٔ شما باید تعیین کند که چه فرمانهایی (PATH) برای محیط آن تنظیم شده است. توضیح دهید برنامه ممکن است در کجا محیط خودش را بیابد. (درخواست برای محیط برنامه، نه محیط کارفرمای DOS)

۲۴-۸. یک برنامه COM. برای اجرا با PSP خودش در موقعیت 2CD4[0]H قرار گرفته است. بارگذار برنامه در هر یک از ثبات‌های زیر چه آدرسی ذخیره می‌سازد (از نکته بایت، معکوس صرف نظر کنید) : (الف) CS ، (ب) DS ، (ج) ES، (د) SS.

۲۴-۹. یک نقشه پیوند برای یک برنامه EXE. بصورت زیر نشان داده شده است:

Start	Stop	Length	Name	Class
00000H	0003FH	00040H	STACK	STACK
00040H	0006BH	0002CH	CODESG	CODE
00070H	0009CH	0002DH	DATASG	DATA

بارگذار، برنامه و PSP را در موقعیت 1B38[0]H قرار می‌دهد. محاسبات نشان داده شده کجا درست می‌باشد، محتویات هرثبات در زمان بارگذاری را تعیین‌می‌کند (از ترتیب بایت معکوس صرف‌نظر کنید). (لف) SS (ب) SP ، (ج) CS، (د) DS ، (ه) ES

۱۰-۲۴. یک میدان عنوان موقعیت 10A4[0] شروع می‌شود و حاوی مقادیر زیر است : ...4D C00E 0A00 (الف) معنی 4D (M) برای سیستم چیست؟ (ب) چگونه محتویات فرق می‌کرد اگر این آخرین بلوک حافظه بود؟ (ج) موقعیت حافظه میدان عنوان بعدی چیست؟ محاسبات خود را نشان دهید؟

۱۱-۲۴. برنامه‌های مقیم عموماً ورودی صفحه کلید را قطع می‌کنند. دقیقاً کجا و چرا این آدرس قطع می‌شود؟

۱۲-۲۴. با چه روشهای علمی برنامه‌های مقیم کدنویسی می‌شود تا برنامه متفاوت با برنامه‌های معمولی خاتمه یابد؟

فصل ۲۵

نواحی داده BIOS و وقفه‌های برنامه

هدف : توصیف نواحی داده BIOS و سرویس‌های وقفه برای BIOS و DOS

مقدمه

BIOS شامل مجموعهٔ گرانقدری از روتین‌های ورودی / خروجی و جداولی است که مبنی بر وضعیت ابزارهای سیستم می‌باشد. DOS و برنامه‌های کاربردی می‌توانند روتینهای BIOS را برای ارتباط با ابزارهایی که با سیستم در ارتباطند، درخواست نمایند. ارتباط با BIOS از طریق وقفه‌های نرم‌افزاری انجام می‌شود. این فصل، نواحی داده‌ای (یا جداولی) که BIOS پشتیبانی می‌کند، روال وقفه، و وقفه‌های BIOS از 00H تا 1BH و وقفه‌های DOS از 20H تا 33H را بررسی می‌کند.

فرآیند راه اندازی

در کامپیوتر شخصی ROM در موقعیت FFFF0H ساکن شده است. روشن کردن کامپیوتر سبب یک راه‌اندازی سرد می‌شود. پردازشگر حالت تنظیم مجدد را وارد می‌کند، همه موقعیت‌های حافظه با صفر تنظیم می‌شوند، تست توازن حافظه انجام می‌شود و ثبات CS با FFFF:0 [0]H و ثبات IP با صفر مقدار می‌گیرند. بنابراین اولین دستوری که اجرا می‌شود در موقعیت FFFF:0 یا به عبارت بهتر در ابتدای BIOS قرار دارد. BIOS نیز مقدار 1234H را در 40[0]:72H می‌گذارد تا علامتی برای Ctrl + Alt + Delهای بعدی باشد بدین ترتیب، خودآزمایی حالت روشن شدن کامپیوتر دیگر انجام نمی‌شود (به این نوع راه‌اندازی، راه‌اندازی گرم گفته می‌شود.)

BIOS درگاه‌های مختلف را برای تشخیص بررسی می‌کند و ابزارهایی که متصل هستند را با استفاده INT 11H (تعیین تجهیزات) و INT 12H (تعیین اندازه حافظه)، تعیین کند. سپس با شروع از موقعیت 0 حافظه معمول، BIOS، جدول برداری وقفه را برقرار می‌سازد که حاوی آدرس روتین‌های وقفه است.

حال BIOS تعیین می‌کند که آیا دیسک حاوی فایل‌های سیستم آماده است، در اینصورت، INT 19H را برای دستیابی اولین سکتور دیسک شامل بارگذار راه انداز اجرا می‌کند. این برنامه یک سیستم عامل موقتی است بطوریکه روتین BIOS پس از بارگذاری آن در حافظه، کنترل را به آن منتقل می‌کند. بارگذار راه‌انداز فقط یک وظیفه دارد : بارگذاری اولین بخش از سیستم عامل واقعی در حافظه. سپس فایل‌های سیستم IO.SYS و MSDOS.SYS و COMMAND.COM از دیسک روی حافظه قرار می‌گیرند.

ناحیه داده BIOS

BIOS، ناحیه داده ۲۵۶ بایتی خودش (100H) را در حافظه پایینی با شروع از آدرس سگمنت 40[0]H با فیلدهایی

شامل داده‌ها بترتیب بایت معکوس، دستکاری می‌کند. یک تجربه با ارزش استفاده از DEBUG برای بررسی این فیلدها است که به ترتیب افست لیست شده‌اند.

ناحیه داده درگاه سریال

- 00H-07H چهار کلمه، آدرسهای چهار درگاه سریال COM1-COM4.

ناحیه داده درگاه موازی

- 08H-0FH چهار کلمه، آدرسهای چهار درگاه موازی LPT1-LPT4.

ناحیه داده تجهیزات سیستم

- 10H-11H وضعیت تجهیزات، یک تعیین ابتدایی وضعیت ابزارهای نصب شده. می‌توانید INT 11H را صادر کنید که موارد زیر را در AX باز می‌گرداند.

بیت	ابزار
15,14	تعداد درگاه‌های موازی متصل
11,9	تعداد تطبیق دهنده‌های موازی RS235
7,6	تعداد ابزارهای دیسکت، که بیت 00 = 1، 01 = 2، 10 = 3 و 11 = 4
5,4	حالت اولیه ویدئو. مقادیر بیت چنین هستند ۰۰ = بلا استفاده، 01 = 20×40 رنگی، 10 = 25×80 رنگی، 11 = 25×80 تک رنگ.
2	ابزار نشانه‌ای (موس) که ۱ = نصب شده.
1	1 = کمک پردازشگر ریاضی وجود دارد.
0	1 = درایو دیسکت وجود دارد.

ناحیه داده متفرقه

- 12H پرچم‌های بررسی کارخانه سازنده

ناحیه داده اندازه حافظه

- 13H-14H میزان حافظه روی برد سیستم به کیلو بایت
- 15H-16H میزان حافظه توسعه یافته به کیلو بایت

اولین ناحیه داده‌های صفحه کلید

بیت	فعالیت	بیت	فعالیت
۷	Insert فعال	۳	Alt فشرده شده
۶	Capslock فعال	۲	Ctrl فشرده شده
۵	NumLock فعال	۱	Shift چپ فشرده شده
۴	Scroll Lock فعال	۰	Shit راست فشرده شده

فعال یعنی کلید فشرده شده و تنظیم می‌باشد. فشرده شده یعنی هنگامی که BIOS موقعیت جاری را ذخیره می‌کند و پایین نگه داشته شده است.

- 18H دومین بایت وضعیت شیفت جاری.

بیت	فعالیت	بیت	فعالیت
۷	Insert فشرده شده	۳	Ctrl/Numlock فشرده شده
۶	CapsLock فشرده شده	۲	SysReq فشرده شده
۵	NumLock فشرده شده	۱	LeftAlt فشرده شده
۴	Scroll Lock فشرده شده	۰	LeftCtrl فشرده شده

- 19H تناوب ورودی صفحه کلید برای کاراکترهای ASCII.
- 1AH-1BH اشاره‌گر به شروع بافر صفحه کلید.
- 1CH-1DH اشاره‌گر به دنباله بافر صفحه کلید.
- 1EH-3DH بافر صفحه کلید (۳۲ بایت)

ناحیه داده درایو دیسکت

- 3EH وضعیت چرخش دیسک. شماره بیت ۰ به درایو A، ۱ به درایو B ، ۲ به درایو C و ۳ به درایو D اشاره دارد. یک مقدار بیت صفر یعنی که چرخش بعد در محل سیلندر 0 به منظور تنظیم درایو می‌باشد.
- 3FH وضعیت موتور دیسک. اگر بیت ۷ = ۱، یک عملیات نوشتن در جریان است. شمارهٔ بیت صفر به درایو A، ۱ به B ، ۲ به C و ۳ به D اشاره دارد و مقدار بیت ۰ یعنی موتور روشن است.
- 40H شمارش برای اتمام وقت موتور، تا زمانیکه موتور خاموش شود.
- 41H وضعیت دیسک مبنی بر خطا در آخرین عملیات دیسک درایو :

00H	خطایی نیست	09H	تلاش برای درست کردن DMA در وسط مرز 64K
01H	پارامتر درایو نامعتبر	0CH	نوع واسطه پیدا نشد
02H	آدرس مشخص شده پیدا نشد	10H	خطای CRC زمان خواندن
03H	خطای محافظت در برابر نوشتن	20H	خطای کنترلر
04H	سکتور پیدا نشد	40H	چرخش موفق نبود
06H	نامعلوم	80H	درایو آماده نیست
08H	اجرای DMA		

- 42H-48H وضعیت کنترلر دیسک درایو

اولین ناحیه داده‌های ویدئو

- 49H حالت هم اکنون ویدئو، با یک بیت مشخص می‌شود.

بیت	فعالیت	بیت	فعالیت
۷	تک رنگ	۳	۲۵ × ۸۰ رنگی
۶	۲۰۰ × ۶۴۰ تک رنگ	۲	۲۵ × ۸۰ تک رنگ
۵	۲۰۰ × ۳۸۰ تک رنگ	۱	۲۵ × ۴۰ رنگی
۴	۲۰۰ × ۳۲۰ رنگی	۰	۲۵ × ۴۰ تک رنگ

- 4AH-4BH تعداد ستون‌های صفحه
- 4CH-4DH اندازه بافر صفحه ویدئو
- 4EH-4FH افست شروع بافر ویدئو

- 50H-5FH ۸ کلمه برای موقعیت شروع جاری برای هر ۸ صفحه، با شماره‌های ۰ - ۷.
- 60H-61H خط شروع و پایان مکان نما
- 62H صفحه نمایش فعال جاری
- 63H-64H آدرس درگاه نمایش فعال، که تک رنگ B4H 03 و رنگی 03D4H خواهد بود.
- 65H تنظیم جاری از ثبات حالت ویدئو
- 66H جعبه رنگ جاری

ناحیه داده سیستم

- 67H-68H زمان صرف شده برای خواندن داده.
- 69H-6AH ثبات CRC.
- 6BH آخرین مقدار ورودی.
- 6CH-6DH نیمه پایینی زمان سنج.
- 6EH-6FH نیمه بالایی زمان سنج.
- 70H سرریز زمان سنج (اگر زمان سنج از نیمه شب گذشته باشد ۱ است).
- 71H کلیدهای Ctrl + Break بیت ۷ تا ۱ را تنظیم می‌کنند.
- 72H-73H پرچم تنظیم حافظه. اگر محتویات آن 1234H باشد، کلیدهای Ctrl + Alt + Del سبب راه اندازی گرم می‌شوند.

ناحیه داده دیسک سخت

- 74H وضعیت آخرین عملیات دیسک سخت (جزئیات در فصل ۱۹ است).
- 75H تعداد دیسکت‌های سخت متصل شده.

ناحیه داده‌های مربوط به اوقات جانبی

- 78H-7BH داده‌های درگاه‌های موازی (LPT1-LPT4)
- 7CH-7FH داده‌های درگاه‌های سریال (COM1-COM4)

دومین ناحیه داده‌های صفحه کلید

- 80H-81H آدرس افست ابتدای بافر صفحه کلید
- 82H-83H آدرس افست انتهای بافر صفحه کلید

دومین ناحیه داده‌های ویدئو

- 84H تعداد سطرهای صفحه (منهای ۱)
- 85H ارتفاع کاراکتر در خطوط پیمایش
- 86H-8AH اطلاعات متفرقه ویدئو

ناحیه داده دیسکت / دیسک سخت

- 8BH-95H کنترلر و وضعیت خطا

سومین ناحیه داده‌های صفحه کلید

- 96H وضعیت حالت صفحه کلید و پرچم‌های نوع

بیت	فعالیت	بیت	فعالیت
۷	خواندن ID در حال انجام	۳	Alt سمت راست فشرده شده
۶	آخرین کد ACK شد.(۱)	۲	Ctrl سمت راست فشرده شده
۵	فعال‌سازی NumLock درصورت‌وجود KBX,ID	۱	آخرین کد پیمایش E0 است
۴	صفحه کلید توسعه یافته نصب شده است	۰	آخرین کد پیمایش E1 است

- 97H پرچم‌های LED صفحه کلید (بیت ScrollLock=0، Numlock=1، CapsLock=2)

ناحیه داده ساعت زمان حقیقی

- 98H-A7H وضعیت پرچم‌های انتظار

ذخیره اشاره‌گر ناحیه داده

- A8H-ABH اشاره‌گر به جداول BIOS مختلف

دومین ناحیه داده‌های متفرقه

- ACH-FFH توسط سیستم برای مصارف داخلی رزرو شده است.

سرویس‌های وقفه

یک عملیات وقفه، اجرای یک برنامه را مطلق گذاشته، تا سیستم بتواند عمل خاصی انجام دهد. تا هم‌اکنون تعدادی وقفه برای نمایش ویدئو، ورودی خروجی دیسک، چاپ کردن و برنامه‌های مقیم استفاده کرده‌اید. روتین وقفه اجرا می‌شود و بطور عادی کنترل به روال متوقف شده باز می‌گردد. بدین ترتیب اجرا از سرگرفته می‌شود. BIOS وقفه‌های 1FH تا INT 00H را دستکاری می‌کند، در حالیکه DOS وقفه‌های INT20H-3FH را دستکاری می‌کند.

جدول بردار وقفه

وقتی کامپیوتر روشن می‌شود، BIOS و DOS در موقعیت 000H-3FFH حافظه معمول یک جدول بردار وقفه برقرار می‌سازند. این جدول برای ۳۵۶ (100H) وقفه فراهم شده است، هر کدام یک آدرس سگمنت : افست ۴ بایتی مرتبط به صورت IP:CS می‌باشند. عملوند یک دستور وقفه مانند INT 05H ، نوع درخواست را مشخص می‌سازد. چون ۲۵۶ ورودی وجود دارد و هر یک ۴ بایت طول دارند، جدول ۱۰۲۴ بایت حافظه را اشغال می‌سازد یعنی از آدرس 00H تا 3FFH هر آدرس جدول با یک روتین BIOS یا DOS برای یک نوع وقفه خاص مرتبط است. بنابراین بایت‌های ۰-۳ حاوی آدرس وقفه ۰ است و بایت‌های ۴-۷ برای وقفه یک و الی آخر.

INT 00H	INT 01H	INT 02H	INT 03H	INT 04H	INT 05H	INT 06H	...
IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	...
00H	04H	08H	0CH	10H	14H	18H	...

۱. تشخیص داده شد. مترجم

اجزای یک وقفه

یک وقفه محتویات ثبات پرچم، CS و IP را روی پشته می‌گذارد. برای مثال، آدرس جدول INT 05H (که ناحیه نمایش ویدئو را وقتی کاربر کلیدهای Ctrl + Prtsc را بفشارد، چاپ می‌کند) 0014H است.
(05H × 4 = 14H). عملیات ۴ بایت آدرس را از موقعیت 0014H اقتباس نموده و ۲ بایت را در IP و ۲ بایت را در CS ذخیره می‌سازد. آدرس CS:IP به شروع یک روتین در ناحیه BIOS اشاره می‌کند، که هم اکنون اجرا می‌شود. وقفه بر طبق یک دستور IRET (بازگشت وقفه) باز می‌گردد، که IP ، CS و ثبات پرچم را از روی پشته بر می‌دارد و کنترل به دستور بعد از INT باز می‌گردد.

وقفه‌های خارجی و داخلی

یک ***وقفه خارجی*** توسط یک ابزار که خارج از پردازشگر است، ایجاد می‌شود. دو خطی که می‌توانند وقفه‌های خارجی را علامت دهند، خط وقفه غیر قابل پوشش (NMI) و خط درخواست وقفه (INTR) می‌باشند. خط NMT ، خطاهای توازن حافظه و ورودی / خروجی را گزارش می‌دهد. پردازشگر همیشه با این وقفه فعال می‌شود، حتی اگر CLI را برای پاک کردن پرچم وقفه صادر کرده باشید تا وقفه‌های خارجی غیر فعال شوند. خط INTR درخواست‌های ابزارهای خارجی، مانند وقفه‌های 05H تا 0FH، برای زمان‌سنج، صفحه کلید، درگاه سریال، تعمیر دیسک، دیسک درایو و درگاه‌های موازی را گزارش می‌دهد.

یک ***وقفه داخلی*** وقتی رخ می‌دهد که حاصل اجرای یک دستور INT باشد یا یک عملیات تقسیم که سبب سرریز می‌شود، اجرا در حالت تک مرحله است، یا درخواستی برای یک وقفه خارجی مانند I/O دیسک باشد. برنامه‌ها عموماً از وقفه‌های داخلی استفاده می‌کنند که غیر قابل پوشش هستند و روال‌های DOS و BIOS را دستیابی می‌کنند.

وقفه‌های BIOS

این بخش وقفه‌های BIOS را از 00H تا 1BH در بر دارد. دیگر عملیاتی که ذکر شده است می‌تواند فقط توسط BIOS اجرا شود.

INT 00H تقسیم بر صفر. تلاش برای تقسیم بر صفر باعث فعال شدن این وقفه می‌شود، یک پیغام نمایش می‌دهد و همیشه سیستم متوقف می‌شود. توسعه دهندگان برنامه، با این خطا آشنا هستند چون حذف یک ثبات سگمنت تصادفاً ممکن است این وقفه را فعال کند.

INT 01H تک مرحله‌ای. توسط DEBUG و دیگر اشکال زداها جهت فعال ساختن مرحله به مرحله اجرای برنامه استفاده می‌شود.

INT 02H وقفه غیر قابل پوشش. برای شرایط سخت‌افزاری مهم، مانند خطای توازن، که همیشه فعال است استفاده می‌شود. بنابراین این یک برنامه با صادر کردن دستور CLI (پاک کردن وقفه) تاثیری بر این شرایط ندارد.

INT 03H نقطه شکست. توسط برنامه‌های اشکال زدا برای توقف اجرا استفاده می‌شود. فرمان‌های G و P از DEBUG این وقفه را در نقطه توقف مناسب در برنامه تنظیم می‌کنند، DUBUG حالت تک مرحله‌ای را انجام نمی‌دهد و به برنامه امکان می‌دهد، تا INT 03H بطور طبیعی اجرا شود و بعد از آن DUBUG مجدداً حالت تک مرحله‌ای را تنظیم می‌کند.

INT 04H سرریز. ممکن است، با انجام یک عملیات محاسباتی به وجود آید. با این حال، غالباً سرریز پدیدهٔ خاصی را به وجود نمی‌آورد.

INT 05H چاپ صفحه نمایش. سبب می‌شود تا محتویات ناحیه نمایش ویدئو چاپ شود. صدور

INT 05H وقفه داخلی را فعال می‌کند و فشردن <Prtscr> + <Ctrl> آن را بطور خارجی فعال می‌کند. عملیات وقفه‌ها را فعال می‌کند و موقعیت مکان نما را ذخیره می‌کند. هیچ ثباتی تحت تاثیر قرار نمی‌گیرد. آدرس 50:00 در ناحیه داده BIOS حاوی وضعیت عملیات است.

INT 08H زمان سنج سیستم. یک وقفه سخت‌افزاری که زمان سیستم و تاریخ را بروز رسانی می‌کند (اگر لازم باشد) یک تراشه زمان سنج قابل برنامه ریزی هر ۵۴۹۲۴۲ میلی ثانیه که ۱۸۰۲ مرتبه آن یک ثانیه می‌شود، یک وقفه ایجاد می‌کند.

INT 09H وقفه صفحه کلید. با فشردن یا رها کردن یک کلید روی صفحه کلید ایجاد می‌شود، در فصل ۱۱ با جزئیات توصیف شد.

INT 0CH , INT 0BH کنترل ابزار سریال. کنترل درگاه COM1. و COM2. (بترتیب).

INT 0FH.INT 0DH کنترل ابزار موازی. کنترل درگاه LPT2 و LPT1 (بترتیب).

INT 0EH کنترل دیسکت. علائم فعالیت دیسکت، مانند تکمیل عملیات I/O.

INT 10H نمایش ویدئو. پذیرش تعدادی تابع در AH برای حالت صفحه، تنظیم مکان نما، حرکت طوماری صفحه و نمایش دادن، با جزئیات در فصل ۱۰ توضیح داده شده است.

INT 11H تعیین تجهیزات. تعیین ابزارهای انتخابی روی سیستم و بازگشت محتویات موقعیت 40:10H از BIOS در AX. (هنگام روشن شدن، سیستم این عملیات را انجام می‌دهد و AX را در موقعیت 40:10H ذخیره می‌سازد، بخش قبلی با عنوان "ناحیه داده BIOS" را ببینید).

INT 12H تعیین اندازه حافظه. در AX اندازه، حافظه پایه را به کیلوبایت باز می‌گرداند، برای مثال، 640K حافظه بصورت 0280H است و همان مقداری است که در هنگام روشن شدن تعیین گردیده است.

INT 13H ورودی / خروجی دیسک. تعدادی تابع در AH برای وضعیت دیسک، خواندن سکتورها، نوشتن سکتورها، تعیین فرمت و گرفتن تشخیص خطا می‌پذیرد، در فصل ۱۹ کاملاً توضیح داده شد.

INT 14H ارتباط ورودی / خروجی. جریان بایت‌های I/O (یک بیت در هر بار) به درگاه ارتباط R5232 را فراهم می‌سازد. DX باید حاوی تعداد تطبیق دهنده‌های RS232 باشد (۰ تا ۳ برای COM1 تا COM4). تعدادی تابع در AH برقرار می‌شوند.

تابع 00H. مقدار دهی درگاه ارتباطی. بر طبق شماره بیت، پارامترهای زیر را در AL تنظیم می‌کند:

BAUDRATE	PARITY	STOP BIT	WORD LENGTH
7 - 5	4 - 3	2	1 - 0
000 = 110	00 = none	0 = 1	10 = 7 bits
001 = 150	01 = odd	1 = 2	11 = 8 bits
010 = 300	10 = none		
011 = 600	11 = even		
100 = 1200			
101 = 2400			
110 = 4800			
111 = 9600			

عملیات، وضعیت درگاه ارتباطی را در AX باز می‌گرداند (تابع 03H را برای جزئیات بیشتر ببینید). در اینجا مثالی را خواهید دید که COM1 را با bood 1200، بدون توازن، یک توقف و ۸ بیت طول داده تنظیم می‌کند.

```
MOV  AH,00H          ; درخواست مقدار دهی درگاه
MOV  AL و 10000011B  ; پارامترها
MOV  DX و 00         ; درگاه سریال COM1
INT  14H             ; فراخوانی سرویس وقفه
```

تابع 01H فرستادن کاراکتر. در AL کاراکتری که روتین می‌فرستد و در DX شماره درگاه را قرار دهید. در بازگشت، عملیات وضعیت درگاه را در AH تنظیم می‌کند. (تابع 03H را ببینید). اگر عملیات قادر به ارسال بایت نباشد، بیت ۷ از AH را نیز تنظیم می‌کند، اگر چه هدف معمول از این بیت، گزارش خطای اتمام وقت است. قبل از استفاده از این سرویس حتماً تابع 00H را تنظیم کند.

تابع 02H دریافت کاراکتر. شماره درگاه را در DX قرار دهید، عملیات یک کاراکتر را از خط ارتباطی به AL می‌پذیرد. همچنین AH را با وضعیت درگاه (تابع 03 را ببینید) برای بیت‌های خطا ۷، ۴، ۳، ۲، ۱، تنظیم می‌کند. بنابراین یک مقدار غیرصفر در AX مبنی بریک خطای ورودی است، قبل از استفاده از این سرویس حتماً تابع 00H را اجرا کنید.

تابع 03H بازگشت وضعیت درگاه ارتباطی. شماره درگاه را در DX قرار دهید، عملیات وضعیت خط را در AH و وضعیت مودم را در AL باز می‌گرداند :

(وضعیت مودم)	AL	(وضعیت خط)	AH
کشف علائم دریافتی خط	۷	اتمام وقت	۷
شاخص زنگ	۶	انتقال ثبات Shift خالی	۶
تنظیم داده آماده است	۵	انتقال ثبات hold خالی	۵
خالی شده برای ارسال	۴	کشف قطع برنامه	۴
تشخیص علامت خط دریافتی Delta	۳	خطای تنظیم کردن	۳
پیمایش خط برای یافتن شاخص زنگ	۲	خطای توازن	۲
آمادگی مجموعه داده‌های Delta	۱	خطای overrun	۱
پاک کردن Delta اجرای انجام عمل ارسال	۰	داده آماده است	۰

دیگر توابع INT 14H تابع 04H (مقدار دهی توسعه یافته) و 05H است (کنترل درگاه ارتباط توسعه یافته).

INT 15H سرویس‌های سیستم. این وقفه دارای توابع بسیاری است. از جمله این توابع که عبارتند از :

تعیین اندازه حافظه توسعه یافته	88H	خودآزمایی هنگام روشن‌شدن	21H
سوئیچ پردازشگر برای حالت محافظت شده	89H	خواندن وضعیت سیستم	43H
رابط موس	C2H	پشتیبانی Joystick	84H

برای مثال، کد تابعی 88H در AH ، INT 15H میزان حافظه توسعه یافته به کیلو بایت را در AX باز می‌گرداند. (برای مثال، 0580H یعنی 1408K بایت). اجرای توابع بدون بازگرداندن وقفه‌ها به حالت اولیه، خاتمه می‌یابد. از آنها باید به شکل زیر استفاده شود.

```
MOV  AH,88H ; درخواست حافظه توسعه یافته
INT  15H    ; درخواست حافظه از BIOS
STI         ; بازیابی وقفه‌ها
```

INT 16H ورودی صفحه کلید. پذیرش تعدادی تابع در AH برای ورودی پایه صفحه کلید در فصل ۱۰ توضیح داده شده است.

INT 17H خروجی چاپگر. فراهم ساختن تعدادی تابع برای چاپ کردن بر طبق BIOS در فصل ۱۰ توضیح داده شده است.

INT 19H بارگذار راه انداز. اگر یک دیسک با برنامه‌های سیستم DOS در دسترس است، تراک صفر، سکتور ۱ را می‌خواند، و در موقعیت راه‌اندازی در حافظه در 7C00H قرار می‌دهد و کنترل را به این موقعیت منتقل می‌کند اگر راه انداز دیسک وجود ندارد، به نقطه ورود ROM BASIC بر طبق INT 18H منتقل می‌شود. این وقفه ممکن است بعنوان یک وقفه نرم‌افزاری استفاده شود، این وقفه صفحه را پاک نمی‌کند یا داده در ROM BIOS را تعداد دهی می‌کند.

INT 1AH خواندن و تنظیم زمان. بر طبق کد تابع AH زمان روز را می‌خواند یا تنظیم می‌کند.

- **00H = خواندن ساعت زمان سنج سیستم.** قسمت بالایی شمارش را در CX و قسمت پائینی را در DX باز می‌گرداند. اگر زمان ۲۴ ساعت از آخرین خواندن گذشته باشد، عملیات AL را با یک مقدار غیر صفر تنظیم می‌کند.
- **01H = تنظیم ساعت زمان سنج سیستم.** قسمت بالایی شمارش را در CX و قسمت پائینی را در DX باز می‌گرداند.
- **02H-07H.** این تابع زمان و تاریخ سرویس‌های ساعت زمان حقیقی را دستکاری می‌کند. برای تعیین مدت زمان اجرای یک روتین، می‌توانید ساعت را با صفر تنظیم کنید و سپس در انتهای پردازش آن را بخوانید.

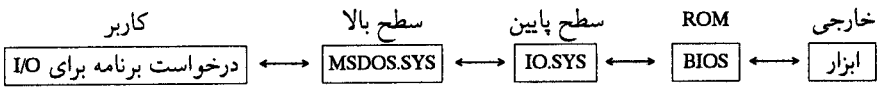
INT 1BH گرفتن کنترل روی قطع صفحه کلید. وقتی کلیدهای Ctrl + Break فشرده می‌شود، سبب می‌شود ROM BIOS کنترل را به آدرس وقفه خود، جایی که یک پرچم تنظیم می‌شود، منتقل کند.

رابط BIOS:DOS

در ماجول سیستم، IO.SYS و MSDOS.SYS استفاده از BIOS را سهولت می‌بخشند. چون این ماجولها، بسیاری از پردازشهای مورد نیاز اضافی را فراهم می‌سازند، عملیات DOS عموماً ساده‌تر از استفاده از نسخهٔ BIOS خود است و عمدتاً وابستگی کمتری به ماشین دارد.

IO.SYS یک رابط سطح پایین برای BIOS است که خواندن داده از ابزار خارجی به حافظه و نوشتن داده از حافظه به ابزار خارجی را، سهولت می‌بخشد.

MSDOS.SYS شامل یک مدیریت فایل است و برخی سرویسها را نیز فراهم می‌سازد. برای مثال، وقتی کاربر برنامه INT 21H را تقاضا می‌کند، عملیات اطلاعات را بر طبق محتویات ثبات‌ها به MSDOS.SYS ارائه می‌دهد. برای تکمیل درخواست MSDOS.SYS ممکن است اطلاعات را به یک یا چند فراخوانی به IO.SYS منتقل کند، که بنوبت BIOS را فرا می‌خواند. شکل زیر این ارتباط را نشان می‌دهد:



وقفه‌های DOS

وقفه‌های 20H تا 3FH برای عملیات DOS در نظر گرفته شده است، همانطور که در بخش‌های بعد توصیف خواهد شد.

INT 20H خاتمه برنامه. یک عملیات مطلق که اجرای یک برنامه COM. را خاتمه می‌دهد، آدرسهای Ctrl + Break و خطاهای بحرانی را بازیابی می‌کند، بافرهای ثبات را خالی می‌کند و کنترل را به DOS باز می‌گرداند. با خروج از این تابع، CS باید حاوی آدرس PSP باشد. INT 21H تابع 4CH با INT 20H جایگزین شده است.

INT 21H درخواست تابع DOS. یک کدتابع در AH نیازدارد و دربخش بعدبه جزئیات آن‌پرداخته‌خواهدشد.

INT 22H آدرس خاتمه. وقتی بارگذار برنامه، یک برنامه را برای اجرا در حافظه قرار می‌دهد، آدرس این وقفه در PSP از برنامه (افست 0AH) کپی می‌شود. در خاتمه برنامه، عملیات کنترل را به آدرس وقفه منتقل می‌کند. برنامه شما نباید این وقفه را صادر کند.

INT 23H آدرس Ctrl + Break. طراحی شده تا وقتی <Break> + <Ctrl> یا <C> + <Ctrl> را می‌فشارد کنترل به روتین DOS منتقل می‌شود (بر طبق افست 0EH در PSP). روتین اجرای برنامه و یا فایل دسته‌ای را خاتمه می‌دهد. برنامه می‌تواند این آدرس را تغییر دهد بطوریکه روتین خودش، فعالیت خاصی را انجام دهد بدون آنکه

برنامه را خاتمه دهد. برنامه شما نباید این وقفه را صادر کند.

INT 24H دستیاب خطای بحرانی. توسط سیستم برای انتقال کنترل در هنگام تشخیص خطای بحرانی استفاده می‌شود (بر طبق افست 12H از PSP). برنامه شما نباید این وقفه را صادر کند.

INT 25H خواندن مطلق دیسک. محتویات یک یا چند سکتور دیسک را می‌خواند، در فصل ۱۷ گفته شد، اما توسط INT 21H تابع 440DH کد فرعی 61H جایگزین شده است.

INT 26H نوشتن مطلق دیسک. محتویات یک یا چند سکتور دیسک را از روی حافظه می‌نویسد اما توسط تابع 440DH از وقفه 21H کد فرعی 61H جایگزین شده است.

INT 27H خاتمه یافتن اما مقیم ماندن. سبب می‌شود که یک برنامه COM. در هنگام خروج در حافظه باقی بماند این وقفه توسط تابع 31H از وقفه 21H جایگزین شده است.

INT 2FH وقفه مختلط. شامل ارتباط بین برنامه‌هاست، مانند ارتباط وضعیت صف چاپگر [1] حضور یک راه‌انداز ابزار، یا فرمان‌های سیستم مانند ASSIGN یا APPEND.

INT 33H دستیاب موس. سرویس‌هایی برای دستکاری موس فراهم می‌سازد (فصل ۲۱ را ببینید).

سرویس‌های INT 21H

در زیر سرویس‌های INT 21H ارائه شده‌اند که به یک کد تابع در ثبات AH نیاز دارند :

00H. خاتمه برنامه. اصولاً شبیه INT 20H است اما با تابع 4CH از وقفه 21H جایگزین شده است.
01H. ورودی صفحه کلید با انعکاس (فصل ۱۱).
02H. نمایش کاراکتر (فصل ۹).
03H. ارتباطات ورودی. خواندن یک کاراکتر از درگاه سریال به ALC برای انجام این کار استفاده از سرویس‌های اولیه و وقفه 14H از BIOS توصیه می‌شود.
04H. ارتباطات خروجی. DL حاوی کاراکتر برای انتقال است. وقفه 14H از BIOS توصیه می‌شود.
05H. خروجی چاپگر. (فصل ۲۰)
06H. صفحه کلید مستقیم. (فصل ۱۱)
07H. ورودی مستقیم صفحه کلید بدون انعکاس.(فصل ۱۱)
08H. ورودی صفحه کلید بدون انعکاس. (فصل ۱۱)
09H. نمایش رشته. (فصل ۹)
0AH. ورودی بافر شده صفحه کلید. (فصل ۱۱)
0BH. بررسی وضعیت صفحه کلید. (فصل ۱۱)
0CH. پاک کردن بافر صفحه کلید و برداشتن ورودی. (فصل ۱۱)
0DH. تنظیم راه انداز دیسک. (فصل ۱۸)
0EH. انتخاب راه انداز دیسک پیش فرض. (فصل ۱۸)
0FH. باز کردن فایل FCB. (فصل ۱۷)
10H. بستن فایل FCB. (فصل ۱۷)
11H. جستجوی اولین مورد موافق ورودی دیسک، این تابع با تابع 4EH جایگزین شده است.
12H. جستجوی مورد موافق بعدی ورودی دیسک، این تابع با تابع 4FH جایگزین شده است.
13H. حذف فایل FCB. این تابع با تابع 41H جایگزین شده است.
14H. خواند رکورد ترتیبی FCB. (فصل ۱۷)
15H. نوشتن رکورد ترتیبی FCB. (فصل ۱۷)
16H. ایجاد فایل FCB. (فصل ۱۷)
17H. نام گذاری مجدد فایل FCB. این با تابع 56H جایگزین شده است.
19H. تعیین راه انداز دیسک پیش فرض. (فصل ۱۸)
1AH. تنظیم ناحیه انتقال دیسک (فصل ۱۷)
1BH. گرفتن اطلاعات راه انداز پیش فرض. (فصل ۱۸)
1CH. گرفتن اطلاعات راه انداز خاص. (فصل ۱۸)
1FH. گرفتن بلوک پارامتر راه انداز پیش فرض. (فصل ۱۸)
21H. خواندن رکورد FCB بطور تصادفی. (فصل ۱۷)
22H. نوشتن رکورد FCB بطور تصادفی. (فصل ۱۷)
23H. گرفتن اندازه فایل FCB این تابع با تابع 42H جایگزین شده است.
24H. تنظیم فیلد رکورد FCB تصادفی. (فصل ۱۷)
25H. تنظیم بردار وقف. (فصل ۲۴) وقتی یک کاربر <Ctrl> + <Break> یا <Ctrl> + <C> را بفشارد.

1. Print Spooler

روال معمول خاتمه برنامه و بازگشت به سیستم عامل است. ممکن است بخواهید برنامه شما روتین خاصی را فراهم سازد و حالت خاصی را دستکاری کند. مثال زیر از تابع 25H برای تنظیم آدرس <Break> + <Ctrl> در جدول بردار وقفه <INT 23H> برای روتین خودش استفاده می‌کند. (C10BRK). این روتین هر عمل لازم را انجام می‌دهد :

```
درخواست تنظیم جدول آدرس ;  MOV  AH,25H
برای INT 23H ;               MOV  AL,23H
آدرس جدید ;                 MOV  DX,C10BRK
فراخوانی سرویس وقفه ;       MOV  21H
روتین Ctrl + Break ;         C10BRK :
                             ...
بازگشت وقفه ;                IRET
```

26H. ایجاد پیشوند سگمنت برنامهٔ جدید. توسط تابع 4B00H جایگزین شده است.

27H. خواندن بلوک دیسک بطور تصادفی. (فصل ۱۷)

28H. نوشتن بلوک دیسک بطور تصادفی. (فصل ۱۷)

29H. تشریح نام فایل. (فصل ۱۹)

2AH. گرفتن تاریخ سیستم. بازگشت این مقادیر دودویی : AL = روز هفته (یک شنبه = 0)، CX = سال (۲۰۹۹ - ۱۹۸۰)، DH = ماه (۱۲ - ۱)، DL = روز (۳۱ - ۱).

2BH. تنظیم تاریخ سیستم. بارگذاری این مقادیر دودویی : CX = سال (۲۰۹۹ - ۱۹۸۰) DH = ماه (۱۲-۱)، DL = روز (۳۱ - ۱) در بازگشت AL مبنی بر یک عملیات معتبر (00H) یا عملیات نامعتبر (FFH) می‌باشد.

2CH. گرفتن زمان سیستم. بازگشت این مقادیر دودویی : CH = ساعت، در ۲۴ ساعت (۰ تا ۲۳ که نیمه شب یعنی ۰۰)، CL = دقیقه (۰ تا ۵۹) ، DH = ثانیه (۰ تا ۵۹)، DL = صدم ثانیه (۰ تا ۹۹).

2DH. تنظیم زمان سیستم. بارگذاری این مقادیر دودویی : CH = ساعت، در قالب ۲۴ ساعت (۰ تا ۲۳ که نیمه شب یعنی ۰۰)، CL = دقیقه (۰ تا ۵۹) ، DH = ثانیه (۰ تا ۵۹)، DL = صدم ثانیه (۰ تا ۹۹). در بازگشت AL به معنی عملیات معتبر (00H) یا یعنی عملیات نامعتبر (FFH) می‌باشد.

2EH. تنظیم/ یا تنظیم مجدد تشخیص‌دیسک (فصل ۱۸)

2FH. گرفتن آدرس ناحیه انتقال دیسک جاری (DTA). (فصل ۱۷ و تابع 1AH برای تنظیم آدرس).

30H. گرفتن شماره نسخه DOS. (تابع 3306H را ببینید). بازگشت این مقادیر:

AL = شماره اصلی، مانند n در نسخه n.11.

AH = شماره فرعی، مانند 11 در نسخه n.11.

BH = شماره کارخانه یا پرچم نسخه. اگر پرچم نسخه 08H باشد، DOS در ROM اجرا می‌شود.

BL:CX = صفر یا شماره سریال کاربر ۲۴ بیتی (وابسته به سازنده)

31H. خاتمه اما مقیم ماندن. (فصل ۲۴)

32H. گرفتن بلوک پارامتر درایو (DPB) (فصل ۱۸)

3300H. گرفتن وضعیت Ctrl + C اگر پرچم Ctrl + C خاموش (0) باشد، سبب می‌شود، هنگام دستکاری کاراکترهای I/0 توابع 0CH-01H سیستم Ctrl + C را بررسی کند. اگر پرچم فعال (۱) باشد، زمان دستکاری دیگر توابع سیستم بررسی می‌کند. برای گرفتن وضعیت 00H را در AL بگذارید. مقدار بازگشتی در DL، = 00H بررسی عدم فعالیت یا 01H = بررسی فعالیت.

3300H. بررسی وضعیت Ctrl + C اگر پرچم Ctrl + C خاموش (0) باشد، سبب می‌شود، هنگام دستکاری کاراکترهای I/0 توابع 01H تا 0CH سیستم Ctrl + C را بررسی کند. اگر پرچم یک باشد هنگام دستکاری دیگر توابع بررسی می‌کند. برای تنظیم وضعیت، زیر تابع 01H را در AL تنظیم کنید. وضعیت را در DL بطوری تنظیم کنید که 00H = تنظیم بررسی خاموش یا 01H = تنظیم بررسی روشن.

3305H. گرفتن درایو راه انداز. در DL درایو (1 = A و غیره) باز می‌گرداند. برای بارگذاری فایل‌های سیستم استفاده می‌شود.

3306H. گرفتن نسخه DOS (تابع 30H را ببینید.) این مقادیر باز می‌گردد.

BL = شماره نسخه اصلی، مانند n در نسخه n.11.

BH = شماره نسخه فرعی، مانند 11 در نسخه n.11.

DL = شماره بازبینی در بیت‌های ۰ تا ۲.

DL = پرچم نسخه DOS (مبنی بر این است که آیا

سیستم در حافظه اصلی، ناحیه بالایی حافظه یا ROM است). اگر چه فرمان SETVER می‌تواند شمارهٔ نسخه را جعل کند، تابع 3306H نسخه صحیح را ارائه می‌دهد.

34H. گرفتن آدرس پرچم (inDOS) مشغولیت DOS (فصل ۲۴)

35H. گرفتن بردار وقفه. (فصل ۲۴)

36H. گرفتن فضای آزاد دیسک. (فصل ۱۸)

38H. گرفتن / تنظیم اطلاعات کشور وابسته. تعدادی‌تابع که اطلاعات‌خاص راجع‌به کشورهای مختلف را پشتیبانی می‌کند، مانند یک سمبول و قالب برای پول رایج، جداکننده صدگان و دهگان و جداکننده تاریخ و زمان. برای عملیات در DX، FFFFH را برای تنظیم کد کشور که سیستم استفاده می‌کند یا هر مقدار دیگری، برای گرفتن کد کشور رایج مورد استفاده، قرار دهید.

39H. ایجاد زیر شاخه (MKDIR). (فصل ۱۸)

3AH. پاک کردن زیر شاخه (RMDIR). (فصل ۱۸)

3BH. تغییر شاخه جاری (CHDIR). (فصل ۱۸)

3CH. ایجاد شاخه با دستگیره. (فصل ۱۷)

3DH. باز کردن فایل با دستگیره. (فصل ۱۷)

3EH. بستن فایل با دستگیره. (فصل ۱۷)

3FH. خواندن فایل / ابزار. (فصل ۹ و ۱۷)

40H. نوشتن فایل / ابزار با دستگیره.(فصل ۹، ۱۷ و ۲۰)

41H. حذف فایل از شاخه. (فصل ۱۸)

42H. انتقال اشاره‌گر فایل. (فصل ۱۷)

43H. بررسی / تغییر صفت فایل (فصل ۱۸)

44H. کنترل I/0 برای ابزارها. مجموعهٔ گرانقدری از زیر تابع‌ها برای بررسی ابزارها و خواندن و نوشتن داده‌ها را پشتیبانی می‌کند که در زیر لیست شده‌اند :

4400H. گرفتن اطلاعات ابزار. (فصل ۱۸)

4401H. تنظیم اطلاعات ابزار. (فصل ۱۸)

4404H. خواندن داده کنترلی از درایو. (فصل ۱۸)

4405H. نوشتن داده کنترلی به درایو. (فصل ۱۸)

4406H. بررسی وضعیت ورودی (فصل ۱۸)

4407H. بررسی وضعیت خروجی (فصل ۱۸)

4408H. تعیین وجود واسطه قابل انتقال برای ابزار (فصل ۱۸)

440DH. کدفرعی 41H: نوشتن‌سکتوردیسک(فصل ۱۸)

440DH. کدفرعی 61H:خواندن‌سکتوردیسک(فصل ۱۸)

440DH. کد فرعی 42H : فرمت تراک (فصل ۱۸)

440DH. کد فرعی 46H : تنظیم ID واسطه (فصل ۱۸)

440DH. کدفرعی 60H:گرفتن‌پارامترهای‌ابزار(فصل ۱۸)

440DH. کد فرعی **66H : ID** واسطه (فصل ۱۸)

440DH. کدفرعی **68H**: تشخیص‌نوع واسطه (فصل ۱۸)

45H. کپی برداری از دستگیره فایل. (فصل ۱۸)

46H. اجبار در کپی برداری از دستگیره فایل. (فصل ۱۸)

47H. گرفتن شاخه جاری (فصل ۱۸)

48H. تخصیص بلوک حافظه (فصل ۲۴)

49H. اجبار در تخصیص بلوک حافظه (فصل ۲۴)

4AH. تنظیم‌اندازه‌بلوک حافظه تخصیص‌یافته. (فصل ۲۴)

4BH. بارگذاری و اجرای یک برنامه. (فصل ۲۴)

4CH. خاتمه برنامه (فصل ۴) عملیات استاندارد برای خاتمه اجرای برنامه.

4DH. بازیابی کدبازگشتی از یک زیرپردازش. (فصل ۲۴)

4EH. یافتن اولین مورد موافق ورودی شاخه. (فصل ۱۸)

4FH. یافتن مورد بعدی موافق ورودی شاخه. (فصل ۱۸)

50H. تنظیم آدرس پیشوند قطعه برنامه (PSP) در BX افست آدرس PSP برای برنامه جاری را قرار دهید. مقداری برنمی‌گردد.

51H. گرفتن آدرس پیشوند قطعه برنامه (PSP) بازگشت افست آدرس PSP برای برنامه جاری. (فصل ۲۴)

52H. گرفتن آدرس لیست DOS داخلی. (بدون توضیحات، فصل ۲۴)

54H. گرفتن آدرس وضعیت تشخیص. (فصل ۱۸)

56H. نام‌گذاری مجدد یک فایل. (فصل ۱۸)

57H. گرفتن / تنظیم تاریخ و زمان فایل. (فصل ۱۸)

5800H. گرفتن استراتژی تخصیص حافظه. (فصل ۲۴)

5801H. تنظیم استراتژی تخصیص حافظه. (فصل ۲۴)

5802H. گرفتن پیوند حافظه فوقانی. (فصل۲۴)

5803H. تنظیم پیوند حافظه فوقانی. (فصل ۲۴)

59H. گرفتن کد خطای توسعه یافته. (فصل ۱۸)

5AH. ایجاد یک فایل کمکی. (فصل ۱۸)

5BH. ایجاد یک فایل جدید.

5CH. قفل کردن یا قفل نکردن دستیابی به فایل. در محیط‌های شبکه و چند کاره استفاده می‌شود.

5DH. تنظیم خطای توسعه یافته. در DX آدرس افست یک جدول از اطلاعات خطا قرار دهید. اجرای بعدی تابع 59H (گرفتن کد خطای توسعه یافته) برای بازیابی یک جدول (تابع 59H در فصل ۱۸ را برای جزئیات بیشتر ببینید.)

5EH. سرویس‌های شبکه ناحیه محلی. یک زیر تابع در AL سرویس را مشخص می‌سازد: 00H= گرفتن نام ماشین، 02H =تنظیم اولیه چاپگر، 03H =گرفتن تنظیم اولیه چاپگر.

5FH. سرویس‌های شبکه ناحیه محلی. یک زیر تابع در AL سرویس را مشخص می‌سازد: 02H= گرفتن لیست مشخص کننده ورودی، 03H = گرفتن ارتباط شبکه، 04H = لغو ارتباط شبکه.

62H. گرفتن آدرس PSP (تابع 51H نیز عملیاتی یکسان دارد).

65H. گرفتن اطلاعات گسترده کشور. تعدادی زیر تابع که اطلاعاتی خاص کشورهای مختلف دارند، پشتیبانی می‌کند.

66H. گرفتن/ تنظیم صفحه کد عمومی.

67H. تنظیم حداکثر تعداد دستگیره. (فصل ۲۴)

68H. درگیر کردن فایل (فصل ۱۸)

6CH. باز کردن فایل گسترش یافته. ترکیب توابع 3CH (ایجاد فایل)، 3DH (باز کردن فایل) و 5BH (ایجاد فایل یگانه). (فصل ۱۸).

نکات کلیدی

- ROM در موقعیت FFFF0H ساکن می‌شود. روشن کردن کامپیوتر یک راه اندازی سرد است. پردازشگر یک حالت تنظیم را وارد می‌کند، همه موقعیت‌های حافظه صفر می‌شوند. یک بررسی توازن حافظه انجام می‌شود و ثبات CS با FFFF[0] H تنظیم می‌شود و ثبات IP صفر می‌شود. بنابراین اولین دستور اجرایی در 0: FFFF یا FFFF0 نقطه ورود به BIOS می‌باشد.
- در زمان راه اندازی، BIOSدرگاه‌های مختلف را برای تعیین و مقدار دهی ابزارهایی که متصل شده‌اند، بررسی می‌کند. سپس BIOS یک جدول بردار وقفه، با شروع از موقعیت صفر حافظه برقرار می‌سازد که شامل آدرسهای وقفه‌هایی است که رخ می‌دهد. دو عملیاتی که BIOS انجام می‌دهند تعیین تجهیزات و اندازه حافظه است. اگر یک دیسک شامل فایل‌های سیستم DOS وجود داشته باشد، BIOS اولین سکتور دیسک شامل بارگذار راه انداز را دستیابی می‌کند. این برنامه فایل‌های سیستم IO.SYS، MSDOS.SYS و COMMAND.COM را از دیسک در حافظه قرار می‌دهد.
- BIOS ناحیه داده خودش در حافظه پایینی با شروع از آدرس سگمنت 40[0] H نگه داری می‌کند. ناحیه داده مربوطه شامل موارد زیر می‌باشد : درگاه سریال، درگاه موازی، تجهیزات سیستم، صفحه کلید، راه‌انداز دیسکت، کنترل ویدئو، دیسک سخت و ساعت زمان واقعی.
- عملوند یک دستور وقفه مانند INT 12H، نوع درخواست برای هر یک از ۲۵۶ نوع ممکن را انتخاب می‌کند، سیستم یک آدرس ۴ بایتی در جدول سرویس‌های وقفه در موقعیت 0000H تا 3FFH نگه می‌دارد.
- محدوده وقفه‌های BIOS از 00H تا 1FH است و شامل چاپ صفحه، زمان سنج، کنترل ویدئو، کنترل دیسکت، نمایش ویدئو، تعیین تجهیزات و اندازه حافظه، I/O دیسک، ورودی صفحه کلید، ارتباطات، خروجی چاپگر و بارگذار راه انداز می‌باشد.
- تابع 3FH از وقفه 20H برای عملیات DOS در نظر گرفته شده است.
- INT 21H توابع موجود در این وقفه شامل ورودی صفحه کلید، خروجی نمایش، خروجی چاپگر، تنظیم دیسک، بازکردن/ بستن فایل، حذف‌فایل، خواندن/ نوشتن رکورد، خاتمه اما مقیم‌ماندن، ایجاد زیرشاخه و خاتمه برنامه‌است.

پرسش‌ها

۲۵-۱. بین وقفه خارجی و داخلی چه تفاوتی هست.

۲۵-۲. بین خط NMT و یک خط INTR چه فرقی هست.

۲۵-۳. (الف) موقعیت حافظه در نقطه ورود به BIOS چیست؟

(ب) هنگام روشن شدن چگونه سیستم به این آدرس دست می‌یابد؟

۲۵-۴. در هنگام راه اندازی BIOS وقفه‌های INT 11H، 12H و 19H را انجام می‌دهد. هدف از هر وقفه را توضیح دهید.

۲۵-۵. نقطه آغازین ناحیه داده BIOS کجاست؟

۲۵-۶. مقادیر دودویی زیر در ناحیه داده BIOS نوشته شده‌اند. برای هر عنصر، فیلد را مشخص کنید و مفهوم علمی یک بیت را شرح دهید.

(الف) 0100010001100111:10-11 H، (ب) 01101010:17H،

(ج) 00010010:18H ، (د) 00001010:96H

۲۵-۷. مقادیر شانزدهی زیر در ناحیه BIOS نوشته شده‌اند. برای هر عنصر فیلد را مشخص کنید و مفهوم علمی هر یک را شرح دهید،

(الف) F8 03 F8 02:00-03H ، (ب) 78 03 00 00:08-0BH،

(ج) 80 20:13-14H، (د) 00 10:15-16H،

(ه) 50 00:4A-4BH، (و) 0E 0D:60-61H

(ز) 18:84H

۲۵-۸. وقفه‌های BIOS زیر را تعیین کنید :

(الف) تعیین اندازه حافظه، (ب) ارتباطات I/O،

(ج) گرفتن وضعیت تجهیزات، (د) خروجی چاپگر،

(ه) ورودی صفحه کلید، (و) I/O دیسک،

(ز) نمایش ویدئو، (ح) وقفه صفحه کلید،

(ط) چاپ صفحه، (ی) تقسیم بر صفحه.

۲۵-۹. کدام عملیات INT برای DOS در نظر گرفته شده است؟

۲۵-۱۰. توابع سرویس‌های INT 21H زیر را تعیین کنید؟

(الف) خاتمه اما مقیم ماندن، (ب) گرفتن آدرس جدول وقفه،

(ج) ایجاد زیر شاخه، (د) گرفتن فضای آزاد دیسک،

(ه) گرفتن آدرس PSP، (و) ارتباطات ورودی

(ز) گرفتن زمان سیستم، (ح)نام گذاری مجدد یک فایل.

۲۵-۱۱. توابع INT 21H زیر را تعیین کنید.

(الف) 03H، (ب) 09H، (ج) 0DH، (د) 19H، (ه) 2AH،

(و) 31H، (ز) 35H، (ح) 39H، (ط) 41H.

فصل ۲۶

عملگرها و پیش پردازنده‌ها

هدف : ارائه توضیحات کامل در مورد عملگرهای اسمبلی و پیش‌پردازنده‌ها.

مقدمه

ویژگیهای مختلف زبان اسمبلی در ابتدا ممکن است قدری پیچیده بنظر آید. اما اکنون که با جنبه‌های ساده‌تر و عمومی‌تر اسمبلی در چند فصل آشنا شدید، فهم ویژگی‌های مختلف انواع شناسه‌ها عملگرها و پیش‌پردازنده‌ها در این فصل ساده‌تر خواهد بود. کتاب حاضر شامل تعدادی از ویژگیهای جنبی مفید دستورات اسمبلی می‌باشد. به قابلیت توربواسمبلر توجه کنید : TASM هم می‌تواند در حالت MASM اجرا شود که مشخصات MASM استاندارد را بپذیرد، یا در حالت ایده‌آل باشد، که در بسیاری حالات، قوانین و عبارات متفاوتی دارد که مشخصات MASM را تشخیص نمی‌دهد. این فصل بسیاری از این استثناءها را بررسی می‌کند.

شاخص نوع

شناسه‌های تعیین نوع می‌توانند اندازهٔ متغیر یا فاصله نسبی یک برچسب دستور را تعریف کنند. شناسه‌هایی که اندازهٔ یک متغیر داده را مشخص می‌کنند، BYTE، WORD، DWORD، FWORD، QWORD، TBYTE هستند. آنهایی که فاصله یک برچسب دستور را مشخص می‌کنند، NWAR، FAR و PROC می‌باشند. یک آدرس نزدیک که فقط یک افست است، در سگمنت جاری فرض می‌شود، یک آدرس دور که شامل افست : سگمنت است می‌تواند برای دستیابی داده در سگمنت دیگر استفاده شود.

عملگردهای PIR و THIS با پیش‌پردازنده‌های COM، EXTRN، LABEL، PROC بعنوان شاخص نوع استفاده می‌شوند.

عملگرها

یک عملگر امکان تغییر یا تجزیه عملوند در طی اسمبل کردن را فراهم می‌سازد. عملگرها به طبقات مختلف زیر تقسیم می‌شوند.

- ***عملگرهای محاسبات :*** ریاضی، نشانه‌گذاری، منطقی، شیفت و نام فیلد ساختار.
- ***عملگرهای ماکرو :*** انواع مختلفی که در فصل ۲۲ گفته شد.
- ***عملگرهای رکورد :*** WIDTH,MASK بعداً در این فصل تحت عنوان پیش‌پردازنده RECORD گفته می‌شود.
- ***عملگرهایی رابطه‌ای :*** EQ ، GE ، GT ، LE ، LT و NE

- **عملگرهای سگمنت** : SEG، OFFSET و بازنویسی سگمنت.
- **عملگرهای نوع (یا صفت)** : HIGH، HIGHWORD، LENGTH، LOW، LOWWORD، PTR، SHORT، SIZE، THIS و TYPE

این عملگرها را بترتیب الفبایی توضیح خواهیم داد.

عملگرهای ریاضی

این عملگرها شامل علائم آشنای ریاضی هستند و در طی مراحل اسمبل محاسبات ریاضی را انجام می‌دهند. در اغلب حالات، می‌توانید محاسبات را شخصاً انجام دهید، اگر چه مزیت استفاده از این عملگرها این است که هر زمان برنامه را تغییر دهید و مجدداً آن را اسمبل کنید، اسمبلر بطور خودکار، مقادیر عملگرهای ریاضی را محاسبه می‌کند. در زیر بیتی از عملگرها همراه با مثالی از استفاده و تاثیر آنها آمده است :

علامت	نوع	مثال	تاثیر
+	جمع	FLDA + 25	جمع ۲۵ با آدرس FLDA
+	مثبت	+FLDA	با FLDA مانند یک رقم مثبت برخورد می‌کند
-	تفریق	FLDB-FLDA	محاسبه اختلاف بین دو افست آدرس
-	منفی	-FLDA	علامت FLDA را معکوس می‌کند
×	ضرب	valve × 3	ضرب valve در ۳
/	تقسیم	valve / 3	تقسیم valve بر ۳
MOD	باقیمانده	valve1 MOD valve2	باقیمانده valve2 / valve1 را ارائه می‌دهد.

بجز جمع و تفریق، همه عملگرها باید ثبات‌های صحیح باشند، مثالهای مرتبط زیر، عبارات صحیح را مشخص می‌سازد:

```
value1 = 12 * 4          ; 48
value1 = valve1/6        ; 48/6 = 8
value1 = -value1 - 3     ; (-8) - (3) = -11
```

عملگرهای HIGH و HIGHWORD

عملگر HIGH بایت بالا (سمت چپ) از یک عبارت را باز می‌گرداند و HIGHWORD (از MASM 6.0) (از MASM 6.0) کلمه بالا رتبه یک عبارت را باز می‌گرداند. (عملگر LOW را نیز ببینید). در اینجا یک مثال ذکر شده

```
EQUVAL   EQU   1234H
         . . .
         MOV   CL,HICH EQUVAL
```

عملگرهای نشانه

برای یک ارجاع مستقیم حافظه، یک عملوند از دستور نام عنصر داده تعریف شده را مشخص می‌کند، مانند COUNTER در دستور ADD CX,COUNTER. در طی اجرا پردازشگر، با ترکیب آدرس سگمنت داده در DS با مقدار افست در عنصر داده، در موقعیت عنصر دادهٔ مشخص شده در حافظه قرار می‌گیرد.

برای آدرسدهی غیر مستقیم، یک عملوند، یک پایه یا ثبات نشانه، ثبات متغیرهای افست و متغیرها مراجعه می‌کند. عملگر نشانه، که در کروشه قرار می‌گیرد، مانند علامت (+) جمع عمل می‌کند. یک نوع استفاده از نشانه‌گذاری، مراجعه به عناصر داده جدول می‌باشد. می‌توانید از عملیات زیر برای مراجعه به حافظهٔ نشانه‌گذاری شده استفاده کنید :

- [ثابت] یک عدد یا نام بلافصل در کروشه. برای مثال ، پنجمین ورودی PARTTBL را در CL قرار دهید (توجه کنید که [0] PARTTBL در اولین ورودی است).

```
PARTTBL  DB   25 DUP (?)          تعریف جدول ;
  . . .
         MOV  CL,PARTTBL [4]  ; گرفتن پنجمین ورودی از PARTTBL
```

- ثبات پایه BX بصورت [BX] در ارتباط با ثبات سگمنت DS است و ثبات پایه BP در ارتباط با ثبات سگمنت SS می‌باشد. برای مثال، از آدرس افست در BX (بصورت DS:BX) استفاده کنید و عنصر ارجاعی را در Dx منتقل کنید:

```
MOV  DX,[BX]  ; ثبات پایه DS:BX
```

- ثبات نشانه DI بصورت [DI] و ثبات نشانه SI بصورت [SI]، هر دو در ارتباط با ثبات سگمنت DS می‌باشد. برای مثال، از آدرس افست در SI (بصورت DS:SI) و انتقال عنصر ارجاعی به AX استفاده کنید :

```
MOV  AX,[SI]  ; ثبات نشانه DS:SI
```

- ثبات‌های نشانه ترکیبی و برای مثال، محتویات AX را به آدرس مشخص شده با جمع آدرس DS، افست BX، افست SI و ثابت ۴، منتقل کنید : ثبات + index + پایه ;

```
MOV  [BX+SI+4],[AX]
```

اولین عملوند در مثال قبل همچنین می‌تواند بصورت 4+ [BX + SI] کدنویسی شود. شما ممکن است این عملوندها را با هر ترتیب، ترکیب کنید، اما نمی‌توانید دو ثبات پایه [BX + BP] یا دو ثبات نشانه [DI + SI] را با هم ترکیب کنید. فقط ثبات‌های نشانه باید در کروشه باشند و در نتیجه اسمبلر می‌داند که چگونه با آن بعنوان یک ورودی نشانه رفتار کند.

عملگر LENGTH

عملگر LENGTH تعداد ورودیهای تعریف شده با یک عملگر DUP را باز می‌گرداند، همانطور که در دستور MOV زیر نشان داده شده است :

```
PARTTBL  DW   10 DUP(?)
  . . .
         MOV  DX,LENGTH PARTTBL:DX      بازگشت طول ۱۰ در
```

اگر عملوند بازگشتی شامل ورودی DUP نباشد ، عملگر مقدار ۰۱ را باز می‌گرداند. (عملگرهای SIZE و TYPE را نیز ببینید).

عملگرهای منطقی

عملگرهای منطقی، روی بیت‌های یک عبارت، عملیات منطقی انجام می‌دهند :

عملگر	نحوه استفاده	تاثیر
AND	عبارت AND عبارت ۱	بیت‌ها را با هم AND می‌کند
OR	عبارت OR عبارت ۱	بیت‌ها را با هم OR می‌کند
XOP	عبارت XOR عبارت ۱	بیت‌ها را با هم OR انحصاری می‌کند
NOT	عبارت 1 NOT	بیت را معکوس می‌کند.

در اینجا دو مثال آورده شده است :

```
MOV  CL,00111100B AND 01010101B   ; CL = 00010100B
MOV  DL,NOT 01010101              ; DL = 10101010B
```

عملگرهای LOW و LOWWORD

عملگرهای LOW بایت پایین (سمت راست) یک عبارت را باز می‌گرداند و LOWWORD (از MASM 6.0) کلمه پایین عبارت را باز می‌گرداند. (عملگر HIGH را نیز ببینید). در اینجا یک مثال آورده شده است.

```
EQUVAL  EQU  1234H
        ...
        MOV  CL,LOW EQUVAL     CL ; قرار دادن 34H در
```

عملگر OFFSET

عملگر OFFSET آدرس افست یک متغیر یا برچسب را باز می‌گرداند (که این یک آدرس نسبی داخل سگمنت داده یا سگمنت کد است). قالب کلی چنین است :

```
OFFSET برچسب یا متغییر
```

دستور MOV زیرآدرس افست از PARTTBL را باز می‌گرداند :

```
MOV DX,OFFSET PARTTBL
```

عملگر MASK

پیش پردازندهٔ RECORD در بخش پیش پردازنده‌ها را ببینید.

عملگر RTR

عملگر RTR می‌تواند با متغیرهای داده و برچسب دستور استفاده شود. این عملگر از شناسه‌های BYTE، WORD، FWORD، DWORD، QWORD، TBYTE برای مشخص کردن اندازه یک عملوند نامشخص یا بازنویسی نوع تعیین شده برای متغیر (DB ، DW ، DF ، DD ، DF یا DT) استفاده می‌کند. همچنین از شناسه‌های نوع NEAR، FAR، PROG برای بازنویسی فاصله ضمنی برچسب استفاده می‌کند. قالب کلی PTR چنین است :

```
type PTR  عبارت
```

این type یک صفت جدید است. مانند BYTE. عبارت می‌تواند یک متغییر یا ثبات باشد. در زیر مثالهای غیر مرتبطی از عملگر PTR آورده شده است (به WORDA دقت کنید، که اسمبلر بایت‌ها را به ترتیب معکوس ذخیره می‌سازد) :

```
BYTEA   DB   22H
        DB   23H
WORDA   DW   2672H                  ; داده بصورت 7226 ذخیره‌می‌شود
        MOV  AH,BYTE PTR WORDA       ; انتقال اولین بایت (72)
        ADD  BL,BYTE PTR WORDA       ; انتقال دومین بایت (26)
        MOV  BYTE PTR WORDA,05       ; انتقال ۰۵ به اولین بایت
        MOV  AX,WORD PTRA BYTEA      ; انتقال دو بایت به AX (2235)
        CALL FAR PTR [BX]            ; فراخوانی روال دور
```

یک مورد دیگر که تابعی مشابه PTR انجام می‌دهد، پیش پردازنده LABEL است بعد از توضیح داده خواهد شد.

عملگر SEG

عملگر SEG آدرس سگمنتی که متغیر مشخص شده یا بر چسب در آن قرار گرفته باز می‌گرداند. برنامه‌ای که در سگمنت اسمبل شدهٔ مجزا را ترکیب می‌کند اغلب، احتمالاً از این عملگر استفاده خواهد کرد.

قالب کلی آن چنین است : برچسب یا متغییر SEG

دستورات MOV زیر آدرس سگمنتی را که نامهای ارجاع شده در آن تعریف شده‌اند، باز می‌گرداند :

```
MOV DX,SEG WORDA ; آدرس سگمنت داده
MOV DX,SEG A10BEGIN ; آدرس سگمنت کد
```

عملگر بازنویسی سگمنت

این عملگر بصورت یک علامت (:) کد می‌شود، آدرس برچسب یا متغیر مرتبط با سگمنت خاص را محاسبه می‌کند. قالب کلی آن چنین است. عبارت : سگمنت

سگمنت نامبرده می‌تواند هر ثبات سگمنت با یک سگمنت یا نام گروه باشد. عبارت می‌تواند یک ثابت، یا عبارت، با یک عبارت SEG باشد. این مثالها، ثبات سگمنت DS پیش فرض را بازنویسی می‌کنند.

```
MOV BH,ES:10H ; دستیابی از ES + 10H
MOV CX,SS:[BX] ; دستیابی از SS + افست در BX
```

یک دستور می‌تواند، عملگر بازنویسی سگمنت را فقط برای یک عملوند استفاده کند.

عملگرهای SHL و SHR

عملگرهای SHL و SHR یک عبارت را در طی اسمبل کردن شیفت می‌دهد. قالب کلی آنها چنین است :

expression SHL/SHR Count

در مثال زیر، عملگر SHR محتویات بیت را ۳ بیت به راست شیفت می‌دهد :

```
MOV BL,01011101 SHR 3 ; قرار دادن مقدار 00001011
```

در اغلب موارد مشابه، عبارت یک نام سمبولیک را ارجاع می‌دهد تا یک مقدار ثابت.

عملگر SHORT

هدف از عملگر SHORT تعیین صفت NEAR برای یک مقصد پرش JMP است که در داخل ۱۲۷+ و ۱۲۸- بایت قرار دارد. قالب آن چنین است : JMP SHORT Label

اسمبلر عملوند کد ماشین را از دو بایت به یک بایت کاهش می‌دهد. این خصیصه برای پرش‌های نزدیک که به طرف جلو منشعب می‌شوند مفید است[1]، در غیر اینصورت اسمبلر فاصله بین آدرس پرش را نمی‌داند و ممکن است دو بایت برای یک پرش نزدیک در نظر بگیرد.

عملگر SIZE

عملگر SIZE حاصل ضرب LENGTH در TYPE را باز می‌گرداند و اگر فقط متغییر ارجاع شده حاوی ورودی DUP باشد، مفید است. تحت مدل ایده‌آل TASM، عبارت تعداد بایت‌های واقعی را باز می‌گرداند، قالب کلی چنین است : SIZE Variable

بخش عملگر TYPE را برای دیدن مثال، ملاحظه کنید.

عملگر THIS

عملگر THIS، یک عملوند با مقادیر سگمنت و افست ایجاد می‌کند که مساوی است با شمارشگر موقعیت جاری

---

۱. پرش‌های نسبی مترجم

است، قالب کلی آن چنین است : THIS type

شاخص type می‌تواند BYTE، WORD، DWORD، FWORD، QWORD یا TBYTE برای متغیرها و NEAR، FAR یا PROC برای برچسب‌ها باشد. THIS نوعاً با پیش پردازنده‌های EQU یا علامت مساوی (=) استفاده می‌شود. مثال زیر PARTREC را تعریف می‌کند : PARTREC EQU THIS BYTE

تاثیر آن مشابه استفاده از پیش‌پردازنده، LABEL به صورت زیر است

```
PARTREC LABEL BYTE
```

عملگر TYPE

عملگر TYPE تعداد بایت‌ها، بر طبق تعریف متغیر ارجاع داده شده را باز می‌گرداند. اما عملیات همیشه مقدار یک برای متغیر رشته‌ای و مقدار صفر را برای ثابت باز می‌گرداند.

تعریف	تعداد بایت‌های متغیرهای عددی
DB/BYTE	1
DW/WORD	2
DD/DWORD	4
DF/FWORD	6
DQ/QWORD	8
DT/TWORD	10
STRUC/STRUCT	تعداد بایت‌های تعریف شده با ساختار
برچسب NEAR	FFFFH
برچسب FAR	FFFEH

قالب کلی آن چنین است : TYPE variable or Label

مثال‌های زیر عملگرهای TYPE، LENGTH، SIZE را مشخص می‌سازد.

```
BYTEA   DB    ?                 ; تعریف یک‌بایت
PARTTBL DW    10 DUP (?)        ; تعریف ۱۰ کلمه
        . . .
        MOV   AX,TYPE BYTEA       ; AX = 0001H
        MOV   AX,TYPE PARTTBL     ; AX = 0002H
        MOV   CX,LENGTH PARTTBL   ; CX = 000AH (10)
        MOV   DX,SIZE PARTTBL     ; DX = 0014H (20)
```

از آنجائیکه PARTTBL بصورت DW تعریف شده است، TYPE مقدار 0002H را باز می‌گرداند، LENGTH مقدار 000AH (10) بر مبنای ورودی DUP و SIZE مقدار TYPE ضرب در LENGTH یا 14H(20) را باز می‌گرداند.

عملگر WIDTH

"پیش پردازنده RECORD" در بخش بعدی را ملاحظه کنید.

پیش پردازنده‌ها

این بخش اغلب پیش‌پردازنده‌های اسمبلر را تعریف می‌کند و فصل ۴، با جزئیات کامل پیش‌پردازنده‌های تعریف داده (DB ، DW و غیره) و فصل ۲۴ پیش‌پردازنده‌های دستورات ماکرو را در بردارند. بنابراین در اینجا تکرار نمی‌شوند.

پیش‌پردازنده‌ها به طبقات مختلف تقسیم می‌شوند :

- بر چسب‌های کد : ALLIGN ، EVEN ، LABEL و PROC
- اسمبل شرطی : IF ، ELSE و غیره که در فصل ۲۱ گفته شد.
- خطاهای شرطی : .ERR ، .ERRI و غیره.
- تخصیص داده : ALLIGN, EQU, EVEN, LABE, ORG, DB, DW, DD, DF, DQ, DT در فصل ۴ گفته شد.
- کنترل لیست‌گیری : .CREF، .LIST، PAGE، SUBTTL، (SUBTITLE)، TITLE، .XCREF و .XLIST. در این فصل توصیف خواهد شد. .LALL، .LFCOND، .SALL، .SFCOND، .TFCOND و .XALL در فصل ۲۲ توصیف شد.
- ماکروها : ENDM , EXITM , LOCAL , MACRO , PURGE مطالب فصل ۲۱.
- متفرقه : COMMCNT , INCLUDE , INCLUDELIB , NAME , OUT & .RADIX.
- پردازشگر : 8086.، 286.، 286P، 386.، 386P، 8087.، 287.، 387. و غیره.
- تکرار بلوک‌ها : IRP , IRPC , REPT در فصل ۲۲.
- حوزه : COMM , EXTRN , PUBLIC.
- سگمنت : .ALPHA، ASSUME.، DOSSEG، END، ENDS، GROUP، SEGMENT، SEQ.
- سگمنت ساده شده : .CODE، .CONST، .DATA، .DATA?، .DOSSEG، .EXIT، .FARDATA، .FARDATA?، .MODEL و .STACK.
- ساختار / رکورد : ENDS , RECORD , STRUCT , TRPEDEF , UNION.

چون دانستن این پیش‌پردازنده‌ها بر مبنای طبقات لازم نیست، این پیش‌پردازنده‌ها را (بجز آنهایی که در ارتباط با ماکروهاست) به ترتیب حروف الفبا بیان می‌کنیم.

پیش پردازنده ALIGN

پیش پردازنده ALIGN سبب می‌شود اسمبلر عنصر داده بعدی یا دستور بعدی را بر طبق آدرس مقدار داده شده قرار دهد. نحوه قرارگیری دستیابی کلمات یا داده‌های دو کلمه‌ای را برای پردازشگر امکان‌پذیر می‌کند. قالب کلی چنین است :

```
ALIGN number
```

تعداد باید توانی از ۲، مانند ۲، ۴، ۸ یا ۱۶ باشد. در مثال زیر، شمارشگر موقعیت 0005 است وقتی که به جمله ALIGN 4 می‌رسد سبب می‌شود تا شمارشگر در موقعیت بعدی قابل تقسیم بر ۴ پیش رود :

```
0005  ALIGN   4
0008  DWORD   DD 0     ; قرار دادن به مرز دو کلمه‌ای
```

اگر شمارشگر موقعیت، در آدرس مورد نظر باشد، پیش نمی‌رود. اسمبلر بایت‌های بلا استفاده را برای داده‌ها با صفر و با دستورات با NOP پر می‌کند. توجه کنید که ALIGN2 همان تاثیر EVEN را دارد.

پیش پردازنده ALPHA

پیش پردازنده ALPHA که در شروع یا حوالی ابتدای برنامه قرار دارد، به اسمبلر می‌گوید که سگمنت‌ها را به ترتیب حروف الفبا، برای مطابقت با نسخه اسمبلر مرتب کند. همچنین می‌توانید از انتخاب A/ در فرمان خطی اسمبلر استفاده کنید. (پیش پردازنده‌های DOSSEG و SEQ. را ملاحظه کنید).

پیش پردازنده ASSUME

ASSUME به اسمبلر می‌گوید نامهای سگمنت را با ثبات‌های CS ، DS ، ES و SS مرتبط سازد. قالب کلی آن چنین است.

```
ASSUME نام سگمنت : ثبات سگمنت [, . . .]
```

ورودیهای ثبات سگمنت معتبر CS ، DS ، ES ، SS به اضافه FS و GS در پردازشگرهای ۸۰۳۸۶ و ما بعد می‌باشد، نامهای سگمنت معتبر، ثبات‌های سگمنت، NOTHING , GROUP و یک عبارت SEG می‌باشد. یک عبارت ASSUME ممکن است تا چهار ثبات سگمنت را با هر ترتیبی مشخص می‌سازد. پیش‌پردازنده‌های سگمنت ساده شده به طور خودکار یک ASSUME تولید می‌کنند. در جملهٔ ASSUME زیر , DATASG , STACK CODESG نام‌هایی است که برنامه برای تعریف سگمنت‌ها استفاده می‌کند :

```
ASSUME CS:CODESG , DS:DATASG , SS:STACK , ES:DATASG
```

حذف یک سگمنت ارجاع شده مانند کد کردن NOTHING است. با استفاده از کلمه کلیدی NOTHING هر ASSUME حتی برای یک ثبات سگمنت مشخص را لغو می‌سازد.

```
ASSUME ES:NOTHING
```

فرض کنید که ثبات ES را تعیین نکرده‌اید و نه از NOTHING برای لغو آن استفاده کرده‌اید. سپس برای مراجعه به یک عنصر در سگمنت داده، یک عملوند دستور ممکن است از عملگر سگمنت (:) برای ارجاع به ثبات ES استفاده کند، که باید شامل یک آدرس سگمنت معتبر باشد :

```
MOV  AX,ES:[BX]    ; استفاده از آدرس نشانه گذاری شده
MOV  AX,ES:WORD    ; انتقال محتویات WORDA
```

پیش پردازنده CODE.

این پیش پردازنده سگمنت ساده شده سگمنت کد را تعریف می‌کند. قالب کلی آن چنین است :

```
.CODE [name]
```

همه کدهای اجرایی باید در این سگمنت قرار گیرند. برای مدل‌های TINY، SMALL ، COMPACT نام سگمنت پیش فرض -TEXT است. مدلهای حافظه MEDIUM و LARGE چندین سگمنت کد را مجاز می‌دانند، که تفاوت بین آنها در عملوند نام می‌باشد. (پیش پردازنده MODEL. را ببینید).

پیش پردازنده COMM

تعریف یک متغییر بصورت COMM به آن صفات PUBLC و EXTRN می‌دهد. با این روش، نباید در یک ماجول، متغییر را بصورت PUBLIC و در دیگری بصورت EXTRN تعریف کنید. قالب کلی آن چنین است :

```
COMM [NEAR / FAR] label: size [:count]
```

- COMM داخل یک سگمنت داده کد می‌شود.
- صفات NEAR یا FAR می‌تواند کد شود یا بسته به مدل حافظه بصورت پیش‌فرض مجاز شود.
- بر چسب نام یک متغیر است. توجه کنید که نام متغیری است که نمی‌تواند مقدار داخلی داشته باشد.
- اندازه می‌تواند هر شاخص نوع BYTE، WORD، DWORD، QWORD، TBYTE باشد یا هر مقدار صحیحی که تعداد بایت‌ها را مشخص کند.
- count مبنی بر تعداد عناصر یک متغیر است. پیش فرض آن یک است.

مثال‌های زیر، عناصری با صفت COMM تعریف می‌کنند.

```
COMM  NEAR  WORD COMFLD1:      ; اندازه کلمه با صفت COMM
COMM  FAR   COMMFLD2: BYTE:25  ; ۲۵ بایت با صفت COMM
```

پیش پردازندهٔ COMMENT

این پیش پردازنده برای چندین خط توضیح مفید است : قالب کلی آن چنین است :

```
COMMENT delimiter [comments]
        [comments]
        delimiter [comments]
```

dclimiter اولین کاراکتر غیر جای خالی است، مانند ٪ یا + که بعد از COMMENT قرار می‌گیرد. توضیحات در خطی که دومین delimiter ظاهر شود ، خاتمه می‌دهد این مثال از '+' بعنوان delimiter استفاده می‌کند : '

```
COMMENT + scans routine This
          the keyword input
          for invalid
        + charcters.
```

مدل ایده‌آل TASM پیش پردازنده، COMMENT را تشخیص می‌دهد.

پیش پردازنده CONST.

این پیش پردازنده سگمنت ساده یک سگمنت داده (یا داده ثابت) با کلاس 'const' تعریف می‌کند (پیش پردازندهٔMODEL. را نیز ببینید)

پیش پردازنده CREF.

این پیش پردازنده (پیش فرض) به اسمبلر می‌گوید یک جدول ارجاع متقابل تولید کند. این پیش پردازنده باید بعد از یک پیش پردازنده XCREF. که سبب جلوگیری از ایجاد جدول می‌شود، بکار می‌رود.

پیش پردازنده DATA. و ?DATA

این پیش پردازنده‌های سگمنت ساده شده سگمنت‌های داده را تعریف می‌کنند. DATA. یک سگمنت برای داده‌های نزدیک مقدار دهی شده تعریف می‌کند،?DATA. یک سگمنت برای داده‌های نزدیک مقدار دهی نشده تعریف می‌کند و اغلب وقتی با یک زبان سطح بالا پیوند زده می‌شود استفاده می‌شود. برای یک برنامه اسمبلی مستقل ممکن است دادهٔ نزدیک مقدار دهی نشده را در یک سگمنت DATA. تعریف کنید.
(پیش پردازنده‌های CODEL. و FARDATA. را ببینید).

پیش پردازنده DOSSEG. / DOSSEG

روشهای متعددی برای کنترل ترتیب مرتب کردن سگمنت‌ها توسط اسمبلر وجود دارد. (برخی نسخه‌های اخیر آن‌ها را بترتیب حروف الفبا مرتب می‌کنند). می‌توان پیش پردازنده‌های SEQ. یا ALPHA. را در شروع برنامه قرار داد. همچنین می‌توان گزینه‌های /S یا /A در خط فرمان اسمبلر وارد کرد. پیش پردازندهٔ DOSSEG (DOSSEG. در MASM 6.0) به اسمبلر می‌گوید که از دیگر درخواست صرف نظر کند و با ترتیب سگمنت DOS بر مبنای کد داده و پشته تطبیق دهد. این پیش‌پردازنده را در آغاز یا نزدیک به آغاز برنامه علی الخصوص برای سهولت استفاده از اشکال‌زدای CODEVIEW برای برنامه‌های مستقل قرار دهید.

پیش پردازنده END

پیش پردازنده END در انتهای برنامهٔ منبع قرار می‌گیرد. قالب کلی آن چنین است :

```
END [start-address]
```

آدرس شروع انتخابی مبنی بر موقعیت سگمنت کد، جایی است که اجرا آغاز می‌شود. بارگذار سیستم با استفاده از این آدرس ثبات CS را مقدار می‌دهد. اگر برنامه شما شامل فقط یک ماجول باشد، یک آدرس شروع تعریف کنید. اگر شامل تعدادی ماجول باشد، فقط یکی (اغلب اولی) یک آدرس شروع دارد.

پیش پردازندهٔ ENDP

این پیش پردازنده بیانگر انتهای روالی است که توسط PROC تعریف شده است، قالب کلی آن چنین است.

MASM:	proc-name ENDP
TASM Ideal model	Endp [proc- name]

نام روال باید همانی باشد که روال تعریف می‌کند.

پیش پردازندهٔ ENDS

این پیش پردازنده بیانگر انتهای یک سگمنت (تعریف شده با SEGMENT) یا ساختار تعریف شده با STRUC یا STRUCT) می‌باشد. قالب کلی آن چنین است :

MASM:	seg-nam ENDS
TASM Ideal mode	ENDS [seg-name]

نام سگمنت باید همانی باشد که سگمنت یا ساختار تعریف می‌کند.

پیش پردازنده EQU

پیش پردازنده EQU برای تعریف مجدد نام داده یا متغیر با دیگر نام داده، متغیر یا مقدار بلافصل استفاده می‌شود. پیش پردازنده باید در یک برنامه قبل از ارجاع به آن تعریف شود. قالب داده عددی و رشته‌ای تفاوت دارد.

```
Numeric equate: name EQU expression
String equate:    name <string> EQU
```

اسمبلر هر رخداد نام را با عملوند جایگزین می‌کند. چون EQU برای جایگزینی ساده استفاده می‌شود، هیچ فضای اضافی توسط آن در برنامه مقصد تولید شده اشغال نمی‌شود. مثال‌هایی از EQU با داده‌های عددی چنین است :

```
COUNTER  DW   0
         SUM  EQU    COUNTER     ; نام دیگری برای COUNTER
         TEN  EQU    10          ; مقدار عددی
              . . .
              INC    SUM         ; افزایش شمارشگر
              ADD    SUM,TEN     ; افزودن 10 به شمارشگر
```

مثال‌هایی از EQU برای داده‌های رشته‌ای :

```
PRODMSG   EQU    <'Enter prodoct number:'>
BYPTR     EQU    <BYTE PTR>
          . . .
MESSGE1   DB     PRODMSG          ; جایگزینی با رشته
          MOV    SAVE,BYPTR [BX]  ; جایگزینی با رشته
```

پرانتز زاویه‌ای بیانگر عملوند رشته‌ای است.

پیش پردازنده .ERR

این پیش پردازنده‌های خطای شرطی می‌تواند برای کمک به بررسی خطا در طی اسمبل کردن استفاده شود.

پیش پردازنده	ثبت خطا
.ERR	وقتی خطا واقع شود
.ERR1	در طی مرحله ۱ از اسمبلی
.ERR2	درطی مرحله ۲ از اسمبلی
.ERRE	توسط عبارت صحیح (۰)
.ERRNZ	توسط عبارت غلط (غیر صفر)
.ERRDEF	توسط عبارت سمبول تعریف شده
.ERRNDEF	توسط سمبول تعریف نشده
.ERRB	توسط یک رشته خالی
.ERRNB	توسط یک رشته غیر خالی
.ERRIDN [I]	توسط رشته‌های مساوی
.ERRDIF [I]	توسط رشته‌های متفاوت

شما می‌توانید از پیش پردازنده‌های فوق الذکر، در ماکروها و در جملات اسمبلی شرطی استفاده کنید. در جملات اسمبلی شرطی زیر ، اسمبلر، در صورت عدم صحت شرط، یک پیغام نمایش می‌دهد :

```
IF      condition
        . . .
ELSE    .ERR
        % OUT [message]
ENDIF
```

در MASM 6.0 نیازی به اشارهٔ مرحله اول (.ERR1) یا مرحله دوم (.ERR2) در اسمبل کردن نیست.

پیش پردازندهٔ EVEN

EVEN به اسمبلر می‌گوید تا در صورت لزوم شمارشگر موقعیت را پیش برد، بنابراین عنصر دادهٔ تعریف شدهٔ بعدی در یک مرز حافظه زوج آغاز می‌شود. این خصیصه، پردازشگر را در دستیابی عنصر داده ۱۶ یا ۳۲ بیتی راحت می‌سازد. (پیش پردازنده ALLIGN را نیز ببینید).

در مثال زیر، BYTELOCN یک فیلد یک بایتی بر روی یک مرز زوج 0016 است. حال شمارشگر موقعیت هم اکنون در 0017 قرار دارد. EVEN سبب می‌شود تا اسمبلر شمارشگر موقعیت را یک بایت جلو برده و به آدرس 0018 ببرد. در این آدرس WORDLOCN تعریف شده است :

```
0016    BYTELOCN    DB ?
0017    EVEN                    (پیش بردن شمارشگر موقعیت)
0018    WORDLOCN    DW ?
```

پیش پردازنده .EXIT

می‌توانید این پیش پردازنده را در سگمنت کد جهت تولید کد خاتمه برنامه بکار ببرید. قالب کلی آن چنین است :

MASM :	.EXIT [return-value]
TASM Ideal mode :	EXITCODE [return-value]

کد مقدار بازگشتی صفر یعنی مشکل نیست و یک یعنی خطایی باعث خاتمه پردازش شده است. که تولید شده چنین است :

```
MOV   AH,4CH
MOV   AL,return-value    ; اگر مقدار بازگشتی کد شود تولید خواهد شد
INT   21H
```

پیش پردازنده EXTRN / EXTERN

پیش پردازنده EXTRN(یا EXTERN در MASM 6.0) اسمبلر و لینکر ، را از متغییرهای داده و برچسب‌هایی که مرحله جاری اسمبل باز می‌گرداند و توسط ماجول دیگری تعریف شده است. آگاه می‌کند.

قالب کلی آن چنین است : `EXTRN/EXTERN name : type [, . . .]`

name عنصری است که در دیگر قسمت‌های اسمبل تعریف شده است و بصورت PUBLIC مشخص شده است. شناسه type می‌تواند به هر یک از موارد زیر اشاره کند :

- عناصرداده: ABS (یک‌ثابت)، BYTE، WORD، DWORD، FWORD، QWORD، TBYTE و EXTRN را در سگمنتی که عناصر رخ می‌دهد قرار دهید.
- فاصله : NEAR یا FAR. NEAR را در سگمنتی که عنصر رخ می‌دهد و FAR را در هر جای دلخواه می‌توانید استفاده کنید.

در مثال بعدی، برنامه فراخوانده CONVAL را بصورت PUBLIC و با DW تعریف می‌کند. برنامهٔ فراخوانده شده CONVAL را بصورت EXTRN و FAR (در دیگر سگمنت) مشخص می‌سازد.

برنامهٔ فراخوان :

```
DSEG1   SEGMENT
        PUBLIC CONVAL
        . . .
CONVAL  DW ?
        . . .
DSEG1   ENDS
```

برنامهٔ فراخوانده شده :

```
        EXTRN CONVAL:FAR
DSEG2   SEGMNT
        . . .
        MOV
        AX,CONVAL
        . . .
DESG2   ENDS
```

برای مثالهایی از EXTRN، فصل ۲۳ را ببینید.

پیش پردازندهٔ FARDATA. و .? FARDATA

این پیش‌پردازنده‌های سگمنت ساده شده، سگمنت داده را تعریف می‌کنند. FARDATA. یک سگمنت برای داده‌های دور مقدار دهی شده تعریف می‌کند و ? FARDATA. یک سگمنت برای داده‌های دور مقدار دهی نشده تعریف می‌کند. برای یک برنامه اسمبلی مستقل، ممکن است داده دور مقدار دهی نشده را در سگمنت FARDATA. تعریف کنید. (پیش پردازنده‌های DATA. و MODEL. را نیز ببینید).

پیش پردازندهٔ GROUP

یک برنامه ممکن است شامل چندین سگمنت با نوع مشابه باشد (کد، داده یا پشته). هدف از پیش پردازندهٔ GROUP جمع آوری سگمنت‌هایی با نوع مشابه تحت یک نام است، بنابراین داخل یک سگمنت اغلب یک سگمنت

داده بازنویسی می‌شود، قالب کلی چنین است : `name GROUP seg-name [,seg-name] . . . .`

GROUP زیرا DSEG1 و DSEG2 را در یک ماجول اسمبلی ترکیب می‌کند :

```
GROUPX    GROUP     DSEG1,DSEG2
DSEG1     DSEGMENT  PARA,'Data'
          ASSUME    DS,GROUPX
          . . .
DSEG1     ENDS
DSEG2     DSEGMGNT  PARA 'Data'
          ASSUME    DS:GROUPX
          . . .
DSEG2     ENDS
```

تأثیر استفاده از GROUP مشابه با تخصیص یک نام و صفت PUBLIC به سگمنت‌ها است.

پیش پردازندهٔ INCLUDE

اگر بخشی از کد اسمبلی یا دستورات ماکرو را در اختیار دارید که برنامه‌های مختلف می‌توانند استفاده کنند، می‌توانید آن را در یک فایل‌های دیسکی مجزا ذخیره کنید تا توسط دیگر برنامه‌ها استفاده شود. فرض کنید یک روتین، کد ASCCII را به دودویی تبدیل می‌کند که در درایو : E تحت نام CONVERT.LIB قرار گرفته است. برای دستیابی به فایل، یک جمله INCLUDE مانند زیر : INCLUDE E:CONVERT.LIB

را در موقعیتی که روتین تبدیل ASCII وجود دارد قرار دهید. اسمبلر فایل را روی دیسک قرار می‌دهد و برنامه شما را شامل جملات روتین می‌سازد. اگر اسمبلر نتواند فایل را پیدا کند، یک پیغام خطا صادر می‌کند و از INCLUDE صرف نظر می‌کند).

برای هر خط INCLUD اسمبلر یک C (بسته به نسخه اسمبلر) در ستون 30 از فایل LST. چاپ می‌کند و کد منبع را از ستون ۳۳ آغاز می‌نماید.

فصل ۲۲ یک مثال عملی از INCLUDE را ارائه می‌دهد و نحوه استفاده از پیش پردازنده فقط در مرحله یک از اسمبل کردن را توضیح می‌دهد.

پیش پردازندهٔ LABEL

پیش پردازندهٔ LABEL امکان تعریف مجدد صفت یک متغییر داده یا برچسب دستور را فراهم می‌سازد. قالب کلی آن چنین است : `name LABEL type-specifier`

برای برچسب‌ها، امکان استفاده از LABEL برای تعریف مجدد کد اجرایی بصورت NEAR ، FAR یا PROC، مانند نقطهٔ ورود دوم به روال وجود دارد. برای متغیرها، می‌توانید از شاخص نوع BYTE، WORD، DWORD، FWORD، QWORD یاTBYTE یا یک نام ساختار جهت تعریف مجدد عناصر داده یا نام ساختارها استفاده کنید. برای مثال، می‌توانید با استفاده از LABEL یک فیلد را بصورت DB یا DW تعریف کنید.

مثالهای زیر انواع BYTE و WORD را مشخص می‌سازند :

```
BYTE1   LABEL   BYTE      ; تعریف اولین بایت بصورت BYTE1
WORD1   DW      2532H     ; تعریف اولین دو بایت بصورت WORD1
WORD2   LABEL   WORD      ; تعریف سومین‌وچهارمین بایت بصورت WORD2
BYTE2   DB      25H       ; تعریف سومین بایت بصورت BYTE2
        DB      32H       ; تعریف چهارمین بایت بصورت BYTE2
        . . .
        MOV     AL,BYTE1  ; انتقال اولین بایت :
        MOV     BX,WORD2  ; انتقال‌سومین‌وچهارمین‌بایت :
```

اولین دستور MOV فقط اولین بایت WORD1 را منتقل می‌کند. دومین MOV دو بایت با شروع از BYTE2 را منتقل می‌کند. عملگر PTR عملی مشابه انجام می‌دهد.

مثال بعد از LABEL با عملگر NEAR استفاده می‌کند. اگرچه روش معمول کدنویسی یک برچسب نزدیک با یک پیشوند (:) است، بصورت A20CALC اما می‌توانید برچسب را بصورت A20ACLC NEAR LABEL کد نمایید.

پیش پردازنده .LIST

پیش پردازنده .LIST (پیش فرض و شناخته شده بصورت %LIST در مدل ایده‌آل (TASM) سبب می‌شود تا اسمبلر برنامهٔ منبع را لیست نماید. ممکن است بلوکی از کد داشته باشید که بدلیل مشترک بودن در دیگر برنامه‌ها نیازی به لیست‌گیری آنها نباشد. در این حالت، با استفاده از .XLIST (.NOLIST) لیست‌گیری ادامه نخواهد یافت و سپس با استفاده از .LIST لیست‌گیری مجدد آغاز خواهد شد. از این پیش پردازنده‌ها بدون عملوند استفاده کنید.

پیش پردازندهٔ .MODEL

این پیش پردازندهٔ سگمنت ساده شده سگمنت‌های پیش فرض و ASSUME مورد نیاز و جمله GROUP را تولید می‌کند. قالب کلی آن چنین است : `.MODEL memory-model`

مدلهای حافظه بشرح زیر می‌باشند :

TINY	از MASM 6.0 و TASM 4.0 برای برنامه‌های COM. استفاده می‌شود.
SMALL	همهٔ داده‌ها در یک سگمنت و همهٔ کد در یک سگمنت است.
MEDIUM	همه داده‌ها در یک سگمنت، اما کد در چندین سگمنت است.
COMPACT	داده‌ها در چندین سگمنت، اما کد در یک سگمنت است.
LARGE	هم داده و هم کد در چندین سگمنت هستند، اما آرایه بیشتر از 64K نمی‌تواند باشد.
HUGE	هم داده و هم کد در چندین سگمنت هستند، اما آرایه بیشتر از 64K می‌تواند باشد.

پیش پردازندهٔ .STACK، پشته را تعریف می‌کند، .CODE، سگمنت کد را تعریف می‌کند و هریک .DATA، .DATA?، .FARDATA، .FADATA؟ سگمنت داده را ممکن است تعریف کنند.

در اینجا یک مثال آورده شده است :

```
.MODEL   SMALL
.STACK   120
.DATA
         [data items]
.CODE
         [instructions]
END
```

در مدل ایده‌آل TASM، پیش پردازنده .MODEL است (بدون نقطه) و سگمنت‌ها .CODESEG، .DATASEG، .UDATASEG، .FARDATA، .UFARDATA است.

پیش پردازنده .NOLIST. (پیش پردازندهٔ .XLIST. را ببینید).

پیش پردازنده .ORG. یک سگمنت داده با تعاریف زیر در نظر بگیرید.

شمارشگر موقعیت	عملوند	عملیات	نام	افست
02	2542H	DW	WORD1	00
03	36H	DB	BYTE1	02
05	212EH	DW	WORD2	03
09	0000705H	DD	BYTE2	05

در ابتدا، شمارشگر موقعیت اسمبلر با 00 تنظیم می‌شود. چون WORD1، ۲ بایت است، شمارشگر موقعیت ۲ واحد افزوده می‌شود تا در موقعیت عنصر بعدی قرار گیرد. چون BYTE1، یک بایت است، شمارشگر موقعیت افزوده شده و

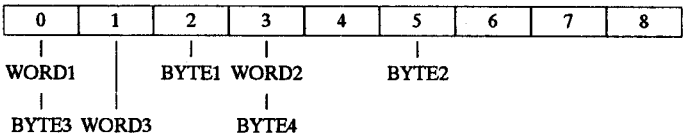
03 خواهد شد و الی آخر. ممکن است با استفاده از پیش پردازنده ORG محتویات شمارشگر موقعیت را افزوده و بر طبق آن در موقعیت عنصر موقعیت بعدی قرار دهید. قالب کلی آن چنین است :

```
ORG expression
```

expression باید یک عدد ۲ بایتی باشد و نباید یک نام سمبولیک باشد. فرض کنید عناصر دادهٔ زیر بلافاصله پس از BYTE 2 در تعریف ما قبل تعریف شده‌اند :

شمارشگر موقعیت	عملوند	عملیات	نام	افست
00	0	0RG		
01	?	DB	BYTE3	00
02	?	DW	WORD3	01
04	?	DB	BYTE4	03
09	$ + 5	ORG		

اولین ORG شمارشگر موقعیت را با صفر تنظیم می‌کند. متغیرهای پس از آن، BYTE3، BYTE4 و WORD3 - موقعیت‌های حافظه تعریف شده بصورت WORD1، BYTE1 و WORD2 را مجدداً تعریف می‌کند :



یک عملوند که شامل یک سمبول دلار ($) است، بصورتی که در آخرین ORG دیده می‌شود به مقدار جاری در شمارشگر موقعیت اشاره دارد. بنابراین عملوند 5 + $ شمارشگر موقعیت را به 5 + 04 یا 09 تنظیم می‌کند، که همان تنظیم بعد از تعریف BYTE2 است.

ارجاع به WORD2 به یک فیلد یک کلمه‌ای در افست 03 است و یک ارجاع به BYTE4، ارجاع به یک فیلد یک بایتی در افست 03 است.

```
MOV AX,WORD2 ; یک کلمه
MOV AL,BYTE4 ; یک بایت
```

وقتی از ORG برای تعریف مجدد موقعیت‌های حافظه استفاده کنید، از تنظیم مجدد شمارشگر موقعیت با مقدار صحیح مطمئن شوید و برای تعریف مجدد موقعیت‌های حافظه حساب آن را داشته باشید. همچنین متغیرهای مجدداً تعریف شده نباید شامل ثابت‌های تعریف شده باشند، زیرا مقادیر جدید روی مقادیر اولیه قرار خواهند گرفت. از شبه عمل ORG در داخل یک STRUC نمی‌توان استفاده کرد.

پیش پردازندهٔ %OUT/ECHO

این پیش‌پردازندهٔ به اسمبلر می‌گوید تا یک پیغام را به ابزار خروجی استاندارد (اغلب صفحه، نمایش) بفرستد.(از MASM 6.0 نام آن ECHO شده است.) قالب کلی آن چنین است.

```
%OUT/ECHO massage
```

بخش پیش پردازندهٔ ERR. یک مثال ارائه می‌دهد.

پیش پردازندهٔ PAGE

پیش پردازندهٔ PAGE در شروع یک برنامه منبع حداکثر تعداد خط یک لیست در صفحه و حداکثر تعداد کاراکتر روی یک خط را مشخص می‌سازد. قالب کلی آن چنین است :

```
PAGE [length],width]
```

مثال زیر ۶۰ خط در هر صفحه و ۱۳۲ کاراکتر در خط را تنظیم می‌کند

```
PAGE 60,132
```

تعداد خط هر صفحه در دامنهٔ ۱۰ تا ۲۵۵ و تعداد کاراکترهای هر خط در دامنه ۶۰ تا ۱۳۲ ممکن است قرار گیرد. حذف جمله PAGE سبب می‌شود تا اسمبلر PAGE 50,80 را در نظر بگیرید. برای اجبار در رد کردن یک صفحه در خط خاص، مانند انتهای یک سگمنت، PAGE را بدون هیچ عملوندی کد نویسی کنید. در مدل ایده‌آل TASM ، PAGE با عملوند PAGESIZE% و PAGE بدون عملوند NEWPAGE% است.

پیش پردازندهٔ PROC

یک روال بلوکی از کد است که با پیش پردازندهٔ PROC آغاز و با پیش پردازندهٔ ENDP خاتمه می‌یابد. اگر چه بطور تکنیکی امکان ورود به یک روال توسط دستور JMP وجود دارد، روش معمول، استفاده از CALL برای ورود و RETN یا RETF برای بازگشت است. عملوند CALL ممکن است شاخص نوع NEAR یا FAR باشد. یک روال که در همان سگمنت روال فراخوان است یک روال NEAR محسوب شده و توسط افست دستیابی می‌شود.

```
proc-neme PROC [NEAR]
```

با حذف عملوند از پیش فرض NEAR استفاده می‌شود. اگر یک روال فراخوانی شده خارج از سگمنت روال فراخوان باشد، باید بعنوان PUBLIC مشخص شده و از CALL برای ورود به آن استفاده کنید. در یک برنامهٔ EXE.، PROC اصلی که نقطه ورود برای اجرا است باید از نوع FAR باشد. در ضمن روال فراخوانی شده تحت یک مقدار متفاوت ASSUME CS باید صفت FAR داشته باشد :

```
            PUBLIC   proc-name
proc-name   PROC     FAR
```

یک برچسب دور ممکن است در سگمنت دیگری باشد، که CALL با افست و آدرس سگمنت به آن دست می‌یابد. مدل ایده‌آل TASM ، PROC و ENDP را چنین تعریف می‌کند :

```
PROC proc-name [NEAR/FAR]
. . .
ENDP [proc-name
```

پیش پردازندهٔ Processor

این پیش پردازنده، پردازشگری را که اسمبلر تشخیص می‌دهد، تعریف می‌کند. جای معمول این پیش پردازندهٔ در شروع برنامه منبع است، اگر چه می‌توانید داخل برنامه، در نقطه‌ای که می‌خواهید یک خصیصهٔ پردازشگر را فعال یا غیر فعال کنید، آن را کد نویسی کنید.

- 8086. مدل 8086/8088 و پردازشگر 8087 را فعال می‌سازد (حالت پیش فرض).
- 186. ، 286. ، 386. ، 486. و 586. تمام دستوراتی که، برای پردازشگرهای نامبرده است و پردازشگرهای وابسته، فعال می‌کند. که هر پیش پردازنده دستورات پردازشگرهای ما قبل را نیز فعال می‌سازد. (برای مثال 386.، همهٔ 387. ، 386. ، 186. و 8086 را فعال می‌سازد).
- 186P. ، 286P. ، 386P. ، 486P. و 586P. همهٔ تنظیمات دستورات پردازشگری که دقیقاً ذکر می‌شود بعلاوهٔ دستورات پردازشگرهای ماقبل را فعال می‌سازد. مدل ایده‌آل TASM از عبارات P8086 تا P586 و P8087 تا P587 استفاده می‌کند.

پیش پردازندهٔ PUBLIC

دستور PUBLIC، اسمبلر و لینکر را از انتقال علائم و یک برنامه اسمبل به دیگر ماجول‌هایی که به آن الحاق شده است آگاه می‌کند. قالب کلی آن چنین است. `PUBLIC symbol [, . . .]`

سمبول می‌تواند یک برچسب یک عدد (تا دو بایت) یا یک متغیر باشد. بخش پیش پردازندهٔ EXTRN و فصل ۲۳ را با مثالها ملاحظه کنید.

پیش پردازندهٔ RECORD

پیش پردازندهٔ RECORD به شما این توانایی را می‌دهد که الگوی بیت‌ها، مانند الگوی رنگ‌ها و شاخص سوئیچ‌ها را بصورت یک بیت یا چند بیت تعریف کنید، قالب کلی آن چنین است :

MASM : record-name RECORD field-name : width [=exp] , [, . . .]
TASM Ideal mode : RECORD name . . .

نام رکورد و نام فیلد ممکن است هر شناسه، معتبر منحصربفردی باشند. بعد از هر نام فیلد یک (:) است و یک عرض که تعداد بیت‌ها مشخص می‌سازد. محدودهٔ عرض از ۱ تا ۳۲ بیت است، طول تا ۸، ۸ بیت، ۹ تا ۱۶ بیت و ۱۷ تا ۳۲ خواهد شد ۳۲ بیت، که در صورت لزوم محتویات از راست تنظیم خواهد شد. مثال زیر از پیش پردازندهٔ RECORD برای تعریف BITREC استفاده می‌کند : BITREC RECORD BIT1:3, BIT2:7, BIT3:6

BIT1، اولین ۳ بایت از BITREC را تعریف می‌کند. BIT7 ۷ بیت بعدی را تعریف می‌کند و BIT3 آخرین ۶ بیت را تعریف می‌کند. مجموع آن ۱۶ بیت یا یک کلمه است. امکان مقدار دهی در یک رکورد بصورت زیر وجود دارد :

BITREC2 RECORD BIT1:3=101B, BIT2:7=0110110B, BIT3:6 = 011010B

توجه کنید که تعریف رکورد واقعاً حافظه‌ای را تولید نمی‌کند. بنابراین، پس از تعریف RECORD در سگمنت داده، شما باید جملهٔ دیگری کد نمایید تا حافظه را برای رکورد تخصیص دهید. یک نام معتبر منحصر بفرد، یک نام رکورد و یک عملوند شامل پرانتز زاویه‌ای (سمبولهای بزرگتر، کوچکتر) تعریف کنید :

DEFBITS BITREC < >

تخصیص برای DEFBITS کد مقصد AD9AH (ذخیره شده بصورت 9AAD) در سگمنت داده تولید می‌کند. پرانتز زاویه‌ای ممکن است شامل ورودیهایی باشد که BITREC را مجدداً تعریف می‌کنند.

برنامه شکل ۱-۲۶، BITREC را بصورت RECORD تعریف می‌کند، اما هیچ مقدار اولیه‌ای در رکوردهای فیلد قرار نمی‌دهد، در این حالت، یک جمله تخصیص دهی در سگمنت داده بصورتی که در پرانتز زاویه‌ای نشان داده شده است هر فیلد را ارزشدهی می‌کند. عملگرهای خاص رکورد WIDTH، شمارش شیفت و MASK می‌باشد. با استفاده از این عملگرها امکان تغییر تعریف RECORD را خواهید داشت، بدون آنکه اجباری در تغییر دستوری که به آن مواجه می‌کند داشته باشید.

عملگر WIDTH. عملگر WIDTH یک عرض بعنوان تعداد بیت‌ها در یک RECORD یا در یک فیلد RECORD باز می‌گرداند. در شکل ۱-۲۶، بعد از A20 دو مثال از WIDTH وجود دارند. اولین MOV عرض RECORD BITREC ورودی (۱۶ بیت) را باز می‌گرداند، دومین MOV عرض فیلد BIT2 رکورد را (۷ بیت) باز می‌گرداند. در هر دو حالت، اسمبلر یک عملوند بلافصل برای عرض تولید می‌کند.

شمارش شیفت. یک مراجعه مستقیم به یک فیلد RECORD مانند MOV CL,BIT2 به محتویات BIT2 دسترس نخواهد داشت. جای آن اسمبلر یک عملوند بلافصل تولید می‌کند که حاوی یک شیفت شمار است که به شما کمک می‌کند تا فیلد را تجزیه کنید. مقدار بلافصل تعداد بیت‌هایی را نشان می‌دهد که برای تغییر مکان BIT2 با تنظیم از راست

آن در دست دارید. در شکل ۱-۲۶ سه مثال بعد از A30، شیفت شمار BIT1 ، BIT2 و BIT3 را باز می‌گرداند. **عملگر MASK.** عملگر MASK یک پوشش از بیت‌های ۱ نشان دهنده میدان مشخص شده را باز می‌گرداند و در نتیجه، مکانهای بیت که میدان را اشغال می‌کنند، تعریف می‌کند. برای مثال، MASK برای هر یک از فیلدهای تعریف شده در BITREC به صورت زیر می‌باشند.

فیلد	دودویی	HEX
BIT1	1110000000000000	E000
BIT2	0001111111000000	IFC0
BIT3	0000000000111111	003F

در شکل ۱-۲۶، سه دستور متعاقب A20 مقادیر MASK را برای BIT1، BIT2، BIT3 باز می‌گرداند. دستورات پس از A50 ، A60 ، BIT2 و BIT1 را بترتیب از BITREC مجزا می‌سازد. A50 رکورد را در ثبات AX می‌گذارد و سپس آن را با استفاده از یک پوشش BIT2 ،AND می‌کند.

```
                    101 0110110  011010          رکورد :
                    000 1111111  000000  : AND MASK BIT2
                    000 0110110  000000          حاصل :
```

```
                TITLE   A26RECOR (COM) Test of RECORD Directive
0000            CODESG  SEGMENT PARA 'Code'
                        ASSUME CS: CODESG,DS:CODESG,SS:CODESG
0100                    ORG     100H
0100 EB 02      BEGIN:  JMP     SHORT A10MAIN
                ;-------------------------------------------------------
                BITREC RECORD BIT1:3,BIT2:7,BIT3:6          ;Define record
0102 AD9A       DEFBITS BITREC <101B,0110110B,011010B>      ;Init. record
                ;-------------------------------------------------------
0104            A10MAIN PROC NEAR
0104            A20:                                    ;Width:
0104 B7 10              MOV  BH,WIDTH BITREC            ;  of record (16)
0106 B0 07              MOV  AL,WIDTH BIT2              ;  of field (07)
0108            A30:                                    ;Shift count:
0108 B1 0D              MOV  CL,BIT1                    ;  hex 0D
010A B1 06              MOV  CL,BIT2                    ;       06
010C B1 00              MOV  CL,BIT3                    ;       00
010E            A40:                                    ;Mask:
010E B8 E000            MOV  AX,MASK BIT1               ;  hex E000
0111 BB 1FC0            MOV  BX,MASK BIT2               ;       1FC0
0114 B9 003F            MOV  CX,MASK BIT3               ;       003F
0117            A50:                                    ;Isolate BIT2:
0117 A1 0102 R          MOV  AX,DEFBITS                 ;  get record
011A 25 1FC0            AND  AX,MASK BIT2               ;  clear BIT1 & 3
011D B1 06              MOV  CL,BIT2                    ;  get shift 06
011F D3 E8              SHR  AX,CL                      ;  shift right
0121            A60:                                      ;Isolate BIT1:
0121 A1 0102 R          MOV  AX,DEFBITS                 ;  get record
0124 B1 0D              MOV  CL,BIT1                    ;  get shift 13
0126 D3 E8              SHR  AX,CL                      ;  shift right
0128 B8 4C00            MOV  AX,4C00H                   ;End processing
012B CD 21              INT  21H
012D            A10MAIN ENDP
012D            CODESG  ENDS
                        END BEGIN
```

شکل الف ۱-۲۶ استفاده از پیش پردازندهٔ *RECORD*

```
Structures and Records:
 N a m e                  Width    # fields
                          Shift    Width   Mask    Initial
BITREC . . . . . .        0010     0003
 BIT1  . . . . . .        000D     0003    E000     0000
 BIT2  . . . . . .        0006     0007    IFC0     0000
 BIT3  . . . . . .        0000     0006    003F     0000
Segments and Groups:
 N a m e                  Length   Align   Combine  Class
CODESG . . . . . .        012D     PARA      NONE   'CODE'
Symbols:
 N a m e                   Type    Value    Attr
A10MAIN . . . . . .       N PROC   0114     CODESG  Length = 0029
A20 . . . . . .           L NEAR   0104     CODESG
A30 . . . . . .           L NEAR   0108     CODESG
A40 . . . . . .           L NEAR   010E     CODESG
A50 . . . . . .           L NEAR   0117     CODESG
A60 . . . . . .           L NEAR   0121     CODESG
BEGIN . . . . . .         L NEAR   0100     CODESG
BIT1 . . . . . .          000D
BIT2 . . . . . .          0006
BIT3 . . . . . .          0000
DEFBITS . . . . . .       L WORD   0102     CODESG
```

شکل ب ۱-۲۶ استفاده از پیش پردازندهٔ *RECORD*

تاثیر آن صفر کردن همه بیت‌ها بجز BIT2 است. دو دستور بعدی سبب می‌شوند تا AX شش بیت به راست تغییر مکان یابد به طوری که BIT2 به سمت راست تنظیم می‌شود.

0000 0000 0011 0110 (0036H

مثال بعد از A60 رکورد را در AX قرار می‌دهد و از آنجا که BIT1 در سمت چپ تنظیم است، روال به سادگی فاکتور تغییر مکان آن را برای تغییر مکان دادن ۱۳ بیت به سمت راست استفاده می‌کند

0000 0000 0000 0101 (0005H)

پیش پردازندهٔ SEGMENT

یک ماجول اسمبلی شامل یک یا چند سگمنت، بخشی از یک سگمنت، یا حتی بخشی از چندین سگمنت است. قالب کلی برای سگمنت چنین است :

```
seg-name    SEGMENT [align] [combine] ['class']
            ...
seg-name    ENDS
```

حالت ایده‌آل TASM از قالب seg-name SEGMENT استفاده می‌کند.

همه عملوندها اختیاری‌اند. بخش‌های زیرین ورودیهای align ، combine و class را توصیف می‌کند.

Align. عملوند align مبنای مرز شروع یک سگمنت است.

BYTE آدرس بعد.

WORD آدرس زوج بعد (قابل تقسیم بر ۲).

DWORD آدرس دو کلمه‌ای بعد (قابل تقسیم بر ۴).

PARA پاراگراف بعد (قابل تقسیم بر ۱۶، یا 10H).

PAGE آدرس صفحهٔ بعد (قابل تقسیم بر ۲۵۶ یا 100H).

PARA عموماً برای همه انواع سگمنت‌ها استفاده می‌شود. BYTE و WORD می‌تواند برای سگمنت‌هایی که داخل سگمنت دیگر، عموماً سگمنت داده ترکیب می‌شوند، مورد استفاده قرار گیرد. DWORD عموماً برای برنامه‌هایی که تحت 80386 و پردازشگرهای ما بعد اجرا می‌شود، مورد استفاده است.

Combine. عملوندهای ترکیب NONE , PUBLIC , STACK , COMMON روشی را که لینکر یک سگمنت را دستیابی می‌کند، مشخص می‌سازد.

- NONE (پیش فرض) سگمنت بطور منطقی از دیگر سگمنت‌ها مجزاست، اگر چه ممکن است در مجاورت فیزیکی دیگری پایان یابد. فرض می‌شود که سگمنت دارای آدرس پایه خودش می‌باشد.
- PUBLIC مرحلهٔ LINK همهٔ سگمنت‌های PUBLIC که یک نام و کلاس مشابه دارند در مجاورت یکدیگر قرار می‌دهد. یک آدرس پایه برای همهٔ چنین سگمنت‌هایی PUBLIC فرض می‌شود.
- STACK مرحلهٔ LINK با STACK مانند PUBLIC رفتار می‌کند. حداقل یک STACK تعریف شده در یک برنامهٔ EXE. پیوند زده باید باشد. اگر چندین پشته باشد، SP با شروع اولین پشته تنظیم می‌شود.
- COMMON اگر سگمنت‌های COMMON، نام و کلاس یکسانی داشته باشند ، لینکر به آنها آدرس پایه یکسانی می‌دهد. در طی اجرا، دومین سگمنت بر روی اولین قرار می‌گیرد. بزرگترین سگمنت، حتی اگر بر روی آن سگمنتی قرار گرفته باشد، طول ناحیه مشترک را مشخص می‌سازد.
- آدرس پاراگراف AT : پاراگراف باید قبلاً تعریف شده باشد. وسیلهٔ ورودی برچسب‌ها و متغیرها در افست ثابت داخل ناحیه‌ای از حافظه را تعریف می‌کنند، مانند جدول وقفه در حافظه پایینی یا ناحیه داده BIOS در 40[0]H برای مثال، کد موجود در ROM موقعیت ناحیه نمایش ویدئو را بصورت زیر مشخص می‌سازد :

```
VIDEO-RAM SEGMENT AT 0B800H
```

اسمبلر یک سگمنت فرض ایجاد می‌کند، که در واقع تصویری از موقعیت‌های حافظه را فراهم می‌سازد.

'class' ورودی کلاس می‌تواند به لینکر کمک کند تا سگمنت‌هایی که نام‌های مختلف دارند را وابسته کرده سگمنت‌ها را تشخیص دهد، و ترتیب آنها را کنترل نماید. کلاس شامل هر نام معتبر است و داخل علامت نقل قول قرار می‌گیرد لینکر از نام برای مرتبط کردن سگمنت‌هایی که نام و کلاس یکسانی دارند استفاده می‌کند. نمونه‌ای از مثال 'Data' و 'Code' می‌باشد. اگر شما یک کلاس را بصورت Code تعریف کنید، لینکر فرض می‌کند که سگمنت شامل کد دستورات می‌باشد. همچنین اشکال زدایی CODEVIEW کلاس Code را برای سگمنت کد در نظر می‌گیرد. لینکر دو سگمنت زیربا نام مشابه (CSEG) و کلاس مشابه ('Code') را دریک سگمنت فیزیکی تحت همان‌ثبات سگمنت ترکیب‌می‌کند.

```
-----------------------------------------
 Assembly  CSEG  SEGMENT PARA PUBLIC 'Code'
 module 1        ASSUME  CS:CSEG
                 ...
           CSEG  ENDS
-----------------------------------------
 Assembly  CSEG  SEGMENT PARA PUBLIC 'Code'
 module 2        ASSUME  CS:CSEG
                 ...
           CSEG  ENDS
-----------------------------------------
```

از آنجائیکه ممکن است بخواهید ترتیب سگمنت‌های داخل یک برنامه را کنترل نمایید، برای درک چگونگی دستکاری پردازش، این مطلب مفید است. ترتیب اولیه نام‌های سگمنت، ترتیب اصلی را فراهم می‌سازد، که ممکن است توسط صفت PUBLIC و نام‌های کلاس بر روی آن قرار دهید. مثال زیر دو ماجول مقصد (هر دو ماجول شامل یک سگمنت با نام DSEG1 با صفت PUBLIC و نام کلاس یکسان است) برای پیوند زدن را نشان می‌دهد :

```
module 1  SSEG   SEGMENT  PARA STACK
module 1  DSEG1  SEGMENT  PARA PUBLIC 'Data'
module 1  DSEG2  SEGMENT  PARA
module 1  CSEG   SEGMENT  PARA 'Code'
module 2  DSEG1  SEGMENT  PARA PUBLIC 'Data'
module 2  DSEG2  SEGMENT  PARA
module 2  CSEG   SEGMENT  PARA 'Code'
```

بعد از آن ماجول OBJ. پیوند زده شد، ماجول EXE. چنین خواهد بود :

```
module 1        CSEG   SEGMENT PARA 'Code'
module 2        CSEG   SEGMENT PARA 'Code'
modules 1 + 2   DSEG1  SEGMENT PARA PUBLIC 'Data'
module 1        DSEG2  SEGMENT PARA
module 2        DSEG2  SEGMENT PARA
module 1        SSEG   SEGMENT PARA STACK
```

در صورتی که یک سگمنت لانه‌ای کاملاً در داخل دیگری باشد، می‌توانید سگمنت‌های لانه‌ای داشته باشید.

```
DSEG1 SEGMENT
  . . .          DSEG1 begins
DSEG2 SEGMENT
  . . .          DSEG2 area
DSEG2 END
  . . .          resumes DSEG1
DSEG1 ENDS
```

پیش پردازنده‌های ALPHA. ، SEQ. و DOSSEG و انتخاب‌های A/ و S/ اسمبلر نیز می‌توانند ترتیب سگمنت‌ها کنترل نمایند. (برای ترکیب سگمنت‌ها در گروه‌ها، پیش پردازندهٔ GROUP را نیز ملاحظه کنید.)

پیش پردازندهٔ SEQ.

این پیش پردازندهٔ (پیش فرض) در شروع یا حوالی شروع یک برنامه قرار می‌گیرد، به اسمبلر می‌گوید که سگمنت‌ها را به ترتیب اولیه خود رها سازد. (برخی اسمبلرهای اخیر سگمنت‌ها را به ترتیب حروف الفبا مرتب می‌کنند.) همچنین می‌توانید از انتخاب A/ خط فرمان اسمبلر استفاده کنید. (پیش پردازنده‌های ALPHA. و DOSSEG را ببینید).

پیش پردازندهٔ STACK.

این پیش پردازندهٔ سگمنت ساده شده پشته را تعریف می‌کند. قالب کلی آن چنین است :

```
.STACK [size]
```

اندازهٔ پیش فرض پشته ۱۰۲۴ بایت است، که ممکن است بر روی آن بازنویسی کنید. (پیش پردازندهٔ MODEL. را نیز ببینید.

پیش پردازندهٔ STARTUP.

می‌توانید با استفاده از این پیش پردازنده در شروع سگمنت کد، ثبات‌های DS، SS، SP، را مقدار دهی کنید. مدل ایده‌آل TASM از عبارت STARTUPCODE استفاده می‌کند. پیش پردازندهٔ EXIT. را نیز ببینید.

پیش پردازندهٔ STRUC / STRUCT

پیش پردازندهٔ STRUC (در MASM 6.0 , STRUCT) تعریف فیلدهای مرتبط یک ساختار را سهولت می‌بخشد. قالب کلی آن چنین است :

```
Struc-name    STRUC/STRUCT
              . . .
              [defined fields]
              . . .
struc-name    ENDS
```

یک ساختار با نام خودش و پیش‌پردازندهٔ STRUC آغاز می‌شود و با نام و پیش‌پردازندهٔ ENDS خاتمه می‌یابد. اسمبلر، فیلدهای تعریف شده را یکی پس از دیگری از شروع ساختار تشخیص می‌دهد. ورودیهای معتبر، تعاریف DB ، DW ، DD ، DQ و DT با نام‌های فیلدهای انتخابی است.

حالت ایده‌آل TASM از قالب STRUC struc-name استفاده می‌کند.

در شکل ۲-۲۶، STRUC یک لیست پارامتر با نام PARLIST تعریف می‌کند که برای INT 21H تابع 0AH جهت دارد یک نام از صفحه کلید استفاده می‌شود. توجه کنید که (مشابه پیش پردازندهٔ STRUC (RECORD واقعاً هیچ حافظه‌ای را تولید نمی‌کند. بنابراین، یک جمله تخصیص دهی برای تخصیص دادن حافظهٔ ساختار مورد نظر است که آن را داخل برنامه قابل آدرسدهی نماید. PARAMS PARLIST < >

پرانتز زاویه‌ای (سمبولهای بزرگتر کوچکتر) عملوند، در این مثال خالی است، اما امکان استفاده از آنها برای تعریف مجدد (یا بازنویسی) داده داخل یک ساختار وجود دارد.

دستورات ممکن است ساختاری را مستقیماً با آن ارجاع دهند. برای ارجاع فیلدهای داخل ساختار، دستورات باید آنها را با استفاده از نام اختصاص یافته ساختار (در این مثال (PARAMS) و به دنبالش یک نقطه که آن را با نام فیلد مرتبط می‌سازد، معرفی کنند. برای مثال، MOV AL ,PARAMS . ACTLEN همچنین می‌توانید از جمله تخصیص دهی (PARAMS در شکل ۲-۲۶) برای تعریف مجدد محتویات فیلد داخل یک ساختار استفاده کنید.

پیش پردازندهٔ SUBTTL / SUBTITLE

پیش پردازندهٔ SUBTTL (SUBTITLE در MASM 6.0 و SUBTTL% در حالت ایده‌آل TASM) سبب می‌شود تا یک زیر عنوان تا ۶۰ کاراکتر در خط ۳ هر صفحه لیست برنامه منبع چاپ می‌شود. امکان کد نویسی این پیش پردازنده تا چندین بار وجود دارد. قالب کلی آن چنین است.

```
SUBTTL/SUBTTLE text
```

پیش پردازندهٔ TEXTEQU

قالب کلی برای این پیش پردازنده (توسط MASM 6.0 معرفی شده) چنین است :

```
TEXTEQU [text-item]
```

عنصر متنی عملوند می‌تواند یک رشته حرفی، یک ثابت که در اول آن % قرار دارد، یا یک رشته‌ای که یک تابع ماکرو باز می‌گرداند، باشد.

پیش پردازندهٔ TITLE

پیش پردازندهٔ TITLE (TITLE% در حالت ایده آل TASM) سبب چاپ یک عنوان حداکثر ۶۰ کاراکتری در خط ۲ از هر صفحه از لیست منبع خواهد شد. امکان کدنویسی TITLE یک بار درشروع وجود دارد که بر طبق قالب متن TEXT باشد.

پیش پردازندهٔ .XCREF/.NOCREF

پیش پردازندهٔ .XCREF (NOCREF در MASM 6.0) به اسمبلر می‌گوید تا از جدول ارجاع متقابل ممانعت نماید. قالب کلی آن چنین است.

```
.XCREF/.NOCREF [name [,name] . . .]
```

حذف عملوند سبب جلوگیری از همه ورودیهای جدول خواهد شد. امکان جلوگیری از برخی عناصر خاص ارجاع متقابل نیز وجود دارد. در اینجا مثالهایی از .XCREF و .CREF آورده شده است.

```
                                page 60,132
                        TITLE   A26STRUC (COM) Defining a structure
0000                    CODESG SEGMENT PARA 'Code'
                                ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
0100                            ORG     100H
0100 EB 30              BEGIN: JMP      SHORT MAIN
                        ;-----------------------------------------------
                        PARLIST STRUCT                  ;Parameter list
0000 19                 MAXLEN  DB      25              ;
0001 00                 ACTLEN  DB      ?               ;
0002 0019 [20]          NAMEIN  DB      25 DUP(' ')     ;
001B                    PARLIST ENDS

0102 19                 PARAMS  PARLIST <>              ;Allocate storage
0103 00
0104 0019 [20]
011D 57 68 61 74 20 69 PROMPT   DB      'What is the part no.?'
73 20 74 68 65 20
70 61 72 74 20 6E
6F 2E 3F
                        ;-----------------------------------------------
0132                    MAIN    PROC    NEAR
0132 B4 40                      MOV     AH,40H          ;Request display
0134 BB 0001                    MOV     BX,01
0137 B9 0015                    MOV     CX,21           ;Length of prompt
013A 8D 16 011D R               LEA     DX,PROMPT       ;Address of prompt
013E CD 21                      INT     21H
0140 B4 0A                      MOV     AH,0AH          ;Accept keyboard
0142 8D 16 0102 R               LEA     DX,PARAMS       ; input
0146 CD 21                      INT     21H
0148 A0 0103 R                  MOV     AL, PARAMS.ACTLEN
                        ;       ...                     ;Length of input
014B B8 4C00                    MOV     AX,4C00H        ;End processing
014E CD 21                      INT     21H
0150                    MAIN    ENDP
0150                    CODESG  ENDS
                                END     BEGIN

Structures and Records:
N a m e                 Width   # fields
                        Shift   Width    Mask      Initial
PARLIST . . . . . .001B 0003
 MAXLEN . . . . . .0000
 ACTLEN . . . . . .0001
 NAMEIN . . . . . .0002
Segments and Groups:
 N a m e                Length  Align    Combine   Class
CODESG . . . . . 0150    PARA     NONE      'CODE'
Symbols:
 N a m e                Type    Value    Attr
BEGIN . . . . . .L NEAR 0100    CODESG
MAIN . . . . . N PROC  0132    CODESG    Length = 001E
PARAMS . . . . . L      0102    CODESG
PROMPT . . . . . L BYTE 011D    CODESG
```

شکل ۲-۲۶ استفاده از یک ساختار

```
                                    ممانعت از ارجاع متقابل :   .XCREF
...
                                    بازیابی ارجاع متقابل :     .CREF
...
                                    ممانعت از ارجاع متقابل FLDA و FLDE ;   .XREF FLDA,FLDB
```

پیش پردازندهٔ XLIST / NOLIST.

امکان استفاده از پیش‌پردازندهٔ XLIST. (با نام NOLIST. در MASM 6.0 , NOLIST% در حالت ایده‌آل TASM) در هر جایی از برنامهٔ منبع جهت عدم لیست‌گیری برنامهٔ اسمبل شده وجود دارد. یک حالت جایی است که جملات با دیگر برنامه‌ها مشترک هستند و نیازی به لیست‌گیری مجدد نیست، با پیش پردازندهٔ LIST. (پیش فرض) لیست‌گیری مجدداً آغاز خواهد شد. از این پیش پردازنده بدون عملوند استفاده کنید.

فصل ۲۷

مجموعه دستورات کامپیوترهای شخصی

هدف : توضیح کد ماشین و فراهم ساختن توصیفی از مجموعه دستورات PC.

مقدمه

این فصل کد ماشین را توضیح می‌دهد و لیستی از دستورات سمبولیک با توضیحی از اهداف آنها را ارائه می‌دهد. خیلی از دستورات اهداف خاص دارند، به طوری که یک کد دستور زبان ماشین یک بایتی به آنها تخصیص داده شده است. نمونه‌هایی از این دستورات را در زیر مشاهده می‌کنید.

کد ماشین	دستور سمبولیک	توضیح
40	INC AX	افزایش AX
50	PUSH AX	قرار دادن AX روی پشته
C3	RET (short)	بازگشت کوتاه از روال
CB	RET (far)	بازگشت دور از روال
FD	STD	تنظیم پرچم جهت

هیچکدام از دستورات فوق ارجاع مستقیم به حافظه ندارند. دستوراتی که یک عملوند بلافصل، دو ثبات، یا یک ارجاع به حافظه را مشخص می‌سازند پیچیده‌ترند و دو یا چند بایت کد ماشین نیاز دارند. کد ماشین مقررات خاصی برای دلالت بر ثبات خاص و مقررات دیگری برای ارجاع حافظه توسط بایت حالت آدرسدهی دارد.

نشانه گذاری ثبات

دستوراتی که به یک ثبات مراجعه می‌کنند ممکن است شامل سه بیت باشند که مبنی بر ثبات خاص و W بیت مبنی بر طول بایت (0) یا کلمه (1) است. همچنین، فقط دستورات خاص ثبات سگمنت را دستیابی می‌کنند. شکل ۱-۲۷ نشانه گذاری کامل ثبات را نشان می‌دهد. برای مثال، مقدار بیت 000 یعنی AH اگر W بیت صفر باشد و اگر یک باشد به معنی AX است.

در اینجا کد سمبولیک و کد ماشین دستور MOV با یک عملوند بلافصل یک بایتی ذکر شده است :

```
MOV  AH,00      10110  100  00000000
                   |   |||
                   W  reg  = AH
```

در این حالت، اولین بایت از کد ماشین مبنی بر یک بایت (W=0) است و به ثبات AH (100) اشاره دارد. در اینجا یک دستور MOV ذکر شده است که شامل یک عملوند بلافصل یک کلمه‌ای است و کد ماشین تولید شده آن نیز آورده شده است :

```
MOV AH,00      10111  000  00000000 00000000
                 |   |||
                 W  reg  = AX
```

اولین بایت کد ماشین مبنی بر طول یک کلمه (W=1) است و به ثبات (000)AX اشاره دارد. برای دیگر دستورات W و ثبات، اشکال مختلف دیگری خواهند داشت.

General, Base, and Index Registers			Bits for Segment Registers	
Bits	w = 0	w = 1	000	ES
000	AL	AX/EAX	001	CS
001	CL	CX/ECX	010	SS
010	DL	DX/EDX	011	DS
011	BL	BX/EBX	100	FS
100	AH	SP	101	GS
101	CH	BP		
110	DH	SI		
111	BH	DI		

شکل ۱-۲۷

آدرسدهی بایت وضعیت

بایت وضعیت، وقتی باشد بایت دوم کد ماشین را اشغال می‌کند و شامل سه عنصر زیر است :

mod ۲ بیت وضعیت، که مقادیر 00 ، 01 و 10 به موقعیت حافظه اشاره دارد و ۱۱ به یک ثبات اشاره می‌کند.

reg یک سه بیت که به ثبات مراجعه می‌کند.

r/m یک سه بیت که به ثبات یا حافظه مراجعه می‌کند، r ثبات را مشخص می‌سازد و m مبنی بر آدرس حافظه است.

همچنین، اولین بایت کد ماشین ممکن است شامل یک بیت d باشد که مبنی بر جهت جریان (راست / چپ) می‌باشد در مثال زیر، جمع AX و BX است.

```
ADD BX,AX      00000011   11  011  000
                  ||     ||  |||  |||
                  dw    mod reg  r/m
```

d=1 یعنی mod(11) و reg(011) اولین عملوند را توصیف می‌کند و r/m (000) دومین عملوند را توصیف می‌کند. چون W=1 طولی یک کلمه است. بنابراین دستور AX(000) را با BX(011) جمع می‌کند. بایت دوم کد مقصد مبنی بر حالات آدرسدهی حافظه است. می‌توانید از DEBUG برای بررسی مثال این روش استفاده کنید : کد ماشین را بصورت DB E1 00 03 وارد کنید و سپس دستور U 100,101 را وارد کنید.

بیت‌های mod. دو بیت mod بین آدرسدهی ثبات‌ها و حافظه تفاوت می‌گذارد. در زیر اهداف آنها را توضیح می‌دهد :

00 بیت‌های r/m انتخاب آدرسدهی دقیق را ارائه می‌دهد، بدون بایت افست.

01 بیت‌های r/m انتخاب آدرسدهی دقیق را ارائه می‌دهد، یک بایت افست.

10 بیت‌های r/m انتخاب آدرسدهی دقیق را ارائه می‌دهد، دو بایت افست.

11 r/m یک ثبات را مشخص می‌سازد. بیت W (در بایت کد عملیاتی) مشخص می‌کند که آیا ارجاع به یک ثبات ۸، ۱۶ یا ۳۲ بیتی است.

بیت‌های Reg. سه بیت reg در ارتباط با بیت W طول واقعی را مشخص می‌سازد.

بیت‌های R/M. سه بیت r/m (حافظه / ثبات) در ارتباط با بیت‌های mod همانطوریکه در شکل ۲-۲۷ نشان داده شده است حالت آدرسدهی را مشخص می‌سازند.

r/m	mod=00	mod=01 or 10	mod=11 w=0	mod=11 w=1
000	BX+SI	DS:[BX+SI+disp]	AL	AX
001	BX+DI	DS:[BX+DI+disp]	CL	CX
010	BP+SI	SS:[BP+SI+disp]	DL	DX
011	BP+DI	SS:[BP+DI+disp]	BL	BX
100	SI	DS:[SI+disp]	AH	SP
101	DI	DS:[DI+disp]	CH	BP
110	Direct	SS:[BP+disp]	DH	SI
111	BX	DS:[BX+disp]	BH	DI

شکل ۲-۲۷

دستورات دو بایتی

دستور دو بایتی زیر BX را با AX جمع می‌کند.

```
ADD AX,BX      0000 0011  11  000 011
                        ||  ||  |||  |||
                       dw mod reg  r/m
```

d=1 reg بعلاوه W اولین عملوند را (AX) توصیف می‌کند و mod بعلاوه r/m بعلاوهٔ w دومین عملوند (BX) را مشخص می‌سازد.

W=1 طول یک کلمه است.

mod=1 دومین عملوند یک ثبات است.

reg=000 اولین عملوند ثبات AX است.

r/m=011 دومین عملوند ثبات BX است.

مثال بعدی AL را در BL ضرب می‌کند :

```
MUL BL      11110110  11  100  011
                    |  ||  |||  |||
                    w mod reg  r/m
```

پردازشگر فرض می‌کند که مضروب در صورتیکه مضروب فیه یک بایتی باشد در AL و در صورتیکه یک کلمه‌ای باشد در AX و اگر دو کلمه‌ای باشد در AX قرار دارد. طول (W=0) یک بایتی است، (mod(1 یک ثبات ارجاع می‌دهد و ثبات (r/m=011) در (011) BL است. Reg=100 در اینجا معنی ندارد.

دستورات سه بایتی

MOV زیر سه بایت کد ماشین تولید می‌کند :

```
MOV mem_word,AX  .  10100011 mmmmmmmm mmmmmmmm
                            ||
                           dw
```

یک انتقال از انباشتگر (AX یا AL) فقط به دانستن اینکه عملیات یک بایتی یا یک کلمه‌ای است، نیاز دارد در این مثال، W=1 یعنی یک کلمه است، و ۱۶ بیت AX مفهوم می‌شود. (AL در دومین عملوند سبب می‌شود که بیت W صفر شود). بایت‌های ۲ و ۳ شامل افست موقعیت حافظه است. استفاده در ثبات انباشتگر اغلب یک دستور با طول کوتاهتر و اجرای سریع‌تری به نسبت استفاده از دیگر ثبات‌ها ایجاد می‌کند.

دستورات چهار بایتی

دستور ۴ بایتی زیر AL را در یک موقعیت حافظه ضرب می‌کند :

```
MUL  mem-byte    11110110  00  100  110  mmmmmmmm mmmmmmmm
                           |   ||   |||  |||
                           w  mod  reg  r/m
```

برای این دستور، اگرچه reg، 100 است، مضروب AL در نظر گرفته می‌شود. MOD=00 مبنی بر ارجاع به حافظه است و r/m=110 به معنی ارجاع مستقیم به حافظه است، دو بایت بعدی افست موقعیت حافظه را فراهم می‌سازد.

مثال بعدی، دستور LEA را مشخص می‌سازد، که یک آدرس کلمه را معین می‌کند :

```
LEA  DX,mem      10001101  00  010  110  mmmmmmmm mmmmmmmm
                           ||  |||  |||
                          LEA mod  reg  r/m
```

Reg=010 ثبات DX را مشخص می‌سازد، mod=00 و r/m=110 مبنی بر ارجاع مستقیم به آدرس حافظه است و دو بایت بعدی افست این موقعیت را فراهم می‌سازد.

مجموعهٔ دستورات

این بخش حاوی مجموعه دستورات به ترتیب حروف الفبایی است، اگر چه دستورات مرتبط با یکدیگر برای سهولت بیشتر، گروه بندی شده‌اند. بعلاوه برای بحث قبلی بایت حالت و بیت طول، خلاصه‌های زیر مناسب است :

addr	آدرس موقعیت حافظه
addr-high	بایت سمت راست یک آدرس
addr-Low	بایت سمت چپ یک آدرس
data	عملوند بلافصل (اگر w=0، ۸ بیت اگر w=1، ۱۶ بیت)
data-high	بایت سمت راست یک عملوند بلافصل
data-Low	بایت سمت چپ یک عملوند بلافصل
disp	جابجایی (مقدار افست)
reg	ارجاع به یک حافظه

پردازشگرهای ۸۰۲۸۶ و ما بعد تعدادی دستور خاص را پشتیبانی می‌کنند که در اینجا آورده نشده است.
ARPL ، BOUND ، CLTS ، ENTER ، LAR ، LEAVE ، LGDT ، LIDT ، LLDT ، LMSW ، LSL ، LTR ، SGDT ، SIDT ، SLDT ، SMSW ، STR ، VERR ، VERW ، دستورات منحصر بفرد 80486 و ما بعد BSWAP ، INVD ، WBINVD و INVLPG نیز در اینجا آورده نشده‌اند.
خلاصه‌ای‌برای‌پرچم‌ها نیز چنین‌است: AF = معین، CF = نقلی، DF = جهت، IF = وقفه، OF = سرریز، PF = توازن، SF = علامت، IF = اجرای مرحله‌ای و ZF = صفر.

AAA : تنظیم ASCII بعد از جمع

عمل. مجموع دو بایت ASCII را در ثبات AL اصلاح می‌کند. اگر سمت راست‌ترین چهار بیت ثبات AL دارای مقداری بزرگتر از ۹ باشد یا پرچم AF دارای مقدار ۱ باشد، AAA یک واحد به AH می‌افزاید و پرچمهای AF و CF را با ۱ مقدار می‌دهد. دستور همواره سمت چپ‌ترین چهار بیت ثبات AL را با صفر پر می‌کند.

پرچم‌ها. بر CF و AF اثر می‌گذارد OF، PF، SF، ZF تعریف نشده‌اند.)

کد منبع. AAA (بدون عملوند)

کد مقصد. 00110111

AAD : تنظیم ASCII قبل از تقسیم

عمل. یک مقدار BCD غیر فشرده (مقسوم) در AX را قبل از تقسیم تنظیم می‌کند. AAD، AH را در ۱۰ ضرب می‌کند، حاصل را با AL جمع می‌کند و AH را پاک می‌کند حال مقدار دودویی در AX مساوی با مقدار BCD غیر فشرده اصلی است و برای یک عمل تقسیم آماده است.

پرچم‌ها. بر PF ، SF و ZF اثر می‌گذارد (AF ، CF و OF تعریف نشده‌اند.)

کد منبع. AAD (بدون عملوند).

کد مقصد. |11010101|00001010|

AAM : تنظیم ASCII بعد از ضرب

عمل. حاصل تولید شده در AL توسط MUL برای ضرب دو رقم BCD غیر فشرده را تنظیم می‌کند. AAM , AL را بر ۱۰ تقسیم می‌کند و خارج قسمت را در AH و باقیمانده را در AL ذخیره می‌کند.

پرچم‌ها. بر پرچم‌های PF ، SF و ZF اثر می‌گذارد (AF ، CF و OF تعریف نشده است).

کد منبع. AAM (بدون عملوند)

کد مقصد. |11010100|00001010|

AAS : تنظیم ASCII بعد از تفریق

عمل. بعد از تفریق (بعد از SUB) دو بایت ASCII و AL را تنظیم می‌کند. اگر مقدار چهار بیت سمت راست بزرگتر از ۹ باشد، یا اگر CF یک باشد، AAS از AL ۶ را کم می‌کند یک را از AH تفریق می‌کند و AF و CF را یک می‌کند. در غیر اینصورت AF و CF صفر می‌شوند. AAS همیشه چهار بیت سمت چپ را از AL را پاک می‌کند.

پرچم‌ها. بر پرچم‌های AF و CF اثر می‌گذارد. (OF ، PF ، SF و ZF تعریف نشده است.)

کد منبع. AAS (بدون عملوند)

کد مقصد. 00111111

ADC : جمع با نقلی

عمل. معمولاً در جمع دودویی چند کلمه‌ای برای انتقال یک بیت ۱ سرریز شده به مرحلهٔ بعدی محاسبات استفاده می‌شود. ADC محتویات CF (۰/۱) را با اولین عملوند جمع می‌کند و دومین عملوند را با اولین عملوند مانند ADD جمع می‌کند (SBB را نیز ببینید).

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد.

کد منبع. ADC register/memory,register/memory/immediate

کد مقصد. سه قالب دارد :

ثبات/ حافظه با ثبات : |000100dw|modregr/m|

بلافصل با انباشتگر : |0001010w|--- data ---| data if w=1|

بلافصل با ثبات / حافظه : |100000sw|mod010r/m|--- data ---|data if sw=01|

ADD : جمع اعداد دودویی

عمل. مقادیر دودویی را از حافظه، ثبات، یا مقدار بلافصل با یک ثبات جمع می‌کند یا مقادیر یک ثبات یا بلافصل را

با حافظه جمع می‌کند مقادیر باید بایت، کلمه یا دو کلمه‌ای باشند (۸۰۳۸۶ و ما بعد).

پرچم‌ها. بر AF ، CF ، OF ، DF ، SF و ZF اثر می‌گذارد.

کد منبع. ADD register/memory,register/memory/immediate

کد مقصد. سه قالب دارد :

ثبات یا حافظه با ثبات : |000000dw|modregr/m|

بلافصل با انباشتگر : |0000010w|--- data ---|data if w=1|

بلافصل با ثبات یا حافظه : |100000sw|mod000r/m|--- data ---|data if sw=01|

AND : AND منطقی

عمل. روی بیت‌های دو عملوند یک عمل AND منطقی انجام می‌دهد. دو عملوند، هر دو بایت یا هر دو کلمه، یا هردو دو کلمه‌ای (در ۸۰۳۸۶ و ما بعد) هستند که دستور AND دو عملوند را بیت به بیت با یکدیگر تطبیق می‌دهد. برای هر دو بیت ۱، یک بیت در اولین عملوند ۱ خواهد شد، در غیر اینصورت بیت صفر خواهد شد. (OR ، XOR و TEST را نیز ببینید.)

پرچم‌ها. بر CF(0) ، OF(0) ، PF ، SF و ZF اثر می‌گذارد. (AF تعریف نشده است.)

کد منبع. AND egister/memory,register/memory/immediate

کد مقصد. سه قالب دارد :

ثبات یا حافظه با ثبات : |001000dw|modregr/m|

بلافصل با انباشتگر : |0010010w|--- data ---|data if w=1|

بلافصل با ثبات یا حافظه : |100000sw|mod100r/m|--- data ---|data if sw=01|

BSR/BSF : پیمایش بیت به سمت جلو / پیمایش بیت معکوس (۸۰۳۸۶ و مابعد)

عمل. یک رشته را برای یافتن اولین بیت ۱ پیمایش می‌کند. BSF را از راست به چپ پیمایش می‌کند، و BSR از چپ به راست پیمایش می‌کند. دومین عملوند (۱۶ یا ۳۲ بیت) شامل رشته‌ای است که باید پیمایش شود. اگر یک بیت ۱ پیداشود، عمل موقعیت آن‌را دراولین ثبات عملوند بازمی‌گرداند و ZF را یک می‌کند، درغیراینصورت ZF صفرمی‌شود.

پرچم‌ها. بر ZF اثر می‌گذارد.

کد منبع. BSF/BSR register,register/memory

کد مقصد. BSF : |00001111|10111100|modregr/m|۲

BSR : |00001111|10111101|modregr/m|

BT/BTC/BTR/BTS : چک کردن بیت (۸۰۳۸۶ و مابعد)

عمل. یک بیت مخصوص را در CF کپی می‌کند. اولین عملوند شامل رشته‌های بیت است که بررسی می‌شود و دومین عملوند شامل مقداری است که بر موقعیت آن دلالت دارد. BT بیت را در CF کپی می‌کند. دیگر دستورات نیز بیت را کپی می‌کنند اما بر روی آن با این روش عمل می‌کنند : BTC بیت را با معکوس کردن مقدار آن در اولین عملوند، مکمل می‌کند. BTR بیت را با صفر کردن مجدداً تنظیم می‌کند، BTS بیت را با یک تنظیم می‌کند ارجاع به مقادیر ۱۶ و ۳۲ بیتی خواهد بود.

پرچم‌ها. بر CF اثر می‌گذارد.

کد منبع. BT/BTC/BTR/BTC register/memory,register/immediate

کد مقصد. دو قالب دارد :

بلافصل با ثبات: |00001111|10111010|mod***r/m |

بلافصل با حافظه یا ثبات : |00001111|10***010|modregr/m|

(*** یعنی 100=BT ، 111=BTC ، 110=BTR و 101=BTS)

CALL : فراخوانی یک روال

عمل. یک روال دور یا نزدیک را فرا می‌خواند. اسمبلر یک CALL نزدیک تولید می‌کند اگر روال فراخوانی شده NEAR باشد و اگر روال فراخوانی شده FAR باشد یک CALL دور تولید می‌کند. یک CALL نزدیک IP را بر روی پشته می‌گذارد (آدرس دستور بعدی)، سپس در IP آدرس افست مقصد را قرار می‌دهد. یک CALL دور CS را روی پشته قرار می‌دهد و یک اشاره‌گر بین سگمنت‌ها روی پشته می‌گذارد، سپس IP روی پشته قرار می‌دهد و در IP آدرس افست مقصد را قرار می‌دهد. در بازگشت، RETN یا RETF عکس مراحل فوق را انجام می‌دهد.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. CALL register/memory

کد مقصد. چهار قالب دارد :

مستقیم در سگمنت : |11101000|disp_low|disp_high|

غیر مستقیم در سگمنت : |11111111|mod010r/m|

غیر مستقیم بین سگمنت : |11111111|mod011r/m|

مستقیم بین سگمنت : |10011010|offset-low|offset-high|seg-low|seg-high|

CBW: تبدیل بایت به کلمه

عمل. یک بایت مقدار علامت دار را به یک کلمه علامت دار بسط می‌دهد بدینوسیله که علامت (بیت ۷) در AL را در بیت‌های AH کپی می‌کند. (CWD ، CWDE و CDQ را نیز ببینید.)

پرچم‌ها. هیچ اثری ندارد.

کد منبع. CBW (بدون عملوند)

کد مقصد. 10011000

CDQ : تبدیل دو کلمه‌ای به چهار کلمه‌ای (۸۰۳۸۶ و ما بعد)

عمل. یک مقدار علامت دار ۳۲ بیت را در یک مقدار علامت دار ۶۴ بیت بسط می‌دهد بدینوسیله که علامت (بیت ۳۱) EAX را در EDX کپی می‌کند. (CBW ، CWD و CWDE را نیز ببینید.)

پرچم‌ها. هیچ اثری ندارد.

کد منبع. CDQ (بدون عملوند)

کد مقصد. 10011001

CLC : صفر کردن پرچم نقلی

عمل. CF را صفر می‌کند. برای مثال، ADC یک بیت ۱ را جمع نمی‌کند (STC را نیز ببینید)

پرچم‌ها. CF (صفر خواهد شد)
کد منبع. CLC (بدون عملوند)
کد مقصد. 11111000

CLD : صفر کردن پرچم جهت

عمل. DF را صفر می‌کند، تا سبب شود عملیات رشته‌ای مانند MOVS پردازش را از چپ به راست انجام دهند (STD را نیز ببینید).
پرچم‌ها. DF (صفر خواهد شد)
کد منبع. CLD بدون عملوند
کد مقصد. 11111100

CLI : صفر کردن پرچم وقفه

عمل. IF را صفر می‌کند، وقفه‌های خارجی قابل پوشش را غیر فعال می‌کند. (STI را نیز ببینید).
پرچم‌ها. IF (صفر خواهد شد).
کد منبع. CLI (بدون عملوند)
کد مقصد. 11 111 010

CMC : مکمل کردن پرچم نقلی

عمل. CF را مکمل می‌کند. مقدار بیت CF معکوس خواهد شد بنابراین صفر تبدیل به یک و یک تبدیل به صفر خواهد شد.
پرچم‌ها. CF (معکوس می‌شود)
کد منبع. CMC (بدون عملوند)
کد مقصد. 11110101

CMP : مقایسه

عمل. محتویات دودویی دو فیلد داده را مقایسه می‌کند. CMP بطور داخلی دومین عملوند را از اولین کم می‌کند و پرچم صفر یا یک می‌کند، اما حاصل را ذخیره نمی‌کند. هر دو عملوند باید بایت، کلمه، یا دو کلمه‌ای (۸۰۳۸۶ و ما بعد) باشند. CMP امکان مقایسه ثبات، حافظه، یا مقدار بلافصل را با یک ثبات یا امکان مقایسه ثبات یا بلافصل را با حافظه، دارد. (CMP مقایسه عددی انجام می‌دهد، CMPS را برای مقایسهٔ رشته‌ای ببینید).
پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF تاثیر می‌گذارد.
کد منبع. CMP register/ memory , register/memory/ immediate
کد مقصد، سه قالب دارد :

ثبات یا حافظه با ثبات : |001110dw|modregr/m|
بلافصل یا انباشتگر : |0011110w|--- data ---|data if w=1|
بلافصل با ثبات یا حافظه : |100000sw|mod111r/m|--- data ---|data if w=0|

CMDSD/CMPSW/CMPSB/CMPS : مقایسه رشته‌ای

عمل. رشته‌هایی با هر طول در حافظه را مقایسه می‌کند. معمولاً یک پیشوند RFPn قبل از این دستورات قرار می‌گیرد، با حداکثر طول که در CX است. CMPSB بایت‌ها را مقایسه می‌کند، CMPSW کلمات را مقایسه می‌کند و CMPSD (۸۰۳۸۶ و ما بعد) دو کلمه‌ای‌ها را مقایسه می‌کند. آدرس اولین عملوند در DS:SI و آدرس دومین عملوند در ES:DI قرار دارد. اگر DF صفر باشد، عملیات از چپ به راست مقایسه می‌کند و SI و DI را برای هر بایت یک واحد، برای هر کلمه دو واحد و برای هر دو کلمه ۴ واحد می‌افزاید، اگر DF یک باشد، عملیات از راست به چپ مقایسه می‌کند و SI و DI را کاهش می‌دهد. REPn برای هر بار تکرار CX را یک واحد می‌کاهد. REPNE هنگامیکه اولین مورد موافق پیدا شد خاتمه می‌یابد، REPE هنگامیکه اولین مورد مخالف پیدا شد خاتمه می‌یابد، و هر دو زمانی که CX صفر شود نیز خاتمه می‌یابند، DI و SI بعد از بایت‌هایی که سبب خاتمه شدند قرار می‌گیرند. آخرین مقایسه پرچم‌ها را تنظیم می‌کند.

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد.

کد منبع. (بدون عملوند) CMPSB/CMPSW/CMPSD [REPnn]

کد مقصد. 1010011w

CMPXCHG : مقایسه و تعویض (۸۰۳۸۶ و ما بعد)

عمل. دومین عملوند (AL، AX یا EAX) را با اولین عملوند (ثبات یا حافظه) مقایسه می‌کند. اگر مساوی باشند، CMPXCHG دومین عملوند را در اولین عملوند را قرار می‌دهد و ZF را یک می‌کند، اگر مساوی نباشند اولین عملوند را در دومین عملوند قرار می‌دهد و ZF را صفر می‌کند.

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد.

کد منبع. CMPXCHG register/memory,AL/AX/EAX

کد مقصد. (شانزدهی) OF BI/r یا OF BO/r

CMPXCHG8B : مقایسه و تعویض (پنتیوم و ما بعد)

عمل. ۶۴ بیت EDX:EAX را بااولین عملوند (ثبات یاحافظه) مقایسه‌می‌کند.اگرمساوی‌باشند،CMPXCHG8B، EDX:EAX را در اولین عملوند قرار می‌دهد و ZF را یک می‌کند، اگر مساوی نباشند، CMPXCHG8B. اولین عملوند را در EDX:EAX قرار می‌دهد و ZF را صفر می‌کند.

پرچم‌ها. بر ZF اثر می‌گذارد.

کد منبع. (یک عملوند ۶۴ بیتی) CMPXCH8B register/memory

کد مقصد. (شانزدهی) OF C7

CWD : تبدیل کلمه به دو کلمه‌ای

عمل. یک مقدار یک کلمه‌ای علامت دار را در DX:AX دو کلمه‌ای علامتدار بسط می‌دهد بدینوسیله که بیت علامت (بیت ۱۵) AX را در AX کپی می‌کند تا ۳۲ بیت تولید شود. (CBW ، CWDE و CDQ را نیز ببینید).

پرچم‌ها. هیچ اثری ندارد.

کد منبع. CWD (بدون عملوند)

کد مقصد. 10011001

CWDE : تبدیل کلمه به دو کلمه‌ای بسط یافته (۸۰۳۸۶ و ما بعد)

عمل. یک مقدار علامتدار یک کلمه‌ای را در یک دو کلمه‌ای EAX بسط می‌دهد بدینوسیله که بیت علامت (بیت ۱۵) AX را کپی می‌کند و یک مقسوم ۳۲ بیتی تولید می‌کند. (CBW ، CWD و CDQ را نیز ببینید.)

پرچم‌ها. هیچ اثری ندارد.

کد منبع. CWDE (بدون عملوند)

کد مقصد. 10011000

DAA : تنظیم دهدهی بعد از جمع

عمل. حاصل موجود در AL را بعد از ADD یا ADCکه دو رقم BCD را جمع کرده است، تصحیح می‌کند. اگر مقدار چهار بیت سمت راست بزرگتر از ۹ باشد، یا اگر AF یک باشد، DAA، ۶ را با AL جمع می‌کند و AF را یک می‌کند. سپس، اگر مقدار AL بزرگتر از 99H باشد، یا اگر CF یک باشد، DAA مقدار 60H را با AL جمع می‌کند و CF را یک می‌کند. در غیر اینصورت AF و CF صفر می‌شوند. حال AL شامل یک حاصل دهدهی فشردهٔ ۲ رقمی صحیح است. (DAS را نیز ببینید.)

پرچم‌ها. بر AF ، CF ، PF ، SF و ZF اثر می‌گذارد (OF تعریف نشده است)

کد منبع. DAA (بدون عملوند)

کد مقصد. 00100111

DAS : تنظیم دهدهی برای تفریق

عمل. بعد از یک SUB یا SBBکه دو عنصر BCD فشرده را تفریق می‌کند، حاصل تولید شده در AL را تصحیح می‌کند. اگر مقدار چهار بیت سمت راست بزرگتر از ۹ باشد، DAS مقدار 60H را از ALکم می‌کند و CF را یک می‌کند، در غیر اینصورت AF و CF صفر می‌شوند. حال AL شامل حاصلی دو رقمی فشرده تصحیح شده است. (DAA را نیز ببینید)

پرچم‌ها. بر AF ، CF ، PF ، SF و ZF اثر می‌گذارد (0F تعریف نشده است).

کد منبع. DAS (بدون عملوند)

کد مقصد. 00101111

DEC : یک واحد کاهش

عمل. یک را از بایت، کلمه، یا دو کلمه‌ای (در ۸۰۳۸۶ و ما بعد) موجود در ثبات حافظه می‌کاهد و با مقدار موجود در آن بصورت یک عدد صحیح بدون علامت برخورد می‌کند. (INC را نیز ببینید.)

پرچم‌ها. بر AF ، OF ، PF ، SF و ZF اثر می‌گذارد.

کد منبع. DEC register/memory

کد مقصد. دو قالب دارد :

ثبات : |01001reg|

ثبات یا حافظه : |1111111w|mod001r/m|

DIV : تقسیم بدون علامت

عمل. یک مقسوم بدون علامت را بر یک مقسوم علیه علامتدار تقسیم می‌کند. DIV سمت چپ‌ترین بیت را بعنوان

بیت دارد. در نظر می‌گیرد و نه بعنوان یک علامت منفی. تقسیم بر صفر سبب ایجاد وقفهٔ تقسیم بر صفر خواهد شد. (IDIV را نیز ببینید). در اینجا عملیات تقسیم بر طبق اندازه مقسوم ذکر شده است :

Size	Dividend (Operand 1)	Divisor (Operand 2)	Quotient	Remainder	Example
16-bit	AX	8-bit reg/memory	AL	AH	DIV BH
32-bit	DX:AX	16-bit reg/memory	AX	DX	DIV CX
64-bit	EDX:EAX	32-bit reg/memory	EAX	EDX	DIV ECX

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد، (همه تعریف نشده‌اند.)

کد منبع. DIV register/memory

کد مقصد. |1111011w|mod110r/m|

ESC : گریز

عمل. استفاده از کمک پردازشگرهایی مانند 87 × 80 را جهت انجام عملیات ویژه سهولت می‌بخشد. ESC، کمک پردازشگر با یک دستور و عملوند را جهت اجرا مهیا می‌سازد. توجه کنید که نسخه 6.1، MASM دیگر ESC را پشتیبانی نمی‌کند، بجای آن کد مقصد کامل مورد نیاز برای دستورات کمک پردازشگر را تولید می‌کنند.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. ESC immediate,register/memory

کد مقصد. (بیت‌های X به کد مقصد کمک پردازشگر اشاره دارد) |11011***|mod***r/m|

HLT : وارد کردن حالت توقف

عمل. سبب می‌شود زمانی که پردازشگر منتظر یک وقفه است به حالت توقف برود، هم اکنون ثبات‌های CS و IP به آدرس دستور بلافاصله بعد اشاره می‌کند. وقتی یک وقفه رخ دهد، پردازشگر CS و IP را بر روی پشته می‌گذارد و روتین وقفه را اجرا می‌کند. در بازگشت یک دستور IRET ، از روی پشته ثبات‌ها را بر می‌دارد و پردازش از HLT اصلی آغاز می‌شود.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. HLT (بدون عملوند)

کد مقصد. 11110100

IDIV : تقسیم علامتدار (صحیح)

عمل. یک مقسوم علامتدار را بر یک مقسوم علیه علامتدار تقسیم می‌کند. IDIV با سمت چپ‌ترین بیت بعنوان علامت برخورد می‌کند (0 = مثبت، 1 = منفی). تقسیم بر صفر سبب وقفه تقسیم بر صفر خواهد شد. (CBW و CWD را برای بسط طول مقسوم علامتدار و همچنین DIV را نیز ملاحظه کنید). در اینجا عملیات تقسیم بر طبق اندازهٔ مقسوم ذکر شده است.

Size	Dividend (Operand 1)	Divisor (Operand 2)	Quotient	Remainder	Example
16-bit	AX	8-bit reg/memory	AL	AH	IDIV BH
32-bit	DX:AX	16-bit reg/memory	AX	DX	IDIV CX
64-bit	EDX:EAX	32-bit reg/memory	EAX	EDX	IDIV ECX

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد.

کد منبع. IDIV register/memory

کد مقصد. |1111011w|mod111r/m|

IMUL : ضرب علامتدار (صحیح)

عمل. یک مضروب علامتدار را در یک مضروب فیه علامتدار ضرب می‌کند. IMUL با سمت چپ‌ترین بیت مانند بیت علامت برخورد می‌کند (0 = مثبت و 1= منفی). این عملیات در نظر می‌گیرد که مضروب در AL ، AX و EAX است و این اندازه بر حسب مضروب فیه بدست می‌آید. (MUL را نیز ببینید) در اینجا عملیات ضرب بر طبق اندازهٔ مضروب فیه آورده شده است.

Size	Multiplicand (Operand 1)	Multiplier (Operand 2)	Product	Example
8-bit	AL	8-bit register/memory	AX	IMUL BL
16-bit	AX	16-bit register/memory	DX:AX	IMUL BX
32-bit	EAX	32-bit register/memory	EDX:EAX	IMUL ECX

پرچم‌ها. بر CF و OF اثر می‌گذارد. (AF ، PF ، SF و ZF تفریق نشده‌اند.)

کد منبع. (در همه پردازشگرها) IMUL register/memory

کد مقصد. (اولین قالب) |1111011w|mod101r/m|

سه قالب دیگر برای این دستور وجود دارد :

- (در ۸۰۳۸۶ و بعد از آن) IMUL register,immediate
- (در ۸۰۲۸۶ و بعد از آن) IMUL register,register/immediate
- (در ۸۰۳۸۶ و بعد از آن) IMUL register,register/memory

IN : واردکردن بایت یا کلمه

عمل. انتقال از یک درگاه ورودی به AL یا یک کلمه به AX. درگاه را بصورت یک عملوند عددی ثابت (IN AX,port#) یا متغییری در DX (بصورت IN AX,DX) کد نمایید. در صورتی از DX استفاده کنید که شماره درگاه بزرگتر از ۲۵۶ باشد. (INS و OUT را نیز ملاحظه کنید.)

پرچم‌ها. هیچ اثری ندارد.

کد منبع. IN AL,AX,Portno/DX

کد مقصد. دو قالب دارد :

درگاه، متغیر : |1110110w|

درگاه، ثابت : |1110010w|--- port ---|

INC : یک واحد افزایش

عمل. یک بایت، یک کلمه، یا دو کلمه‌ای (در ۸۰۳۸۶ و ما بعد) موجود در ثبات یا حافظه را یک واحد افزایش می‌دهد و با مقدار موجود در آن بعنوان یک عدد صحیح بدون علامت برخورد می‌کند، برای مثال بصورت INC CX کد می‌شود. (DEC را نیز ببینید)

پرچم‌ها. بر AF ، OF ، PF ، SF و ZF اثر می‌گذارد.

کد منبع. INC register/memory

کد مقصد. دو قالب دارد :

ثبات : |01000reg|

ثبات یا حافظه : |1111111w|mod000r/m|

INS/INSB/INSW/INSD وارد کردن رشته (در ۸۰۲۸۶ و ما بعد)

عمل. یک رشته را از درگاه (مقصد) دریافت می‌کند. مقصد توسط ES:DI آدرسدهی می‌شود و DX حاوی شمارهٔ درگاه است. تجربهٔ عملی استفاده از INSn با پیشوند REP است که CX حاوی تعداد عناصر دریافتی است (بصورت بایت، کلمه، یا دو کلمه‌ای). بسته به DF (0/1)، عملیات بر طبق اندازهٔ عنصر DI را افزایش یا کاهش می‌دهد. (IN و OUTS را نیز ببینید).

پرچم‌ها. هیچ اثری ندارد.

کد منبع. (بدون عملوند) INSB/INSW/INSD [REP]

کد مقصد. 0110110w

INT : وقفه

عمل. پردازش را متوقف می‌کند و کنترل را به یکی از ۲۵۶ آدرس وقفه که از سگمنت 0 ، افست 0 آغاز می‌شود. منتقل می‌کند. INT چنین عمل می‌کند : (۱) پرچم‌ها را روی پشته می‌گذارد و IF و TF را یک می‌کند، (۲) CS را روی پشته می‌گذارد و کلمه بالا رتبهٔ آدرس وقفه را در CS قرار می‌دهد، (۳) IP را بر روی پشته می‌گذارد و IP را با کلمهٔ پایین رتبه آدرس وقفه پر می‌کند. برای ۸۰۳۸۶ و ما بعد INT یک IP ۱۶ بیتی برای سگمنت ۱۶ بیتی و یک IP ، ۳۲ بیتی برای سگمنت ۳۲ بیتی بر روی پشته می‌گذارد. IRET از روتین وقفه باز می‌گردد.

پرچم‌ها. IF و TF را صفر می‌کند.

کد منبع. INT numberc

کد مقصد. (if v=0 , typeis 3) |11000110V|--- type ---|

INTO : وقفه در حالت سرریز.

عمل. سبب یک وقفه خواهد شد اگر سرریز رخ داده باشد (OF یک باشد) و INT 04H را اجرا می‌کند. آدرس وقفه در موقعیت 10H از جدول سرویس وقفه قرار دارد (INT را نیز ببینید).

پرچم‌ها. بر IF و TF اثر می‌گذارد.

کد منبع. (بدون عملوند) INTO

کد مقصد. 11001110

IRET/IRETD : بازگشت از وقفه

عمل. یک بازگشت دور را از یک روتین وقفه فراهم می‌سازد. IRET روال زیر را انجام می‌دهد : (۱) کلمهٔ بالای پشته را برداشته و در IP می‌گذارد، SP را ۲ واحد می‌افزاید و کلمه بالای پشته را در CS قرار می‌دهد، (۲) SP را ۲ واحد می‌افزاید و کلمه بالای پشته را در پرچم‌ها می‌گذارد این روال عکس مراحل وقفه را برای بازگشت انجام می‌دهد. در ۸۰۳۸۶ و ما بعد از IRETD (دو کلمه‌ای) برای برداشتن IP ، ۳۲ بیتی از روی پشته استفاده کنید. RET(را نیز ببینید).

پرچم‌ها. بر همه اثر دارد.
کد منبع. IRET (بدون عملوند)
کد مقصد. 11001111

Jcondition : پرش بر حسب شرط

این بخش، دستورات پرش شرطی را بطور خلاصه بیان می‌کند که در صورت پرچم شرط بررسی شده به عملوند منتقل می‌شود. اگر شرط صحیح باشد، عملیات افست عملوند را با IP جمع می‌کند و پرش را انجام می‌دهد، اگر شرط صحیح نباشد، پردازش از دستور بعدی ادامه می‌یابد. برای ۸۰۸۶ - ۸۰۲۸۶ پرش باید کوتاه باشد (۱۲۸- تا ۱۲۷ بایت)، برای ۸۰۳۸۶ و ما بعد اسمبلر یک پرش نزدیک در نظر می‌گیرد (۳۲۷۶۸ تا ۳۲۷۶۷) اما شما باید عملگر SHORT را در این روش کوتاه استفاده کنید. عملیات پرچم‌ها را بررسی می‌کند ولی آنها را تغییر نمی‌دهد. کد منبع Jcondition Label می‌باشد. همهٔ کدهای مقصد بصورت |--- disp ---|distnnnn| در حالیکه بیت‌های disp برای پرش‌های کوتاه 0111 و برای پرش‌های نزدیک 1000 می‌باشد. در اولین لیست، دستورات، بعد از عملیات مقایسه که اولین عملوند و دومین را با هم مقایسه می‌کند، مورد استفاده قرار می‌گیرد.

SOURCE CODE	OBJECT CODE	FLAGS CHECKED	USED AFTER COMPARISON
JA	|dist0111|	CF = 0, ZF = 0	Unsigned data, above (higher)
JAE	|dist0011|	CF = 0	Unsigned data, above/equal
JB	|dist0010|	CF = 1	Unsigned data, below (lower)
JBE	|dist0110|	CF = 1 or AF = 1	Unsigned data, below/equal
JE	|dist0100|	ZF = 1	Signed/unsigned data, equal
JG	|dist1111|	ZF = 0, SF = OF	Signed data, greater
JGE	|dist1101|	SF = OF	Signed data, greater/equal
JL	|dist1100|	SF not= OF	Signed data, lower
JLE	|dist1110|	ZF = 1 or SF not= OF	Signed data, lower/equal
JNA	|dist0110|	CF = 1 or AF = 1	Unsigned data, not above
JNAE	|dist0010|	CF = 1	Unsigned data, not above/equal
JNB	|dist0011|	CF = 0	Unsigned data, not below
JNBE	|dist0111|	CF = 0, ZF = 0	Unsigned data, not below/equal
JNE	|dist0101|	ZF = 0	Signed/unsigned, not equal
JNG	|dist1110|	ZF = 1 or SF not= OF	Signed data, not greater
JNGE	|dist1100|	SF not= OF	Signed data, not greater/equal
JNL	|dist1101|	SF = OF	Signed data, not lower
JNLE	|dist1111|	ZF = 0, SF = OF	Signed data, not lower/equal

در لیست دوم، دستورات بعد از محاسبه یا عملیات دیگری که بر طبق نتیجه بیت‌ها را صفر یا یک می‌کند، مورد استفاده قرار می‌گیرد.

SOURCE CODE	OBJECT CODE	FLAGS CHECKED	USED TO TEST
JC	|dist0010|	CF = 1	If CF set (same as JB/JNAE)
JNC	|dist0011|	CF = 0	If CF off (same as JAE/JNB)
JNO	|dist0001|	OF = 0	If OF off
JNP	|dist1011|	PF = 0	If no (odd) parity: odd number of bits set in low-order 8 bits
JNS	|dist1001|	SF = 0	If sign is positive
JNZ	|dist0101|	ZF = 0	If signed/unsigned data not zero
JO	|dist0000|	OF = 1	If OF set
JP	|dist1010|	PF = 1	If even parity: even number of bits set in low

```
                                order 8 bits
JPE     |dist1010|  PF = 1   Same as JP
JPO     |dist1011|  PF = 0   Same as JNP
JS      |dist1000|  SF = 1   If sign is negative
JZ      |dist0100|  ZF = 1   If signed/unsigned data is zero
```

JCXZ/JECXZ : پرش در صورتیکه CX/ECX صفر است.

عمل. در صورتیکه CX یا ECX (در ۸۰۳۸۶ و ما بعد) حاوی صفر است به آدرس خاص پرش می‌کند. این عملیات در شروع یک حلقه می‌تواند مفید باشد، اگر چه محدود به یک پرش کوتاه است.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. JCXZ/JECXZ label

کد مقصد. |11100011|--- disp ---|

JMP : پرش غیر شرطی

عمل. بر آدرس معین شده در هر شرایطی پرش می‌کند. آدرس JMP ممکن است کوتاه (۱۲۸- تا ۱۲۷+) نزدیک (داخل 32K± پیش فرض) یا دور (در سگمنت کد دیگری) باشد. یک JMP نزدیک یا کوتاه IP را با آدرس افست مقصد جایگزین می‌کند. یک پرش دور (مانند JMP FAR PTR Label) CS:IP را با آدرس سگمنت جدید جایگزین می‌کند.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. JMP register/memory

کد مقصد. پنج قالب دارد :

مستقیم داخل سگمنت کوتاه : |11101011|--- disp ---|

مستقیم داخل سگمنت کوتاه : |11101001|disp-low|disp-high|

غیر مستقیم داخل سگمنت کوتاه : |11111111|mod100r/m|

غیر مستقیم بین سگمنت : |11111111|mod101r/m|

مستقیم بین سگمنت : |11101010|offset-low|offset-high|seg-low|seg-high|

LAHF : قرار دادن پرچم‌ها در AH

عمل. ۸ بیت سمت راست ثبات پرچم را در AH قرار می‌دهد (SAHF را نیز ببینید).

پرچم‌ها. هیچ اثری ندارد.

کد منبع. LAHF (بدون عملوند)

کد مقصد. 10011111

LDS/LES/LFS/LGS/LSS : بارگذاری ثبات سگمنت

عمل. آدرس و افست دور یک عنصر داده را مقدار دهی می‌کند بطوریکه دستور بعدی بتواند به آن دسترسی داشته باشد. اولین عملوند هر ثبات عمومی، شاخص یا اشاره‌گر را ارجاع می‌کند. دومین، عملوند چهار بایت حافظه شامل یک افست و یک آدرس سگمنت را ارجاع می‌کند. عملیات آدرس سگمنت را در ثبات سگمنت و آدرس افست را در ثبات اولین عملوند قرار می‌دهد. برای مثال LDS یعنی بارگذاری ثبات سگمنت داده. LFS ، LGS و LSS توسط ۸۰۳۸۶ و ما بعد پشتیبانی می‌شوند.

پرچم‌ها. هیچ اثری ندارد.
کد منبع. LDS/LES/LFS/LGS/LSS register,memory
کد مقصد. LDS: |11000101|modregr/m|
LES: |11000100|modregr/m|
LFS: |00001111|10110100|modregr/m|
LGS: |00001111|10110101|modregr/m|
LSS: |00001111|10110010|modregr/m|

LEA : بارگذاری آدرس موثر

عمل. یک آدرس (افست) نزدیک را در یک ثبات قرار می‌دهد.
پرچم‌ها. هیچ اثری ندارد
کد منبع. LEA register,memory
کد مقصد. 10001101

LES/LFS/LGS : بارگذاری ثبات سگمنت اضافی

عمل. LDS را ببینید.

LOCK : قفل درگاه

عمل. 80X87 یا دیگر پردازشگرها را از تغییر یک عنصر داده در لحظه‌ای خاص باز می‌دارد. LOCK یک پیشوند یک بایتی است که ممکن است قبل از هر دستور کد نمایید. عملیات یک سیگنال به پردازشگر می‌فرستد تا از استفاده داده قبل از تکمیل دستور جاری جلوگیری کند.
پرچم‌ها. هیچ اثری ندارد.
کد منبع. LOCK instruction
کد مقصد. 11110000

LODS/LODSB/LODSW/LODSD : بارگذاری رشتهٔ بایت، کلمه یا دو کلمه‌ای

عمل. ثبات انباشتگر را با مقداری از حافظه بارگذاری می‌کند. اگر چه LODS یک عملیات رشته‌ای است، نیازی به پیشوند REP ندارد آدرس ثبات‌های DS:SI یک بایت (در صورت LODSB) یک کلمه (در صورت LODSW) یا دو کلمه‌ای (در صورت LODSD در ۸۰۳۸۶ و ما بعد) هستند و بترتیب از حافظه در AL ، AX و EAX قرار می‌دهند. اگر DF صفر باشد، عملیات یک (بایت)، ۲ (کلمه) یا ۴ (دو کلمه‌ای) را با SI جمع می‌کند، در غیر اینصورت ۱، ۲، ۴ را تفریق می‌کند.
پرچم‌ها. هیچ اثری ندارد.
کد منبع. (بدون عملوند LODSB/LODSW/LODSD
کد مقصد. 1010110w

LOOP/LOOPW/LOOPD : حلقه تا زمانی که کامل شود.

عمل. اجرای یک روتین را به تعداد مرتبه مشخص شده کنترل می‌کند. قبل از شروع حلقه، CX باید حاوی تعداد

شمارش باشد. LOOP در انتهای حلقه ظاهر می‌شود و CX را یک واحد می‌کاهد. اگر CX صفر نباشد، LOOP به آدرس عملوند (یک پرش کوتاه) منتقل می‌شود که به شروع حلقه اشاره دارد (افست با IP جمع می‌شود) در غیر اینصورت به دستور بعدی می‌رود.

برای ۸۰۳۸۶ و ما بعد، LOOP در حالت ۱۶ بایتی و از ECX در حالت ۳۲ بیتی استفاده می‌کند. برای مشخص کردن CX با ۱۶ بیت از LOOPW و ECX با ۳۲ بیت از LOOPD استفاده کنید.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. LOOPnnlabel

کد مقصد. |--- disp ---|11100010|

LOOPE/LOOPZ/LOOPEW/LOOPZW/LOOPED/LOOPZD : تکرار حلقه تا زمانی که مساوی یا صفر است.

عمل. تکرار اجرای یک روتین را کنترل می‌کند. LOOPE و LOOPZ مشابه LOOP است، بجز آنکه به آدرس عملوند (یک پرش کوتاه) در صورتیکه CX صفر نباشد و ZF یک باشد منتقل می‌شود(شرط صفر، با دیگر دستورات تنظیم می‌شود)، در غیر اینصورت، عملیات به دستور بعدی می‌رود (LOOPNE/LOOPNZ را نیز ببینید.)

برای ۸۰۳۸۶ و ما بعد، LOOPE و LOOPZ از CX در حالت ۱۶ بیتی و از ECX در حالت ۳۲ بیتی استفاده کنید. می‌توانید از LOOPEW و LOOPZW برای ۱۶ بیت CX و LOOPED و LOOPZD برای ۳۲ بیت ECX استفاده کنید.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. LOOPnnlabel

کد مقصد. |--- disp ---|11100001|

LOOPNE/LOOPNZ/LOOPNEW/LOOPNZW : تکرارحلقه تا زمانیکه مساوی‌نیست یا صفرنیست

عمل. تکرار اجرای یک روتین را کنترل می‌کند. LOOPNE و LOOPNZ مشابه LOOP است بجز آنکه در صورتیکه CX صفر نباشد و ZF صفر باشد به آدرس عملوند منتقل می‌شود (یک پرش کوتاه)، شرط غیر صفر توسط دستورات دیگر تنظیم می‌شود.) در غیر اینصورت عملیات به دستور بعدی می‌رود. (LOOPE/LOOPZ را نیز ببینید).

برای ۸۰۳۸۶ و ما بعد، LOOPNZ و LOOPNE از CX در حالت ۱۶ بیتی و از ECX در حالت ۳۲ بیتی استفاده می‌کند. می‌توانید از LOOPNZW و LOOPNEW برای مشخص کردن ۱۶ بیت CX و LOOPNZD و LOOPNED برای مشخص کردن ۳۲ بیت ECX استفاده کنید.

پرچم‌ها. هیچ اثری ندارند.

کد منبع. LOOPNE/LOOPNZ label

کد مقصد. |--- disp ---|111000000|

LSS : بارگذاری ثبات سگمنت پشته

عمل. LDS را ببینید.

MOV. انتقال داده.

عمل. داده‌ها را بین دو ثبات یا بین یک ثبات و حافظه منتقل می‌کند و داده بلافصل را به یک ثبات یا حافظه منتقل

می‌کند. داده ارجاع شده تعداد بایت‌های انتقالی (۴،۲،۱) را تعریف می‌کند، عملوندها باید هم اندازه باشند. MOV نمی‌تواند بین دو موقعیت حافظه، (از MOVS استفاده کنید)، یا از داده بلافصل به ثبات سگمنت، یا از ثبات سگمنت به ثبات سگمنت انتقال انجام دهد. (MOVSX/MOVZX را نیز ببینید.)

پرچم‌ها. هیچ اثری ندارد

کد منبع. MOV register/memory,register/memory/immediate

کد مقصد. هفت قالب :

ثبات یا حافظه به / از ثبات :	|100010dw|modregr/m|
بلافصل به ثبات یا حافظه :	|1100011w|mod000r/m|--- data ---|data if w=1|
بلافصل به ثبات :	|1011wreg|--- data ---|data if w=1|
حافظه به انباشتگر :	|1010000w|addr-low|addr-high|
انباشتگر به حافظه :	|10100001w|addr-low|addr-high|
ثبات یا حافظه به ثبات سگمنت :	|10001110|mod0sgr/m|(sg=segreg)|
ثبات سگمنت به ثبات یا حافظه :	|10001100|mod0sgr/m|(sg=segreg)|

MOVS/MOVSB/MOVSW/MOVSD : انتقال رشته

عمل. داده‌ها را بین موقعیت‌های حافظه منتقل می‌کند. عموماً با پیشوند REP استفاده می‌شود و طول در CX قرار می‌گیرد، MOVSB بایت‌ها را منتقل می‌کند، MOVSW کلمات را منتقل می‌کند و MOVSD دو کلمه‌ای را منتقل می‌کند (در ۸۰۳۸۶ و ما بعد). اولین عملوند توسط ES:DI و دومین عملوند توسط DS:SI آدرسدهی می‌شود. اگر DF صفر باشد، عملیات، داده‌ها را از چپ به راست اولین مقصد عملوند منتقل می‌کند و DI و SI را ۱، ۲، یا ۴ واحد می‌افزاید. اگر DF یک باشد، عملیات داده‌ها را از راست به چپ منتقل می‌کند و DI و SI را می‌کاهد. REP ، CX را برای هر بار تکرار می‌کاهد. عملیات وقتی CX صفر شود خاتمه می‌یابد، DI و SI بعد از آخرین بایت انتقالی قرار دارند.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. (بدون عملوند) [REP] MOVSB/MOVSW/MOVSD

کد مقصد. 1010010w

MOVZX : MOVSX : انتقال با بسط علامت یا بسط صفر (در ۸۰۳۸۶ و مابعد)

عمل. یک عملوند منبع ۸ یا ۱۶ بیتی را به عملوند مقصد ۱۶ یا ۳۲ بیتی کپی می‌کند. MOVSX بیت علامت را در بیت‌های سمت چپ پر می‌کند، و MOVZX بیت‌ها را با صفر پر می‌کند.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. MOVSX/MOVZX register/memory,register/memory/immediate

کد مقصد. MOVSX : |00001111|1011111w|modregr/m|

MOVZX : |00001111|1011011w|modregr/m|

MUL : ضرب بدون علامت

عمل. یک مضروب بدون علامت را در مضروب فیه علامتدار ضرب می‌کند. MUL با سمت چپ‌ترین بیت بصورت بیت داده، نه یک بیت علامت برخورد می‌کند. عملیات مضروب را در AZ، AX یا EAX فرض می‌کند و اندازه

آن را از مضروب فیه در نظر می‌گیرد. (LMUL را نیز ببینید.) در اینجا عملیات ضرب بر طبق اندازهٔ مضروب فیه آورده شده است.

Size	Multiplicand (Operand 1)	Multiplier (Operand 2)	Product	Example
8-bit	AL	8-bit register/memory	AX	MUL BL
16-bit	AX	16-bit register/memory	DX:AX	MUL BX
32-bit	EAX	32-bit register/memory	EDX:EAX	MUL ECX

پرچم‌ها. بر CF و OF اثر می‌گذارد (AF ، PF ، SF و ZF تفریق نشده‌اند.)
کد منبع. MUL register/memory
کد مقصد. |1111011w|mod100r/m|

NEG : منفی کردن

عمل. یک مقدار دودویی را از مثبت به منفی یا از منفی به مثبت معکوس می‌کند. NEG مکمل ۲ عملوند خاص را با تفریق عملوند از صفحه و جمع آن با یک مهیا می‌سازد. عملوندها باید یک بایت، کلمه یا دو کلمه‌ای (در ۸۰۳۸۶ و ما بعد) در یک ثبات یا حافظه باشند. (NOT را نیز ببینید.)
پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF. اثر می‌گذارد.
کد منبع. NEG registr/memory
کد مقصد. |1111011w|mod011r/m|

NOP : عملی انجام نده

عمل. برای حذف یا جایگزینی کد ماشین یا تاخیر اجرا برای زمان استفاده می‌شود. NOP یک عمل پوچ با اجرای XCHG AX,AX انجام می‌دهد.
پرچم‌ها. هیچ اثری ندارد.
کد منبع. NOP (بدون عملوند)
کد مقصد. 10010000

NOT : NOT منطقی

عمل. بیت‌های 0 را با یک و بر عکس تعویض می‌کند. عملوند یک بایت، کلمه، یا دو کلمه‌ای (در ۸۰۳۸۶ و ما بعد) در یک ثبات یا حافظه است (NEG را نیز ببینید.)
پرچم‌ها. هیچ اثری ندارد.
کد منبع. NOT register/memory
کد مقصد. |1111011w|mod010r/m|

OR : OR منطقی

عمل. یک عملیات OR منطقی بر روی بیت‌های دو عملوند انجام می‌دهد. هر دو عملوند، بایت، کلمه، یا دو کلمه‌ای‌اند (در ۸۰۳۸۶ و مابعد) که OR بیت به بیت وفق می‌دهد. برای هر جفت بیت جور شده، اگر هر دو یا یکی از آنها ۱ باشد، بیت اولین عملوند یک خواهد شد، در غیر اینصورت بیت تغییری نمی‌کند. (AND و XOR را نیز ببینید.)

پرچم‌ها. بر CF(0) ، OF(0) ، PF ، SF و ZF اثر می‌گذارد و (AF تعریف نشده است).
کد منبع. OR register/memory,register/memory/immediate
کد مقصد. سه قالب دارد :

ثبات یا حافظه با ثبات : |000010dw|modregr/m|
بلافصل با انباشتگر : |0000110w|--- data ---|data if w =1|
بلافصل با ثبات یا حافظه : |100000sw|mod001r/m|--- data ---|data if w=1|

OUT : خروجی بایت یا کلمه

عمل. یک بایت را از AL یا کلمه را از AX به درگاه خروجی می‌فرستد. درگاه یک عملوند عددی ثابت یا یک متغییر در DX است. از DX هنگامی استفاده کنید که شمارهٔ درگاه بزرگتر از ۲۵۶ باشد. (IN و OUTS را نیز ببینید.)
پرچم‌ها. هیچ اثری ندارد.
کد منبع. درگاه ثابت : OUT port#,AX
درگاه متغیر : OUT DX,AX
کد مقصد. درگاه ثابت : |1110011w|--- port ---|
درگاه متغیر : |1110111w|

OUTS/OUTSB/OUTSW/OUTSD : خروج رشته (۸۰۲۸۶ و ما بعد)

عمل. یک رشته (منبع) را به درگاه ارسال می‌دارد. منبع توسط DS:SI آدرسدهی می‌شود و DX حاوی شماره درگاه است. تجربهٔ عملی استفاده از OUTSn با پیشوند REP است، با CX که حاوی تعداد عناصری (بصورت بایت، کلمه، یا دو کلمه‌ای) ارسالی است. بسته به DF (0/1) عملیات SI را بر طبق اندازه عنصر افزایش یا کاهش می‌دهد (IN و OUTS را نیز ببینید.)
پرچم‌ها. هیچ اثری ندارد.
کد منبع. (بدون عملوند) [REP] OUTSB/OUTSW/OUTSD
کد مقصد. |0110111w|

POP : برداشتن کلمه از روی پشته

عمل. یک کلمه یا دو کلمه‌ای (در ۸۰۳۸۶ و ما بعد) را که قبلاً روی پشته گذاشته شده بر می‌دارد و در یک مقصد خاص قرار می‌دهد - یک موقعیت حافظه، ثبات عمومی، یا ثبات سگمنت. SP به کلمهٔ جاری بالای پشته اشاره دارد، POP آن را به مقصد خاص منتقل می‌کند و SP را ۲ واحد افزایش می‌دهد. در ۸۰۳۸۶ و ما بعد، یک عملوند ۳۲ بیتی بر یک مقدار دو کلمه‌ای دلالت دارد و ESP ۴ واحد افزایش می‌یابد. (PUSH را نیز ببینید)
پرچم‌ها. هیچ اثری ندارد.
کد منبع. POP register/memory
کد مقصد. سه قالب دارد :

ثبات : |01011 reg|
ثبات سگمنت : |000sg111|sg=(segment reg)|
ثبات / حافظه : |10001111|mod000r/m|

POPA : (در ۸۰۲۸۶ و ما بعد) POPAD : (در ۸۰۳۸۶ و ما بعد)
همهٔ ثبات‌های عمومی را از روی پشته برمی‌دارد.

عمل. POPA ، هشت کلمه بالای پشته را برداشته و در DI ، SI ، BP ، SP ، BX ، DX ، CX و AX بترتیب قرار می‌دهد. POPAD هشت دو کلمه‌ای بالای پشته را برداشته و در EDI ، ESI ، EBP ، ESP ، EBX ، EDX ، ECX ، EAX قرار می‌دهد. مقدار SP بعد از بارگذاری دور انداخته می‌شود. معمولاً، یک PUSHA /PUSHAD قبلاً ثبات‌ها را بر روی پشته گذارده‌اند.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. (بدون عملوند) POPA/POPAD

کد مقصد. 01100001

POPF/POPFD : برداشتن پرچم‌ها از روی پشته

عمل. POPF کلمه بالای پشته را برداشته و در ثبات پرچم قرار می‌دهد و SP را ۲ واحد می‌افزاید. POPFD (در ۸۰۳۸۶ و ما بعد) دو کلمه‌ای بالای پشته را برداشته و در ثبات پرچم ۳۲ بیتی قرار می‌دهد و SP را ۴ واحد افزایش می‌دهد. معمولاً یک PUSHF قبلاً پرچم‌ها را روی پشته گذاشته‌اند.

پرچم‌ها. بر همه اثر می‌گذارد.

کد منبع. POPF/POPFD (بدون عملوند)

کد مقصد. 10011101

PUSH : قرار دادن بر روی پشته

عمل. یک کلمه یا دو کلمه‌ای (در ۸۰۳۸۶ و ما بعد) را جهت استفادهٔ بعدی بر روی پشته می‌گذارد. ثبات SP به کلمه جاری بالای پشته اشاره می‌کند. PUSH و SP را ۲ واحد کاهش می‌دهد یا ESP را ۴ واحد کاهش می‌دهد و یک (دو) کلمه را از عملوند خاص به بالای جاری پشته منتقل می‌کند. منبع ممکن است یک ثبات عمومی، ثبات سگمنت، یا حافظه باشد. (POP و PUSHF را نیز ببینید.)

پرچم‌ها. هیچ اثری ندارد.

کد منبع. PUSH register/memory (همه پردازشگرها)
PUSH immediate (۸۰۲۸۶ و ما بعد)

کد مقصد. سه قالب دارد :

ثبات : |01010reg|
ثبات سگمنت : |000sg110| (sg=segment reg)|
ثبات یا حافظه : |11111111|mod100r/m|

PUSHA (۸۰۲۸۶ و مابعد) PUSHAD (۸۰۳۷۶ و مابعد) قراردادن همهٔ ثبات‌های عمومی برروی پشته.

عمل. PUSHA همه ثبات‌های AX ، CX ، DX ، BX ، SP ، BP ، SI و DI را به ترتیب روی پشته قرار می‌دهد و SP را ۱۶ واحد کاهش می‌دهد. PUSHAD همهٔ ثبات‌های ESP ، EBP ، ESI ، EDI ، EAX ، ECX ، EDX و EBX را روی پشته قرار می‌دهد و SP را ۳۲ واحد می‌کاهد. معمولاً یک POPA/POPAP بعداً ثبات‌ها را برمی دارد.

پرچم‌ها. هیچ اثری ندارد.
کد منبع. PUSHA/PUSHAD (بدون عملوند)
کد مقصد. 01100000

PUSHF/PUSHFD : گذاشتن پرچم‌ها روی پشته

عمل. محتویات ثبات پرچم را جهت استفادهٔ بعدی روی پشته می‌گذارد. PUSHF و SP را ۲ واحد می‌کاهد. PUSHFD (در ۸۰۳۸۶ و ما بعد) ثبات پرچم ۳۲ بیتی را روی پشته می‌گذارد و SP را ۴ واحد می‌کاهد. (POPF و PUSH را نیز ببینید).

پرچم‌ها. هیچ اثری ندارد
کد منبع. PUSHF (بدون عملوند)
کد مقصد. 10011100

RCL/RCR : چرخش به چپ با رقم نقلی و چرخش به راست با رقم نقلی.

عمل. بیت‌ها را در CF چرخش می‌دهد. عملیات بیت‌های یک بایت، کلمه، یا دو کلمه‌ای (در ۸۰۳۸۶ و ما بعد) را در یک ثبات یا حافظه به چپ یا راست می‌چرخاند. عملوند ممکن است یک ثابت بلافصل یا ارجاع به CL باشد. در 8088/86، ثبات فقط می‌تواند ۱ باشد، و تعداد چرخش بیشتر در CL قرار می‌گیرد. در پردازشگرهای ما بعد، مقدار ثبات تا ۳۱ نیز می‌تواند باشد. برای RCL، سمت چپ‌ترین بیت وارد CF می‌شود و CF وارد بیت 0 مقصد می‌شود، و همه دیگر بیت‌ها به سمت چپ می‌چرخند. برای RCR، بیت 0 وارد CF می‌شود و بیت CF وارد بیت سمت چپ مقصد می‌شود و همهٔ دیگر بیت‌ها به راست می‌چرخند. (ROL و ROR را نیز ببینید.)

پرچم‌ها. بر CF و OF اثر می‌گذارد.
کد منبع. RCL/RCR register/memory,CL/immediate
کد مقصد.
RCL: |110100cw|mod 010 r/m|(if c=0, shift is 1;
RCR: |110100cw|mod011r/m|if c=1, shift is in CL)

REP : تکرار رشته

عمل. یک عملیات رشته‌ای به تعداد خاص تکرار می‌شود. REP یک پیشوند تکرار انتخابی است که قبل از دستورات رشته‌ای MOVS ، STOS ، INS و OUTS کد می‌شود). قبل از اجرا در CX تعداد شمارش را قرار دهید. برای هر بار اجرای دستور رشته‌ای، REP و CX را یک واحد می‌کاهد و عملیات تا زمانی که CX صفر شود تکرار می‌شود و در اینجا پردازش با اجرای دستور بعدی ادامه می‌یابد. REPNE/REPNZ (REPE/REPZ را نیز ببینید.)

پرچم‌ها. دستورات رشته‌ای مرتبط را ببینید.
کد منبع. REP string-instruction
کد مقصد. 11110010

REPE/REPZ/REPNE/REPNZ : تکرار مشروط رشته

عمل. یک عملیات رشته‌ای را به تعداد مرتبه خاصی تکرار می‌کند تا شرط خاصی برقرار شود. REPE، REPNE و REPNZ پیشوند تکرار انتخابی هستند که قبل از دستورات رشته‌ای SCAS و CMPS که می‌شوند که این دستورات ZF را تغییر می‌دهند. در CX قبل از اجرا تعداد تکرار را قرار دهید. برای REPE/REPZ (تکرار تا مساوی / صفرشدن)،

عملیات تا زمانیکه ZF یک شود (شرط صفر / مساوی) و CX صفر نباشد ادامه می‌یابد. برای REPNE/REPNZ (تکرار تا هنگامیکه مساوی/ صفر نیست)، عملیات تا زمانیکه ZF صفر است (شرط نامساوی/ غیر صفر) و CX صفر نیست، تکرار می‌شود. تا زمانیکه شرط برقرار است، عملیات CX را یک واحد می‌کاهد و دستور رشته‌ای اجرا می‌شود.

پرچم‌ها. دستورات رشته‌ای مرتبط را ببینید.

کد منبع. دستور رشته‌ای REPE/REPZ/REPNE/REPNZ

کد مقصد. REPNE/REPNZ : 1110010

REPE/REPZ : 1110011

RET/RETN/RETF : بازگشت از روال

عمل. از یک روال که قبلاً توسط یک CALL نزدیک یا دور به آن وارد شده، باز می‌گردد. اسمبلر، در صورتیکه روال برچسب NEAR داشته باشد، یک RET نزدیک تولید می‌کند و در صورتیکه روال برچسب FAR داشته باشد یک RET دور تولید می‌کند. در حالت نزدیک، RET کلمه بالای پشته را به IP منتقل می‌کند و SP را ۲ واحد می‌افزاید. برای حالت دور، RET کلمه بالای پشته را به IP و CS منتقل می‌کند و SP را ۴ واحد می‌افزاید. هر عملوند عددی (مثلاً RET 4) با SP جمع می‌شود. RETN و RETF توسط MASM 50 معرفی شده‌اند که می‌توانید بازگشت نزدیک یا دور را دقیقاً کد نمایید.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. RET/RETN/RETF [مقدار POP]

کد مقصد. چهار قالب دارد :

داخل یک سگمنت : |11000011|

داخل یک سگمنت با مقدار pop : |11000010|data-low|data-high|

بین سگمنت‌ها : |11001011|

بین سگمنت‌ها با مقدار pop : |11001010|data-low|data-high|

ROL/ROR : چرخش به چپ یا چرخش به راست

عمل. بیت‌های یک بایت، کلمه، یا دو کلمه‌ای (۸۰۳۸۶ و ما بعد) در یک ثبات یا حافظه را به چپ یا راست می‌چرخاند. عملوند ممکن است یک ثابت بلافصل یا ارجاعی به CL باشد. در 8088/86 مقدار ثابت فقط یک می‌تواند باشد و مقادیر چرخش بزرگتر در CL قرار می‌گیرند. در پردازشگرهای ما بعد، ثابت حداکثر ۳۱ می‌تواند باشد. برای ROL ، سمت چپ‌ترین بیت وارد بیت 0 مقصد می‌شود. دیگر بیت‌ها به سمت چپ چرخش دارند. در ROR، بیت 0 وارد سمت چپ‌ترین بیت مقصد می‌شود، دیگر بیت‌ها به سمت راست چرخش دارند. (RCL و RCR را نیز ببینید). بیت چرخش یافته نیز وارد CF می‌شود.

پرچم‌ها. بر CF و OF اثر می‌گذارد.

کد منبع. ROL/ROR register/memory,CL/immediate

کد مقصد.

ROL: |110100cw|mod000r/m|(if c=0 count=1)

ROR : |110100cw|mod001r/m|(if c=1 count is in cl)

SAHE : ذخیره سازی محتویات AH در پرچم‌ها

عمل. بیت‌های AH را در بیت‌های سمت راست ثبات پرچم ذخیره می‌کند. (LAHF را نیز ببینید).

پرچم‌ها. بر AF ، CF ، PF ، SF و ZF اثر می‌گذارد.
کد منبع. SAHF (بدون عملوند)
کد مقصد. 10011110

SAL/SAR : شیفت جبری به چپ یا شیفت جبری به راست

عمل. بیت‌های یک بایت، کلمه یا دو کلمه‌ای در یک ثبات یا حافظه را به راست یا چپ شیفت می‌دهد. عملوند باید یک مقدار بلافصل یا ارجاع به CL باشد. در 8088/86، ثابت باید فقط یک باشد و مقادیر بزرگتر شیفت باید در CL قرار گیرند. در پردازشگرهای مابعد مقدار ثبات حداکثر ۳۱ می‌تواند باشد. SAL به تعداد خاصی بیت‌ها را به سمت چپ شیفت می‌دهد و بیت‌های 0 در موقعیت‌های خالی سمت راست قرارمی‌دهد. SAL دقیقاً مانند SHL عمل می‌کند. SAR یک شیفت محاسباتی است که علامت فیلد ارجاع شده را در نظر می‌گیرد. SAR بیت‌ها را به تعداد خاصی به سمت راست شیفت می‌دهد و بیت علامت (1 یا 0) در سمت چپ پر می‌کند. همه بیت‌های شیفت داده شده از بین می‌روند.

پرچم‌ها. بر CF ، OF ، PF ، SF و ZF اثر می‌گذارد. (AF تعریف نشده است.)
کد منبع. SAL/SAR register/memory,cl/immediate
کد مقصد.
SAL: 110100cw|mod100r/m|(if c=0 count=1)
SAR: 110100cw|mod111r/m|(if c=1 count in cl)

SBB : تفریق با رقم فرضی

عمل. نوعاً در تفریق دودویی چند کلمه‌ای، برای انتقال رقم سرریز شده به مرحله حسابی بعدی استفاده می‌شود SBB ابتدا محتویات CF(0/1) را از اولین عملوند تفریق میکند و سپس دومین عملوند را از اولین عملوند مانند SUB تفریق می‌کند. (ADC را نیز ببینید.)

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد.
کد منبع. SBB register/memory,register/memory/immediate
کد مقصد. سه قالب دارد :

ثبات / حافظه با ثبات : |000110dw|modregr/m|
بلافصل با انباشتگر : |0001110w|--- data ---|data if w=1|
بلافصل با ثبات/حافظه : |100000sw|mod011r/m |--- data ---|data if sw=01|

SCAS/SCASB/SCASW/SCASD : پیمایش رشته

عمل. یک رشته موجود در حافظه را برای یافتن مقدار خاصی پیمایش می‌کند. برای SCASB مقدار را در AL قرار دهید، برای SCASW مقدار را در AX و برای SCASD (۸۰۳۸۶ و ما بعد) مقدار را در EAX قرار دهید. جفت ES:DI رشته‌ای در حافظه راکه باید پیمایش شود ارجاع می‌دهند. این عملیات عموماً با یک پیشوند REPE/REPNE استفاده می‌شود که تعداد شمارش در CX قرار می‌گیرد، از REPE برای یافتن اولین مورد مخالف و REPNE برای یافتن اولین مورد موافق استفاده کنید. اگر DF صفر باشد، عملیات حافظه را از چپ به راست پیمایش می‌کند و DI را می‌افزاید. اگر DI یک باشد، عملیات حافظه را از راست به چپ پیمایش می‌کند و DI را می‌کاهد. REPn برای هر بار تکرار CX را کاهش می‌دهد. وقتی یک شرط مساوی (REPNE) یا نامساوی (REPE) رخ دهد یا CX صفر شود، عملیات خاتمه می‌یابد. آخرین مقایسه پرچم‌ها را صفر / یک می‌کند. اگر شرط خاصی پیدا نشود، REP ، CX را کاهش می‌دهد تا صفر شود، در غیر اینصورت، DI و SI حاوی آدرس عناصر بعدی است.

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد.
کد منبع. (بدون عملوند) [REPnn] SCASB/SCASW/SCASD
کد مقصد. 1010111w

SETnn : تنظیم مشروط بایت (در ۸۰۳۸۶ و مابعد)

عمل. یک بایت خاص را بر حسب یک شرط تنظیم می‌کند. یک گروه با ۳۰ دستور، شامل SET(N)E ، SET(N)C ، SET(N)S و SET(N)L وجود دارد که مشابه تنظیم پرش‌های شرطی هستند. اگر یک شرط بررسی شده صحت داشته باشد، عملیات بایت عملوند را با یک و در غیر اینصورت با صفر تنظیم می‌کند. یک مثال چنین است :

```
CMP   AX,BX          ; مقایسه محتویات BX,AX
SETE  CL             ; اگر مساوی است، CL را با یک و در غیر اینصورت با صفر تنظیم کن
```

پرچم‌ها. هیچ اثری ندارد.
کد منبع. SETnn register/memory
کد مقصد. |00001111|1001cond|mod000r/m|
(cond بر طبق شرط بررسی شده تغییر می‌کند)

SHR/SHL شیفت منطقی به چپ یا شیفت منطقی به راست

عمل. بیت‌های یک بایت، کلمه، یا دو کلمه‌ای در یک ثبات یا حافظه را به چپ یا راست شیفت می‌دهد. عملوند ممکن است یک ثبات بلافصل یا ارجاعی به CL باشد. در 8088/86 ثابت ممکن است یک باشد و اگر مقدار بزرگتری باشد باید در CL قرار گیرد. در پردازشگرهای مابعد، ثابت حداکثر ۳۱ می‌تواند باشد. SHL و SHR شیفت‌های منطقی‌اند که با بیت علامت مانند یک بیت داده برخورد می‌کنند.
SHL بیت‌ها را به تعداد خاص به چپ شیفت می‌دهد و موقعیت‌های سمت راست خالی شده را با بیت‌های صفر پر می‌کند. SHL دقیقاً مانند SAL عمل می‌کند. SHR بیت‌ها را به تعداد خاص به راست شیفت می‌دهد و بیت‌های سمت چپ را با بیت صفر پر می‌کند. همه بیت‌های شیفت داده شده از بین می‌روند.

پرچم‌ها. بر CF ، OF ، PF ، SF و ZF اثر می‌گذارد. (AF تعریف نشده است)
کد منبع. SHL/SHR register/memory,cl/immediate
کد مقصد. SHL : |110100cw|mod100r/m|(if c=0 , count=1)
SHR : |110100cw|mod101r/m|(if c=1 , count in cl)

SHRD/SHLD : شیفت با دقت مضاعف (۸۰۳۸۶ و ما بعد)

عمل. چندین بیت را به یک عملوند شیفت می‌دهد. دستورات به سه عملوند نیاز دارند. اولین عملوند یک ثبات ۱۶ یا ۳۲ بیتی است با موقعیت حافظه است که شامل مقداری است که باید شیفت داده شود. دومین عملوند یک ثبات (با همان اندازهٔ عملوند اول) است که حاوی بیت‌هایی است که در اولین عملوند شیفت داده می‌شود. سومین عملوند CL یا ثبات بلافصل است که حاوی مقدار شیفت است.

پرچم‌ها. بر CF ، OF ، PF ، SF و ZF اثر می‌گذارد. (AF تعریف نشده است).
کد منبع. SHLD/SHRD register/memory,register,CL/immediate
کد مقصد. |00001111|10100100|modregr/m|

STC : یک کردن پرچم نقلی

عمل. CF را یک می‌کند (CLC پرچم نقلی را صفر می‌کند)

پرچم‌ها. CF را یک می‌کند

کد منبع. STC (بدون عملوند)

کد مقصد. 11111001

STD : یک کردن پرچم جهت

عمل. DF را یک می‌کند تا سبب شود عملیات رشته‌ای مانند MOVS پردازش را از راست به چپ انجام دهد. (CLD را برای صفر کردن DF ببینید.)

پرچم‌ها. DF را یک می‌کند.

کد منبع. STD (بدون عملوند)

کد مقصد. 11111101

STI : یک کردن پرچم وقفه

عمل. IF را یک می‌کند تا وقفه‌های خارجی را بعد از اجرای دستور بعدی فعال سازد. (CLI را برای صفر کردن IF ببینید.)

پرچم‌ها. IF را یک می‌کند.

کد منبع. STI (بدون عملوند)

کد مقصد. 11111011

STOS/STOSB/STOSW/STOSD

عمل. محتویات انباشتگر را در حافظه ذخیره می‌کند. وقتی با یک پیشوند REP با تعداد شمارش در CX استفاده شود، عملیات مقادیر یک رشته را به دفعات مشخص شده کپی می‌کند، این عملیات برای فعالیتهایی نظیر پاک کردن یک ناحیه حافظه مناسب است. برای STOSD مقدار را در AL قرار دهید، برای STOSW مقدار را در AX و برای STOSD مقدار را در EAX قرار دهید. جفت ES:DI موقعیتی در حافظه را ارجاع می‌دهند که مقدار باید ذخیره شود. اگر DF صفر باشد، عملیات ذخیره سازی در حافظه از چپ به راست انجام می‌گیرد و DI را می‌افزاید. اگر DF یک باشد، عملیات از راست به چپ، ذخیره سازی را انجام می‌دهد و DI را می‌کاهد. REP برای هر بار تکرار CX را می‌کاهد و وقتی صفر شد عملیات خاتمه می‌یابد.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. [REP] STOSB/STOSW/STOSD (بدون عملوند)

کد مقصد. 1010101w

SUB : تفریق مقادیر دودویی

عمل. مقادیر دودویی در یک ثبات، حافظه یا بلافصل را از یک ثبات تفریق می‌کند، یا مقادیر در یک ثبات یا بلافصل را از حافظه تفریق می‌کند. مقادیر ممکن است بایت، کلمه یا دو کلمه‌ای باشند. (۸۰۳۸۶ و ما بعد). (SBB را ببینید.)

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد.

کد منبع. SUB register/memory , register/memory/immediate

کد مقصد. سه قالب دارد :

ثبات / حافظه با ثبات : |001010dw|modregr/m |

بلافصل از انباشتگر : |0010110w|--- data ---|data if w=1|

بلافصل از ثبات/ حافظه : |100000sw|mod101r/m|--- data ---|data if sw=01|

TEST : بررسی بیت‌ها

عمل. عمل AND منطقی را انجام می‌دهد، اما عملوند مقصد را تغییر نمی‌دهد. هر دو عملوند باید بایت، کلمه، یا دو کلمه‌ای (۸۰۳۸۶ و مابعد) در یک ثبات یا حافظه باشد، دومین عملوند ممکن است بلافصل باشد. بعد از اجرای آن، ممکن است از JE یا JNE برای بررسی پرچم‌ها استفاده کنید.

پرچم‌ها. CF و OF را صفر می‌کند و بر PF ، SF و ZF اثر می‌گذارد. (AF تعریف نشده است.)

کد منبع. TEST register/memory , register/memory/immediate

کد مقصد. سه قالب دارد :

ثبات/ حافظه و ثبات : |1000010w|modregr/m|

بلافصل با انباشتگر : |1010100w|--- data ---|data if w=1|

بلافصل با ثبات/ حافظه : |1111011w|mod000r/m|--- data ---|data if w=1

WAIT : قرار دادن پردازشگر در حالت انتظار

عمل. به پردازشگر اصلی اجازه می‌دهد تا در حالت انتظار باقی بماند تا هنگامیکه یک وقفه خارجی رخ دهد، بدین منظور که با یک کمک پردازشگر همزمان شود. پردازشگر اصلی منتظر می‌ماند تا کمک پردازشگر اجرا را خاتمه دهد و با دریافت یک علامت از پایه TEST پردازش را آغاز کند.

پرچم‌ها. هیچ اثری ندارد.

کد منبع. WAIT (بدون عملوند)

کد مقصد. 10011011

XADD : معاوضه و جمع (۸۰۴۸۶ و ما بعد)

عمل. عملوندهای منبع و مقصد را با هم جمع می‌کند و حاصل را در مقصد ذخیره می‌سازد. همچنین مقدار اصلی مقصد را به منبع منتقل می‌کند.

پرچم‌ها. بر AF ، CF ، OF ، PF ، SF و ZF اثر می‌گذارد.

کد منبع. XADD register/memory,register

کد مقصد. |00001111|1100000b|modregr/m|

XCHG : معاوضه

عمل. داده‌ها را بین دو ثبات (بصورت XCHGAH,BL) یا بین یک ثبات و حافظه (بصورت XCHG (CX,word معاوضه می‌کند.

پرچم‌ها. هیچ اثری ندارد.
کد منبع. XCHG register/memory,register/memory
کد مقصد. دو قالب دارد :
ثبات با انباشتگر : |10010reg|
ثبات/ حافظه با ثبات : |1000011w|modregr/m|

XLAT/XLATB : تبدیل

عمل. بایت‌ها را به یک قالب متفاوت مانند، ASCII به EBCDLC تبدیل می‌کند. یک جدول را تعریف کنید، آدرس آن را در BX یا EBX برای اندازهٔ ۳۲ بیت قرار دهید و در AL مقداری را که باید تبدیل شود قرار دهید. عملیات از مقدار AL بعنوان یک افست جدول استفاده می‌کند، بایت را از جدول انتخاب می‌کند و آن را در AL ذخیره می‌سازد. (XLATB معادل XLAT است.)

پرچم‌ها. هیچ اثری ندارد.
کد منبع. XLAT [AL] (عملوند AL اختیاری است)
کد مقصد. 11010111

XOR : OR انحصاری

عمل. یک OR انحصاری منطقی روی بیت‌های دو عملوند انجام می‌دهد. هر دو عملوند باید بایت، کلمه، یا دو کلمه‌ای (۸۰۳۸۶ و مابعد) باشند که XOR بیت‌ها را با هم جور کند. برای هر جفت بیت جور شده، اگر هر دو یکسان باشند، بیت اولین عملوند صفر خواهد شد، اگر بیت‌های جور شده متفاوت باشند بیت اولین عملوند یک خواهد شد.(AND و OR را نیز ببینید.)

پرچم‌ها. بر CF(0) ، OF(0) ، PF ، SF و ZF اثر می‌گذارد. (AF تعریف نشده است)
کد منبع. XOR /memory,register/memory/immediate register
کد مقصد. سه قالب دارد :
ثبات / حافظه با ثبات : |001100dw|modregr/m |
بلافصل با ثبات / حافظه : |1000000w|mod110r/m|--- data ---|data if w=1 |
بلافصل با انباشتگر : |0011010w|--- data ---|data if w=1 |

ضمیمهٔ الف

تبدیل اعداد شانزدهی و دهدهی

این ضمیمه، مراحل مورد نیاز تبدیلات بین اعداد شانزدهی و دهدهی را ارائه می‌کند. اولین بخش نحوهٔ تبدیل A7B8 شانزدهی به 42936 دهدهی و بخش دوم نحوه تبدیل 42936 به A7B8 را نشان می‌دهد.

تبدیل اعداد شانزدهی به دهدهی

برای تبدیل عدد شانزدهی به عدد دهدهی، با سمت چپ‌ترین رقم آغاز کنید و هر رقم شانزدهی را در ۱۶ ضرب و حاصل را ذخیره کنید، چون ضرب در مبنای ده می‌باشد اعداد شانزدهی A تا F را به دهدهی 10 تا 15 تبدیل کنید. مراحل تبدیل A7B8H به قالب دهدهی چنین است :

اولین رقم : A(10)	۱۰
ضرب در ۱۶	×۱۶
	۱۶۰
جمع با رقم بعدی ۷	+۷
	۱۶۷
ضرب در ۱۶	×۱۶
	۲۶۷۲
جمع با رقم بعدی، B(۱۱)	+۱۱
	۲۶۸۳
ضرب در ۱۶	×۱۶
	۴۲۹۲۸
جمع با رقم بعدی، ۸	+۸
مقدار دهدهی	۴۲۹۳۶

همچنین می‌توانید از جدول تبدیل استفاده کنید. برای A7B8H، رقم راست (8) را در موقعیت اول، رقم بعدی سمت چپ (B) را در موقعیت دوم، رقم بعدی (7) را در موقعیت سوم و رقم بعدی (A) را در موقعیت چهارم در نظر بگیرید. به جدول A-1 مراجعه و مقدار هر رقم را در موقعیت خودش قرار دهید :

رقم 8 در موقعیت ۱، ستون ۱ =	8
رقم B در موقعیت ۲، ستون ۲ =	176
رقم 7 در موقعیت ۳، ستون ۳ =	1792
رقم A در موقعیت ۴، ستون ۴ =	40960
مقدار دهدهی	42936

جدول A-1 جدول تبدیلات شانزدهی ـ دهدهی

Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240
	8		7		6		5		4		3		2

Hex	Dec
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15
	1

تبدیل یک عدد دهدهی به شانزدهی

برای تبدیل عدد دهدهی : ۴۲۹۳۶ به شانزدهی، ابتدا ۴۲۹۳۶ را بر ۱۶ تقسیم کنید، باقیمانده سمت راست‌ترین رقم شانزدهی است، ۸. سپس خارج قسمت بعدی ۲۶۸۳ را بر ۱۶ تقسیم کنید، باقیمانده ۱۱ یا B رقم شانزدهی سمت چپ ۸ است. با این روش آنقدر ادامه دهید تا خارج قسمت صفر شود. مراحل طی شده چنین می‌باشد :

شانزدهی	باقیمانده	خارج قسمت	عملیات
8 (سمت راست)	8	2683	42936/16
B	11	167	3683/16
7	7	10	167/16
A (سمت چپ)	10	0	10/16

از جدول A-1 برای تبدیل دهدهی به شانزدهی نیز می‌توانید استفاده نمایید. برای عدد دهدهی ۴۲۹۳۶، عدد را در جدولی که مساوی یا کوچکتر نزدیک به آن می‌باشد، قرار دهید. عدد شانزدهی معادل و موقعیت آن در جدول را یادداشت کنید. مقدار دهدهی آن رقم شانزدهی را از ۴۲۹۳۶ کم کنید و اختلاف را در جدول قرار دهید. این سئوال چنین عمل می‌کند :

	شانزدهی	دهدهی
مقدار دهدهی اولیه		۴۲۹۳۶
تفریق عدد کوچکتر نزدیک به آن	A000	-۴۰۹۶۰
اختلاف		۱۹۷۶
تفریق عدد کوچکتر نزدیک به آن	700	-۱۷۹۲
اختلاف		۱۸۴
تفریق عدد کوچکتر نزدیک به آن	B0	-۱۷۶
	8	۸
مقدار شانزدهی نهایی	A7B8	

ضمیمه ب

کد کاراکترهای ASCII

عبارت ASCII مخفف "کد استاندارد امریکایی برای تبادل اطلاعات" است. جدول B-1 همهٔ ۲۵۶ کد کاراکتر (00H تا FFH) ASCII را همراه با مقدار شانزدهی آنها بیان می‌کند. طبقه بندی کدهای کاراکتر چنین است :

00-1FH کدهای کنترلی برای صفحه نمایش، چاپگرها و انتقال داده، که برای ایجاد یک فعالیت در نظر گرفته شده است.

20-7FH کدهای کاراکتر اعداد، حروف و نقطه گذاری. (20H جای خالی استاندارد است).

80-FFH کدهای ASCII توسعه یافته، کاراکترهای زبان فارسی، سمبول‌های یونانی و ریاضی و کاراکترهای گرافیکی برای رسم کادر.

در اینجا کدهای کنترلی از 00H تا 1FH است، که آنهایی که داخل پرانتز هستند سمبول قابل چاپ ندارند :

HEX	CHARACTER	HEX	CHARACTER	HEX	CHARACTER
00	(Null)	01	Happy face	02	Happy face
03	Heart	04	Diamond	05	Club
06	Spade	07	(Beep)	08	(Back space)
09	(Tab)	0A	(Line feed)	0B	(Vertical tab)
0C	(Form Feed)	0D	(Return)	0E	(Shift out)
0F	(Shift in)	10	(Data line esc)	11	(Dev ctl 1)
12	(Dev ctl 2)	13	(Dev ctl 3)	14	(Dev ctl 4)
15	(Neg acknowledge)	16	(Synch idle)	17	(End tran block)
18	(Cancel)	19	(End of medium)	1A	(Substitute)
1B	(Escape)	1C	(File separator)	1D	(Group separator)
1E	(Record separator)	1F	(Unit separator)		

جدول B-1 مجموعهٔ کاراکترهای ASCII

00		20		40	@	60	`	80	Ç	A0	á	C0	└	E0	α
01	☺	21	!	41	A	61	a	81	ü	A1	í	C1	┴	E1	ß
02	●	22	"	42	B	62	b	82	é	A2	ó	C2	┬	E2	Γ
03	♥	23	#	43	C	63	c	83	â	A3	ú	C3	├	E3	π
04	♦	24	$	44	D	64	d	84	ä	A4	ñ	C4	-	E4	Σ
05	♣	25	%	45	E	65	e	85	à	A5	Ñ	C5	┼	E5	σ
06	♠	26	&	46	F	66	f	86	å	A6	ª	C6	╞	E6	µ
07		27	'	47	G	67	g	87	ç	A7	º	C7	╟	E7	τ
08		28	(	48	H	68	h	88	ê	A8	¿	C8	╚	E8	Φ
09		29	)	49	I	69	i	89	ë	A9	⌐	C9	╔	E9	θ
0A		2A	*	4A	J	6A	j	8A	è	AA	¬	CA	╩	EA	Ω
0B		2B	+	4B	K	6B	k	8B	ï	AB	½	CB	╦	EB	δ
0C		2C	,	4C	L	6C	l	8C	î	AC	¼	CC	╠	EC	∞
0D		2D	-	4D	M	6D	m	8D	ì	AD	¡	CD	═	ED	φ
0E		2E	.	4E	N	6E	n	8E	Ä	AE	«	CE	╬	EE	ε
0F		2F	/	4F	O	6F	o	8F	Å	AF	»	CF	╧	EF	∩
10	►	30	0	50	P	70	p	90	É	B0	▒	D0	╨	F0	≡
11	◄	31	1	51	Q	71	q	91	æ	B1	▒	D1	╤	F1	±
12	↕	32	2	52	R	72	r	92	Æ	B2	▓	D2	╥	F2	≥
13	‼	33	3	53	S	73	s	93	ô	B3	│	D3	╙	F3	≤
14	¶	34	4	54	T	74	t	94	ö	B4	┤	D4	╘	F4	⌠
15	§	35	5	55	U	75	u	95	ò	B5	╡	D5	╒	F5	⌡
16	▬	36	6	56	V	76	v	96	û	B6	╢	D6	╓	F6	÷
17	↨	37	7	57	W	77	w	97	ù	B7	╖	D7	╫	F7	≈
18	↑	38	8	58	X	78	x	98	ÿ	B8	╕	D8	╪	F8	°
19	↓	39	9	59	Y	79	y	99	Ö	B9	╣	D9	┘	F9	·
1A		3A	:	5A	Z	7A	z	9A	Ü	BA	║	DA	┌	FA	·
1B		3B	;	5B	[	7B	{	9B	¢	BB	╗	DB	▉	FB	√
1C	∟	3C	<	5C	\	7C	|	9C	£	BC	╝	DC	▄	FC	n
1D	↔	3D	=	5D	]	7D	}	9D	¥	BD	╜	DD	▌	FD	²
1E	▲	3E	>	5E	^	7E	~	9E	₧	BE	╛	DE	▐	FE	■
1F		3F	?	5F	_	7F	⌂	9F	ƒ	BF	┐	DF	▀	FF	

ضمیمه پ

کلمات رزرو شده

اسمبلر برخی کلمات که منظور خاصی دارند را تشخیص می‌دهد، استفاده از این کلمات فقط تحت شرایط مقرر شده امکان دارد. کلماتی که برای اسمبلر رزرو شده در چهار دسته طبقه بندی می‌شوند :

- نام‌های ثبات مانند AX و AH.
- دستورات سمبولیک، مانند ADD و MOV.
- پیش پردازنده‌ها(دستورات به اسمبلر) مانند PROC و END.
- عملگرها مانند DUP و SEG.

اگر خیلی از کلمات رزرو شدهٔ زیر را برای تعریف عنصر داده استفاده کنید سبب گمراه شدن اسمبلر یا ایجاد خطای اسمبلر خواهدشد. یک اسمبلر خاص ممکن‌است کلمات رزرو شده‌ای علاوه‌بر آنهایی که لیست شده‌است، داشته‌باشد.

نام‌های ثبات

AH, AL, AX, BH, BL, BP, BX, CH, CL, CS, CX, DH, DI, DL, DS, DX, EAX, EBP, EBX, ECX, EDI, EDX, EIP, ES, ESI, FS, GS, IP, SI, SP, SS

دستورات سمبولیک

AAA, AAD, AAM, AAS, ADC, ADD, AND, ARPL, BOUND, BSF, BSR, BTn, CALL, CBW, CDQ, CLC, CLD, CLI, CLTS, CMC, CMP, CMPSn, CMPXCHG, CMPXCHG8B, CWDn, DAA, DAS, DEC, DIV, ENTER, ESC, HLT, IDIV, IMUL, IN, INC, INSn, INT, INTO, IRET, JA, JAE, JB, JBE, JCXZ, JE, JECXZ, JG, JGE, JL, JLE, JMP, JNA, JNAE, JNB, JNBE, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ, LAHF, LAR, LDS, LEA, LEAVE, LES, LFS, LGDT, LGS, LIDT, LLDT, LMSW, LOCK, LODSn, LOOP, LOOPE, LOOPNEn, LOOPNZn, LOOPZ, LSL, LSS, LSS, LTR, MOV, MOVSn, MOVSX, MOVZX, MUL, NEG, NOP, NOT, OR, OUTn, POP, POPA, POPAD, POPF, POPFD, PUSH, PUSHAD, PUSHF, PUSHFD, RCL, RCR, REN, REP, REPE, REPNE, REPNZ, REPZ, RET, RETF, ROL, ROR, SAHF, SAL, SAR, SBB, SCASn, SETnn, SGDT, SHL, SHLD, SHR, SHRD, SIDT, SLDT, SMSW, STC, STD, STI, STOSn, STR, SUB, TEST, VERR, VERRW, WAIT, XADD, XCHG, XLAT, XOR

پیش پردازنده‌ها

ALIGN, .ALPHA, ASSUME, BYTE, .CODE, COMM, COMMENT, .CONST, .CREF, .DATA, .DATA?, DB, DD, DF, DOSSEG, DQ, DT, DW, DWORD, ELSE, END, ENDIF, ENDM, ENDP, ENDS, EQU, .ERRnn, EVEN, EXITM, EXTRN, EXTERN, .FARDATA, .FARDATA?, FWORD, GROUP, IF, IF1, IF2, IFB, IFDEF, IFDIF, IFE, IFIDN, IFNB, IFNDEF, INCLUDE, INCLUDELIB, IRP, IRPC, LABEL, .LALL, .LFCOND, .LIST, LOCAL, MACRO, .MODEL, NAME, ORG, &OUT, PAGE, PROC, PUBLIC, PURGE, QWORD, .RADIX, RECORD, REPT, .SALL, SEGMENT, .SEQ, .SFCOND, .STACK, STRUC, SUBTTL, .TFCOND, TITLE, TWORD, UNION, WORD, .XALL, .XCREF, .XLIST

عملگرها

AND, BYTE, COMMENT, CON, DUP, EQ, FAR, GE, GT, HIGH, LE, LENGTH, LINE, LOW, LT, MASK, MOD, NE, NEAR, NOT, NOTHING, OFFSET, OR, PTR, SEG, SHL, SHORT, SHR, SIZE, STACK, THIS, TYPE, WHILE, WIDTH, WORD, XOR

اسمبلر و انتخاب‌های لینک

این ضمیمه قوانین اسمبل کردن، لینک کردن، تولید فایلهای ارجاع متقابل و تبدیل برنامه‌های EXE. به COM. را در بر دارد. نسخهٔ مایکروسافت اسمبلر MASM و بورلند TASM است که هر دو مشابه هستند. از نسخهٔ 6.0 اسمبلر مایکروسافت از فرمان ML استفاده می‌کند که اسمبل و لینک را با یک فرمان انجام می‌دهد. مقاله‌های این ضمیمه بطور اختیاری از درایو : D به عنوان مسیر همهٔ برنامه‌ها و فایلها استفاده می‌کنند، برای بکار بردن دیگر درایوها مسیر مناسب مانند : C: یا C:\subdirectory را جایگزین نمایید.

نسخه‌های اسمبلر مختلف مجموعه انتخابهای بی‌پایانی دارند که همه آنها در اینجا آورده نشده است.

اسمبل کردن یک برنامه

شمامی‌توانید یک خط فرمان برای‌احضار اسمبل استفاده کنید اگرچه MASM حالت اعلان را نیز فراهم‌نموده‌است.

اسمبل کردن با یک خط فرمان

قالب کلی استفاده از یک خط فرمان برای اسمبل کردن چنین است :

```
MASM/TASM  [option] source [,Object] [, Listing] [, crossref]
```

- Options بعداً توضیح داده خواهد شد.
- source برنامهٔ منبع را مشخص می‌سازد. اسمبلر توسعهٔ آن را ASM. فرض می‌کند بنابراین نیازی نیست آن را وارد کنید. مسیر را وارد کنید مانند :D یا D:\subdirectory \filename
- Object نام یک فایل Obj. تولید شده است. مسیر و نام فایل می‌تواند مشابه یا متفاوت با فایل منبع باشد.
- Listing نام فایل LST. است که حاوی کد منبع و مقصد است. مسیر و نام فایل می‌تواند مشابه یا متفاوت با فایل منبع باشد.
- Crossref نام فایل تولید شده‌ای است که حاوی سمبولهای لیست ارجاع متقابل است. توسعه CRF. برای MASM و XRF. برای TASM است. مسیر و نام فایل می‌تواند مشابه یا متفاوت با فایل منبع باشد.

مثال زیر همه فایل‌ها را هجی کرده است : `MASM D:name.ASM , D:name.OBJ , D:name.LST,D:name.CRF`

فرمان کوتاه زیر از پیش فرض برای فایلهای مقصد، لیستگیری و ارجاع متقابل، همه با یک نام استفاده می‌کند :

```
MASM  D:filename , D: , D: , D:
```

مثال بعدی فایل ارجاع متقابل را درخواست می‌کند، اما فایل لیستگیری را درخواست نمی‌کند(دو علامت کاما را یادداشت کنید) : `MASM  D:filename , D: , , D:`

اسمبل کردن با اعلان

همچنین می‌توانید نام اسمبلر خط فرمان وارد کنید، اما TASM و MASM (تا نسخهٔ ۵۰) بطور متفاوتی پاسخ می‌دهند، TASM قالب کلی برای خط فرمان بدون توضیح انتخاب‌ها را نشان می‌دهد، در حالیکه MASM لیستی از اعلان‌هایی که پاسخ می‌دهید/ نشان می‌دهد :

نام فایل منبع	[ASM]:
نام فایل مقصد	[SOURCE.OBJ]:
لیست منبع	[NUL.LST]:
ارجاع متقابل	[NUL.CRF]:

- ***نام فایل منبع*** نام فایل منبع را تعیین می‌کند. مسیر (اگر پیش فرض استفاده نمی‌کنید) و نام فایل منبع را بدون توسعهٔ ASM. وارد کنید.
- ***نام فایل مقصد*** نام فایل مقصد را مهیا می‌سازد. اعلان نام فایل را مشابه در نظر می‌گیرد، اگر چه می‌توانید آن را تغییر دهید، برای دریافت یک فایل مقصد روی درایو، :D وارد کنید :D و سپس <Entetr> را فشار دهید.
- ***لیست منبع*** نام لیست اسمبل شده را مهیا می‌سازد. اگر چه اعلان در نظر می‌گیرد که شما آن را نمی‌خواهید برای دریافت یک لیست روی درایو :D وارد کنید :D و سپس <Enter> را فشار دهید.
- ***ارجاع متقابل*** نام لیست ارجاع متقابل را مهیا می‌سازد، اگر چه اعلان در نظر می‌گیرد که شما آن را نمی‌خواهید برای دریافت آن روی درایو D وارد کنید:D و <Enter> را فشار دهید. توسعه آن در مایکروسافتی CRF. و در بورلند XRF. است.

برای سه اعلان آخر، با فشردن <Enter> پیش فرض را می‌پذیرید.

انتخاب‌های اسمبلر

انتخاب‌های اسمبلر برای MASM و TASM شامل موارد زیر است :

/A مرتب کردن سگمنت‌های منبع به ترتیب حروف الفبا.

/C ایجاد یک جدول ارجاع متقابل در فایل LST.

/D در MASM: در هر دو مرحلهٔ فایل لیست برای خطاها ایجاد می‌کند. در TASM /D symbol یک سمبول را تعریف می‌کند.

/E دستورات کمک پردازشگر 80X87 را می‌پذیرد و یک پیوند با BASIC ، C یا FORTRAN برای دستورات ممیز شناور، تولید می‌کند.

/H انتخاب‌های‌اسمبلر و توضیح مختصری ازآنها را نمایش‌می‌دهد.H/ را بدون نام فایل و دیگرانتخاب‌ها واردکنید.

/L ایجاد یک فایل لیست معمولی (LST.) خط فرمان نیز یک مسیر برای این انتخاب فراهم می‌سازد.

/LA برای TASM ، یک فایل لیست توسعه یافته (LST.) ایجاد می‌کند.

/M# برای TASM ، امکان # تعداد از مراحل برای حل ارجاع‌های به جلو را می‌دهد.

/ML همهٔ نام‌ها را نسبت به حالت حروف حساس می‌سازد.

/MV همهٔ نام‌ها را به حروف بزرگ تبدیل می‌کند.

/MX نام‌های عمومی و خارجی را به حالت حروف حساس می‌سازد.

/N جلوگیری از ایجاد جدول سمبول در فایل LST.

/R فراهم سازی پشتیبانی کمک پردازشگر ریاضی حقیقی.

/S باقی ماندن سگمنت‌های منبع در حالت اصلی.

/T نمایش تشخیص‌ها در انتهای اسمبل کردن اگر فقط یک خطا دیده شده است. (کوتاه نویس).

/V در انتهای اسمبل کردن، تعداد خطوط و سمبول‌های پردازش شده را نمایش می‌دهد. (طولانی‌نویس).

/Wn تنظیم سطح پیغامهای هشدار دهنده : 0 = فقط نمایش خطاها، 1= نمایش خطاها و هشدارهای مهم (پیش فرض)، 2 = نمایش خطاها و هشدارهای مهم و هشدارهای مشورت‌آمیز.

/Z نمایش خط‌های منبع روی صفحه برای خطاها.

/ZD مشمول ساختن اطلاعاتی راجع به شمارهٔ خط در فایل مقصد برای برنامه‌های Codeview یا TurboDebugger.

این انتخاب‌ها را هم در اعلان و عم در حالت خط فرمان می‌توانید درخواست کنید. برای مثال در حالت اعلان، باید MASM /A/V را وارد کنید و سپس نام فایل مورد استفاده را وارد کنید. یا هر انتخاب را در جلوی خط اعلان وارد کنید، به صورت filename /A/V <Enter> یا /A/V filename : [.ASM] نام فایل

انتخاب‌های /A/V به اسمبلر می‌گوید که سگمنت‌ها را به ترتیب حروف الفبا مرتب کند و تشخیص‌ها را در انتهای اسمبل کردن نمایش دهد. راهنمای اسمبلر خودتان را برای انتخاب‌های دیگر ببینید.

نسخهٔ مایکروسافت 6.x

خط فرمان اسمبلر مایکروسافت از نسخهٔ 6.0 چنین است :

```
ML [Option] filenames [Options] Filenames] . . . [/Link options]
```

اسمبلر به شما این امکان را می‌دهد که هر تعداد برنامه را در یک ماجول اجرایی اسمبل و لینک نمایید، یک انتخاب مفید ML-? است که املاء کامل خط فرمان و انتخاب‌های آن را نمایش می‌دهد.

خصیصه‌های اضافی توربواسمبلر

توربواسمبلر این امکان را دارد که چندین فایل را در یک خط فرمان اسمبل کنید که هر کدام انتخاب‌های خود را دارد. نیز می‌توانید از حروف جایگزین (* , ?) استفاده نمایید برای اسمبل کردن همهٔ برنامه‌های منبع در شاخهٔ جاری وارد کنید : *.TASM. برای اسمبل کردن همه برنامه‌های منبع با نام PROG2.ASM و PROG1.ASM و غیره وارد کنید : TASM PROG? همچنین می‌توانید گروهی (یا مجموعه‌ای) از نام‌های فایل که هر گروه توسط یک علامت بعلاوه (+) جدا شده است وارد کنید فرمان زیر PROGB و PROGA را با انتخاب /C و PROGC را با انتخاب /A اسمبل می‌کند. TASM /C PROGA PROGB + /A PROGC

درخواست انتخاب /W سبب می‌شود TASM پیغامهای هشدار دهنده برای کدهای بدون کارآیی نیز تولید نماید. حالت ایده‌آل خصیصه‌های اضافی بسیاری دارد. بورلند دو نسخهٔ اسمبلر دیگر، TASMX و TASM 32 در حالت محافظت شده نیز دارد.

جداول

در زیر یک لیست LST اسمبلر است که جدول سگمنت‌ها و گروهها و یک جدول سمبول دارد.

جدول سگمنت‌ها و گروه‌ها. این جدول یک سرفصل مشابه زیر دارد : Name Length Align Combine Class

- ستون ***نام***، نام همه سگمنت‌ها و گروه‌ها را به ترتیب حروف الفبا ارائه می‌دهد.
- ستون ***طول***،اندازهٔ شانزدهی هر سگمنت را ارائه می‌دهد.
- ستون *align* نوع قرارگیری مانند BYTE ، WORD یا PARA را ارائه می‌دهد.
- ستون *combine* نوع ترکیب تعریف شده، مانند STACK برای پشته، NONE برای نوع کد نشده PUBLIC برای

تعاریف خارجی، یا یک آدرس شانزدهی برای انواع AT ارائه می‌دهد.

- ستون *Class* نام کلاس سگمنت را که در جملهٔ SGMENT کد شده، لیست می‌کند.

جدول سمبول. یک جدول سمبول عنوانی مشابه این عنوان دارد : Name Type Value Attribute

و ستون *name* نام همهٔ عناصر تعریف شده به ترتیب حروف الفبا را خواهد داشت.

- ستون type انواع زیر را ارائه می‌دهد :

LNEAR یا LFAR : برچسب نزدیک یا برچسب دور

NPROG یا FPROG : یک روال نزدیک یا دور

BYTE ، WORD ، DWORD ، FWORD ، QWORD ، TBYTE : یک عنصر داده

ALIAS : یک ناحیه کاری برای سمبول دیگر

NUMBER : یک برچسب مطلق

OPCODE : یک معادل برای عملوند دستور

TEXT : یک معادل برای متن

- ستون Value افست شانزدهی از شروع سگمنت را برای نام‌ها، برچسب‌ها و روال‌ها ارائه می‌دهد.
- ستون attribute صفات سمبول‌ها را لیست می‌کند که شامل سگمنت‌ها و طول آنهاست.

فایل ارجاع متقابل

یک فایل CRE. یا XRF. یک لیست ارجاع متقابل از برچسب‌های برنامه، سمبول‌ها و متغیرها تولید می‌کند، اما باید برای مایکروسافت از CREF و برای بورلند از TCREF جهت تبدیل لیست به یک فایل ارجاع متقابل مرتب شده استفاده کنید. می‌توانید CREF یا TCREF را با خط فرمان یا با اعلان وارد کنید.

استفاده از یک خط فرمان

قالب کلی برای استفاده از یک خط فرمان چنین است :

```
CREF/TCREF d:xreffile , d:reffile
```

- Xreffile فایل ارجاع متقابل تولید شده توسط اسمبلر را مشخص می‌سازد. برنامه توسعه را در نظر می‌گیرد، بنابراین نیازی به وارد کردن آن ندارید.
- Reffile برای تولید یک فایل REF. فراهم شده‌است. مسیر و نام فایل ممکن است مشابه یا متفاوت بانام منبع باشد.

مثال زیر یک فایل ارجاع متقابل با نام ASMPROG.REF روی درایو :D می‌نویسد :

```
CREF/TCREF D:ASMPROG , D:
```

استفاده از اعلان

می‌توانید CREF یا TCREF بدون خط فرمان نیز وارد کنید. TCREF قالب کلی فرمان و توضیحی از انتخاب‌های آن را وارد می‌کند، در حالیکه CREF این اعلان‌ها را نمایش می‌دهد :

```
Cross-reference [.CRF]:

Listing [filename.REF]:
```

برای اولین اعلان نام فایل بدون توسعهٔ CRF. را وارد کنید. برای دومین اعلان فقط مسیر را می‌توانید وارد کنید و نام فایل پیش فرض را می‌پذیرد. این انتخاب سبب می‌شود CREF یک فایل ارجاع متقابل با نام REF. را بنویسید.

لینک کردن یک برنامه

لینکر مایکروسافت LINK و بورلند TLINK است. هر دو لینکر خط فرمان را برای درخواست لینک می‌پذیرند، اما LINK اعلان نیز فراهم نموده است.

لینک کردن با یک خط فرمان

قالب کلی برای استفاده از خط فرمان جهت لینک کردن چنین است :

```
LINK/TLINK [option] objfile,exefile[, mapfile][, Library file]
```

- Option قبلاً توضیح داده شد.
- Objfile فایل مقصد تولید شده توسط اسمبلر را مشخص می‌سازد. لینکر توسعهٔ آن را OBJ فرض می‌کند پس نیاز به وارد کردن آن نیست. همچنین می‌توانید یک مسیر را وارد کنید.
- Exefile برای تولید کردن یک فایل EXE. فراهم شده است. مسیر و نام فایل می‌تواند مشابه یا متفاوت با منبع باشد.
- Mapfile برای تولید یک فایل با توسعهٔ MAP. فراهم شده است که این فایل مبنی بر ارتباط موقعیت و اندازهٔ هر سگمنت و هر خطایی است که لینکر می‌یابد، یک نوع خطا عدم موفقیت در تعریف سگمنت پشته است. وارد کردن CON به لینکر می‌گوید که نقشه را بر روی صفحه نمایش دهد(بجای آنکه بر روی دیسک بنویسد) بنابراین می‌توانید خطاها را بلافاصله ببینید.
- Library file برای انتخاب کتابخانه‌ها فراهم شده است.

برای لینک کردن چندین فایل مقصد در یک ماجول اجرایی، آنها را در یک خط ترکیب کنید :

```
LINK/TLINK D:PROGA+D:PROGB+D:PROGC
```

لینک کردن با استفاده از اعلان

می‌توانید نام لینکر را بدون خط فرمان وارد کنید، اگر چه TLINK و LINK بطور متفاوتی پاسخ می‌دهند. TLINK قالب کلی برای فرمان و توضیح هر انتخاب را فراهم می‌سازد، در حالیکه LINK لیستی از اعلان‌ها را نمایش می‌دهد. اعلان‌های LINK که باید پاسخ دهید :

ماجول‌های مقصد	[.OBJ]:
فایل اجرایی	[EXASM1.EXE]:
فایل لیست	[NUL.MAP]:
کتابخانه‌ها	[.LIB]:

- ***ماجول‌های مقصد :*** نام ماجول(های) مقصد را برای لینک کردن می‌پرسد، اگر توسعه را حذف کنید OBJ. پیش فرض خواهد بود.
- ***فایل اجرایی :*** نام فایلی که اجرا می‌شود و می‌پرسد و امکان پیش فرض برای نام فایل ماجول مقصد را دارد. باید دقیقاً مسیر را وارد کنید.
- ***فایل لیست :*** برای فایل نقشه فراهم شده است، اگر چه پیش فرض MAP. NUL یعنی نقشه‌ای تولید نشود) است. پاسخ CON، یک انتخاب مرسوم است که به لینکر می‌گوید تا نقشه را بر روی صفحه نمایش دهد.
- ***کتابخانه‌ها :*** برای انتخاب کتابخانه است که خارج از حد این متن می‌باشد.

برای سه اعلان آخر، جهت پذیرش پیش فرض <Enter> را بفشارید. مثال زیر به لینکر می‌گوید که فایل‌های EXE. و CON. را ایجاد کنید.

ماجول‌های مقصد	[.OBJ]: D:ASM ROG <Enter>
فایل اجرایی	[ASMPROG.EXE]: D: <Enter>
فایل لیست	[NUL.MAP] :CON <Enter>
کتابخانه‌ها	[.LIB]: <Enter>

انتخاب‌های اشکال زدا

اگر مقصد دارید از Code View یا TurboDebugger استفاده کنید، انتخاب ZI/ از اسمبلر را برای اسمبل کردن بکار ببرید. برای لینک کردن، انتخاب LINK /CO مایکروسافت در حالت اعلان یا خط فرمان یا انتخاب TLINK /V توربو را بکار ببرید :

LINK /CO filename . . . یا TLINK /V filename . . .

تبدیل فایل‌های مقصد توربو به برنامه‌های COM.

TLINK بورلند امکان تبدیل مستقیم یک برنامهٔ مقصد به قالب COM. را دارد، که برای برنامه‌های منبع که در اصل بر طبق نیازمندیهای COM. کد شده است، فراهم گردیده است. از انتخاب T/ استفاده کنید :

TLINK /T objfile,comfile,CON

انتخاب‌های EXE2BIN

برنامهٔ EXE2BIN مایکروسافت ماجولهای EXE. تولید شده توسط MASM را به ماجول COM. تبدیل می‌کند، برای برنامه‌هایی که در اصل بر طبق نیازمندیهای COM. کد شده است، مهیا است. فرمان زیر را وارد کنید :

EXE2BIN D:filename, D:filemame.COM

اولین عملوند نام فایل EXE. است که می‌توانید بدون یک توسعه وارد نمایید. عملوند دوم نام فایل COM. است، که شما امکان تغییر نام را دارید، اما از کدنویسی توسعهٔ COM اطمینان حاصل کنید، فایلهای OBJ و EXE را حذف کنید.

ضمیمهٔ ث

برنامهٔ DEBUG

برنامهٔ DOS DEBUG برای نوشتن اغلب برنامه‌های کوچک به اشکال زدایی برنامه‌های اسمبلی، و برای بررسی محتویات یک فایل یا حافظه بکار می‌رود. امکان واردکردن یکی از دو فرمان، زیر برای آغاز DEBUG وجود دارد :

۱) برای ایجاد یک فایل یا بررسی حافظهٔ DEBUG را بدون مشخصهٔ فایل وارد کنید.

۲) برای تغییر یا اشکال زدایی یک برنامه (COM. یا EXE.) یا تغییر یک فایل، DEBUG را با مشخصه فایل وارد کنید. مثل : DEBUG D:PROGC.COM

بارگذار برنامه DEBUG را در حافظه قرار می‌دهد و DEBUG یک علامت (-) را به عنوان اعلان نمایش می‌دهد. ناحیه حافظه‌ای که برای برنامهٔ شماست با عنوان سگمنت برنامه شناخته می‌شود. ثبات‌های CS، DS، ES و SS با آدرس PSP (100H) ارزشدهی می‌شوند و ناحیهٔ کاری شما از PSP + 100H آغاز می‌شود.

ارجاع به یک آدرس حافظه با عبارت سگمنت و افست امکان‌پذیر است مانند DS:120 ، یا فقط یک افست مانند 120. نیز امکان ارجاع مستقیم به آدرس حافظه وجود دارد مانند 40:17 که 40[0]H سگمنت و 17H افست است. DEBUG همه اعداد ورودی را شانزدهی فرض می‌کند، بنابراین نیازی نیست که در انتهای اعداد H را وارد کنید. کلیدهای F1 و F3 در DEBUG مانند DOS کار می‌کنند که F1 کلید به کلید دستور قبلی را تکرار می‌کند، در حالیکه F3 همهٔ فرمان ورودی قبلی را تکرار می‌کند. همچنین DEBUG بین حالت حروف بزرگ و کوچک تفاوتی قائل نیست و زیر توضیف دستورات DEBUG به ترتیب حروف الفبایی آورده شده است.

A (Assemble) جملات منبع اسمبلی را به کد ماشین تبدیل می‌کند. این عمل بطور خاص برای نوشتن و بررسی برنامه‌های اسمبلی کوتاه و بررسی سگمنت‌های کوچک کد مفید است. آدرس شروع پیش فرض برای کد CS:0100H است و قالب کلی برای فرمان A چنین است : A [آدرس]

مثال زیر برنامهٔ اسمبلی شامل ۵ دستور را ایجاد می‌کند. دستورات را (بدون توضیحات) کد نمایید. DEBUG سگمنت کد را ایجاد می‌کند (در اینجا به صورت : xxxx نشان داده شده است) و یک افست که از 0100H آغاز می‌شود.

```
                        A (0r A 100) <Enter>          ; Explanation
xxxx : 0100             MOV CX,[10D] <Enter>          ; گرفتن محتویات از 10D
xxxx : 0104             ADD CX,1A <Enter>             ; جمع با مقدار بلافصل
xxxx : 0107             MOV [10D],CX <Enter>          ; ذخیرهٔ CX در 10D
xxxx : 010B             MOV 100 <Enter>               ; پرش به ابتدا
xxxx : 010D             DW 2500 <Enter>               ; تعریف ثابت
                        <Enter>                       ; انتهای فرمان
```

به دلیل اندازهٔ PSP ، DEBUG مقدار IP را با 100H تنظیم می‌کند، بنابراین جملات از 100H آغاز می‌شوند. آخرین <Enter> (دوتا در یک سطر) به DEBUG می‌گوید که برنامه خاتمه یافته است. حال می‌توانید از فرمان U(Unassemble) برای بررسی کد ماشین و T(Trace) برای پیگیری اجرای برنامه استفاده کنید. توجه کنید که می‌توانید DB و DW را برای تعریف عناصر داده‌ای که برنامه به آن مراجعه می‌کند، بکار ببرید.

امکان تغییر هر یک از دستورات قبلی یا عناصر داده وجود دارد، در حالتی که طول دستور جدید با دستور قبلی یکسان باشد. برای مثال، برای تغییر ADD در 104H به SUB تایپ کنید :

```
A 104 <Enter>
xxxx : 0104  SUB CX,1A <Enter> <Enter>
```

وقتی مجدداً برنامه را اجرا کنید، IP افزوده خواهد شد. از فرمان (R) ثبات برای تنظیم مجدد آن به 100H استفاده کنید و برای خروج Q را وارد کنید.

C(compare). دو ناحیه از حافظه را با هم مقایسه می‌کند. ثبات پیش فرض DS است و قالب کلی آن چنین است.

C [آدرس] [دامنه]

امکان کد کردن این زمان را به یکی از دو روش زیر دارید :

۱) یک آدرس شروع (مقایسه از) ، یک طول و یک آدرس شروع (مقایسه با). مثال زیر 20H بایت در DS:050 را با بایت‌های DS:200 مقایسه می‌کند :

```
C 050  L 20 200  ; مقایسه‌ای به طول 20H
```

۲) یک آدرس شروع و یک آدرس خاتمه (مقایسه از) و یک آدرس شروع (مقایسه با). این مثال بایت‌هایی با شروع از DS:050 را با بایت‌های DS:200 مقایسه می‌کند :

```
C 050 070 200  ; مقایسه با استفاده از دامنه
```

این عمل آدرسها و محتویات بایت‌های نامساوی را نمایش می‌دهد.

D(Display یا Dump) محتویات بخشی‌از برنامهٔ حافظه را در مبنای شانزده و ASCII نمایش می‌دهد. ثبات پیش‌فرض DS است و قالب کلی آن چنین است : D [آدرس] یا D [محدوده]

می‌توانید یک آدرس شروع یا یک آدرس شروع با یک محدوده را مشخص کنید. حذف یک محدوده یا طول سبب می‌شود تا پیش فرض 80H انتخاب شود. مثال‌های فرمان D چنین هستند :

```
D 200          ; نمایش 80H بایت با شروع از DS:200H
D              ; نمایش 80H بایت با شروع از انتهای آخرین نمایش
D CS:150       ; نمایش 80H بایت با شروع از CS:150H
D DS:20 L5     ; نمایش ۵ بایت با شروع از DS:20H
D 30032C       ; نمایش بایت‌های 300H تا 32CH
```

E (Enter) وارد کردن داده یا دستورات ماشین را فعال می‌سازد. ثبات پیش فرض DS است و قالب کلی آن چنین است :

E address [لیست]

عملیات دو حالت دارد :

۱) جایگزینی بایت‌ها با آنهایی که در لیست هستند، همانطوریکه بعداً نشان داده خواهد شد :

```
E 105 3A 21        ; تایپ سه بایت با شروع از DS:105H
E CS:211 21 2A     ; تایپ سه بایت با شروع از CS:211H
E 110 'anything'   ; تایپ یک رشته کاراکتری در DS:110H
```

امکان استفاده از کامای تک یا کامای دوتایی برای رشته‌های کاراکتری وجود دارد.

۲) کردن ویرایش ترتیبی بایت‌ها، آدرس را همانطوریکه می‌خواهید نمایش داده شود وارد کنید :

محتویات DS:12CH را نمایش می‌دهد ; E 12C

عملیات منتظر یک ورودی از صفحه کلید خواهد شد. یا چندین بایت شانزدهی که با یک فاصله مجزا شده‌اند را وارد کنید که از DS:12CH آغاز می‌شود.

F (Fill) یک محدوده موقعیت‌های حافظه را با مقادیر یک لیست پر می‌کند. ثبات پیش فرض DS است قالب کلی آن چنین است : F rang list

مثالهای بعد موقعیت حافظه DS:210H با بایت‌های حاوی 'Help' پر می‌کند :

F 210 L19 'Help ; استفاده از طول 19H (25)

F 210 229 'Help' ; استفاده از محدوده 210H تا 229H

G (Go) یک برنامه زبان ماشین که اشکال زدایی نموده‌اید تا نقطهٔ قطع مشخص شده[1] اجرا می‌کند. ابتدا لیست کد ماشین را برای آدرس‌های IP معتبر بررسی کنید، چون یک آدرس نامعتبر سبب نتایج غیر قابل پیش‌بینی خواهد شد. همچنین، نقطهٔ قطع را فقط در برنامه خودتان تنظیم کنید، نه در ماجولهای DOS یا BIOS. عملیات تا عمل‌های INT اجرا شده و متوقف می‌شود، در صورت لزوم برای ورودی صفحه کلید منتظر می‌ماند. ثبات پیش‌فرض CS است. قالب کلی آن چنین است : G[=address] address [addres ...]

ورودی addres = یک آدرس شروع انتخابی را مهیا می‌سازد. دیگر ورودی‌ها تا 10 آدرس نقطهٔ قطع را مشخص می‌سازد. مثال زیر به DEBUG می‌گوید که همه دستورات از موقعیت جاری IP تا موقعیت 11AH اجرا کند. G 11A

H (Hexadecimal) مجموع و اختلاف دو مقدار شانزدهی که به صورت، "مقدار، مقدار H کد" شده‌اند را نمایش می‌دهد. حداکثر طول چهار رقم مبنای شانزده است. برای مثال فرمان H 14F 22 حاصل ۱۷۱ (مجموع) و 12D (اختلاف) را نشان می‌دهد.

I (Input) یک بایت را از حافظه وارد کرده و نمایش می‌دهد، به صورت "آدرس پورت I کد" می‌شود.

L (Load) یک فایل یا سکتورهای دیسک را در حافظه قرار می‌دهد. توجه کنید که یک فایل ممکن است 'named' باشد بنابراین آن را به یکی از دو روش زیر می‌شناسد : هم با درخواست اجرای DEBUG با یک مشخصه فایل، یا از داخل DEBUG با صدور فرمان N(Name). دو قالب کلی برای فرمان L وجود دارد :

۱) بارگذاری یک فایل نامبرده : L [آدرس]

از پارامتر *آدرس* برای بارگذاری در یک موقعیت خاص استفاده کنید. حذف آدرس سبب می‌شود تا L از آدرس CS:100 بارگذاری نماید. برای بارگذاری فایلی که نامگذاری نشده، ابتدا باید نامگذاری شود (N را ببینید).

N filespec ; نام فایل

L ; بارگذاری فایل در CS:100H

برای بارگذاری مجدد فایل، L را بدون آدرس صادر کنید، DEBUG فایل را مجدداً بارگذاری نموده و ثبات‌ها را بر طبق آن ارزشدهی می‌کند.

۲) بارگذاری داده از سکتورهای دیسک : L [address [drive start nvmber]]

- *Address* برای موقعیت شروع حافظه برای بارگذاری داده فراهم شده است (پیش فرض CS:100 است)
- *Drive* دیسک درایو را مشخص می‌سازد که A = 0 و B = 1 و غیره.
- *Start* شماره شانزدهی اولین سکتوری که باید بارگذاری شود را مشخص می‌سازد. (این یک شماره رابطه‌ای است که سیلندر 0 ، تراک 0) سکتور را یک سکتور رابطه‌ای 0 است).
- *Number* تعداد شانزدهی سکتورهای متوالی که باید بارگذاری شوند، را مشخص می‌سازد. مثال زیر با شروع از

1) breukpaint

CS:100 از درایو 0(A) ، سکتور 20H تا 15H سکتور را بارگذاری می‌کند :

L 100 0 20 15

عملیات L تعداد بایت‌های بارگذاری شده را در BX:CX باز می‌گرداند. برای یک فایل .EXE ، DEBUGاز پارامتر آدرس (در صورت وجود) صرفنظر می‌کند و از آدرس بارگذاری در عنوان EXE. استفاده می‌کند. همچنین عنوان آن را جدا کرده، تا نگهداری نماید، فایل را قبل از اجرای DEBUG با یک توسعهٔ متفاوت مجدداً نامگذاری می‌کند.

M(Move) محتویات موقعیت‌های حافظه رامنتقل (یا کپی) می‌کند. ثبات پیش‌فرض DS است و قالب‌کلی آن چنین‌است

M range address

این مثال بایت‌های DS:050H تا 150H را به آدرس DS:400H کپی می‌کند.

استفاده از طول برای انتقال ;	M	DS:50	L100	DS:400
استفاده از محدوده برای انتقال ;	M	DS:50	150	DS:400

N(Name) یک برنامه یا فایلی را که قصد دارید از دیسک بخوانید یا روی دیسک بنویسید، نام گذاری می‌کند فرمان را بدین صورت کد نمایید. مشخصه فایل N، مانند N D:SAM.COM

عملیات نام را در CS:80 در PSP ذخیره می‌سازد. اولین بایت در CS:80 حاوی طول (0AH) است، که پس از آن یک فاصله و مشخصه فایل قرار دارد. حال امکان استفاده از L برای بارگذاری و W برای نوشتن فایل وجود دارد.

O(Output) یک بایت را به درگاه ارسال می‌کند، به صورت زیر کد می‌شود : بایت ارسالی آدرس درگاه O

P(Proceed)یک سابروتین فراخوانی شده (CALL)، حلقه (LOOP)، وقفه (INT) یا دستور تکرار رشته (REP) را تا رسیدن به دستور بعدی اجرا می‌کند. قالب کلی آن چنین است P [= address] [value]

که address = یک آدرس رشتهٔ انتخابی valve تعداد دستورات انتخابی است که باید اجرا شود. حذف address = سبب انتخاب پیش فرض جفت ثبات CS:IP می‌شود. برای مثال اگر در پیگیری اجرا به یک عملیات INT 21H رسیدید، P را وارد کنید تا عملیات کاملاً اجرا شود. G و T را نیز ببینید.

Q(Quit) از DEBUG خارج می‌شود. این عمل فایل‌ها را ذخیره نمی‌کند، بدین منظور از W استفاده کنید.

R(Register) محتویات ثبات‌ها و دستور بعدی را نشان می‌دهد قالب کلی آن چنین است : R [نام ثبات]

مثال‌های زیر استفاده از این فرمان را مشخص می‌کند :

R همهٔ ثبات‌ها را نمایش می‌دهد.

R DX DX را نمایش می‌دهد، DEBUG به شما امکان یکی از این انتخاب‌ها را می‌دهد :

۱- <Enter> را بفشارید، که DX بدون تغییر باقی می‌ماند،

۲- یک تا چهار رقم شانزدهی را وارد کنید تا محتویات DX تغییر کند.

R IP IP را نمایش می‌دهد. مقدار دیگری وارد کنید تا محتویات آن تغییر کند.

RF تنظیم هر پرچم را به صورت یک کد دو حرفی نمایش می‌دهد. با هر ترتیبی می‌توانید هر تعداد از پرچم‌ها را تغییر دهید :

پرچم	یک کردن	صفر کردن
سرریز	ov	nv
جهت	dn	up
علامت	ng(-)	pl(+)
صفر	zr	nz
نقلی	cy	nc

S(Search) حافظه را برای یافتن کاراکترهای یک لیست جستجو می‌کند. ثبات پیش فرض DS است، و قالب کلی آن چنین است : لیست دامنه S

اگر کارکترها پیدا شوند، عملیات آدرس آنها را ارائه می‌دهد، در غیر اینصورت پاسخ نمی‌دهد. مثال زیر کلمه "VIRUS" را در آدرس DS:300 تا 2000H بایت جستجو می‌کند : S 300 L2000 "VIRUS"

این مثال از CS:100 تا CS:400 برای یافتن یک بایت حاوی 51H جستجو می‌کند :

S CS:100 400 51

T(Trace) یک برنامه را در حالتی تک مرحله‌ای اجرا می‌کند. توجه کنید که معمولاً از P برای اجرای دستورات INT استفاده می‌شود. ثبات پیش فرض CS:IP است و قالب کلی آن چنین است : T [=آدرس] [مقدار]

ورودی اختیاری آدرس = به DEBUG می‌گوید که پیگیری را آغاز کند و مقدار انتخابی تعداد دستوراتی که باید پیگیری شود مشخص می‌سازد. حذف عملوندها سبب می‌شود تا DEBUG دستور بعدی را اجرا کند و ثبات‌ها را نمایش می‌دهد. در اینجا دو مثال آورده شده است :

```
T                ; اجرای دستور بعدی
T 10             ; اجرای 10H (16) دستور بعدی
```

U(Unassemble) دستورات ماشین را Unassemble می‌کند. ثبات پیش فرض جفت CS:IP است و قالب کلی آن چنین خواهد بود U یا U [آدرس] [محدوده]

ناحیه مشخص شده حاوی کد ماشین معتبر است که عملیات دستورات سمبولیک معادل آن را نمایش می‌دهد. در اینجا سه مثال آورده شده است :

```
U 100            ; ۳۲ بایت از آدرس CS:100 را unassembel می‌کند
U                ; ۳۲ بایت از آخرین U در صورت وجود را unassembel می‌کند
U 100 140        ; از 100H تا 140H را در صورت وجود unassembel می‌کند
```

توجه کنید که DEBUG برخی پرسش‌های شرطی و دستورات خاص پردازشگرهای ۸۰۳۸۶ و ما بعد را به نحو صحیحی تبدیل نمی‌کند، اگر چه آنها را صحیح اجرا می‌کند.

- W (Write) یک فایل را از DEBUG می‌نویسد، فایل ابتدا باید نامگذاری شود اگر تا هم اکنون بارگذاری نشده است (N را ببینید). ثبات پیش فرض در CS است و قالب کلی آن چنین است.

W [آدرس] [شروع درایو] [تعداد سکتورها سکتور

این فرمان فقط فایل برنامه با توسعه COM. را می‌نویسد، چون W قالب EXE. را پشتیبانی نمی‌کند. (برای تغییر یک برنامه EXE. توسعه آن را موقتاً تغییر دهید). مثال‌های زیر از W بدون عملوند استفاده کرده و اندازهٔ فایل را در BX : CX تنظیم می‌کند.

```
R BX             ; درخواست ثبات BX
Q                ; تنظیم BX با صفر
N filespec       ; نام فایل
R CX             ; درخواست ثبات CX
Length           ; جایگذاری اندازهٔ فایل به صورت مقدار شانزدهی در CX
W                ; نوشتن فایل
```

اگر شما یک فایل را تغییر دهید ولی طول یا نام آن تغییر نکند، DEBUG می‌تواند فایل را در موقعیت اولیه روی دیسک بنویسد. همچنین می‌توانید یک فایل را مستقیماً در سکتورهای خاص دیسک بنویسید، اگر چه این تجربه نیازمند دقت بالایی است.

فرامین DEBUG که در اینجا ذکر شد :

- XA : تخصیص حافظهٔ توسعه یافته.
- XD : آزاد سازی حافظه توسعه یافته.
- XM : صفحات منطقی را بر روی صفحات فیزیکی نقشه کردن.
- XS : نمایش وضعیت حافظهٔ توسعه یافته.

کد پیمایشی صفحه کلید و کدهای ASCII

در لیست زیر، کلیدها به طور دلخواه طبقه بندی شده‌اند. برای هر طبقه ستونها نشان دهندهٔ قالب یک کلید معمولی (بدون ترکیب با دیگر کلیدها) و قالبی که کلید با دیگر کلیدها مانند Shift، Ctrl و کلیدهای Alt ترکیب شود را نشان می‌دهد. تحت ستونهایی با عنوان معمولی 'Shif' ، 'Ctrl' ، 'Alt' دو بایت هستند که وقتی عملیات صفحه کلید انجام می‌شود در ثبات‌های AH و AL ارائه داده می‌شود. برای مثال، فشردن حرف 'a' 1EH را در AH و 61H را در AL ارائه می‌دهد. وقتی در حالت بزرگ باشد 'A' صفحه کلید در 1EH را در AH و 41H را در AL ارائه می‌دهد. کدهای پیمایش 85H و بیشتر برای صفحه کلیدهای توسعه یافته است.

LETTERS	NORMAL		SHIFT		CTRL		ALT	
a and A	1E	61	1E	41	1E	01	1E	00
b and B	30	62	30	42	30	02	30	00
c and C	2E	63	2E	43	2E	03	2E	00
d and D	20	64	20	44	20	04	20	00
e and E	12	65	12	45	12	05	12	00
f and F	21	66	21	46	21	06	21	00
g and G	22	67	22	47	22	07	22	00
h and H	23	68	23	48	23	08	23	00
i and I	17	69	17	49	17	09	17	00
j and J	24	6A	24	4A	24	0A	24	00
k and K	25	6B	25	4B	25	0B	25	00
l and L	26	6C	26	4C	26	0C	26	00
m and M	32	6D	32	4D	32	0D	32	00
n and N	31	6E	31	4E	31	0E	31	00
o and O	18	6F	18	4F	18	0F	18	00
p and P	19	70	19	50	19	10	19	00
q and Q	10	71	10	51	10	11	10	00
r and R	13	72	13	52	13	12	13	00
s and S	1F	73	1F	53	1F	13	1F	00

t and T	14	74	14	54	14	14	14	00
u and U	16	75	16	55	16	15	16	00
v and V	2F	76	2F	56	2F	16	2F	00
w and W	11	77	11	57	11	17	11	00
x and X	2D	78	2D	58	2D	18	2D	00
y and Y	15	79	15	59	15	19	15	00
z and Z	2C	7A	2C	5C	2C	1A	2C	00
Spacebar	39	20	39	20	39	20	39	20

FUNCTION KEYS	NORMAL		SHIFT		CTRL		ALT	
F1	3B	00	54	00	5E	00	68	00
F2	3C	00	55	00	5F	00	69	00
F3	3D	00	56	00	60	00	6A	00
F4	3E	00	57	00	61	00	6B	00
F5	3F	00	58	00	62	00	6C	00
F6	40	00	59	00	63	00	6D	00
F7	41	00	5A	00	64	00	6E	00
F8	42	00	5B	00	65	00	6F	00
F9	43	00	5C	00	66	00	70	00
F10	44	00	5D	00	67	00	71	00
F11	85	00	87	00	89	00	8B	00
F12	86	00	88	00	8A	00	8C	00

NUMERIC KEYPAD	NORMAL		SHIFT		CTRL		ALT	
Ins and 0	52	00	52	30	92	00		
End and 1	4F	00	4F	31	75	00	00	01
DnArrow and 2	50	00	50	32	91	00	00	02
PgDn and 3	51	00	51	33	76	00	00	03
LtArrow and 4	4B	00	4B	34	73	00	00	04
5 (keypad)	4C	00	4C	35	8F	00	00	05
RtArrow and 6	4D	00	4D	36	74	00	00	06
Home and 7	47	00	47	37	77	00	00	07
UpArrow and 8	48	00	48	38	8D	00	00	08
PgUp and 9	49	00	49	39	84	00	00	09
+ (gray key)	4E	2B	4E	2B	90	00	4E	00

− (gray key)	4A	2D	4A	2D	8E	00	4A	00
Del and .	53	00	53	2E	93	00		
* (gray key)	37	2A	37	2A	96	00	37	00

TOP ROW	NORMAL		SHIFT		CTRL		ALT	
and ~	29	60	29	7E			29	00
1 and !	02	31	02	21			78	00
2 and @	03	32	03	40	03	00	79	00
3 and #,	04	33	04	23			7A	00
4 and $	05	34	05	24			7B	00
5 and %	06	35	06	25			7C	00
6 and ^	07	36	07	5E	07	1E	7D	00
7 and &	08	37	08	26			7E	00
8 and *	09	38	09	2A			7F	00
9 and (	0A	39	0A	38			80	00
0 and)	0B	30	0B	29			81	00
− and _	0C	2D	0C	5F	0C	1F	82	00
= and +	0D	3D	0D	2B			83	00

OPERATION KEYS	NORMAL		SHIFT		CTRL		ALT	
Esc	01	1B	01	1B	01	1B	01	00
Backspace	0E	08	0E	08	0E	7F	0E	00
Tab	0F	09	0F	00	94	00	A5	00
Enter	1C	0D	1C	0D	1C	0A	1C	00

PUNCTUATION	NORMAL		SHIFT		CTRL		ALT	
[and {	1A	5B	1A	7B	1A	1B	1A	00
] and }	1B	5D	1B	7D	1B	1D	1B	00
; and :	27	3B	27	3A			27	00
‘ and “	28	27	28	22			28	00
\ and |	2B	5C	2B	7C	2B	1C	2B	00
, and <	33	2C	33	3C			33	00
. and >	34	2E	34	3E			34	00
/ and ?	35	2F	35	3F			35	00

در زیر کلیدهای دوتایی که فقط در صفحه کلیدهای پیشرفته هستند آورده شده است (اولین ستون کاراکترهای ASCII و دومی کلیدهای مکان نما است).

KEY	NORMAL		SHIFT		CTRL		ALT	
Slash (/)	E0	2F	E0	2F	95	00	A4	00
Enter	E0	0D	E0	0D	E0	0A	A6	00
Home	47	E0	47	E0	77	E0	97	00
End	4F	E0	4F	E0	75	E0	9F	00
PageUp	49	E0	49	E0	84	E0	99	00
PageDown	51	E0	51	E0	76	E0	A1	00
DownArrow	50	E0	50	E0	91	E0	A0	00
LeftArrow	4B	E0	4B	E0	73	E0	9B	00
RightArrow	4D	E0	4D	E0	74	E0	9D	00
UpArrow	48	E0	48	E0	8D	E0	98	00
Ins	52	E0	52	E0	92	E0	A2	00
Del	53	E0	53	E0	93	E0	A3	00

کلیدهای کنترلی، کدهای پیمایش معینی دارند، اگر چه BIOS آنها را به بافر صفحه کلید ارائه نمی‌دهد، در اینجا کدهای پیمایش آنها آورده شده است :

CapsLock	3A	Shift (Right)	36
NumLock	45	Alt	38
ScrollLock	46	Ctrl	1D
Shift (Left)	2A	PrtScreen	37

پاسخ به پرسشهای انتخابی

فصل ۱

۱-۱. (الف) بیت.

۱-۲. (الف) بایت.

۱-۳. (الف) دو.

۱-۴. (الف) 0101، (ج) 10 111.

۱-۵. (الف) 00100010، (ج) 00100000.

۱-۶. (الف) 11011010، (ج) 10001000.

۱-۷. (الف) 00111100، (ج) 00000100.

۱-۸. (الف) ۵۷، (ج) ۵۷.

۱-۹. (الف) 24D8، (ج) 8000.

۱-۱۰. (الف) ۱۲، (ج) ۵۷، (ه) FFF.

۱-۱۱. (الف) 01010001، (ج) 11 11 11 00

۱-۱۳. ROM (حافظه فقط خواندنی) پایدار است، روال‌های راه‌اندازی را انجام می‌دهد و ورودی خروجی را دستکاری می‌کند. RAM (حافظه با دستیابی تصادفی) موقتی است و ناحیه‌ای است که داده‌ها و برنامه‌ها در هنگام اجرا در آن ساکن می‌شوند.

۱-۱۵. (الف) بخشی از یک برنامه حداکثر تا 64K، حاوی کد، داده یا پشته.

۱-۱۶. (الف) پشته، داده و کد.

۱-۱۸. (الف) CS، DS، SS، (ج) AX، BX، CX، DX، DI، SI، (ه) CX

۱-۲۰. (الف) MOV BX,36

فصل ۲

۲-۴. (الف) پیشوند سگمنت برنامه (PSP).

۲-۵. (الف) DS و ES = آدرس PSP.

۲-۷. (الف) سیستم پشتهٔ یک برنامه COM. را تعریف می‌کند.

۲-۸. (الف) در بالاترین موقعیت، اندازه پشته (مانند 64) که سیستم SP را با آن ارزشدهی می‌کند.

۲-۹. (الف) 63E2H.

۲-۱۰. (الف) 74AA4H.

۲-۱۱. (الف) 4D 766H.

فصل ۳

۳-۱. فرامینی که در ابتدای فصل معین شده‌اند.

۳-۲. (الف) D DS:1AS (ج) E DS:18A 444E41.

۳-۳. (الف) B84B32.

۳-۴. E CS:101 54.

۳-۵. (الف)

```
MOV AX,2000
ADD AX,3000
NOP
```

(ج) از R IP برای تنظیم IP با 100 استفاده کنید.

۳-۶. حاصل 0568H است.

۳-۸. از فرمان N برای نامگذاری برنامه استفاده کنید، طول را در BX:CX تنظیم کنید و از فرمان W برای نوشتن برنامه استفاده کنید.

۳-۱۱. از MOV AH,09 برای نمایش استفاده کنید.

فصل ۴

۴-۳. نام (یک عنصر داده) و برچسب (یک دستور).

۴-۴. (الف) فقط در صورتیکه به ثبات CX اشاره کند معتبر است، (ج) معتبر.

۴-۶. (الف) PAGE.

۴-۸. (الف) سبب می‌شود یک سگمنت بر روی یک مرز مانند یک پاراگراف قرار گیرد.

۴-۹. (الف) بخشی از یک کد مرتبط مانند یک سابروتین را فراهم می‌سازد.

۴-۱۰. (الف) ENDP.

۴-۱۱. پیش پردازنده END به اسمبلر می‌گوید که دستور دیگری برای اسمبل کردن وجود ندارد، دستوراتی که سبب می‌شود تا کنترل به سیستم عامل باز گردد MOV AX,4C00H و INT 21H است.

۴-۱۲. ASSUME SS:STKSEG,DS:DATSEG,CS:CDSEG.

۴-۱۵. (الف) ۱ (ج) ۱۰.

۴-۱۶. (الف) CONAME DB 'Computer Services'.

۴-۱۷. (الف) AREA1 DB 00100011B (ج) AREA3 DW.

۴-۱۸. (الف) Hex 22.

۴-۱۹. (الف) B2254A00 (ج) ۳۵.

فصل ۵

۵-۱. MASM/TASM E:SQUEEZE E: , E: , E:.

۵-۳. (الف) E:SQUEEZE.

۵-۴. (الف) فایل اجرایی (ماجول)، (ج) نقشهٔ پیوند، (ه) فایل ارجاع متقابل.

۵-۵.
```
MOV AX,DATSEGM
MOV DS,AX
```

۵-۶. بخشی از کد نویسی :

```
MOV AL,50H  ; بارگذاری مقدار بلافصل
SHL AL,1    ; شیفت به چپ
MOV CL,18H  ; ضرب AL
MUL CL      ; در CL
```

۵-۸. سگمنت داده باید حاوی این عناصر داده باشد :

```
FIELDX  DB  50H
FIELDY  DB  18H
FIELDZ  DW  ?
```

۵-۱۰. به تأثیر پیش پردازندهٔ EVEN بر روی شمارشگر موقعیت توجه کنید.

فصل ۶

۶-۲. (الف) اولین ADD مقدار بلافصل 2584H را با CX جمع می‌کند، دومین ADD محتویات موقعیت 2548H و 2549H را با CX جمع می‌کند.

۶-۴. محتویات DX را با موقعیت حافظه‌ای، که توسط مجموع آدرس افست در BX بعلاوه SI بعلاوه 8 اشاره می‌شود، جمع می‌کند ([DS: [BX + SI + 8])

۶-۵. (الف) پردازشگر نمی‌تواند داده را مستقیماً بین موقعیت‌های حافظه پردازش کند.

۶-۶. (الف) یک عملیات حافظه به حافظه نامعتبر است و بجای آن از دو دستور استفاده کنید :

```
MOV AL,BYTEY
ADD BYTEX,AL
```

۶-۷. (الف) ADD CX,48H (ج) SHR DH,1 (ه) MOV CX,248

۶-۸. از XCHNG استفاده کنید.

۶-۹. از LEA (یا MOV با OFFSET) استفاده کنید.

۶-۱۱. (الف) پرچم‌ها IP و CS را بر روی پشته قرار دهید، پرچم‌های IF و TF را جایگزین کنید و آدرس وقفه را در CS: IP ذخیره نمایید.

فصل ۷

۷-۱. 64K.

۷-۴. از ناحیه بالایی برنامه COM. استفاده می‌کند، اگر حافظه کافی نباشد، از انتهای حافظه استفاده می‌کند

۷-۵. (الف)
```
EXE2BIN PRESSURE,PRESSURE.COM
```

فصل ۸

۸-۱. (الف) داخل ۱۲۸- و ۱۲۷+ بایت.

۸-۲. (الف) داخل ۱۲۸- و ۱۲۷+ بایت. (ب) عملوند یک مقدار یک بایتی است که 00H تا 7FH (0 تا ۱۲۸+) و 80H تا FFH (۱۲۸- تا ۱-) می‌تواند باشد.

۸-۳. (الف) 05DCH (ج) به علامت FA34H توجه کنید.

۸-۴. در اینجا با یکی از چندین حل ممکن آورده شده است.

```
        MOV   AX,00
        MOV   BX,01
        MOV   CX,12
        MOV   DX,00
B20:    ADD   AX,BX     ; عدد در AX است
        MOV   BX,DX
        MOV   DX,AX
        LOOP  B20
```

۸-۵. (الف)
```
CMP AX,BX
JBE address
```
(ج)
```
CMP CX,CX
JG address
```
(هـ)
```
CMP DX,0
JE یا JZ
```

۸-۶. (الف) (۱) TF (ج) (۱) SF

۸-۸. اولین PROC (اصلی) باید FAR باشد چون سیستم عامل آن را به آدرس خودش برای اجرا پیوند می‌زند. صفت NEAR یعنی آدرس داخل همین سگمنت خاص است.

۸-۱۰. سه (یکی برای هر CALL).

۸-۱۱. (الف) 11111011 ، (ج) 10011010.

۸-۱۲. حروف کوچک a تا Z ، 61H تا 7AH هستند.

۸-۱۳. (الف) B972H (ج) 1737H (هـ) 72B9H.

فصل ۹

۹-۱. (الف) سطر = 18H و ستون = 4FH.

۹-۳.
```
        MOV   AX,0613H      ; درخواست
        MOV   BH,attribute  ; پاک کردن
        MOV   CX,0600H      ; پنجره
        MOV   DX,184FH
        INT   10H           ; فراخوانی سرویس وقفه
```

۹-۴.
```
MSSGE   DB    'what is the date (mm/dd/yy)? , '07H , '$'
        MOV   AH,09H        ; درخواست نمایش
        LEA   DX,MSSGE      ; تاریخ
        INT   21H           ; فراخوانی سرویس وقفه
```

۹-۵.
```
DATEPAR LABEL   BYTE
MAXLEN  DB      9           ; فاصله برای / و Enter
        ACTLEN  DB
DATEFLD DB      9 DUP (' ')
        ...
        MOV     AH,0AH      ; درخواست ورودی
        LEA     DX,DATEPAR  ; تاریخ
        INT     21H         ; فراخوانی سرویس وقفه
```

۹-۸. (الف) 04.

فصل ۱۰

۱۰-۱. (الف) 01101110.

۱۰-۲. (الف) 00000001.

۱۰-۳. (الف)

```
MOV   AH,00H        ; درخواست تنظیم حالت
MOV   AL,02         ; ۸۰ ستون تک رنگ
INT   10H           ; فراخوانی سرویس وقفه
MOV   AH,060FH      ; (ج) درخواست چرخش طوماری ۱۴ خط
MOV   BH,07         ; ویدئو معمول
MOV   CX,0000       ; صفحهٔ ورودی
MOV   DX,184FH
INT   10H           ; فراخوانی سرویس وقفه
```

۱۰-۴. ۸ رنگ برای پس زمینه و ۱۶ رنگ برای پیش‌زمینه.

۱۰-۵.

```
MOV   AH,09H        ; درخواست نمایش
MOV   AL,051        ; club
MOV   BH,00         ; شماره صفحه 0
MOV   BL,00011110B  ; زرد روی آبی
MOV   CX,06         ; ۶ مرتبه
INT   10H           ; فراخوانی سرویس وقفه
```

۱۰-۸. توجه کنید که INT 16H مکان نما را پیش نمی‌برد یا کاراکترهای وارد شده را روی صفحه نمایش نمی‌دهد. بطور مشابه، INT 10H تابع 09H مکان نما را پیش نمی‌برد، ولی فقط یک کاراکتر را در هر بار نمایش می‌دهد،

۱۰-۱۰. (الف)

```
MOV   AH,00H        ; درخواست حالت
MOV   AL,04         ; دقت 320 x 200
INT   10H           ; فراخوانی سرویس وقفه
```

۱۰-۱۱. ابتدا حالت گرافیک را تنظیم می‌کند، سپس از INT 10H تابع 0BH برای تنظیم رنگ پس زمینه استفاده می‌کند.

۱۰-۱۲. ابتدا حالت گرافیک را تنظیم می‌کند، سپس از INT 10H تابع 0DH برای خواندن نقطه استفاده می‌کند.

فصل ۱۱

۱۱-۱. (الف) موقعیت 40:17H (417H)

۱۱-۲. (الف) ورودی صفحه کلید با انعکاس، در صورتیکه یک تابع صفحه کلید توسعه یافته باشد به دو عملیات INT نیاز دارد.

۱۱-۴. (الف) 47H، (ج) 50H.

۱۱-۶. از INT 16H تابع 10H برای ورودی صفحه کلید استفاده کنید. از CMP برای بررسی کد پیمایش و از INT 10H تابع 02H برای تنظیم مکان نما استفاده کنید.

۱۱-۸. در هر فشردن یا رها کردن کلید.

۱۱-۱۰. (الف) موقعیت 40 :1EH (41EH).

۱۱-۱۲. (الف) CapsLock و NumLock بیت ۶ و ۳ هستند.

فصل ۱۲

۱۲-۱. (الف) DS:SI و ES:DI

۱۲-۳. (الف)

```
        JCXZ   label 2   ; CX صفر است؟
label1: MOV    AX,[SI]   ; گرفتن کاراکتر
        MOV    [DI],AX   ; ذخیرهٔ کاراکتر
        INC    DI        ; افزایش
        INC    DI        ; DI و
        INC    SI        ; SI
        INC    SI        ; دوبار
        LOOP   label1
label2: . . .
```

۱۲-۴. DF را برای حرکت راست به چپ تنظیم کنید. برای MOVSB در HEADG1+9 و HEADG2+9 مقدار دهی کنید. برای MOVSW در HEADG1+8 و HEADG2 + 8 مقدار دهی کنید.

۱۲-۶. (الف)

```
CLD                ; چپ به راست
MOV   CX,18        ; مقدار دهی
LEA   DI,OUTAREA   ; برای انتقال
LEA   SI,DESCRIP   ; ۱۸ بایت
REP   MOSB         ; انتقال رشته
```

(ج)

```
CLD                ; چپ به راست
LEA   SI,DESCRIP   ; شروع از بایت ۵
LODSW              ; قرار دادن ۲ بایت
```

(ه)

```
CLD                ; چپ به راست
MOV   CX,18        ; ۱۸ بایت
LEA   DI,OUTAREA   ; مقدار دهی
LEA   SI,DESCRIP   ; آدرس
REPE  CMPSB        ; مقایسه رشته‌ها
```

۱۲-۷. در اینجا یک راه حل است :

```
H10SCAS PROC   NEAR
        CLD                    ; چپ به راست
        MOV    CX,10           ; ۱۰ بایت
        LEA    DI,HEADG1       ; مقدار دهی آدرس
        MOV    AL,'n'          ; و کاراکتر پیمایش
H20:    REPNE  SCASB           ; پیمایش
        JNE    H30             ; پیدا شد؟
        CMP    PRT [DI] ,'a' BYTE ; بله بایت بعدی
        JNE    H20             ; مساوی 'a'؟
        MOV    AL,03
H30:    RET
        ENDP
        H10SCAS
```

۱۲-۸. PATTERN را بلافاصله قبل از DISPLAY تعریف کنید، CX ، DI و SI را مقدار دهی کنید و از REP MOVSW استفاده کنید سپس INT 21H تابع 09H را برای نمایش عنصر داده DISPLAY استفاده کنید.

فصل ۱۳

۱۳-۱. (الف) ۳۲۷۶۷ و ۶۵۵۳۵.

۱۳-۳. CF = 0 و 0F = 0

۱۳-۵ (الف)

```
MOV  AX,VALUE2
ADD  AX,VALUE1   ; جمع VALUE1
MOV  VALUE2,AX   ; با VALUE2
```

(ب) شکل ۲-۱۳ را برای جمع چند کلمه‌ای ببینید.

۱۳-۶. STC پرچم نقلی را یک می‌کند. مجموع = 0153H + 0328H + 1.

۱۳-۷. (الف)

```
MOV  AX,VALUE1
MUL  VALUE2  ; حاصل در DX:AX است
```

(ب) شکل ۴-۱۳ را برای ضرب دو کلمه‌ای در کلمه ببینید.

۱۳-۸. (الف)

```
MOV   AX,NALUE1
MOV   BL,36      ; تقسیم VALUE1
DIV   BL         ; بر 36
```

فصل ۱۴

۱۴-۱. (الف) ADD مقدار 006DH را تولید می‌کند و AAA مقدار 0103H را تولید می‌کند.
(ج) SUB مقدار 0002H را تولید می‌کند و AAS هیچ تأثیری ندارد.

۱۴-۲. (الف) |33|37|39|36|

۱۴-۳.

```
        LEA   SI,BCDAMT   ; مقدار دهی آدرس
        MOV   CX,04       ; و ۴ حلقه
        0R    [SI],30H    ; جایگزینی ASCII3
B20:    INC   SI          ; افزایش برای بایت بعدی
        LOOP  B20         ; ۴ بار حلقه
```

۱۴-۴. از شکل ۲-۱۴ برای راهنمایی استفاده کنید ولی CX را با 03 مقدار دهی کنید.

۱۴-۵. از شکل ۳-۱۴ برای راهنمایی استفاده کنید ولی CX را با 03 مقدار دهی کنید.

۱۴-۶. (الف) ضمیمه A را برای روال بنگرید، جواب 9BA6H است.

۱۴-۷. توجه کنید که INT 12H در AX اندازه حافظه را با عبارت 1K بایت باز می‌گرداند.

فصل ۱۵

۱۵-۲. (الف)

```
TEMPTBL DW 365 DUP(0)
```

۱۵-۳. (الف)

```
ITEMNO DB '05', '09', '12', '19', '23'
```

(ج)

```
ITEPRICE DW 1250,9375,8745,7935,1595
```

۱۵-۴.

```
ITEMTBL  DB  '05',  'Videotape'
         DW  1250
```

۱۵-۵ INT 16H تابع 10H برای ورودی صفحه کلید پیشنهاد می‌شود، شکل ۲-۱۲ راهنمای مفیدی است.

۱۵-۶. یک امکان سازماندهی در روال‌های زیر است :

روال	هدف
A10MAIN	مقدار دهی ثبات، همهٔ روالها.
B10READ	نمایش اعلان، پذیرش تعداد عنصر.
C10SRCH	جدول جستجو، نمایش پیغام در صورت نامعتبر بودن عنصر.
D10MOVE	اقتباس توضیح و قیمت از جدول.
E10CONV	تبدیل کمیت از ASCII به دودویی.
F10CALC	محاسبه مقدار (قیمت × کمیت)
G10CONV	تبدیل مقدار از دودویی به ASCII.
K10DISP	نمایش توضیح و مقدار بر روی صفحه.

۱۵-۷. روتین زیر، جدول را کپی می‌کند. به شکل ۸-۱۵ برای مرتب کردن ورودیهای جدول مراجعه کنید.

```
SORTABL DB   5 DUP (و DUP (?))
        . . .
        LEA  SI,ITDESC      ; مقدار دهی
        LEA  DI,SORTABL     ; آدرسهای جدول و
        MOV  CX,45          ; تعداد کاراکترها
        CLD                 ; چپ به راست
        REP  MOVSB          ; انتقال رشته
```

۱۵-۸. مقادیر ASCII را در جدول برای تبدیل شدن، تعریف کنید :

```
ASCTABL DB 75 DUP (20H) ;
        . . .
```

۱۵-۹. هدف استفاده از XLAT برای تبدیل است.

فصل ۱۶

۱۶-۱. ۵۱۲.

۱۶-۴. (الف) یک گروه از سکتورهاست (۱، ۲، ۴ یا ۸) که سیستم با آنها مانند یک واحد فضای حافظه روی دیسک برخورد می‌کند.

۱۶-۵. (الف) ۸۰ سیلندر × ۱۸ سکتور × ۲ طرف × ۵۱۲ بایت = ۱۴۷۴۵۶۰.

۱۶-۷. (الف) سیستم کمک می‌کند تا برنامهٔ خودش را در حافظه قرار دهد.

۱۶-۸. در شاخهٔ، اولین بایت نام فایل با E5H تنظیم می‌شود.

۱۶-۹. (الف) 00H.

۱۶-۱۱. (الف) موقعیت‌های ۳۱-۲۸ شاخه، (ب) 0C5DH که به صورت 5D0C.

۱۶-۱۲. (الف) اولین بایت حاوی F8H است.

فصل ۱۷

۱۷-۱. (الف) 06.

۱۷-۳. (الف) FHANDLE DW?

(ب) تعریف را چنین آغاز کنید :

```
PATNTOUT LABEL BYTE
```

که پس از آن یک DB برای هر عنصر داده قرار دارد.

(ج) از INT 21H تابع 3CH برای ایجاد فایل استفاده کنید، از JC برای بررسی خطا استفاده کنید و دستگیره را ذخیره نمایید.

۱۷-۴. (الف)
```
MOV  AH,3DH        ; درخواست باز کردن
MOV  AL,00         ; فقط خواندنی
LEA  DX,ASCPATH    ; رشتهٔ ASCII
INT  21H           ; فراخوانی سرویس وقفه
JC   error         ; خروج در صورت خطا
MOV  FHANDLE,AX    ; ذخیرهٔ دستگیره
```

۱۷-۵. وقتی یک برنامه فایلهای زیادی را باز می‌کند.

۱۷-۷. از شکل ۲-۱۷ به عنوان راهنما برای ایجاد یک فایل دیسک و شکل ۵-۱۴ برای تبدیل ASCII به دودویی استفاده کنید.

۱۷-۸. در شکل ۳-۱۷ به‌عنوان راهنما برای خواندن فایل و شکل ۶-۱۴ برای تبدیل دودویی به ASCII استفاده کنید.

۱۷-۱۰. شکل ۴-۱۷ را برای استفاده تابع 42H ببینید.

۱۷-۱۱. همه توابع INT 21H هستند (الف) 16H (ج) 15H.

۱۷-۱۲. (الف) ۱۲۸ بایت، (ج) ۱۴۴ (۹ سکتور × ۴ تراک × ۴ رکورد / سکتور)

فصل ۱۸

تمام پرسش‌های این فصل تمریناتی است که مستلزم استفاده از DEBUG می‌باشد.

فصل ۱۹

۱۹-۲. اغلب مشابه توسعه دهندگان برنامه‌های کاربردی دیسک است.

۱۹-۳. (الف) در AH.

۱۹-۵. از INT 13H تابع 00H استفاده کنید.

۱۹-۶. از INT 13H تابع 01H استفاده کنید.

۱۹-۸.
```
MOV  AH,03H        ; درخواست نوشتن
MOV  AL,01         ; یک سکتور
LEA  BX,DATAOUT    ; ناحیه خروجی
MOV  CH,07         ; تراک ۷
MOV  CL,03         ; سکتور ۰۳
MOV  DH,00         ; هد #0
MOV  DL,00         ; درایو A
INT  13H           ; فراخوانی سرویس وقفه
```

۱۹-۹. بایت وضعیت در AH حاوی 00000011 است.

فصل ۲۰

۲۰-۱. (الف) 0DH.

۲۰-۳. (الف)

```
        MOV   AH,05H      ; درخواست چاپ ;
        MOV   DL,0CH      ; Form Feed
        INT   21H         ; فراخوانی سرویس وقفه ;
        LEA   SI,NAMEFLD  ; مقدار دهی نام ;
        MOV   CX,Length   ; و طول ;
        MOV   AH,05H      ; درخواست چاپ ;
B20:    MOV   DL,[SI]     ; کاراکتر نام ;
        INT   21H         ; فراخوانی سرویس وقفه ;
        INC   SI          ; کاراکتر بعدی در نام ;
        LOOP  B20         ; حلقه به تعداد طول ;
```

باید یک (0AH) در جلوی آدرس کد نمایید. راه حل آن مشابه بخش (ب) است.

(ه) یک Form Feed (0CH) دیگر صادر کنید.

۲۰-۴. 12 و 'Title' و 15 و 10 و 13 HEADNG DB

۲۰-۶. (الف) خطای ورودی / خروجی.

۲۰-۸. CX برای حلقه در دسترس نیست زیرا حلقه‌ای که نام را چاپ می‌کند از CX استفاده می‌کند. شما باید از BX چنین استفاده کنید.

```
        MOV   BX,05       ; تنظیم ۵ بار حلقه ;
C20:    . . .
        DEC   BX          ; افزایش شمارشگر حلقه ;
        JNZ   C20         ; تکرار حلقه اگر هنوز غیر صفر است ;
```

۲۰-۹. شکل ۱-۲۰ از INT 21H تابع 40H استفاده می‌کند. برای تابع 05H، راه حل را بر طبق پرسش ۳-۲۰ بازنویسی کنید.

فصل ۲۱

۲۱-۱. (الف) واحد اندازه‌گیری حرکت موس ۱/۲۰۰ یک اینچی افزایش می‌یابد.

۲۱-۲. همهٔ این توابع در ابتدای فصل نزدیک معرفی شده‌اند.

۲۱-۳. نمایش نشانهٔ موس را کنترل می‌کند، وقتی صفر باشد نمایش می‌دهد و وقتی غیر صفر باشد لغو می‌کند.

۲۱-۴. (الف)

```
MOV AX,00H
INT 33H
```

۲۱-۵. این برنامه را تحت DEBUG اجرا کنید تا مقادیر بازگشتی را ببینید.

۲۱-۶. توجه کنید که شکل درگاه‌های موازی LPT1 و LPT2 را معکوس نموده است.

فصل ۲۲

۲۲-۱. مقدمهٔ این فصل سه دلیل را ارائه می‌دهد.

۲۲-۲. جملات شامل MACRO و ENDM می‌باشند.

۲۲-۵. (الف) XALL

۲۲-۶. (الف)

```
MULTBYTE  MACRO   MULTPR,MULTCD
          MOV     AL,MULTCD
          MUL     MULTPR
          ENDM
```

۲۲-۷. برای مشمول نمودن ماکرو در مرحله یک، چنین کد نمایید.

```
IF1
        INCLUDE  name-Library
ENDIF
```

۲۲-۸. تعریف ماکرو می‌تواند چنین آغاز شود :

```
PRINT17 MACRO PRTLINE,PRLEN
```

PRTLINE و PRLEN آرگونهای فرضی برای آدرس و طول خطی هستند که باید چاپ شود. فصل ۲۰ را برای استفاده از INT 17H برای چاپ کردن ببینید.

۲۲-۹. توجه کنید که نمی‌توانید از یک IF شرطی برای بررسی یک تقسیم صفر استفاده کنید. یک IF شرطی فقط در طی اسمبل کردن کار می‌کند. در حالیکه بررسی باید در طی اجرای برنامه باشد. دستورات اسمبلی را چنین کد نمایید.

```
CMP   DIVISOR,00      ; مقسوم علیه صفر است؟
JNZ   (bypass)        ; نه : بگذر
      CALL (error-message routine)
```

۲۲-۱۰. بخش (الف) و (ب) مستلزم پرسش ۶-۲۲ است، بخش (ج) مستلزم پرسش ۸-۲۲ برای چاپ کردن می‌باشد و فصل ۱۴ که تبدیل قالب دودویی به ASCII را دارد.

فصل ۲۳

۲۳-۱. مقدمه فصل دلایل را ارائه می‌دهد.

۲۳-۲. (الف) PARA.

۲۳-۳. (الف) NONE.

۲۳-۴. (الف) 'code'.

۲۳-۶. (الف) EXTRN SCALC23:FAR.

۲۳-۷. (الف) PRICE ، VALUE و PUBLIC QTY.

۲۳-۸. از شکل ۶-۲۳ به عنوان راهنما استفاده کنید.

۲۳-۹. از شکل ۸-۲۳ به عنوان راهنما برای ارسال پارامترها استفاده کنید. اما، این پرسش مستلزم قرار دادن سه متغیر بر روی پشته است. بنابراین برنامهٔ فراخوانده شده باید [BP + 10] را برای سومین ورودی روی پشته (UNITCOST) دستیابی کند. شما می‌توانید از استاندارد خودتان برای بازگرداندن UNITCOST روی پشته استفاده کنید. مراقب مقادیر برداشته شده از روی پشته در عملوند RET باشید.

۲۳-۱۰. این برنامه مستلزم مواردی از فصل ۹ (I/O صفحه)، ۱۳ (ضرب دودویی)، ۱۴ (تبدیل بین ASCII و دودویی) و ۲۳ (پیوند زدن با زیر برنامه‌ها) است در استفاده از پشته دقت کنید.

فصل ۲۴

۲۴-۱. (الف) در سکتور یک، تراک ۰.

۲۴-۲. مانند یک رابط سطح پایین با روتین‌های BIOS در ROM عمل می‌کند.

۲۴-۴. (الف) پس از MSDOS.SYS.

۲۴-۵. (الف) ۲۵۶ بایت اول برنامه هنگامی که برای اجرا در حافظه قرار می‌گیرد.

۲۴-۶. 5CH : 05 53 4C 49 4D 20 20 20 20 41 53 4D
80H : 0B 20 45 3A 53 4C 49 4D 2E 41 53 4D 0D

۲۴-۸. (الف) 2CD4

۲۴-۹. (الف) 1B38[0] + 100H [PSP] + 0H = 1B48[0]

۲۴-۱۰ (الف) شروع یک بلوک حافظه (نه یکی آخر).

۲۴-۱۱. INT 09H در جدول برداری وقفه در 24H.

فصل ۲۵

۲۵-۱. بخش وقفه‌ها در شروع این فصل، انواع را توصیف می‌کند.

۲۵-۲. بخش وقفه‌ها در شروع این فصل، این خطوط را توصیف می‌کند.

۲۵-۳. (الف) FFFF[0]H.

۲۵-۵. در آدرس سگمنت 40[0]H.

۲۵-۶. (الف) وضعیت تجهیزات، (ج) بایت دوم وضعیت شیفت.

۲۵-۷. (الف) آدرس (به ترتیب بایت معکوس) COM1 و COM2.

۲۵-۸. (الف) INT 21H، (ج) INT 11H.

۲۵-۹. INT 20H تا 3FH

۲۵-۱۰. (الف) 31H، (ج) 39H.

۲۵-۱۱. (الف) ارتباطات ورودی، (ج) تنظیم مجدد دیسک درایو.