



محمد علی مزیدی
جانیس گیلیسپی مزیدی

زبان اسمبلی

برنامه نویسی به زبان اسمبلی در کامپیوترهای آی بی ام

80x86 و سازگار با آن

(ویرایش دوم)

دکتر قدرت سپیدنام



زبان اسمبلی

برنامه نویسی کامپیوترهای آی بی ام 86 x 80 و سازگار با آن

کتابفروشی نوین



عرضه کننده کلیه کتب دانشگاهی و تست
فنی حرفه ای و کارودانش
۶۶۴۰۴۱۳۲

محمد علی مزیدی

جانیس گیلیسپی مزیدی

دکتر قدرت سپیدنام

مزیدی، محمد علی
 زبان اسمبلی: برنامه نویسی کامپیوترهای آی.بی.ام 80x86 و سازگار با آن / محمد علی
 مزیدی، سپیدنام. - مشهد باغانی، ۱۳۷۹
 چهار ۳۶۸ ص.: جدول، نمودار.
 ISBN 964-91532-4-1
 فهرست نویسی بر اساس اطلاعات فیپا.
 Assembly Language
 Programming on the IBM Pc, Ps, and compatibles, 2nd, ed
 ۱. ریزپردازنده های اینتل سری ۸۰x۸۶ -- برنامه نویسی. ۱.۲ اسمبلر (زبان برنامه
 نویسی کامپیوتر).
 الف. مزیدی، جانیس گیلیسپی
 د. سپیدنام، قدرت، ۱۳۲۴ - ج. عنوان.
 ۱۹م ۱۹الف/۸/ QAV۶ QAV۶/۸/ ۱۹الف ۲۶۵/۰۰۵
 ۱۳۷۹
 کتابخانه ملی ایران
 ۱۷۴۶۶-۷۹م

زبان اسمبلی

برنامه نویسی کامپیوترهای آی بی ام 80 x 86 و سازگار با آن

نوشته: محمد علی مزیدی، جانیس گیلیسپی مزیدی

ترجمه: دکتر قدرت سپیدنام

چاپ یازدهم، ۳۰۰۰ نسخه، بهار ۸۸، انتشارات باغانی

تهران: انقلاب - خیابان اردیبهشت (منیری جاوید) نرسیده به جمهوری

کوچه صابر - پلاک ۳ - تلفن: ۶۶۴۸۸۱۳۱ - ۶۶۴۸۸۱۳۰ - ۶۶۴۸۸۱۲۹

چاپخانه سروش

قیمت: ۶۰۰۰ تومان

بخش ۴-۱: مقدمه‌ای بر قطعات برنامه	۲۷
مبداء و تعریف قطعه	۲۷
آدرس منطقی و آدرس فیزیکی	۲۷
قطعه کد	۲۷
آدرس منطقی در برابر آدرس فیزیکی در قطعه کد	۲۸
قطعه داده	۲۹
آدرس منطقی و آدرس فیزیکی در قطعه داده	۳۱
قرارداد مبتنی بر کوچکتر	۳۱
قطعه اضافی (ES)	۳۲
نقشه حافظه IBM PC	۳۲
مطالبی بیشتر درباره RAM	۳۲
RAM تصویر	۳۳
مطالبی بیشتر در مورد ROM	۳۳
وظیفه BIOS ROM	۳۳
بخش ۵-۱: مطالبی بیشتر در مورد قطعات در 80x86	۳۴
پشته چیست و چرا لازم است؟	۳۴
چگونه پشته‌ها دستیابی می‌شوند	۳۴
درج در پشته	۳۵
باز یافت پشته	۳۵
آدرس منطقی در برابر آدرس فیزیکی برای پشته	۳۶
جمله‌ای چند درباره قطعات در 80x86	۳۶
همپوشی	۳۷
ثبات پرچم	۳۸
بیت‌های ثبات پرچم	۳۸
ثبات پرچم و دستور ADD	۳۹
استفاده از پرچم صفر در حلقه	۴۰
بخش ۶-۱: روش‌های آدرس‌دهی 80x86	۴۱
روش آدرس‌دهی ثباتی	۴۲
روش آدرس‌دهی فوری	۴۲
روش آدرس‌دهی مستقیم	۴۲
روش آدرس‌دهی غیر مستقیم ثباتی	۴۳
روش آدرس‌دهی نسبی پایه	۴۳
روش آدرس‌دهی نسبی اندیس دار	۴۴
روش آدرس‌دهی نسبی اندیس دار پایه	۴۴
لغو قطعه	۴۵
خلاصه	۴۶
فصل دوم: برنامه‌نویسی به زبان اسمبلی	۵۰
بخش ۱-۲: رهنمون‌ها و یک برنامه نمونه	۵۱
قطعه یک برنامه	۵۱
تعریف قطعه پشته	۵۲
تعریف قطعه داده	۵۲
تعریف قطعه کد	۵۳

فصل صفر: مقدمه‌ای بر محاسبات	۱
بخش ۱-۰: اعداد و سیستم‌های کدبندی	۲
سیستم اعداد دهدهی و دودویی	۲
تبدیل از دهدهی به دودویی	۲
تبدیل از دودویی به دهدهی	۲
سیستم شانزده شانزدهی	۳
تبدیل بین دودویی و شانزده شانزدهی	۴
تبدیل از دهدهی به شانزده شانزدهی	۴
تبدیل شانزده شانزدهی به دهدهی	۴
شمارش در مبنای ۱۰ و ۲ و ۱۶	۵
جمع اعداد دودویی و شانزده شانزدهی	۵
متمم ۲	۶
جمع و تفریق اعداد شانزده شانزدهی	۶
جمع اعداد مبنای شانزده	۶
تفریق دو عدد مبنای شانزده	۷
کد اسکی	۷
بخش ۲-۰: درون کامپیوتر	۸
بعضی اصطلاحات مهم	۸
سارمان درونی کامپیوترها	۹
توضیحی بیشتر درباره گذرگاه داده	۹
توضیحی بیشتر درباره گذرگاه آدرس	۱۰
CPU و ارتباط آن با RAM و ROM	۱۰
درون CPU ها	۱۱
عملیات درونی کامپیوتر	۱۲
بخش ۳-۰: تاریخچه مختصری از CPU	۱۳
CISC در مقایسه با RISC	۱۴
فصل اول: ریزپردازنده 80x86	۱۸
بخش ۱-۱: تاریخچه خلاصه‌ای از خانواده 80x86	۱۹
تکامل از 8080/8085 تا 8086	۱۹
تکامل از 8086 تا 8088	۱۹
موفقیت 8088	۱۹
دیگر ریزپردازنده‌ها: 80286 و 80386 و 80486	۱۹
بخش ۲-۱: درون 8088/86	۲۱
خط لوله گی	۲۲
ثبات‌ها	۲۳
بخش ۳-۱: مقدمه‌ای بر برنامه‌نویسی اسمبلی	۲۴
برنامه‌نویسی ب زبان اسمبلی	۲۴
دستور MOV	۲۴
دستور ADD	۲۶

۸۴	جمع اعداد بدون علامت
۸۴	مورد ۱: جمع داده‌های بایتی یا کلمه‌ای
۸۵	تحلیل برنامه ۱-۳ (الف)
۸۶	مورد ۲: جمع اعداد چند کلمه‌ای
۸۷	تحلیل برنامه ۲-۳
۸۸	تفریق اعداد بدون علامت
۸۹	تفریق با قرض، SBB

۸۹	بخش ۲-۳: ضرب و تقسیم بدون علامت
۹۰	ضرب اعداد بدون علامت
۹۱	تقسیم اعداد بدون علامت

۹۴	بخش ۳-۳: دستورات منطقی و برنامه‌های نمونه
۹۶	نقل مکان
۹۷	COMPARE برای اعداد بدون علامت
۱۰۰	تبدیل حروف کوچک به بزرگ به روش IBM BIOS
۱۰۱	مثالهایی از BIOS برای دستورات منطقی
۱۰۲	توضیح خط به خط

۱۰۳	بخش ۳-۴: دستورات و عملوندهای BCD و اسکی
۱۰۳	سیستم اعداد BCD
۱۰۳	BCD فشرده نشده
۱۰۳	اعداد اسکی
۱۰۳	تبدیل اسکی به BCD
۱۰۴	تبدیل اسکی به BCD فشرده نشده
۱۰۵	تبدیل اسکی به BCD فشرده
۱۰۵	تبدیل BCD فشرده به اسکی
۱۰۶	جمع و تفریق BCD
۱۰۶	جمع و تصحیح BCD
۱۰۷	خلاصه عمل DAA
۱۰۷	تفریق BCD و اصلاح آن
۱۰۷	خلاصه عمل DAS
۱۰۹	جمع و تفریق اسکی
۱۱۲	ضرب و تقسیم BCD غیر فشرده

۱۱۳	بخش ۳-۵: دستورات چرخش
۱۱۳	چرخش بیت‌های یک عملوند به راست و به چپ
۱۱۳	چرخش به راست ROR
۱۱۳	چرخش به چپ ROL
۱۱۴	RCR چرخش به راست از طریق نقلی
۱۱۵	چرخش به چپ از طریق نقلی RCL

۱۱۶	بخش ۳-۶: عملیات بیتی در زبان C
۱۱۶	عملگرهای بیتی در C
۱۱۶	عملگرهای نقل مکان بیتی در C
۱۱۷	تبدیل BCD فشرده به اسکی در C
۱۱۸	تست بیت‌ها در C
۱۱۹	خلاصه

۵۵	بخش ۲-۲: اسمبل کردن، پیوند و اجرای یک برنامه
۵۶	فایل‌های asm و obj
۵۶	فایل list
۵۷	رهنمون‌های PAGE و TITLE
۵۷	فایل crf
۵۷	پیوند برنامه
۵۸	فایل map

۵۸	بخش ۲-۳: چند برنامه نمونه دیگر
۵۹	تحلیل برنامه ۱-۲
۶۱	روش‌های مختلف طراحی برنامه ۱-۲
۶۳	تحلیل برنامه ۲-۲
۶۳	تحلیل برنامه ۲-۳
۶۳	بررسی مجدد تعریف قطعه پشته

۶۵	بخش ۲-۴: دستورات انتقال کنترل
۶۵	FAR (دور) و NEAR (نزدیک)
۶۵	پرش‌های شرطی
۶۵	پرش‌های کوتاه
۶۷	پرش‌های بدون شرط
۶۷	عبارات فراخوانی (CALL)
۶۸	زیر روال‌های زبان اسمبلی
۶۹	قوانین نام‌گذاری در زبان اسمبلی

۷۰	بخش ۲-۵: انواع داده‌ها و تعریف داده
۷۰	انواع داده‌های 80x86
۷۰	رهنمون‌های داده اسمبلر
۷۰	ORG (میداء)
۷۰	DB (تعریف بایت)
۷۱	DUP (کپی)
۷۲	DW (تعریف کلمه)
۷۲	EQU (برابر پنداشتن)
۷۲	DD (تعریف جفت کلمه)
۷۳	DQ (تعریف چهار کلمه)
۷۳	DT (تعریف ده بایت)

۷۴	بخش ۲-۶: تعریف قطعه ساده شده
۷۵	مدل حافظه
۷۵	تعریف قطعه

۷۷	بخش ۲-۷: فایل‌های EXE در برابر COM
۷۷	چرا فایل‌های COM؟
۷۸	تبدیل از EXE به COM
۷۸	خلاصه

فصل سوم: دستورات محاسباتی، منطقی و برنامه‌ها ۸۳

۸۴	بخش ۳-۱: جمع و تفریق بدون علامت

فصل چهارم: برنامه نویسی BIOS ۱۲۳

بخش ۱-۴: برنامه نویسی INT10H از BIOS ۱۲۴

- ۱۲۴ صفحه نمایش در روش متن
- ۱۲۵ پاک کردن صفحه نمایش بکمک INT10H تابع 06H
- ۱۲۵ INT 10H تابع 02: استقرار مکان نما در یک محل خاص
- ۱۲۶ INT 10H تابع 03: تعیین محل فعلی مکان نما
- ۱۲۶ تغییر مد تصویر
- ۱۲۷ بایت خصیصه در مانیتورهای تک رنگ
- ۱۲۷ بایت خصیصه در مد متن CGA
- ۱۲۹ گرافیک: وضوح پیکسل و رنگ
- ۱۳۰ INT 10H و برنامه ریزی پیکسل
- ۱۳۰ ترسیم خطوط افقی و عمودی در مد گرافیک
- ۱۳۰ تغییر رنگ پس زمینه

بخش ۲-۴: وقفه 21H در DOS ۱۳۱

- ۱۳۲ INT21H انتخاب 09: خروج رشته ای از داده روی مانیتور
- ۱۳۲ INT21H انتخاب 02: خروج یک کاراکتر به مانیتور
- ۱۳۲ INT21H انتخاب 01: ورود یک کاراکتر همراه با انعکاس
- ۱۳۲ INT21H انتخاب 0AH: ورود یک رشته داده از صفحه کلید
- ۱۳۴ ورود بیش از سائز بافر
- ۱۳۶ استفاده از بازگشت و تعویض خط
- ۱۳۷ INT21H انتخاب 07: ورود از صفحه کلید بدون انعکاس
- ۱۳۸ استفاده از رهنمون LABEL برای تعریف یک بافر رشته

بخش ۳-۴: برنامه نویسی صفحه کلید INT16H ۱۴۱

- ۱۴۱ تست فشردن کلید
- ۱۴۱ کدام کلید فشرده شده است؟

بخش ۴-۴: برنامه نویسی فایل دیسک ۱۴۳

- ۱۴۳ شناسه فایل و کد خطا
- ۱۴۳ AH = 3CH, INT21H (باز کردن یک فایل)
- ۱۴۴ AH = 40H, INT21H (نوشتن در فایل)
- ۱۴۴ AH = 3EH, INT21H (بستن یک فایل)
- ۱۴۴ AH = 3FH, INT21H (خواندن یک فایل)
- ۱۴۴ AH = 3DH, INT21H (بازکردن یک فایل)

بخش ۵-۴: برنامه نویسی وقفه با C ۱۴۵

- ۱۴۵ برنامه نویسی وقفه های BIOS با ++C
- ۱۵۰ برنامه نویسی تابع INT21H DOS با ++C
- ۱۵۰ دستیابی به ثبات های قطعه
- ۱۵۱ دستیابی به پرچم نقلی در توابع int86 و intdos
- ۱۵۲ ترکیب C با اسمبلی و تست ZF
- ۱۵۲ تابع kbhit از C در برابر ورودی صفحه کلید INT16H
- ۱۵۳ خلاصه

بخش پنجم: ماکروها و ماوس ۱۵۸

بخش ۱-۵: ماکرو چیست و چگونه بکار می رود ۱۵۹

- ۱۵۹ تعریف MACRO
- ۱۶۰ توضیحات در ماکرو
- ۱۶۲ تحلیل برنامه ۵-۱
- ۱۶۳ رهنمون LOCAL و کاربرد آن در ماکرو
- ۱۶۶ رهنمون INCLUDE

بخش ۲-۵: برنامه نویسی ماوس با وقفه 33H ۱۶۹

- ۱۶۹ تشخیص وجود ماوس
- ۱۷۰ برخی از اصطلاحات ماوس
- ۱۷۰ نمایش و پنهان کردن مکان نمای ماوس
- ۱۷۱ وضوح تصویر در مقابل دقت ماوس در مد متن
- ۱۷۱ وضوح تصویر در برابر دقت ماوس در مد گرافیک
- ۱۷۱ بدست آوردن مکان جاری مکان نمای ماوس
- ۱۷۴ استقرار محل اشاره گر ماوس (AX=04)
- ۱۷۴ بدست آوردن اطلاعات حاصل از فشردن کلید ماوس
- ۱۷۵ برنامه ردیابی و نمایش دفعات فشردن کلید
- ۱۷۶ بدست آوردن اطلاعات دکمه رها شده در ماوس
- ۱۷۶ تنظیم محدوده افقی برای اشاره گر ماوس (AX=07)
- ۱۷۶ تنظیم محدوده عمودی برای اشاره گر ماوس (AX=08)
- ۱۷۷ تنظیم ناحیه خارج از محدوده (exclusion)
- ۱۷۷ بدست آوردن اطلاعات راه انداز ماوس (ویرایش)
- ۱۷۸ خلاصه

فصل ششم: اعداد علامت دار، رشته ها و جداول ۱۸۱

بخش ۱-۶: عملیات محاسباتی اعداد علامت دار ۱۸۲

- ۱۸۲ مفهوم اعداد علامت دار در کامپیوترها
- ۱۸۲ عملوندهای باینری علامت دار
- ۱۸۲ اعداد مثبت
- ۱۸۲ اعداد منفی
- ۱۸۴ مسئله سرریز در عملیات اعداد علامت دار
- ۱۸۴ چه وقت پرچم سرریز در عملیات 8 بیت نشانه می شود
- ۱۸۵ پرچم سرریز در عملیات 16 بیتی
- ۱۸۵ پیشگیری از نتایج غلط در عملیات اعداد علامت دار
- ۱۸۷ IDIV (تقسیم عدد علامت دار)
- ۱۸۸ IMUL (ضرب عدد علامت دار)
- ۱۸۸ نقل مکان حسابی
- ۱۸۸ SAR (نقل مکان حسابی به راست)
- ۱۸۹ SAL (نقل مکان حسابی به چپ) و SHL (نقل مکان به چپ)
- ۱۹۰ مقایسه عدد علامت دار

بخش ۲-۶: عملیات رشته ای و جدولی ۱۹۱

- ۱۹۲ استفاده از SI و DI، DS و ES در دستورات رشته ای
- ۱۹۲ عملوندهای باینری و کلمه ای در دستورات رشته ای
- ۱۹۲ پرچم جهت، DF
- ۱۹۳ پیشوند REP
- ۱۹۳ دستورالعمل های STOS و LODS
- ۱۹۴ تست حافظه با STOSB و LODSE
- ۱۹۴ پیشوندهای REPZ و REPNZ

۲۱۸	وارد کردن کد اسمبلی 80x86 در برنامه های C
۲۱۹	برنامه های C فراخوان رویه های اسمبلی
۲۲۰	استاندارد فراخوانی C
۲۲۱	چگونه پارامترها به C بازگردانده می شوند
۲۲۲	اسمبلرهای جدید و پیوند با C
۲۲۳	انتقال آدرس های آرایه ای از C به پشته
۲۲۳	پیوند روال های زبان اسمبلی با C
۲۲۴	خلاصه

فصل هشتم: برنامه نویسی 32 بیتی برای ماشین های

۲۲۸	386 و 486

۲۲۹	بخش ۱-۸: ماشین های 80386/80486 در مد واقعی
۲۳۰	ثبات های همه منظوره به عنوان اشاره گر در 386/486
۲۳۱	محدوده حداکثر حافظه در مد واقعی برای 386/486: 1M
۲۳۲	دستیابی به ثبات های 32 بیتی با اسمبلی های رایج
۲۳۴	بازدید مجدد منتهی به کوچکتر

۲۳۴	بخش ۲-۸: تعدادی برنامه ساده 386/486
۲۳۴	جمع کلمات 16 بیتی با استفاده از ثبات های 32 بیتی
۲۳۶	جمع داده چند کلمه ای در ماشین های 386/486
۲۳۷	ضرب عملوند 32 بیتی در عملوند 16 بیتی در 386/486
۲۳۷	ضرب 32 بیت در 16 بیت با ثبات های 8086/286

۲۳۹	بخش ۳-۸: مقایسه عملکرد 80x86
۲۳۹	اجرای یک برنامه 8086 در خانواده 80x86
۲۴۱	خلاصه

۲۴۳	پیوست الف: برنامه نویسی DEBUG
۲۶۵	پیوست ب: دستور العمل ها و زمان بندی
۳۰۳	پیوست پ: رهنمون اسمبلر و قوانین نام گذاری
۳۱۸	پیوست ت: لیست وقفه
۳۴۰	پیوست ث: وقفه های
۳۵۵	پیوست ج: کدهای اسکی
۳۵۷	پیوست چ: فلوچارت ها، شبه کد و اسمبلر ها

۱۹۵	SCAS (پویش رشته)
۱۹۶	جایگزینی کار کتر پویش یافته
۱۹۶	دستور XLAT و جداول نظاره
۱۹۷	تبدیل کد بکمک XLAT
۱۹۸	خلاصه

فصل هفتم: مدول ها: برنامه نویسی مدولی و C

۲۰۱	بخش ۱-۷: نوشتن و پیوند مدول ها
۲۰۱	چرا مدول ها؟
۲۰۱	نوشتن مدول ها
۲۰۱	رهنمون EXTRN
۲۰۱	رهنمون PUBLIC
۲۰۲	رهنمون END در مدول ها
۲۰۳	پیوند مدول ها و تبدیل آن به یک واحد قابل اجرا
۲۰۴	تحلیل برنامه ۱-۷
۲۰۵	رهنمون SEGMENT
۲۰۵	تعریف قطعه پشته کامل
۲۰۵	تعاریف قطعه کد و داده
۲۰۷	تحلیل نقشه پیوند برنامه ۲-۷
۲۰۸	برنامه نویسی مدولی و استفاده از تعریف قطعه جدید

۲۱۰	بخش ۲-۷: برخی از مدول های بسیار مفید
۲۱۰	تبدیل دودویی (مبنای شانزده) به اسکی (دهدی)
۲۱۱	تبدیل اسکی (دهدی) به دودویی (مبنای 16)
۲۱۲	مدول دودویی به اسکی
۲۱۴	مدول اسکی به دودویی
۲۱۴	مدول فراخوان

۲۱۵	بخش ۳-۷: تبادل پارامترها در میان مدول ها
۲۱۵	تبدیل پارامترها از طریق ثبات ها
۲۱۵	تبادل پارامترها از طریق حافظه
۲۱۵	تبادل پارامترها از طریق پشته
۲۱۶	تحلیل محتوای پشته برای برنامه ۹-۷

۲۱۷	بخش ۴-۷: ترکیب زبان اسمبلی و برنامه های C
۲۱۸	چرا C؟

مقدمه‌ای بر محاسبات

اهداف

در پایان این فصل قادر خواهید بود:

- « هر عدد در مبنای 2، مبنای 10، یا مبنای 16 را به هر یک از دو مبنای دیگر تبدیل کنید.
- « در مبنای 2 یا 16 بشمارید.
- « اعداد مبنای 16 را جمع یا تفریق کنید.
- « اعداد مبنای 2 را جمع نمایید.
- « هر عدد دودویی را به صورت متمم 2 نشان دهید.
- « یک رشته الفبا عدد را در کد اسکی (ASCII) نمایش دهید.
- « اختلاف بین بیت، نیکل، بایت و کلمه را توضیح دهید.
- « تعریف ریاضی دقیقی از اصطلاحات کیلوبایت، مگابایت، ترابایت و گیگابایت ارائه دهید.
- « اختلاف بین RAM و ROM و کاربرد آنها را توصیف نمایید.
- « اجزاء مهم یک سیستم کامپیوتر را فهرست نموده و عملکرد آنها را توضیح دهید.
- « سه نوع گذرگاه موجود در سیستم‌های کامپیوتر را ذکر نموده و نقش هر یک از آنها را توصیف نمایید.
- « نقش CPU را در سیستم‌های کامپیوتری تشریح کنید.
- « اجزاء اصلی CPU را فهرست نموده و نقش هر یک را توضیح دهید.
- « تکامل کامپیوتر را از لامپ‌های خلاء تا ترانزیستور و تا تراشه‌های IC دنبال کنید.
- « اختلاف بین طرح‌های CISC و RISC را بیان نمایید.

برای درک نرم افزار و سخت افزار کامپیوتر، باید ابتدا برخی از مفاهیم اصلی طراحی کامپیوتر را دانست. در این فصل (که بر اساس سنت کامپیوترهای دیجیتال آن را فصل ۰ می نامیم)، اصول عددگذاری و سیستم های کدبندی ارائه شده اند. سپس مقدمه ای بر کار داخل کامپیوتر بیان شده است. نهایتاً در بخش آخر مختصری از تاریخچه معماری CPU را خواهیم گفت. گرچه بعضی از خوانندگان ممکن است زمینه کافی در بسیاری از عنوانهای این فصل داشته باشند، ولی توصیه می شود که مطالب، هر چند بطور مختصر، مرور شود.

بخش ۱-۰: اعداد و سیستم های کدبندی

در حالیکه انسان از حساب مبنای ۱۰ (دهدهی) استفاده می کند، کامپیوترها سیستم مبنای ۲ (دودویی) را بکار می برند. در این بخش چگونگی تبدیل سیستم دهدهی را به سیستم دودویی و برعکس توضیح می دهیم. نمایش معمول اعداد دودویی که شانزده شازدهی (hex) نامیده می شود نیز ارائه شده است. نهایتاً قالب دودویی کد الفبا عددی، که اسکی خوانده می شود، تشریح شده است.

سیستم اعداد دهدهی و دودویی

گرچه تصور بر این است که منشاء سیستم اعداد در مبنای ۱۰ انگشت انسان بوده است، ولی مطلقاً هیچگونه گمانی در مورد دلیل بکارگیری سیستم دودویی در کامپیوتر وجود ندارد. سیستم دودویی به منظور نمایش دو سطح ولتاژ خاموش و روشن با ۰ و ۱ استفاده شده است. در سیستم مبنای ۱۰، ده نماد جدا از هم ۰، ۱، ۲، ۳، ۴، ۵، ۶، ۷، ۸، ۹ برای تولید اعداد وجود دارند، در حالیکه در سیستم دودویی فقط دو نماد ۰ و ۱ موجود است. به بیان دیگر مبنای ده حاوی ارقام ۰ تا ۹ و دودویی حاوی ارقام ۰ و ۱ است. این دو رقم دودویی ۰ و ۱ غالباً بیت خوانده می شوند.

تبدیل از دهدهی به دودویی

یکی از طرق تبدیل دهدهی به دودویی تقسیم متوالی عدد بر ۲ است که طی آن باقیمانده ها مورد استفاده قرار می گیرند. این عمل تا صفر شدن خارج قسمت ادامه می یابد. سپس باقیمانده ها بصورت معکوس بدنبال هم نوشته می شوند تا عدد دودویی بدست آید. این عمل در مثال ۱-۰ نشان داده شده است.

مثال ۱-۰: ۲۵۱۰ را به دودویی تبدیل کنید.

حل:

باقیمانده	خارج قسمت			
۱	۱۲	=	۲۵/۲	کم ارزشترین بیت (LSB)
۰	۶	=	۱۲/۲	
۰	۳	=	۶/۲	
۱	۱	=	۳/۲	
۱	۰	=	۱/۲	با ارزشترین بیت (MSB)

بنابراین $25_{10} = 11001_2$.

تبدیل از دودویی به دهدهی

هنگام تبدیل دودویی به دهدهی، دانستن مفهوم وزن مربوط به هر مکان رقم اهمیت دارد. ابتدا برای مقایسه، وزن اعداد را در سیستم مبنای ۱۰ مرور می کنیم:

$$740683_{10} =$$

$$\begin{array}{rcl} 3 \times 10^0 & = & 3 \\ 8 \times 10^1 & = & 80 \\ 6 \times 10^2 & = & 600 \\ 0 \times 10^3 & = & 0000 \\ 4 \times 10^4 & = & 40000 \\ 7 \times 10^5 & = & 700000 \\ \hline & & 740683 \end{array}$$

به روشی مشابه، مکان هر رقم در عدد مبنای 2 دارای وزن مربوط به خود است.

$$110101_2 =$$

			دهدیی	دودویی
1×2^0	$= 1 \times 1$	$=$	1	1
0×2^1	$= 0 \times 2$	$=$	0	00
1×2^2	$= 1 \times 4$	$=$	4	100
0×2^3	$= 0 \times 8$	$=$	0	0000
1×2^4	$= 1 \times 16$	$=$	16	10000
1×2^5	$= 1 \times 32$	$=$	32	100000
			53	110101

اطلاع از وزن هر بیت در یک عدد دودویی، جمع آنها را برای تهیه معادل دهدیی آسان می‌سازد. به مثال

۲-۰ توجه کنید.

مثال ۲-۰: 11001_2 را به دهدیی تبدیل کنید.

حل:

16	8	4	2	1	وزن:
1	1	0	0	1	ارقام:
$16 +$	$8 +$	$0 +$	$0 +$	$1 = 25_{10}$	جمع:

دانستن وزن مربوط به مکان بیت دودویی این امکان را به فرد می‌دهد تا به جای استفاده از روند تقسیم متوالی، مستقیماً یک عدد دهدیی را به دودویی تبدیل کند. این روش در مثال ۳-۰ نشان داده شده است.

مثال ۳-۰: با استفاده از مفهوم وزن، 39_{10} را به دودویی تبدیل کنید.

حل:

32	16	8	4	2	1	وزن:
1	0	0	1	1	1	
$32 +$	$0 +$	$0 +$	$4 +$	$2 +$	$1 = 39$	

$$39_{10} = 100111_2 \text{ بنابراین}$$

جدول ۱-۰ دهدیی و شانزده شانزدهی

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

سیستم شانزده شانزدهی

سیستم شانزده شانزدهی یا مبنای 16، به عنوان نمایش مناسبی از ارقام دودویی بکار می‌رود. مثلاً، برای انسان نمایش ترکیبی از 0 و 1 ها مثل 100010010110 به صورت معادل شانزده شانزدهی 896H آسانتر است. سیستم دودویی دارای دو رقم 0 و 1 است. سیستم مبنای 16 دارای ده رقم 0 تا 9 می‌باشد. سیستم مبنای 16 باید شانزده رقم داشته باشد. 16، ده رقم اول همان 0 تا 9 مبنای 10، و 10 تا 15 را با حروف A، B، C، D، E و F می‌نویسند. جدول ۱-۰ می‌شود. نمایش انزده شانزدهی را برای 0 تا 15 نشان می‌دهد.

تبدیل بین دودویی و شانزده شانزدهی

برای نمایش یک عدد دودویی به معادل شانزده شانزدهی اش، از سمت راست گروه‌های چهاربیتی را جدا کرده و هر عدد چهاربیتی دودویی را با معادل شانزده شانزدهی اش طبق جدول ۱-۵، جایگزین کنید.
برای تبدیل از شانزده شانزدهی به دودویی، هر رقم شانزده شانزدهی را با چهاربیت دودویی معادلش جایگزین نمایید. تبدیل دودویی و شانزده شانزدهی در مثال‌های ۴-۵ و ۵-۵ نشان داده شده است.

تبدیل از ددهی به شانزده شانزدهی

تبدیل ددهی به شانزده شانزدهی به دو طریق صورت می‌گیرد:

- ۱- ابتدا آن را به دودویی تبدیل کنید و سپس آن را به شانزده شانزدهی برگردانید. تمرین این روش به خواننده واگذار می‌شود.
- ۲- با استفاده از روش تقسیم تکراری آن را مستقیماً به شانزده شانزدهی تبدیل نمایید و باقیمانده‌ها را به شرحی که قبلاً گذشت به دنبال هم قرار دهید. مثال ۶-۵ این روش را نمایش می‌دهد.

تبدیل شانزده شانزدهی به ددهی

تبدیل از شانزده شانزدهی به ددهی نیز می‌تواند به دو طریق انجام شود:

- ۱- از شانزده شانزدهی به دودویی و سپس به ددهی تبدیل کنید.
 - ۲- با جمع همه ارقام وزین، عمل تبدیل شانزدهی به ددهی را مستقیماً انجام دهید.
- مثال ۷-۵ روش دوم تبدیل شانزدهی را به ددهی نشان می‌دهد.

مثال ۴-۵: عدد دودویی 100111110101 را به شانزده شانزدهی نمایش دهید.

حل:

ابتدا عدد به صورت گروه‌های چهاربیتی درمی‌آید: 1001 1111 0101
سپس هر گروه چهاربیتی با معادل شانزدهی اش جایگزین می‌شود:

1001	1111	0101
9	F	5

بنابراین $100111110101_2 = 9F5_{16}$.

مثال ۵-۵: 29B در مبنای شانزده را به دودویی تبدیل کنید.

حل:

2	9	B
0010	1001	1011

با حذف صفرهای سمت چپ خواهیم داشت، 1010011011.

مثال ۶-۵: (الف) 45_{10} را به شانزده شانزدهی تبدیل نمایید.

حل:

باقیمانده	خارج قسمت	
13	2	$45/16 =$
2	0	$2/16 =$

(کم ارزشترین رقم) (hexD)
(با ارزشترین رقم)
بنابراین $45_{10} = 2D_{16}$ خواهد بود.

(ب) 629 در مبنای 10 را به مبنای شانزده ببرید.

حل:

باقیمانده	خارج قسمت	
5	39	$629/16 =$
7	2	$39/16 =$
2	0	$2/16 =$

(کم ارزشترین رقم)
(با ارزشترین رقم)
بنابراین $629_{10} = 275_{16}$.

(پ) 1714 در مبنای ده را به مبنای شانزده ببرید.

خارج قسمت	باقیمانده	حل :
$1714/16 = 107$	2	(کم ارزشترین رقم)
$107/16 = 6$	11	(hexB)
$6/16 = 0$	6	(با ارزشترین رقم)

بنابراین $1714_{10} = 6B2_{16}$

مثال ۷-۰ : اعداد مبنای شانزده زیر را به دهدهی تبدیل کنید.

(الف) $6B2_{16}$

حل :

$$\begin{aligned}
 2 \times 16^0 &= 2 \times 1 = 2 \\
 11 \times 16^1 &= 11 \times 16 = 176 \\
 6 \times 16^2 &= 6 \times 256 = 1536 \\
 &\quad \underline{1714}
 \end{aligned}$$

بنابراین $6B2_{16} = 1714_{10}$

(ب) $9F2D_{16}$

حل :

$9F2D$ شانزده شانزدهی =

$$\begin{aligned}
 13 \times 16^0 &= 13 \times 1 = 13 \\
 2 \times 16^1 &= 2 \times 16 = 32 \\
 15 \times 16^2 &= 15 \times 256 = 3840 \\
 9 \times 16^3 &= 9 \times 4096 = 36864 \\
 &\quad \underline{40749}
 \end{aligned}$$

بنابراین $9F2D_{16} = 40749_{10}$

دهدهی	دودویی	شانزده شانزدهی
0	00000	0
1	00001	1
2	00010	2
3	00011	3
4	00100	4
5	00101	5
6	00110	6
7	00111	7
8	01000	8
9	01001	9
10	01010	A
11	01011	B
12	01100	C
13	01101	D
14	01110	E
15	01111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14
21	10101	15
22	10110	16
23	10111	17
24	11000	18
25	11001	19
26	11010	1A
27	11011	1B
28	11100	1C
29	11101	1D
30	11110	1E
31	11111	1F

شکل ۱-۰ - شمارش در سه مبنای

شمارش در مبنای ۱۰ و ۲ و ۱۶

برای نمایش رابطه بین هر سه مبنای، رشته اعداد 0 تا 31 همراه با معادل دودویی و شانزده شانزدهی آنها در شکل ۱-۰ نشان داده شده‌اند. توجه داشته باشید که در هر مبنای، اگر یکی به بالاترین رقم بیفزائیم، آن رقم صفر شده و یک 1 به مکان رقم بالاتر منتقل می‌شود. مثلاً در دهدهی، $0 + 1 = 9$ است که به مکان بالاتر بعدی یکی اضافه می‌شود. در دودویی $0 + 1 = 1$ ، همراه با یک رقم نقلی، و بطور مشابه در شانزده شانزدهی $0 + 1 = F$ به همراه رقم نقلی خواهد بود.

جمع اعداد دودویی و شانزده شانزدهی

جمع اعداد دودویی یک عمل سر راست است. جدول ۲-۰ جمع دو بیت را نشان می‌دهد. از بحث تفریق، به دلیل اینکه همه کامپیوترها از فرآیند جمع برای پیاده سازی تفریق استفاده می‌کنند چشم‌پوشی می‌شود.

جدول ۲-۰ جمع دودویی

A+B	نقلی	جمع
0+0	0	0
0+1	0	1
1+0	0	1
1+1	1	0

گرچه کامپیوترها دارای مدارات جمع می‌باشند، ولی مدار تفریق‌کننده مجزایی برای تفریق در آنها وجود ندارد. در عوض، جمع‌کننده‌ها همراه با مدارهای متمم 2 برای اجرای تفریق بکار می‌روند. به بیان دیگر در اجرای " $x - y$ "، کامپیوتر متمم 2 کمیت y را بدست آورده و x را به آن می‌افزاید. مفهوم متمم 2 بعداً مرور شده است. مثال ۸-۰ جمع اعداد دودویی را نشان می‌دهد.

مثال ۸-۰: اعداد دودویی زیر را جمع کنید. آنها را با معادل دهدهی‌شان چک نمایید.
حل:

	دودویی	دهدهی
	1101	13
	1001	9
+	10110	22
	101100	44

متمم 2

برای تهیه متمم 2 عدد دودویی، همه بیت‌ها را معکوس کنید و سپس به حاصل 1 واحد بیفزائید. معکوس هر بیت به معنی تبدیل همه 0 ها به 1 و همه 1 ها به 0 است. این عمل را متمم 1 می‌نامیم. مثال ۹-۰ ملاحظه شود.

مثال ۹-۰: متمم 2 عدد 10011101 را بدست آورید.

	عدد دودویی	حل:
	10011101	
	01100010	متمم 1
+	1	
	01100011	متمم 2

جمع و تفریق اعداد شانزده شانه‌ای

در مطالعه موضوعات مرتبط با نرم‌افزار و سخت‌افزار کامپیوترها، اغلب لازم است تا اعداد شانزده شانه‌ای را جمع و تفریق کنیم. آگاهی از این تکنیک‌ها الزامی است. جمع و تفریق شانزده شانه‌ای در زیر بطور جداگانه بحث شده‌اند.

جمع اعداد مبنای شانزده

این بخش فرایند جمع اعداد شانزده شانه‌ای را توصیف می‌کند. ارقام، با شروع از کم ارزشترین رقم، با هم جمع می‌شوند. اگر جواب کمتر از 16 باشد، حاصل جمع را برای آن مکان بنویسید. اگر حاصل جمع بزرگتر از 16 باشد، 16 را از آن کم کنید تا رقم آن مکان و نیز رقم نقلی 1 را برای رقم بعدی بدست آورید. بهترین راه برای توضیح، ارائه آن با مثالی همانند مثال ۱۰-۰ است.

مثال ۱۰-۰: جمع مبنای شانزده را انجام دهید: $23D9 + 94BE$.
حل:

$$\begin{array}{r} 23D9 \\ + 94BE \\ \hline B897 \end{array}$$

$$\begin{array}{lcl}
 \text{LSD :} & 9+14 & = 23 & 23-16=7 \text{ با رقم نقلی به رقم بعد} \\
 & 1+13+11 & = 25 & 25-16=9 \text{ با رقم نقلی به رقم بعد} \\
 & 1+3+4 & = 8 & \\
 \text{MSD:} & 2+9 & = B &
 \end{array}$$

تفریق دو عدد مبنای شانزده

در تفریق دو عدد شانزده شانزدهی (hex)، اگر رقم دوم بزرگتر از اولی باشد، از رقم بعدی 16 واحد قرض کنید. به مثال ۱۱-۰ توجه نمایید.

مثال ۱۱-۰: تفریق مبنای شانزده را اجرا نمایید: $59F - 2B8$

$$\begin{array}{r}
 59F \\
 - 2B8 \\
 \hline
 2E7
 \end{array}$$

حل:

LSD: $15 = 7 \rightarrow 8 \rightarrow 15$

که برابر E است، $25(9+16) = 14 \rightarrow 11$

MSD: $2 \rightarrow 4(5-1) = 2$

کد اسکی

بحث ما تا کنون حول محور نمایش سیستم اعداد بوده است. چون همه اطلاعات در کامپیوتر باید با 0 و 1 نشان داده شوند، بنابراین الگوهای دودویی باید به همه حروف و دیگر کاراکترها تخصیص یابند.

در ۱۹۶۰ نمایش استاندارد به نام اسکی (کد استاندارد امریکائی برای تبادل اطلاعات) ایجاد گردید. کد اسکی الگوهای دودویی را به اعداد 0 تا 9، همه حروف الفبایی از هر دو نوع حروف بزرگ و کوچک و بسیاری از کدهای کنترل و علائم نقطه گذاری، اختصاص می دهد. مزیت مهم این سیستم این است که در اغلب کامپیوترها بکار رفته است، بنابراین اطلاعات می تواند بین کامپیوترها مشترک باشد. سیستم اسکی جمعاً 7 بیت را برای نمایش هر یک بکار می برد. مثلاً 0001 100 به حرف بزرگ "A" و 0001 110 به حرف کوچک "a" اختصاص یافته است. غالباً یک صفر در با اهمیت ترین مکان بیت قرار می گیرد تا آن را به یک کد 8 بیتی تبدیل نماید. شکل ۲-۰ کدهای منتخبی را از اسکی نشان می دهد. لیست کاملی از کدهای اسکی در پیوست (ج) داده شده است. استفاده از اسکی نه تنها برای استاندارد صفحات کلید در امریکا بکار می رود بلکه در سایر کشورها نیز استفاده می شود. این استاندارد برای چاپ و نمایش کاراکترها با وسایل خروجی مانند چاپگرها و مانیتورها نیز مورد استفاده اند.

الگوی کدهای اسکی بخاطر ساده کردن دستکاری داده ها طراحی شد. مثلاً ارقام 0 تا 9 با کدهای اسکی، 30 تا 39 نشان داده می شوند. این انتخاب موجب می شود تا با پوشش یا حذف 3 در نیل (چهار بیت) مرتبه بالا، تبدیل اسکی به سادگی انجام شود. بعنوان مثالی دیگر به لیست کدهای زیر توجه کنید که در آن رابطه ای بین حروف بزرگ و کوچک دیده می شود. حروف بزرگ با کدهای اسکی 41 تا 5A نشان داده می شوند، در حالیکه حروف کوچک با کدهای اسکی، 61 تا 7A نمایش داده شده اند. با ملاحظه کد دودویی آن ها دیده می شود که تنها بیت متفاوت در حروف بزرگ "A" و حروف کوچک "a"، بیت 5 است. بنابراین تبدیل بین حروف بزرگ و کوچک تنها با تغییر بیت 5 در اسکی امکان پذیر است.

شانزده شانزدهی	نماد	شانزده شانزدهی	نماد
41	A	61	a
42	B	62	b
43	C	63	c
44	D	64	d
45	E	65	e
46	F	66	f
47	G	67	g
48	H	68	h
49	I	69	i
4A	J	6A	j
4B	K	6B	k
4C	L	6C	l
4D	M	6D	m
4E	N	6E	n
4F	O	6F	o
50	P	70	p
51	Q	71	q
52	R	72	r
53	S	73	s
54	T	74	t
55	U	75	u
56	V	76	v
57	W	77	w
58	X	78	x
59	Y	79	y
5A	Z	7A	z

شکل ۲-۰ کدهای اسکی الفبا عددی

- ۱ - چرا کامپیوترها به جای استفاده از سیستم اعداد دهدهی از سیستم اعداد دودویی استفاده می کنند؟
- ۲ - 34_{10} را به دودویی و شانزده شانزدهی تبدیل کنید.
- ۳ - 110101_2 را به شانزده شانزدهی و دهدهی تبدیل نمایید.
- ۴ - جمع دودویی مقابل را انجام دهید: $101 + 101100$
- ۵ - 101100_2 را به متمم 2 نمایش دهید.
- ۶ - $36BH + F6H$ را جمع نمایید.
- ۷ - $36BH - F6H$ را تفریق کنید.
- ۸ - جمله "80x86 CPU" را به کد اسکی (به شکل شانزده شانزدهی) بنویسید.

بخش ۲-۰: درون کامپیوتر

در این بخش مقدمه‌ای را برای سازمان و کار درونی کامپیوترها فراهم می آوریم. مدل بکار رفته یک مدل عمومی است، ولی مفاهیم مورد استفاده قابل اعمال به همه کامپیوترها از جمله IBM PC، PS/2 و سازگار با آنهاست. قبل از آغاز این مبحث، مروری بر تعاریف برخی از اصطلاحات در کامپیوتر، مانند کیلو (K)، مگا، گیگا، بایت، ROM، RAM و غیره مفید است.

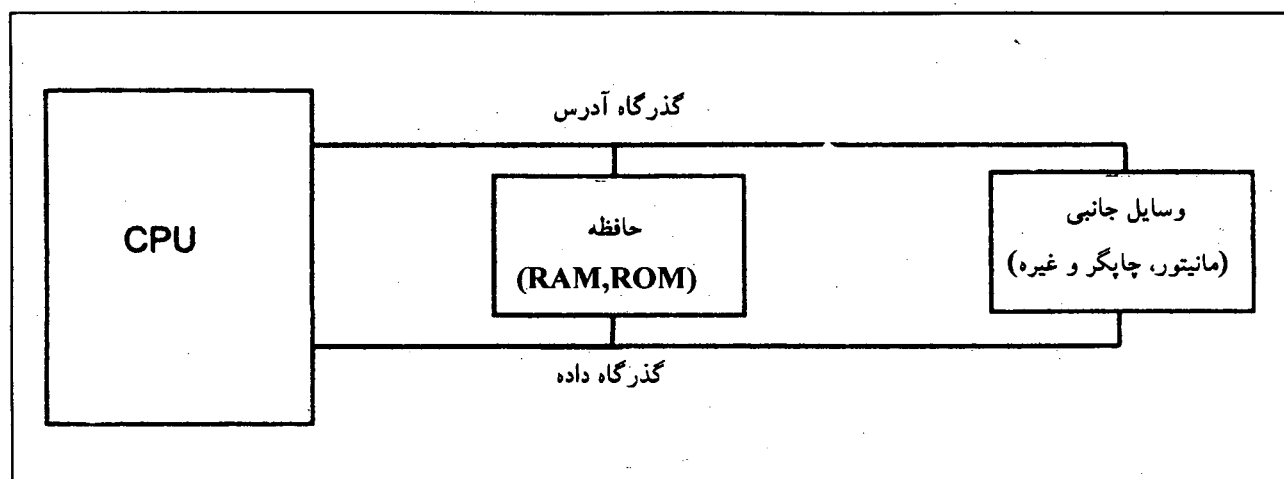
بعضی اصطلاحات مهم

یکی از امکانات مهم یک کامپیوتر حافظه موجود در آن است. بنابراین اکنون اصطلاحات بکار رفته برای اندازه حافظه در IBM PC ها و سازگار با آن‌ها را بیان می کنیم. از بحث قبل بیاد دارید که بیت یک رقم دودویی بود که می توانست مقدار 0 یا 1 داشته باشد. بایت یک مجموعه 8 بیتی است. نیل نصف یک بایت، یا ۴ بیت است. کلمه دو بایت یا 16 بیت می باشد. نمایش زیر به منظور نشان دادن اندازه نسبی این واحدها ارائه شده است. البته، آنها می توانند هر ترکیبی از صفرها و یک ها باشند.

0	بیت
0000	نیل
0000 0000	بایت
0000 0000 0000 0000	کلمه

یک کیلوبایت، 2^{10} بایت یا 1024 بایت است. اغلب از K برای بیان آن استفاده می شود. مثلاً برخی از فلاپی دیسک‌ها (یا دیسک نرم) 356K داده را نگه می دارند. یک مگابایت، یا ساده تر مگ، 2^{20} بایت است. این مقدار، کمی بیش از یک میلیون بایت است و مقدار دقیق آن 1,048,576 می باشد. با گذری سریع در ظرفیت به گیگابایت یا 2^{30} بایت می رسیم (بیش از 1 بیلیون یا میلیارد)، و یک ترابایت نیز 2^{40} بایت است (بیش از 1 تریلیون). برای مثالی از چگونگی کاربرد آنها، فرض کنید که کامپیوتری دارای 16 مگابایت حافظه باشد. این مقدار برابر با 16×2^{20} یا $2^4 \times 2^{24}$ یا 2^{24} است. بنابراین 16 مگابایت 2^{24} بایت می باشد.

در میکروکامپیوترها معمولاً از دو نوع حافظه استفاده می شود که عبارتند از RAM، که به معنی حافظه با دستیابی تصادفی است (گاهی هم حافظه خواندن | نوشتن نامیده می شود) و ROM که به معنی حافظه فقط خواندنی می باشد. RAM بوسیله کامپیوتر برای ذخیره سازی موقت برنامه‌های در حال اجرا مورد استفاده قرار می گیرد. این برنامه‌ها یا اطلاعات بعد از خاموش شدن کامپیوتر از بین می روند. به همین دلیل، RAM را گاهی حافظه فرار هم می خوانند. ROM برای برنامه‌ها و اطلاعات لازم در عملکرد کامپیوتر لازم است. اطلاعات در ROM دائمی است و قابل تعویض بوسیله کاربر نمی باشد و پس از خاموش شدن کامپیوتر هم از بین نمی رود. بنابراین آن را حافظه غیرفرار گوئیم.



شکل ۳-۰ درون کامپیوتر

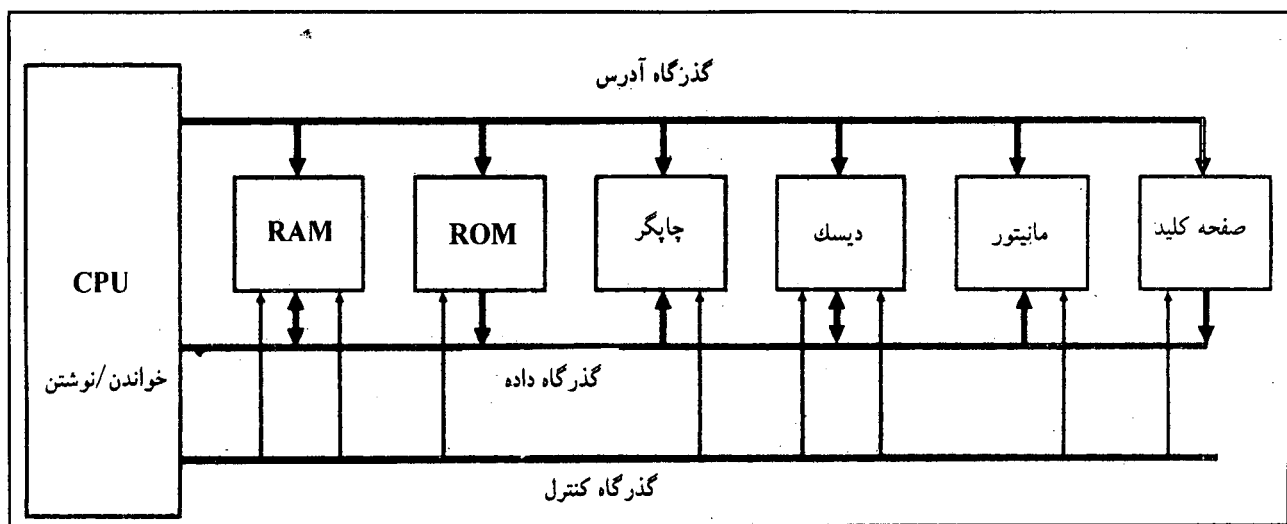
سازمان درونی کامپیوترها

بخش عملیاتی هر کامپیوتر قابل تفکیک به سه قسمت است: وسایل CPU (واحد پردازش مرکزی)، حافظه و I/O (ورودی/خروجی)، شکل ۳-۰ ملاحظه شود. نقش CPU اجرای (پردازش) اطلاعات ذخیره شده در حافظه است. عمل وسایل I/O همچون صفحه کلید، مانیتور تصویر (ویدئو)، تهیه مفاهیم ارتباط و محاوره با CPU است. CPU از طریق رشته‌ای از سیم‌ها به نام گذرگاه به حافظه و I/O متصل است. گذرگاه داخل یک کامپیوتر، درست مثل گذرگاه‌های خیابانی که مردم را از مکانی به مکانی دیگر هدایت می‌کنند، اطلاعات را از جایی به جای دیگر انتقال می‌دهند. در هر کامپیوتر سه نوع گذرگاه موجود است: گذرگاه آدرس، گذرگاه داده و گذرگاه کنترل.

به منظور شناسائی یک وسیله (حافظه یا I/O) توسط CPU، باید آدرسی به آن تخصیص داد. آدرس اختصاص یافته به یک وسیله موردنظر باید منحصر به فرد باشد؛ یعنی دو وسیله مختلف مجاز به داشتن یک آدرس نیستند. CPU آدرس را روی گذرگاه آدرس قرار می‌دهد (البته به شکل دودویی) و مدار دیکد وسیله را می‌یابد. آنگاه CPU از گذرگاه داده برای بدست آوردن داده از وسیله یا ارسال داده به آن استفاده می‌نماید. گذرگاه‌های کنترل برای تهیه سیگنال‌های خواندن و نوشتن در وسیله و مطلع ساختن آن از تصمیم CPU برای دریافت اطلاعات و یا ارسال اطلاعات به آن است. از سه گذرگاه فوق، آدرس و داده، توانمندی یک CPU را نشان می‌دهند.

توضیحی بیشتر درباره گذرگاه داده

چون گذرگاه‌های داده برای انتقال اطلاعات به و یا از CPU بکار می‌روند، هرچه گذرگاه‌های داده بیشتر باشند، CPU بهتر است. اگر گذرگاه‌های داده را همچون خطوط اتوبان تصور کنیم، واضح است که هر چه خطوط بیشتر باشند، مسیر بین CPU و وسایل بیرونی (مانند چاپگرها، RAM، ROM و غیره، شکل ۴-۰ ملاحظه شود) بهتر خواهد بود. اما افزایش در تعداد خطوط، هزینه ساخت را افزایش می‌دهد. گذرگاه‌های حافظه بیشتر، بمعنای CPU و کامپیوتر گرانتر می‌باشد. اندازه متوسط گذرگاه داده در CPUها بین 8 و 64 متغیر است، کامپیوترهای اولیه مانند Apple2 از یک گذرگاه داده 8 بیت استفاده می‌کردند، در حالی که سوپر کامپیوترهایی همچون Cray گذرگاه داده 64 بیتی را به کار می‌برند. گذرگاه‌های داده دو طرفه هستند، زیرا CPU از آنها به هنگام دریافت و یا ارسال داده استفاده می‌کند. توان پردازش CPU به اندازه این گذرگاه‌ها وابسته است، زیرا یک گذرگاه 8 بیتی هر بار قادر است 1 بایت داده را بفرستد، ولی گذرگاه 16 بیتی، 2 بایت را هر بار ارسال می‌کند که در نتیجه دو برابر سریعتر خواهد بود.



شکل ۴ - سازمان درونی کامپیوتر

توضیحی بیشتر درباره گذرگاه آدرس

چون گذرگاه آدرس برای شناسایی وسیله و حافظه متصل به CPU بکار می‌رود، هرچه گذرگاه‌های آدرس بیشتر باشند، تعداد وسایلی که آدرس‌دهی می‌شوند بیشتر خواهند بود. به بیان دیگر تعداد گذرگاه‌های آدرس برای یک CPU، تعداد مکان‌هایی را که با آن محاوره می‌کند افزایش می‌دهد. همواره تعداد مکان‌ها 2^x است که در آن x تعداد خطوط آدرس می‌باشد و ربطی به اندازه خطوط داده ندارد. مثلاً یک CPU با 16 خط آدرس می‌تواند 65536 (2^{16}) یا 64k حافظه را آدرس‌دهی کند. هر مکان حداکثر 1 بایت داده دارد. دلیلش این است که غالباً تمام ریزپردازنده‌های همه منظوره را بایت آدرس‌پذیر می‌نامند. برای مثالی دیگر، کامپیوتر IBM PC AT از یک CPU با 24 خط آدرس و 16 خط داده استفاده می‌کند. در این حالت، کل حافظه قابل دسترس، 16 مگابایت خواهد بود (مگابایت $2^{24} = 16$). در این مثال 2^{24} مکان وجود دارد، و چون هر مکان یک بایت است، 16 مگابایت حافظه موجود خواهد بود. گذرگاه آدرس یک گذرگاه یک طرفه می‌باشد، و به این معنی است که CPU از گذرگاه آدرس فقط برای ارسال آدرس به خارج از خود استفاده می‌کند. بطور خلاصه: تعداد کل حافظه‌های آدرس‌پذیر بوسیله یک CPU همیشه برابر با 2^x می‌باشد که در آن x تعداد بیت‌های آدرس است و ربطی به اندازه گذرگاه داده ندارد.

CPU و ارتباط آن با RAM و ROM

در پردازش اطلاعات بوسیله CPU، داده باید در RAM یا ROM ذخیره شود. وظیفه ROM در کامپیوترها ارائه اطلاعات ثابت و دائمی است. این اطلاعات عبارتند از، جداول برای الگوی کاراکترهای مورد نمایش روی صفحه مانیتور، یا برنامه‌هایی که در کار کامپیوتر نقش اساسی دارند، مانند برنامه‌هایی که کل RAM موجود در سیستم را می‌یابند و یا تست می‌کنند، و یا برنامه‌هایی که اطلاعات را روی مانیتور نمایش می‌دهند. برعکس، RAM برای ذخیره اطلاعاتی بکار می‌رود که غیردائمی و قابل تغییر با زمان می‌باشند، مانند انواع سیستم‌های عامل و بسته‌های کاربردی مثل بسته‌های پردازش کلمات و محاسبه مالیاتی. این برنامه‌ها برای پردازش توسط CPU در RAM قرار داده می‌شوند. CPU قادر به دریافت مستقیم اطلاعات از دیسک نیست زیرا دیسک خیلی کند است. به بیان دیگر، CPU اطلاعات مورد پردازش را از RAM یا ROM دریافت می‌نماید. در صورتی که آن را در آنجا نیابد شروع به جستجو در وسایل ذخیره سازی حجیم مانند دیسک می‌نماید، و سپس اطلاعات را به RAM منتقل می‌کند. به این دلیل، گاهی RAM و ROM را حافظه اصلی می‌نامند و دیسک‌ها نیز حافظه ثانوی خوانده می‌شوند. شکل ۵ - نمودار بلوکی سازمان داخلی یک PC را نشان می‌دهد.

درون CPU ها

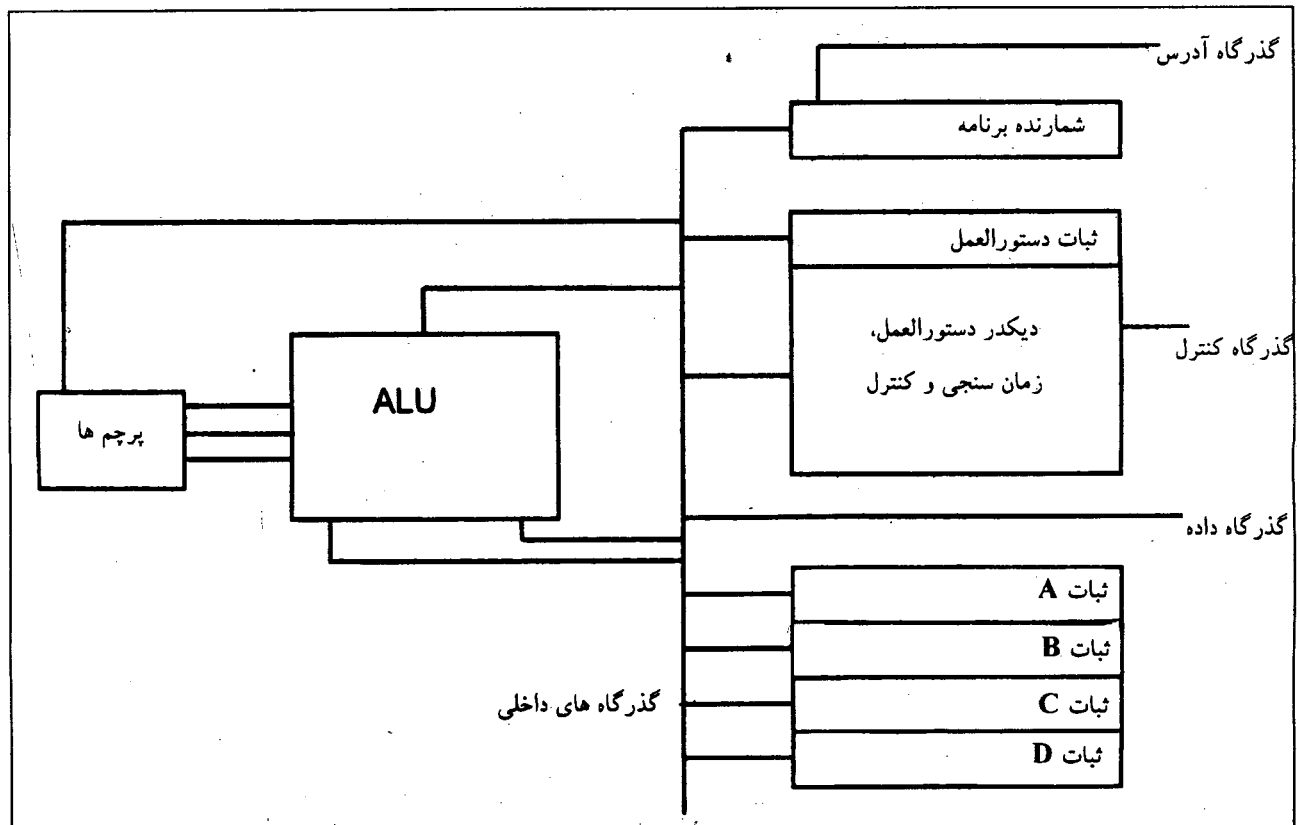
برنامه ذخیره شده در حافظه دستورالعمل‌هائی را برای CPU فراهم می‌سازد تا بر اساس آن عملی را انجام دهد. عمل می‌تواند یک جمع داده ساده همچون صورتحساب و یا کنترل یک ماشین مانند روپات باشد. برداشت این دستورات از حافظه و اجرای آنها بعهده CPU است. برای انجام اعمال برداشت و اجرا، تمام CPU ها مجهز به امکانات زیر هستند:

۱- قبل از هر چیز تعدادی ثابت در اختیار CPU قرار دارد. CPU از این ثابت‌ها برای ذخیره موقت اطلاعات استفاده می‌کند. اطلاعات می‌تواند دو مقدار مورد پردازش و یا آدرس مقدار مورد نظری باشد که باید از حافظه برداشت شود. ثابت‌های درون CPU می‌توانند 8 بیت، 16 بیت، 32 بیت، و یا حتی 64 بیت باشند. اندازه آنها به CPU بستگی دارد. بطور کلی هرچه ثابت‌ها بیشتر و بزرگتر باشند، CPU مناسب‌تر است. عیب ثابت‌های بیشتر و بزرگتر، گرانی CPU می‌باشد.

۲- CPU دارای بخشی بنام ALU (واحد حساب/منطق) است. بخش ALU در CPU مسئول انجام اعمال حسابی مانند جمع، تفریق، ضرب و تقسیم، و اعمال منطقی مانند AND، OR و NOT می‌باشد.

۳- هر CPU دارای یک شمارنده برنامه است. نقش شمارنده برنامه اشاره به آدرس دستورالعمل بعدی برای اجرا است. با اجرای هر دستورالعمل، شمارنده برنامه افزایش یافته و به آدرس دستورالعمل بعدی برای اجرا اشاره خواهد کرد. در این اشاره، محتوای شمارنده برنامه روی گذرگاه آدرس قرار گرفته و دستورالعمل موردنظر را یافته و آن را از مبدأ برداشت می‌کند. در IBM PC شمارنده برنامه را IP یا اشاره‌گر دستورالعمل می‌خوانند.

۴- نقش دیگر دستورالعمل، تفسیر دستور برداشت شده توسط CPU است. می‌توان دیگر دستورالعمل را همانند یک فرهنگ لغت تصور کرد که مفهوم هر دستورالعمل را ذخیره نموده و CPU را در برداشت قدم‌های بعدی پس از دریافت دستورالعمل هدایت می‌کند. همانطور که فرهنگ لغت با تعریف هرچه بیشتر لغات نیاز به صفحات بیشتری دارد، CPU هم در درک دستورالعمل‌های بیشتر نیاز به ترازستورهای بیشتری خواهد داشت.



شکل ۵ - نمودار بلوکی داخل یک CPU

عملیات درونی کامپیوتر

برای نمایش برخی از مفاهیم مورد بحث فوق، تحلیل قدم به قدمی از پردازش یک CPU برای جمع سه عدد در زیر داده شده است. فرض کنید که یک CPU فرضی دارای چهار ثبات با نام‌های A، B، C، D باشد. این پردازشگر دارای گذرگاه داده 8 بیتی و گذرگاه آدرس 16 بیتی است. بنابراین CPU می‌تواند به حافظه‌هایی از 0000 تا FFFFH دسترسی داشته باشد (جمعاً 10000H مکان). عملی که CPU می‌خواهد انجام دهد عبارتست از قراردادن مقدار 21 در ثبات A و سپس جمع ثبات A با مقادیر 42H و 12H. فرض کنید که کد انتقال مقدار به ثبات A برابر 10110000(B0H) و کد جمع یک مقدار به ثبات A نیز 0000 0100(04H) باشد. مراحل لازم و کد اجرای آنها برابر زیر است:

داده	کد	عمل
21H	B0H	انتقال مقدار 21H به ثبات A
42H	04H	جمع مقدار 42H با ثبات A
12H	04H	جمع مقدار 12H با ثبات A

اگر برنامه اجرایی فوق در مکان‌هایی از حافظه قرار گیرد که از 1400H شروع می‌شود، محتوای هر مکان حافظه بقرار زیر است.

آدرس حافظه	محتوای آدرس حافظه
1400	(B0) کد جابجایی یک مقدار به ثبات A
1401	(21) مقدار مورد جابجایی
1402	(04) کد جمع یک مقدار با ثبات A
1403	(42) مقدار مورد جمع
1404	(04) کد برای جمع یک مقدار به ثبات A
1405	(12) مقدار مورد جمع
1406	(F4) کد برای توقف

عملیاتی که CPU برای اجرای برنامه فوق طی می‌کند بقرار زیر است:

- ۱ - شمارنده برنامه CPU می‌تواند مقداری بین 0000 و FFFFH داشته باشد. باید 1400 را در شمارنده برنامه نشانند تا آدرس اولین دستورالعمل برای اجرا مشخص گردد. پس از بارکردن شمارنده برنامه با آدرس اولین دستورالعمل، CPU آماده اجرا است.
- ۲ - CPU، 1400H را روی گذرگاه آدرس قرارداده و آن را به خارج ارسال می‌دارد. مدار حافظه مکان را می‌یابد و در این هنگام CPU نیز سیگنال READ را فعال می‌نماید و به این ترتیب بابت مکان 1400H را از حافظه درخواست می‌کند. این موجب می‌شود تا محتوای حافظه در مکان 1400H، که B0 است، روی گذرگاه قرارگیرد و به CPU انتقال یابد.
- ۳ - CPU دستورالعمل B0 را به کمک مدار دیکد، دستورالعمل، دیکد می‌کند. پس از یافتن تعریف دستورالعمل متوجه می‌شود که باید محتوای مکان حافظه بعدی را به ثبات A در داخل CPU بیاورد. بنابراین به مدار کنترل خود فرمان اجرای دقیق آن را صادر می‌نماید. وقتی که مقدار 21H را از مکان 1401 حافظه به دورن آورد، درچه‌های ورودی همه ثبات‌ها را بجز ثبات A، می‌بندد. بنابراین مقدار 21H وقتی وارد CPU شود مستقیماً وارد ثبات A می‌گردد. پس از تکمیل یک دستورالعمل، شمارنده برنامه به آدرس دستورالعمل بعدی برای اجرا اشاره می‌کند، که در این حالت 1402 است. سپس آدرس 1402H به روی گذرگاه آدرس ارسال می‌شود تا دستورالعمل بعدی برداشت شود.

۴ - آنگاه از مکان 1402H، کد 04 را برمی دارد. پس از دیکد کردن، CPU می فهمد که باید محتوای ثبات A را با بایتی که در آدرس بعدی قرار دارد (1403) جمع کند. پس از آوردن مقدار (در این حالت 42H) به درون CPU، مقدار درون ثبات A را همراه با این مقدار به ALU برای انجام جمع تحویل می دهد. سپس نتیجه جمع را از خروجی ALU دریافت کرده و در ثبات A قرار می دهد. در این هنگام شمارنده برنامه برابر با 1404H، یعنی آدرس دستورالعمل بعدی می گردد.

۵ - آدرس 1404H روی گذرگاه آدرس قرار می گیرد و کد درون آن آدرس به داخل CPU آورده شده و سپس دیکد و اجرا می گردد. مجدداً این کد مقداری را به ثبات A می افزاید. شمارنده برنامه به 1406H اصلاح می شود.

۶ - نهایتاً، محتوای آدرس 1406 برداشت و اجرا می گردد. این دستورالعمل، HALT، به CPU می فهماند تا افزایش شمارنده برنامه را متوقف نماید. در غیاب HALT، CPU به اصلاح شمارنده برنامه ادامه داده و دستورالعمل هارا برداشت می نماید.

اکنون فرض کنید که آدرس 1403H به جای 42H، حاوی 04 باشد. CPU چگونه داده 04 را برای جمع از کد 04 تفکیک می کند؟ بخاطر آورید که برای این CPU، کد 04 به معنی انتقال یک مقدار به داخل ثبات A است. بنابراین CPU سعی بر دیکد مقدار بعدی نخواهد کرد، بلکه محتوای مکان حافظه بعدی را بدون توجه به مقدار آن بداخل ثبات A منتقل می سازد.

سوالات مروری

- ۱ - چند بایت 24 کیلوبایت است؟
- ۲ - "RAM" به معنی چیست؟ در سیستم های کامپیوتری چگونه بکار می رود؟
- ۳ - "ROM" به چه معنی است؟ در سیستم های کامپیوتری چگونه بکار می رود؟
- ۴ - چرا به RAM حافظه فرار می گویند؟
- ۵ - سه جزء مهم از یک سیستم کامپیوتر را ذکر کنید.
- ۶ - CPU به معنی چیست؟ وظیفه آن را در کامپیوتر بیان کنید.
- ۷ - سه نوع گذرگاهی را که در سیستم های کامپیوتری می توان یافت نام ببرید و بطور خلاصه نقش هر یک را بیان نمایید.
- ۸ - بیان کنید که کدامیک از گذرگاه ها یک طرفه و کدامیک دوطرفه است.
(الف) گذرگاه داده (ب) گذرگاه آدرس
- ۹ - اگر گذرگاه آدرس برای یک کامپیوتر مفروض دارای 16 خط باشد، حداکثر حافظه قابل دسترس چقدر است؟
- ۱۰ - ALU به معنی چیست؟ هدف از بکارگیری آن چه می باشد؟
- ۱۱ - ثبات ها چگونه در سیستم های کامپیوتر بکار می روند؟
- ۱۲ - شمارنده برنامه به چه منظوری بکار می رود؟
- ۱۳ - دیکدر دستورالعمل به چه منظوری بکار می رود؟

بخش ۳-۰ : تاریخچه مختصری از CPU

در دهه ۱۹۴۰، CPU با استفاده از لامپ های خلاء ساخته می شد. لامپ خلاء حجیم و مصرف الکتریسته زیادی داشت. مثلاً اولین کامپیوتر دیجیتال با مقیاس بزرگ، ENIAC، 130000 وات توان مصرفی همراه با 1500 فوت مربع فضا لازم داشت. اختراع ترانزیستورها تمام این موارد را تغییر داد. در دهه ۱۹۵۰، ترانزیستورها جایگزین لامپ ها در کامپیوترها شدند. سپس در ۱۹۵۹ اولین مدار مجتمع (IC) اختراع شد. این اختراع موجب حرکتی گردید که بسیاری از مردم آن را انقلاب دوم صنعتی می نامند. در دهه ۱۹۶۰ استفاده از IC در ساخت مدارات CPU معمول شد، تا اینکه در سال ۱۹۷۰ تمام CPU در یک تراشه IC قرار گرفت. اولین CPU قابل

کار در تراشه بوسیله شرکت اینتل در سال ۱۹۷۱ صورت گرفت. CPU را ریزپردازنده نام نهادند. اولین ریزپردازنده 4004 دارای گذرگاه داده 4 بیت بود و از 2300 ترانزیستور تشکیل می شد. در آغاز این وسیله برای یک ماشین حساب دستی طراحی شده بود ولی بزودی در کنترل کننده های چراغ های ترافیک نیز استفاده شد. پیشرفت در ساخت IC در دهه ۱۹۷۰، موجب طراحی ریزپردازنده هایی با گذرگاه داده 8 بیتی و گذرگاه آدرس 16 بیتی گردید. تا اواخر دهه ۱۹۷۰، 8080/85 اینتل یکی از پرمصرف ترین ریزپردازنده ها بود، و در هر وسیله ای از اجاق های ماکروویو گرفته تا کامپیوترهای مورد استفاده در خانه دیده می شد. در این هنگام کمپانی های دیگر بسیاری در این رقابت برای ساخت ریزپردازنده های سریعتر و بهتر وارد شدند. مهمترین آنها، شرکت موتورولا با ریزپردازنده های 6800 و 68000 بود. کامپیوترهای Apple's Macintosh از ریزپردازنده های 68000 استفاده می کرد.

CISC در مقایسه با RISC

تا اوایل دهه ۱۹۸۰، همه CPU ها، چه تک تراشه ای یا تمام برد، فلسفه CISC (کامپیوتر همراه با دستورالعمل های پیچیده) را در طراحی بکار می بردند. CISC به CPU هایی گفته می شود که در آن صدها دستورالعمل برای کارهای ممکن طراحی شده اند. طراحی CPU ها با تعداد زیادی دستورالعمل نه تنها صدها هزار ترانزیستور را بکار می برد، بلکه خود طراحی رانیز بسیار پیچیده، وقت گیر و گران می نمود. در اوایل دهه ۱۹۸۰، فلسفه جدیدی در طراحی CPU بنام RISC (کامپیوتر با مجموعه دستورات کم) بکار رفت. حامیان RISC مدعی بودند که هیچکس همه دستورات حک شده در حافظه CPU های نوع CISC را بکار نمی برد. چرا با کاهش دستورالعمل ها از صدها به حدود 40، آنها را کاراتر نکنیم، و تمام ترانزیستورهای باقیمانده را برای یالا بردن توان CPU بکار نگیریم. گرچه مفهوم RISC در دهه ۱۹۷۰ در IBM مطرح شد، ولی اولین ریزپردازنده RISC قابل کار تراشه ای بوسیله گروهی از محققان در دانشگاه کالیفرنیا در برکلی و در سال ۱۹۸۰ ساخته شد. امروزه فلسفه طراحی RISC از مرز یک آزمایش محدود برای آزمایشگاه های تحقیقاتی فزاینده رفته است. تا اواخر دهه ۱۹۸۰، بسیاری از کمپانی هایی که CPU های جدید طراحی می کردند (تک تراشه ای یا تمام برد) از فلسفه RISC استفاده می نمودند. بنظر می رسد که نهایتاً ریزپردازنده های CISC باقیمانده در کاربردهای مختلف خانواده 80x86 (8086، 8088، 80286، 80386، 80486، 80586 و غیره) و خانواده 680x0 (68000، 68010، 68020، 68030، 68040، 68050 و غیره) باشند. 80x86 به دلیل وجود حجم زیاد تولیدات IBM PC، PS، و کامپیوترهای سازگار با آنها تداوم خواهد داشت و Apple Macintosh نیز ریزپردازنده های 680x0 را تداوم خواهد بخشید.

سؤالات مروری

- ۱ - ریزپردازنده چیست؟
- ۲ - توضیح دهید که چگونه پیشرفت در تکنولوژی، اندازه، قیمت و فراوانی سیستم های کامپیوتری را موجب شد.
- ۳ - اختلاف مهم بین کامپیوترهای CISC و RISC را بیان کنید.

خلاصه

سیستم اعداد دودویی تمام اعداد را با ترکیب ارقام دودویی، یعنی 0 و 1 نشان داد. کاربرد سیستم های دودویی در کامپیوترهای دیجیتال لازم است زیرا فقط دو حالت می تواند نشان داده شود: روشن یا خاموش. هر عدد دودویی، بخاطر سهولت کار انسان، مستقیماً قابل کد شدن به معادل شانزده شانزدهی اش می باشد. تبدیل از دودویی یا شانزده شانزدهی به دهدهی و برعکس یک روند سر راست است که با ممارست، بسیار ساده خواهد

شد. کد اسکی یک کد دودویی است که برای نمایش داده‌های الفبا عددی بکار می‌رود. این کد مکرراً در وسایل جانبی برای ورودی یا خروجی بکار می‌رود.

اجزاء اصلی هر سیستم کامپیوتر عبارتند از CPU، حافظه و وسایل I/O. "حافظه" برای ذخیره‌سازی موقت یا دائمی بکار می‌رود. در بسیاری از سیستم‌ها، حافظه بصورت بایت یا کلمه قابل دسترسی است. جملات کیلوبایت، مگابایت، گیگابایت، و ترابایت برای اشاره به مقدار زیاد بایت‌ها بکار می‌رود. در سیستم‌های کامپیوتر دو نوع حافظه اصلی وجود دارد: RAM و ROM. RAM برای ذخیره موقت برنامه‌ها و داده‌هاست. ROM برای ذخیره‌سازی دائمی برنامه‌ها و داده‌هایی است که کامپیوتر برای کار خود استفاده می‌کند. همه اجزاء کامپیوتر تحت کنترل CPU هستند. وسایل جانبی مانند وسایل I/O به CPU اجازه می‌دهند تا با انسان یا دیگر کامپیوترها تبادل اطلاعات نماید. سه نوع گذرگاه در کامپیوتر وجود دارد: گذرگاه آدرس، گذرگاه داده و گذرگاه کنترل. گذرگاه کنترل برای هدایت سایر وسایل بوسیله CPU بکار می‌رود. گذرگاه آدرس برای شناسایی وسیله یا حافظه توسط CPU بکار می‌رود. گذرگاه داده برای ارسال و دریافت داده بین CPU و سایر وسایل استفاده می‌شود.

بتدریج که تغییر تکنولوژی در طراحی کامپیوتر وارد شد، ارزش و اندازه آنها بشدت کاهش یافت. کامپیوترهای اولیه بطور متوسط به اندازه یک خانه بودند و تنها در اختیار عده بخصوصی از دانشمندان قرار داشتند. اختراع ترانزیستورها و پیشرفت‌های بعدی آن در طراحی، کامپیوتر را در اختیار عموم قرار داد. بتدریج که نوآوری‌های سخت‌افزار محدودیت یافت، طراحان بدنال تکنیک جدیدی مانند معماری RISC براه افتادند، تا عملکرد کامپیوتر را اصلاح نمایند.

مسائل

- ۱ - اعداد دهدهی زیر را به دودویی تبدیل کنید.
(الف) 12 (ب) 123 (پ) 63 (ت) 128 (ث) 1000
- ۲ - اعداد دودویی زیر را به دهدهی تبدیل نمائید.
(الف) 100100 (ب) 1000001 (پ) 11101 (ت) 1010 (ث) 00100010
- ۳ - مقادیر مسئله ۲ را به مبنای 16 تبدیل کنید.
- ۴ - مقادیر شانزده شانزدهی زیر را به مبنای 10 و مبنای 2 بپسید.
(الف) 2B9H (ب) F44H (پ) 912H (ت) 2BH (ث) FFFFH
- ۵ - مقادیر مسئله ۱ را به مبنای 16 تبدیل کنید.
- ۶ - متمم 2 اعداد دودویی زیر را بدست آورید.
(الف) 1001010 (ب) 111001 (پ) 10000010 (ت) 111110001
- ۷ - مقادیر شانزده شانزدهی زیر را جمع کنید.
(الف) 2CH + 3FH (ب) F34H + 5D6H
(پ) 20000H + 12FFFH (ت) FFFFH + 2222H
- ۸ - تفریق‌های مبنای 16 زیر را انجام دهید.
(الف) 24FH - 129H (ب) FE9H - 5CCH
(پ) 2FFFF - FFFFFH (ت) 9FF25H - 4DD99H
- ۹ - کد اسکی را برای 0، 1، 2، 3، 9 در هر دو حالت دودویی و شانزده شانزدهی نشان دهید.
- ۱۰ - کد اسکی را برای رشته زیر بدست آورید (به شانزده شانزدهی):

۱۱ - موارد زیر را پاسخ دهید:

(الف) چند نیبل برابر 16 بیت است؟

(ب) چند بایت 32 بیت است؟

(پ) اگر کلمه‌ای بصورت 16 بیت تعریف شود، یک قلم داده 64 بیت چند کلمه است؟

(ت) مقدار دقیق 1 مگا (به دهدهی) چقدر است؟

(ث) چند K برابر با 1 مگا است؟

(ج) مقدار دقیق گیگا به دهدهی چقدر است؟

(چ) چند K برابر 1 گیگا است؟

(ح) چند مگا برابر با 1 گیگا است؟

(خ) اگر یک کامپیوتر مفروض دارای 8 مگابایت حافظه باشد، این حافظه به دهدهی چقدر است؟ چند کیلوبایت است؟

۱۲ - یک وسیله ذخیره سازی حجیم مانند دیسک سخت می تواند 2 گیگابایت اطلاعات را ذخیره کند. فرض کنید که هر صفحه متن دارای 25 سطر و هر سطر 80 ستون کاراکتر اسکی داشته باشد (هر کاراکتر در یک بایت)، تقریباً چند صفحه اطلاعات در این دیسک قابل ذخیره سازی است.

۱۳ - در یک کامپیوتر با حافظه بایتی، مکان های 10000H تا 9FFFFH برای برنامه های کاربر در اختیار است. اولین مکان 10000H و آخرین مکان 9FFFFH می باشد. موارد زیر را محاسبه کنید.

(الف) تعداد بایت های موجود کل (به دهدهی)

(ب) تعداد کیلوبایت های کل (به دهدهی)

۱۴ - یک کامپیوتر مفروض دارای گذرگاه داده 32 بیتی است. بزرگترین عددی که در هر لحظه به CPU می تواند حمل شود، چقدر است؟

۱۵ - در زیر چند کامپیوتر به همراه عرض گذرگاه داده آنها دیده می شود. برای هر کامپیوتر، حداکثر مقداری که هر بار به داخل CPU می رود چقدر است (به شانزده شانزدهی و دهدهی)

(الف) Apple2 با گذرگاه داده 8 بیتی

(ب) IBM PS/2 با گذرگاه داده 16 بیتی

(پ) IBM PS/2 مدل 80 با گذرگاه داده 32 بیتی

(ت) سوپر کامپیوتر CRAY با گذرگاه داده 64 بیتی

۱۶ - حافظه کل را بر حسب واحد مورد نظر در هریک از CPU های زیر برای گذرگاه های داده شده بیابید.

(الف) گذرگاه 16- بیت (به K)

(ب) گذرگاه آدرس 24- بیت (به مگا)

(پ) گذرگاه آدرس 32- بیت (به مگابایت، گیگابایت)

(ت) گذرگاه آدرس 48- بیت (به مگا بایت، گیگابایت و ترابایت)

۱۷ - کدامیک از گذرگاه ها، یعنی گذرگاه داده و گذرگاه آدرس یک طرفه و کدامیک دوطرفه است؟

۱۸ - کدامیک از ثبات های CPU آدرس دستورالعمل مورد برداشت را نگهدارند؟

۱۹ - کدام بخش CPU مسئول اجرای جمع است؟

۲۰ - کدام نوع CPU (CISC یا RISC) دارای تعداد بیشتری دستورالعمل است؟

بخش ۱-۰: اعداد و سیستم‌های کدبندی

۱ - کامپیوترها از سیستم دودویی استفاده می‌کنند زیرا هر بیت می‌تواند یکی از دو سطح ولتاژ را داشته باشد: روشن و خاموش.

بخش ۲-۰: درون کامپیوتر

۱ - 24,576

۲ - حافظه با دستیابی تصادفی؛ این حافظه برای ذخیره‌سازی موقت برنامه‌هایی که CPU آن‌ها را اجرا می‌کند، مانند سیستم‌های عامل و برنامه‌های پردازش کلمات و غیره بکار می‌رود.

۳ - حافظه فقط خواندنی؛ این حافظه برای ذخیره‌سازی دائمی برنامه‌ها مانند کنترل صفحه کلید و غیره بکار می‌رود.

۴ - هنگامی که کامپیوتر را خاموش کنیم محتوای RAM از بین می‌رود.

۵ - CPU، حافظه و وسایل I/O

۶ - واحد پردازش مرکزی؛ می‌توان آن را مغز کامپیوتر تصور کرد، که برنامه‌ها را اجرا نموده و همه وسایل را در کامپیوتر کنترل می‌نماید.

۷ - گذرگاه آدرس، مکان (آدرس) لازم برای CPU را حمل می‌کند؛ گذرگاه داده اطلاعات را به و یا از CPU حمل می‌نماید؛ گذرگاه کنترل بوسیله CPU برای ارسال سیگنال‌های کنترل وسایل I/O مورد استفاده قرار می‌گیرد.

۸ - (الف) دوطرفه (ب) یک طرفه

۹ - 64K یا 65536 بایت

۱۰ - واحد حساب / منطق؛ تمام عملیات حسابی منطقی را انجام می‌دهد.

۱۱ - برای ذخیره موقت اطلاعات

۱۲ - آدرس دستورالعمل بعدی مورد اجرا را نگهداری می‌کند.

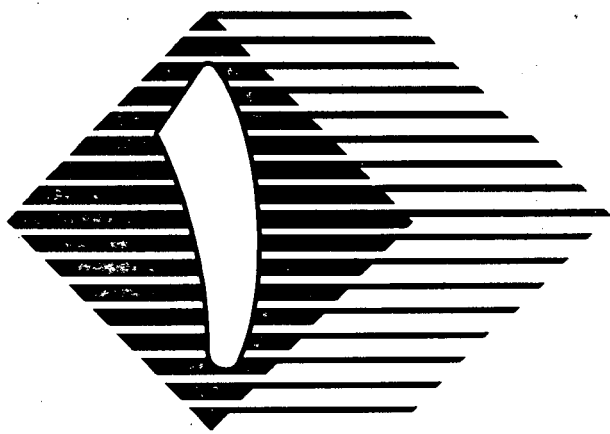
۱۳ - به CPU می‌گوید برای هر دستور چه مرحله‌ای را انجام دهد.

بخش ۳-۰: تاریخچه مختصری از CPU

۱ - یک CPU در یک تراشه

۲ - گذر از لامپ‌های خلاء به ترانزیستورها و به IC اندازه و قیمت کامپیوترها را کاهش داد، بنابراین بیشتر در دسترس قرار گرفتند.

۳ - کامپیوترهای CISC دستورالعمل‌های زیادی بکار می‌برند در صورتی که کامپیوترهای RISC دستورات کمتری دارند.



ریزپردازنده 80x86

اهداف

در پایان این فصل قادر خواهید بود:

- « انواع ریزپردازنده های خانواده اینتل از 8085 تا 80486 را برحسب سائز گذرگاه، حافظه فیزیکی و امکانات ویژه آنها توصیف نمایید.
- « نقش EU (واحد اجرا) و BIU (واحد اتصال گذرگاه) را بدانید.
- « تکنیک خط لوله را توصیف کرده و بگویید چگونه این تکنیک سرعت CPU را بالا می برد.
- « ثبات های 8086 را لیست نمایید.
- « دستورات ساده MOV و ADD را کد نموده و اثر آنها را روی عملوندهایشان توصیف نمایید.
- « هدف از قطعه کد، قطعه داده، قطعه پشته و قطعه اضافی را بیان کنید.
- « تفاوت بین یک آدرس منطقی و آدرس فیزیکی را توضیح دهید.
- « قرار داد مبتنی بر کوچکتر در ریزپردازنده های 80x86 را توصیف نمایید.
- « هدف از پشته را بدانید.
- « کار دستورات PUSH و POP را بگویید.
- « بیت های ثبات پرچم را لیست کرده و بطور خلاصه نقش هر بیت را بیان نمایید.
- « اثر دستور ADD را بر روی ثبات پرچم نشان دهید.
- « روش های آدرس دهی 8086 را لیست کرده و متوالی را برای هر روش شناسایی کنید.

این فصل با تاریخچه تکامل ریزپردازنده‌های خانواده اینتل آغاز می‌شود. بخش دوم مروری بر واحدهای کاری ریزپردازنده‌های 80x86 است. مقدمه‌ای بر برنامه‌نویسی به زبان اسمبلی در بخش سوم داده شده است. بخش‌های چهار و پنج قطعات برنامه‌های زبان اسمبلی و چگونگی تولید آدرس‌های فیزیکی را پوشش می‌دهد. نهایتاً در آخرین بخش روش‌های آدرس‌دهی 80x86 مشروحاً توضیح داده شده است.

بخش ۱-۱: تاریخچه خلاصه‌ای از خانواده 80x86

در این بخش ما تکامل ریزپردازنده‌های خانوار اینتل را از اواخر دهه ۱۹۷۰ پی می‌گیریم. در آن زمان هنوز کامپیوتر شخصی بطور گسترده‌ای پذیرفته نشده بود. این بررسی تا ریزکامپیوترهای قوی امروزی ادامه خواهد یافت.

تکامل از 8080/8085 تا 8086

در ۱۹۷۸ شرکت اینتل یک ریزپردازنده 16 بیتی بنام 8086 را معرفی کرد. این پردازنده از جهات مختلف دارای اصلاحات عمده‌ای نسبت به سری 8080/85 داشت. یکی اینکه 8086 قادر بود 1 مگابایت حافظه را پشتیبانی کند در حالی که 8080/85 تنها 64K حافظه داشت. دوم اینکه 8080/85 یک سیستم 8 بیتی بود، به این معنی که ریزپردازنده در هر لحظه فقط می‌توانست با داده‌های 8 بیتی کار کند. داده‌های بزرگتر از 8 بیت باید به دو بخش 8 بیت شکسته می‌شدند تا بوسیله CPU قابل پردازش باشند. در مقابل 8086 یک پردازنده 16 بیت بود. سوم، 8086 یک پردازنده خط لوله‌ای بود در حالی که 8080/85 بصورت غیر خط لوله‌ای ساخته شده بود. در یک سیستم خط لوله‌ای گذرگاه‌های داده و آدرس، بهنگام پردازش اطلاعات توسط CPU، بطور همزمان مشغول انتقال داده می‌باشند. به این ترتیب توان پردازشی ریزپردازنده بالا می‌رود. گرچه تکنیک خط لوله از ویژگی‌های مرسوم در کامپیوترهای مینی و بزرگ بود، اینتل پیشنهاد قرار دادن خط لوله در ریزپردازنده‌های تک تراشه بوده است.

تکامل از 8086 تا 8088

8086 ریزپردازنده‌ای با گذرگاه داخلی و خارجی داده 16 بیت است، این بدان معنی است که همه ثبات‌ها 16 بیت عرض دارند و برای انتقال داده به داخل یا خارج آن هم گذرگاه داده 16 بیت است. گرچه معرفی 8086 پیشرفت‌های عمده نسبت به نسل‌های قدیمی‌تر را نشان می‌داد، با این وجود مقاومت‌هایی در استفاده از گذرگاه داده خارجی 16 بیت وجود داشت، زیرا در آن زمان همه وسایل جانبی حول ریزپردازنده‌های 8 بیتی ایجاد شده بودند. بعلاوه یک مدار چاپی با گذرگاه داده 16 بیت بسیار گرانتر بود. بنابراین اینتل 8088 را ساخت. از نظر برنامه‌نویسی، این پردازنده با 8086 سازگار است ولی در بیرون، این پردازنده در عوض گذرگاه داده 16 بیتی، از گذرگاه 8 بیتی استفاده می‌کند. ظرفیت حافظه آن هم 1 مگابایت است.

موفقیت 8088

در سال ۱۹۸۱ شانس به اینتل روی آورد به این ترتیب که شرکت IBM، ریزپردازنده 8088 را برای طراحی IBM PC انتخاب کرد. IBM PC مبتنی بر 8088 موفقیت قابل توجهی داشت. این موفقیت بیشتر به این علت بود که IBM و میکروسافت (سازنده سیستم عامل MS-DOS) آن را بصورت یک سیستم باز درآوردند، به این معنی که مدارک و خصوصیات سخت‌افزار و نرم‌افزار PC در اختیار عام قرار گرفت. این خود سبب شد تا دیگر سازندگان، سخت‌افزار را با موفقیت تولید کرده و موجب رشد زیادی در سخت‌افزار و نرم‌افزار طرح‌های مبتنی بر IBM PC گردیدند. در مقابل کامپیوتر Apple بصورت بسته طراحی شد و هرگونه تلاش برای مشابه‌سازی دیگر سازندگان محلی و خارجی بلوکه شده بود.

دیگر ریزپردازنده‌ها: 80286 و 80386 و 80486

با موفقیت‌هایی که اینتل پشت‌سر گذاشته بود و نیاز کاربران PC برای ریزپردازنده‌های قوی‌تر، اینتل در سال

۱۹۸۲ ریزپردازنده 80286 را معرفی کرد. ویژگی‌های آن عبارت بودند از گذرگاه‌های درونی و بیرونی 16 بیتی؛ 24 خط آدرس، که 16 مگابایت ($2^{24}=16MB$) حافظه را میسر می‌سازد، و از همه بااهمیت‌تر حافظه مجازی بود. 80286 در یکی از دو حالت مد واقعی و مد حفاظت شده کار می‌کند. مد واقعی در واقع یک 8088/86 سریع است که حداکثر به 1 مگابایت حافظه می‌تواند دسترسی داشته باشد. مد حفاظت شده اجازه می‌دهد تا 16M بایت حافظه بکار گرفته شود ولی از نظر تخریب‌های عمدی یا سهوی کاربر تحت‌الحفظ سیستم عامل می‌باشد. این ویژگی در 8088/86 وجود نداشت. حافظه مجازی راهی است که ریزپردازنده بر اساس آن در مورد داشتن حافظه‌ای نامحدود گول‌زده می‌شود. به این ترتیب که داده بین دیسک و RAM ردوبدل می‌گردد. IBM ریزپردازنده 80286 را برای طراحی PC AT بکار برد و دیگر مشابه‌سازان، IBM را دنبال کردند.

با تقاضای بیشتر کاربران برای سیستم قوی‌تر، اینتل در سال ۱۹۸۵ ریزپردازنده 80386 را معرفی کرد (گاهی آن را 80386 DX هم می‌گویند). که در درون و بیرون یک ریزپردازنده 32 بیتی بود و 32 بیت هم گذرگاه آدرس داشت. این پردازنده می‌توانست تا 4 گیگابایت (2^{32}) حافظه داشته باشد. حافظه مجازی تا 64 ترابایت (2^{46}) افزایش یافت. همه ریزپردازنده‌هایی که تا آن زمان مورد مطالعه قرار گرفتند ریزپردازنده‌های همه‌منظوره بودند و نمی‌توانستند محاسبات ریاضی را مکرراً انجام دهند. به این دلیل اینتل تراشه‌های پردازش داده عددی را که پردازنده کمکی نام گرفتند معرفی کرد. از جمله آنها می‌توان از 8087، 80287، و 80387 نام برد. بعداً اینتل 386SX، که در درون مشابه 80386 و در بیرون گذرگاه داده 16 بیتی و آدرس 24 بیتی داشت و می‌توانست تا 16 مگابایت (2^{24}) حافظه داشته باشد، را معرفی کرد. این امکانات سیستم 386SX را خیلی ارزان کرد. با معرفی 486 در سال ۱۹۸۹، اینتل نوع بسیار پیشرفته 386 و پردازنده کمکی محاسباتی و امکانات دیگری مانند حافظه نهان را در یک تراشه جای داد، حافظه نهان نوعی RAM ایستا (استاتیک) با سرعت دستیابی زیاد است.

در سال ۱۹۹۲ اینتل پنتیوم را معرفی کرد. در زمان معرفی، پنتیوم سرعت‌های 60 و 66 MHZ را داشت، ولی ویژگی‌های طراحی جدید اجازه داد تا سرعت آن دو برابر 80486 با سرعت 66 MHZ گردد. گرچه پنتیوم دارای گذرگاه داده 64 بیتی است ولی ثبات‌ها در آن 32 بیتی‌اند و با 32 بیت گذرگاه آدرس، قادرست تا 4 گیگابایت حافظه را دارا باشد. در سال ۱۹۹۵ اینتل پنتیوم پرو را عرضه کرد که نسل ششم خانواده x86 بود، که پردازنده نوع پیشرفته پنتیوم را داراست. جدول ۱-۱ تکامل ریزپردازنده‌های اینتل را نشان می‌دهد. باید توجه داشت که همه برنامه‌های نوشته شده در 8086/88 روی کامپیوترهای 286، 386، 486، پنتیوم و پنتیوم پرو قابل اجرا هستند.

جدول ۱-۱ تکامل ریزپردازنده‌های اینتل

تولید	8085	8086	8088	80286	80386	80486	پنتیوم	پنتیوم پرو
سال معرفی	1976	1978	1979	1982	1985	1989	1992	1995
سرعت ساعت (MHZ)	3-8	5-10	5-8	6-16	16-33	25-50	60,66	150
تعداد ترانزیستورها	6500	29,000	29,000	130,000	275,000	1.2 میلیون	3.1 میلیون	5.5 میلیون
حافظه فیزیکی	64K	1M	1M	16M	4G	4G	4G	64G
گذرگاه داده درونی	8	16	16	16	32	32	32	32
گذرگاه داده بیرونی	8	16	8	16	32	32	64	64
گذرگاه آدرس	16	20	20	24	32	32	32	36
نوع داده (بیت)	8	8,16	8,16	8,16	8,16,32	8,16,32	8,16,32	8,16,32

! توجه:

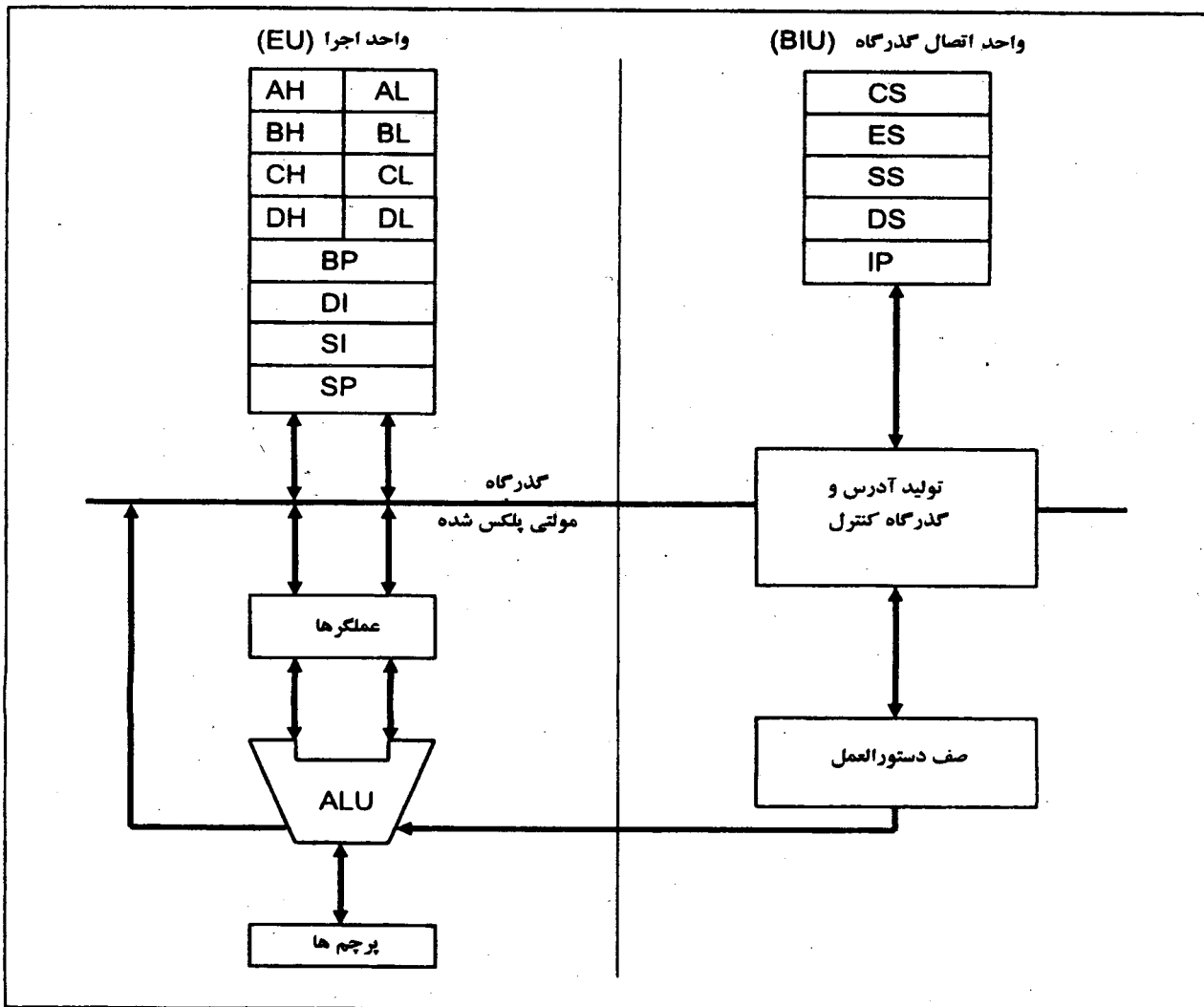
۱- معماری 80386SX مشابه 80386 است با این تفاوت که گذرگاه داده بیرونی در SX، بجای 32 بیت، 16 بیت است، و گذرگاه آدرس هم بجای 32 بیت، 24 بیت می‌باشد؛ بنابراین حافظه فیزیکی 16MB خواهد شد.

۲- سرعت ساعت از سرعت اولیه تا سرعت فعلی را شامل می‌شود. بعضی از خصوصیت‌ها در این فاصله بهتر شده‌اند.

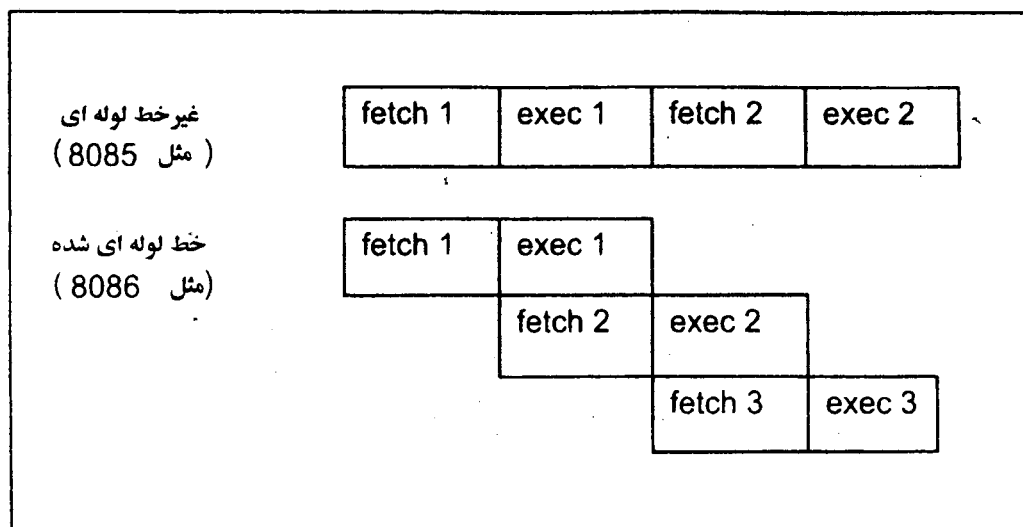
- ۱- سه ویژگی اصلاح شده 8086 را نسبت به 8080/85 نام ببرید.
- ۲- تفاوت عمده میان ریزپردازنده‌های 8088 و 8086 چیست؟
- ۳- ساینز گذرگاه آدرس و ظرفیت حافظه را برای پردازنده‌های زیر مشخص نمایید:
(الف) 8086 (ب) 80286 (پ) 80386
- ۴- 80286 یک ریزپردازنده بیتی است، در صورتی که 80386 یک ریزپردازنده می‌باشد.
- ۵- تفاوت عمده بین 80386 و 80386SX را بیان کنید.
- ۶- ویژگی‌های اضافی معرفی شده در 80286 نسبت به 8086 چیست؟
- ۷- ویژگی‌های اضافی 80486 را در برابر 80386 لیست کنید.

بخش ۲-۱: درون 8088/86

در این بخش ما مفاهیم مهمی را برای عملیات 8088/86 مانند عملیات خط لوله و ثبات‌ها می‌شکافیم. نمودار شکل ۱-۱ را ملاحظه کنید.



شکل ۱-۱. نمودار بلوکی درون CPU 8088/86



شکل ۱-۲. اجرای خط لوله‌ای در برابر غیر خط لوله‌ای

خط لوله‌گی

برای تسریع در پردازش اطلاعات CPU دو راه وجود دارد: افزایش فرکانس کاری یا تغییر در معماری داخلی CPU. انتخاب اول به تکنولوژی وابسته است، به این معنی که طراح باید از هر نوع تکنولوژی که در آن زمان موجود است استفاده کند و ضمناً قیمت را هم در نظر بگیرد. تکنولوژی و مواد بکار رفته در ساخت IC (مدار مجتمع) تعیین کننده فرکانس کاری، توان مصرفی و تعداد ترانزیستور بسته‌بندی شده در تراشه یک ریزپردازنده است. بحث مفصل در مورد تکنولوژی IC و رای دیدگاه این کتاب است. در اینجا فقط کافی است که طراح برای بالا بردن سرعت CPU فرکانس را تا جایی که تکنولوژی و قیمت اجازه بدهد بالا ببرد. انتخاب دوم در اصلاح قدرت پردازش CPU در رابطه با کار درونی CPU است. در ریزپردازنده 8085، CPU فقط می‌تواند در هر لحظه یک کد را برداشت و یا اجرا کند. به بیان دیگر، CPU باید دستوری را از حافظه بردارد، سپس اجرا کند، و باز دوباره برداشت کند و اجرا نماید و به همین ترتیب. ایده خط لوله در ساده‌ترین شکل به CPU اجازه می‌دهد تا به طور همزمان طبق شکل ۱-۲ عمل برداشت و اجرا را انجام دهد. توجه کنید که در شکل ۱-۲ زمان برداشت و اجرا یکی نیستند.

اینترنت مفهوم خط لوله را در 8088/86 با تفکیک ساختار درونی ریزپردازنده به دو بخش تحقق بخشید: واحد اجرا (EU) و واحد ارتباط (اتصال) گذرگاه (BIU). این دو بخش بطور همزمان کار می‌کنند. BIU حافظه و وسایل جانبی را دستیابی می‌کند در حالی که EU دستورات برداشت شده قبلی را اجرا می‌نماید. این هنگامی درست عمل خواهد کرد که BIU از EU جلوتر باشد؛ بنابراین BIU در 8088/86 دارای بافر یا صف است (شکل ۱-۱ ملاحظه شود). بافر در 8088 دارای 4 بایت و در 8086، 6 بایت طول دارد. اگر اجرای دستوری طول بکشد صف تا حداکثر ظرفیتش پر می‌شود و گذرگاه‌ها موقتاً بیکار می‌مانند. هرگاه در 8086 صف به میزان 2 بایت و در 8088 به میزان 1 بایت خالی شد BIU عمل برداشت را انجام می‌دهد. در بعضی موارد ریزپردازنده باید صف را تخلیه کند. مثلاً وقتی یک دستور پرش اجرا شود، BIU شروع به برداشت اطلاعات از مکان جدید حافظه نموده و اطلاعات قبلی برداشت شده در صف حذف می‌شوند. در این وضعیت EU باید برای برداشت دستور جدید بوسیله BIU منتظر بماند. در اصطلاحات کامپیوتر به این انتظار جریمه اشعاب گفته می‌شود. بنابراین در یک CPU خط لوله‌ای، پرش‌های بی‌رویه راندمان برنامه را کاهش خواهد داد. خط لوله در 8088/86 دارای دو مرحله برداشت (واکشی) و اجرا است. در کامپیوترهای قوی‌تر خط لوله ممکن است دارای مراحل متعددی باشد. در سال‌های اخیر ترکیب مفهوم خط لوله با افزایش پایه‌های گذرگاه داده منجر به طراحی ریزپردازنده‌های بسیار قوی شده است.

ثبات‌ها

AX ثبات 16 بیت	
AH ثبات 8 بیت	AL ثبات 8 بیت

در CPU، ثبات‌ها برای ذخیره موقت اطلاعات بکار می‌روند. این اطلاعات می‌توانند یک یا دو بایت داده و یا آدرس داده باشند. ثبات‌های 8088/86 به شش دسته، طبق جدول ۱-۲، تقسیم می‌شوند. در 8088/86 ثبات‌های همه منظوره می‌توانند بصورت ثبات‌های 8 یا 16 بیتی مورد استفاده قرار بگیرند. دیگر ثبات‌ها همگی بصورت کامل 16 بیت در

دسترس هستند. در 8088/86، نوع داده‌ها 8 یا 16 بیت است. مثلاً برای دستیابی به داده 12 بیتی باید از ثبات 16 بیتی استفاده کرد و در آن 4 بیت بالا را صفر قرار داد. بیت‌های یک ثبات مطابق شکل زیر شماره‌گذاری می‌شوند.

ثبات 8 بیت:

D7	D6	D5	D4	D3	D2	D1	D0

ثبات 16 بیت:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

در 8088/86 ثبات‌های مختلف برای اعمال متفاوتی بکار گرفته می‌شوند، و چون بعضی دستورات فقط از ثبات‌های خاصی استفاده می‌کنند تا کار خود را انجام دهند، استفاده از ثبات‌ها برحسب دستورات و کاربرد آن‌ها در قالب یک برنامه مفروض توصیف می‌گردد. اولین حرف هر ثبات همه منظوره کاربرد آن را مشخص می‌کند. AC برای انبار، BX بعنوان ثبات آدرس پایه، CX بعنوان شمارنده در عملیات حلقه‌ای، و DX برای اشاره به داده در عملیات I/O نام‌گذاری شده‌اند.

جدول ۱-۲. دسته‌بندی ثبات‌های 8086/286

دسته	بیت‌ها	نام ثبات‌ها
AX, BX, CX, DX	16	عمومی
AH, AL, BH, BL, CH, CL, DH, DL	8	
SP (اشاره گر پشته)، BP (اشاره گر پایه)	16	اشاره گر
SI (اندیس مبدأ)، DI (قطعه داده)	16	اندیس
CS (قطعه کد)، DS (قطعه داده)	16	قطعه
SS (قطعه پشته)، ES (قطعه اضافی)	16	
IP (اشاره گر دستور)	16	دستور
FR (ثبات پرچم)	16	پرچم

! توجه: ثبات‌های همه منظوره می‌توانند بصورت کامل 16 بیت (مانند AX) یا فقط با بایت بالایی (AH) و یا بایت پایینی (AL) دستیابی شوند.

سؤالات مروری

- ۱- وظیفه EU و BIU را توضیح دهید.
- ۲- خط لوله چیست و چگونه موجب افزایش سرعت CPU می‌گردد.
- ۳- ثبات‌های 8086 ... بیت یا ... بیت هستند.
- ۴- ثبات‌های 16 بیتی 8086 را لیست کنید.

بخش ۳-۱: مقدمه‌ای بر برنامه‌نویسی اسمبلی

گرچه CPU به دودویی کار می‌کند ولی می‌تواند اعمال را با سزعت بسیار بالایی انجام دهد. با این وجود برای انسان برنامه‌نویسی یک کامپیوتر با 0 و 1 کاری طاقت‌فرسا است. برنامه‌ای که از 0ها و 1ها ساخته شده باشد زبان ماشین نام دارد، و در روزگار اولیه پیدایش کامپیوتر، برنامه‌نویسها در واقع برنامه‌ها را به زبان ماشین می‌نوشتند. گرچه سیستم مبنای شانزده به عنوان راه کاراتری برای نمایش اعداد دودویی بکار رفت، فرآیند کار با کد ماشین هنوز برای انسان خسته‌کننده بود. نهایتاً زبان‌های اسمبلی بوجود آمده‌اند و برای دستورات کد ماشین، نماد رمزی (خلاصه) را همراه با امکانات دیگری که برنامه‌نویسی را سریع و کم خطا می‌نمود، فراهم کردند. کلمه نماد رمزی بارها در مقالات علوم و مهندسی کامپیوتر برای اشاره به کدها و خلاصه‌سازی، برای به خاطر سپردن آنها به کار رفته است. برنامه‌های اسمبلی نوشته شده در این سبک باید بوسیله برنامه‌ای بنام اسمبلر به کد ماشین ترجمه شوند. زبان اسمبلی به یک زبان سطح پایین معروف است زیرا مستقیماً با ساختار درونی CPU سروکار دارد. برای برنامه‌نویسی به زبان اسمبلی برنامه‌نویس باید تعداد ثبات‌ها و سائز آنها همراه با جزئیات CPU را بداند.

امروزه هر کس می‌تواند زبان‌های برنامه‌نویسی متعددی، همچون Pascal، BISC، C و نظایر آنها را بکاربرد. این برنامه‌ها را زبانهای سطح بالا می‌گویند زیرا برنامه‌نویس نیازی به دانستن جزئیات درون CPU ندارد. در حالی که برای ترجمه یک برنامه بزبان اسمبلی به کد ماشین، از اسمبلر استفاده می‌شود، (گاهی آن را کد مقصود می‌نامند)، زبان‌های سطح بالا بوسیله کامپایلر به کد ماشین تبدیل می‌شوند. مثلاً برای نوشتن برنامه‌ای در C، باید از کامپایلر C برای ترجمه برنامه به زبان ماشین استفاده کرد.

برای ترجمه برنامه‌های زبان اسمبلی 80x86 به کد ماشین، اسمبلرهای بسیاری در دسترسند. یکی از متداول‌ترین اسمبلرها، MASM ساخت میکروسافت می‌باشد که در فصل ۲ معرفی شده است. این فصل بنحوی ارائه شده تا به پیوست (الف) مرتبط باشد: در پیوست (الف) برنامه‌نویسی بکمک DEBUG آورده شده است. اگر چنانچه با DEBUG آشنا نیستید به پیوست (الف) مراجعه کنید. برنامه کمکی DEBUG به همراه سیستم عامل DOS ارائه می‌شود و لذا به سادگی در دسترس است.

برنامه‌نویسی بزبان اسمبلی

یک برنامه زبان اسمبلی شامل مجموعه‌ای از خطوط دستورات زبان اسمبلی و دیگر چیزهاست. یک دستور زبان اسمبلی شامل نماد رمزی است که بطور اختیاری با یک یا دو عملوند دنبال می‌شود. عملوندها اقلام داده‌ای هستند که باید دستکاری شوند و نمادهای رمزی فرمانهای CPU می‌باشند که به CPU می‌گویند با این اقلام داده چه بکند. ما زبان برنامه‌نویسی اسمبلی را با دو دستور پر مصرف MOV (انتقال) و ADD (جمع) آغاز می‌کنیم.

دستور MOV

ساده بگویم، دستور MOV داده را از یک مکان به مکان دیگر کپی می‌کند. این دستور دارای قالب زیر است:

MOV destination,source ;copy source operand to destination

این دستور به CPU فرمان می‌دهد تا عملوند مبدأ را به عملوند مقصد منتقل سازد (در واقع آن را کپی می‌کند). مثلاً دستور "MOV DX,CX" محتوای ثبات CX را در DX کپی می‌کند. پس از اجرای این دستور، ثبات DX همان مقدار CX را خواهد داشت. دستور MOV عملوند مبدأ را تغییر نمی‌دهد. برنامه‌ریز ابتدا، CL را با 55H بار کرده و سپس این مقدار را به ثبات‌های مختلف داخل CPU منتقل می‌کند.

MOV	CL,55H	;move 55H into register CL
MOV	DL,CL	;copy the contents of CL into DL (now DL=CL=55H)
MOV	AH,DL	;copy the contents of DL into AH (now AH=DL=55H)
MOV	AL,AH	;copy the contents of AH into AL (now AL=AH=55H)
MOV	BH,CL	;copy the contents of CL into BH (now BH=CL=55H)
MOV	CH,BH	;copy the contents of BH into CH (now CH=BH=55H)

استفاده از ثبات‌های 16 بیتی در زیر نشان داده شده است.

MOV	CX,468FH	;move 468FH into CX (now CH=46,CL=8F)
MOV	AX,CX	;copy contents of CX to AX (now AX=CX=468FH)
MOV	DX,AX	;copy contents of AX to DX (now DX=AX=468FH)
MOV	BX,DX	;copy contents of DX to BX (now BX=DX=468FH)
MOV	DI,BX	;now DI=BX=468FH
MOV	SI,DI	;now SI=DI=468FH
MOV	DS,SI	;now DS=SI=468FH
MOV	BP,DI	;now BP=DI=468FH

در CPU 8086، داده می‌تواند در میان همه ثبات‌ها طبق جدول ۱-۲ (بجز ثبات پرچم) جابجا شود. بشرط اینکه ثبات‌های مبدأ و مقصد از نظر سائز یکی باشند. نوشتن کدی مانند "MOV AL,DX" خطا تولید می‌کند زیرا نمی‌توان محتوای یک ثبات 16 بیتی را در یک ثبات 8 بیت ذخیره کرد. مستثنی شدن ثبات پرچم به این معنی است که دستوری همچون "MOV FR,AX" وجود ندارد. بارکردن ثبات پرچم به طریقی دیگر انجام می‌شود و در فصل‌های بعدی توضیح داده شده است.

اگر داده بتواند در میان همه ثبات‌ها از جمله ثبات‌های قطعه انتقال یابد، آیا داده مستقیماً به همه آنها وارد شود؟ پاسخ منفی است. داده فقط با استفاده از دستور MOV مستقیماً می‌تواند به ثبات‌های غیرقطعه‌ای برود. مثلاً، به دستورات زیر نگاه کنید تا ببینید کدام مجاز و کدام غیرمجاز است.

MOV	AX,58FCH	;move 58FCH into AX	(LEGAL)
MOV	DX,6678H	;move 6678H into DX	(LEGAL)
MOV	SI,924BH	;move 924B into SI	(LEGAL)
MOV	BP,2459H	;move 2459H into BP	(LEGAL)
MOV	DS,2341H	;move 2341H into DS	(ILLEGAL)
MOV	CX,8876H	;move 8876H into CX	(LEGAL)
MOV	CS,3F47H	;move 3F47H into CS	(ILLEGAL)
MOV	BH,99H	;move 99H into BH	(LEGAL)

از بحث فوق سه نکته زیر نتیجه می‌شود.

۱- مقادیر نمی‌توانند به هر ثبات قطعه‌ای مثل CS، DS، ES یا SS بار شوند. برای بارکردن یک ثبات قطعه ابتدا باید آن را وارد ثبات غیرقطعه‌ای کرد و سپس آن را بداخل ثبات قطعه فرستاد، مثل زیر:

MOV	AX,2345H	;load 2345H into AX
MOV	DS,AX	;then load the value of AX into DS
MOV	DI,1400H	;load 1400H into DI
MOV	ES,DI	;then move it into ES, now ES=DI=1400

۲- اگر مقدار انتقال یافته به یک ثبات 16 بیتی کمتر از FFH باشد بقیه ثبات صفر فرض می‌شود. مثلاً در "MOV BX,5" نتیجه BX = 0005 خواهد شد که در آن BH = 00 و BL = 05 است.

۳- انتقال مقدار بزرگ در یک ثبات، تولید خطا می‌کند.

MOV	BL,7F2H	;ILLEGAL: 7F2H is larger than 8 bits
MOV	AX,2FE456H	;ILLEGAL: the value is larger than AX

دستور ADD

دستور ADD دارای قالب زیر است:

ADD destination,source ;ADD the source operand to the destination

دستور ADD به CPU فرمان می‌دهد تا عملوندهای مبدأ و مقصد را جمع کرده و حاصل را در مقصد قرار دهد. برای جمع دو عدد مانند 25H و 34H، هر کدام به یک ثبات انتقال یافته، سپس با هم جمع می‌شوند.

```
MOV AL,25H      ;move 25 into AL
MOV BL,34H      ;move 34 into BL
ADD AL,BL       ;AL = AL + BL
```

اجرای برنامه فوق منجر به $AL = 59H$ ($25H + 34H = 59H$) و $BL = 34H$ منجر می‌شود. توجه کنید که محتوای BL تغییر نمی‌کند. برنامه فوق بسته به ثبات‌های بکار رفته بطرق مختلف نوشته می‌شود. راه دیگر ممکن است چنین باشد:

```
MOV DH,25H      ;move 25 into DH
MOV CL,34H      ;move 34 into CL
ADD DH,CL       ;add CL to DH: DH = DH + CL
```

برنامه فوق $DH = 59H$ و $CL = 34H$ را نتیجه می‌دهد. همواره برای نوشتن یک برنامه راه‌های مختلفی وجود دارد. سؤالی که پس از ملاحظه برنامه فوق ممکن است به مغز خطور کند این است که آیا ختماً برای جمع دو عدد لازم است آنها را به ثبات‌ها منتقل کنیم. پاسخ منفی است، و در واقع الزامی نیست. به نوع دیگری از همان برنامه نگاه کنید.

```
MOV DH,25H      ;load one operand into DH
ADD DH,34H      ;add the second operand to DH
```

در مورد فوق، ضمن اینکه یک ثبات یکی از مقادیر را داراست، دومین مقدار بعنوان عملوند بدنبال دستور آمده است. به این عملوند، عملوند فوری می‌گوئیم. مثال‌هایی که تاکنون برای ADD و MOV ارائه شده‌اند. نشان می‌دهند که عملوند مبدأ می‌تواند یک ثبات یا یک داده فوری باشد. در مثال‌های فوق عملوند مقصد همراه یک ثبات بوده است. قالب دستورات زبان اسمبلی، توصیف کاربری آنها و لیست انواع عملوندهای مجاز در پیوست (ب) آمده است.

بزرگترین عددی که یک ثبات 8 بیتی می‌تواند نگهدارد FFH است. برای بکارگیری اعداد بزرگتر از FFH (255 ددهی)، ثبات‌های 16 بیتی همچون AX، BX، CX، یا DX باید بکار روند. مثلاً برای جمع دو عدد مثل 34EH و 6A5H از برنامه زیر باید استفاده شود.

```
MOV AX,34EH      ;move 34EH into AX
MOV DX,6A5H      ;move 6A5H into DX
ADD DX,AX        ;add AX to DX: DX = DX + AX
```

اجرای برنامه فوق $DX = 9F3H$ ($34E + 6A5 = 9F3$) و $AX = 34E$ را نتیجه می‌دهد. مجدداً هر ثبات غیر قطعه‌ای می‌تواند برای اجرای عمل فوق بکار رود.

```
MOV CX,34EH      ;load 34EH into CX
ADD CX,6A5H      ;add 6A5H to CX (now CX=9F3H)
```

ثبات‌های همه منظوره معمولاً در عملیات حسابی بکار می‌روند. ثبات AX را گاهی انباره هم می‌گویند.

- ۱- برنامه‌ای بزبان اسمبلی برای انتقال 1234H به ثبات BX بنویسید.
- ۲- برنامه‌ای بزبان اسمبلی برای جمع 16H و ABH بنویسید. نتیجه را در ثبات AX قرار دهید.
- ۳- داخل کدام ثبات‌ها نمی‌توان مستقیماً نوشت؟
- ۴- بزرگترین عدد مبنای 16 که در یک ثبات 16 بیت قرار می‌گیرد چقدر است؟ در داخل یک ثبات 8 بیت چطور؟ معادل دهدهی این مقادیر شانزده شانزدهی چیست؟

بخش ۱-۴: مقدمه‌ای بر قطعات برنامه

یک برنامه به زبان اسمبلی متشکل از حداقل سه قطعه است: قطعه کد، قطعه داده و قطعه پشته. قطعه کد از دستورات زبان اسمبلی که وظایف برنامه را انجام می‌دهند تشکیل شده است. قطعه داده برای ذخیره اطلاعاتی که باید بوسیله دستورات داخل قطعه کد پردازش شوند، بکار می‌رود. پشته برای ذخیره اطلاعات موقت بکار گرفته می‌شود. در این بخش ما قطعات کد و داده یک برنامه را در قالب مثال‌ها و نیز چگونگی ذخیره داده در حافظه را بحث می‌کنیم. قطعه پشته در بخش ۵-۱ پوشش یافته است.

مبداء و تعریف قطعه

قطعه، ناحیه‌ای از حافظه است که 64K بایت را شامل می‌شود و از آدرسی که قابل تقسیم بر 16 باشد (مانند آدرسی که به 0 مبنای 16 ختم شود) آغاز می‌گردد. سایز قطعه 64K بایتی از کار در آمده است زیرا 8085 حداکثر تا 64K بایت حافظه فیزیکی را آدرس دهی می‌کرد. دلیل این است که تعداد پایه‌های آدرس 16 بوده است. ($2^{16} = 64K$) این محدودیت به طراحی 8088/86 منتقل گشت تا سازگاری حفظ شود. در حالی که 8085 برای قطعات کد، داده و پشته فقط 64K حافظه داشت، در 8088/86 امکان تخصیص 64K بایت به هر یک از بخش‌ها وجود دارد. در یک برنامه زبان اسمبلی این بخش‌ها قطعه کد، قطعه داده و قطعه پشته نامیده می‌شوند. به این دلیل 8088/86 علیرغم داشتن 20 پایه و لذا 1 مگابایت حافظه، در هر لحظه از زمان فقط قادر است 64K بایت داده، 64K بایت کد و 64K بایت پشته را مدیریت کند. چگونگی جابجایی این درجه 64K بایتی برای پوشش تمام 1 مگا بایت حافظه، پس از بررسی آدرس منطقی و فیزیکی در این فصل بحث شده است.

آدرس منطقی و آدرس فیزیکی

در پروشورهای 8086، اغلب سه نوع آدرس ذکر شده است: آدرس فیزیکی، آدرس تفاوت مکان و آدرس منطقی. آدرس فیزیکی یک آدرس 20 بیتی است که در واقع روی پایه‌های آدرس ریزپردازنده 8086 قرار گرفته و بوسیله مدار ارتباط حافظه دیکد می‌شود. این آدرس می‌تواند در محدوده 00000H تا FFFFFH برای 8086 و مدو واقعی 286، 386 و 486 باشد. این آدرس یک آدرس واقعی واقع در RAM یا ROM و در داخل محدوده حافظه 1 مگابایتی است. آدرس تفاوت مکان، مکانی در محدوده 64K بایتی قطعه می‌باشد. بنابراین آدرس تفاوت مکان می‌تواند از 0000H تا FFFFH باشد. آدرس منطقی متشکل از مقدار قطعه و آدرس تفاوت مکان است. تفاوت میان این آدرس‌ها و روال تبدیلی یکی به دیگری بصورت مثال‌هایی که بعداً آمده تفهیم خواهد شد.

قطعه کد

CS	IP
2 5 0 0	9 5 F 3

برای اجرای یک برنامه، 8086 دستورالعمل (کد عمل و عملوند) را از قطعه کد برمی‌دارد. آدرس منطقی یک آدرس، همیشه از CS (قطعه کد)، IP (اشاره‌گر دستور) که در قالب

CS:IP نشان داده می‌شود تشکیل شده است. آدرس فیزیکی محل دستور، با نقل مکان (شیفت یا

جابجایی) CS به میزان یک رقم مبنای 16 به چپ و جمع آن با IP بدست می آید. IP حاوی آدرس تفاوت مکان است. آدرس 20 بیتی حاصل آدرس فیزیکی است زیرا بر روی پایه های گذرگاه آدرس فیزیکی بیرونی قرار می گیرد تا بوسیله مدار دیکد حافظه، دیکد شود. برای روشن شدن این مطلب مهم فرض کنید که مقادیر CS و IP طبق نمودار مفروض باشند. آدرس تفاوت مکان در IP واقع است که در این حالت 95F3H است. آدرس منطقی CS:IP یا 2500:95F3 می باشد. آدرس فیزیکی $25000 + 95F3 = 2E5F3H$ خواهد شد. آدرس فیزیکی یک دستور بطریق زیر قابل محاسبه است.

2 5 0 0

۱- با CS شروع کنید.

2 5 0 0 0

۲- CS را به چپ شیفต์ بدهید.

2 E 5 F 3

۳- با IP جمع کنید.

ریزپردازنده دستور را از مکان های حافظه ای که با 2E5F3 شروع می شود باز می یابد. چون IP دارای حداقل مقدار 0000H و حداکثر FFFFH است. محدوده آدرس منطقی برابر با 2500:0000 تا 2500:FFFF خواهد شد. این بدان معنی است که پایین ترین مکان حافظه قطعه کد فوق (25000+0000) 25000H و بالاترین مکان حافظه (25000+FFFF) 34FFFH خواهد بود. اگر دستور مورد نظر ورای این محدوده باشد چه می شود؟ پاسخ این است که مقدار CS باید برای دستیابی به آن مکان ها تغییر یابد. مثال ۱-۱ را ملاحظه نمائید.

مثال ۱-۱: اگر CS=24F6H و IP=634AH باشد:

- (الف) آدرس منطقی را نشان دهید.
(ب) آدرس تفاوت مکان را بدست آورید و نیز:
(پ) آدرس فیزیکی را محاسبه کنید
(ت) محدوده بالای قطعه کد را محاسبه کنید

حل:

$$\begin{aligned} \text{(الف)} \quad 24F6:634A & \quad \text{(ب)} \quad 634A \\ \text{(پ)} \quad 2B2AA &= (24F60 + 634A) \\ \text{(ت)} \quad 34F5F &= (24F60 + FFFF) \end{aligned}$$

آدرس منطقی در برابر آدرس فیزیکی در قطعه کد

در قطعه کد، CS و IP آدرس منطقی دستورات را برای اجرا نگه می دارند. دستورات زبان اسمبلی زیر، اسمبل شده (بصورت کد ماشین در آمده) و در حافظه ذخیره شده است.

آدرس منطقی	زبان ماشین کد عمل و عملوند	زبان اسمبلی نماد خلاصه و عملوند
CS:IP		
1132:0100	B057	MOV AL,57
1132:0102	B686	MOV DH,86
1132:0104	B272	MOV DL,72
1132:0106	89D1	MOV CX,DX
1132:0108	88C7	MOV BH,AL
1132:010A	B39F	MOV BL,9F
1132:010C	B420	MOV AH,20
1132:010E	01D0	ADD AX,DX
1132:0110	01D9	ADD CX,BX
1132:0112	05351F	ADD AX,1F35

سه ستون مربوط بترتیب آدرس‌های منطقی، CS:IP، کد ماشین ذخیره شده در آن آدرس و کد زبان اسمبلی مربوط را نشان می‌دهد. این اطلاعات بسادگی بوسیله DEBUG و استفاده از فرمان عکس اسمبل قابل تولید است.

برنامه فوق نشان می‌دهد که بایت واقع در آدرس 1132:0100 حاوی B0 است که کد عمل، انتقال یک مقدار به ثبات AL، و آدرس 1132:0101 حاوی عملوند مورد انتقال (در این حال 57) به AL است. بنابراین دستور "MOV, AL, 57" دارای کد ماشین B057 است که در آن B0 کد عمل و 57 عملوند می‌باشد. بطور مشابه کد ماشین B686 در مکان‌های حافظه 1132:0102 و 1132:0103 بوده و کد و عملوند دستور "MOV DH, 86" را نشان می‌دهد. آدرس فیزیکی واقعی در داخل RAM و حتی ROM است. در زیر آدرس‌های فیزیکی و محتوای هر یک از مکان‌های برنامه فوق آورده شده است. بیاد داشته باشید که آنچه بر روی گذرگاه آدرس CPU 8086 قرار می‌گیرد تا بوسیله مدار و دیکدر حافظه دیکد شود، آدرس فیزیکی است.

آدرس منطقی	آدرس فیزیکی	محتوای کد ماشین
1132:0100	11420	B0
1132:0101	11421	57
1132:0102	11422	B6
1132:0103	11423	86
1132:0104	11424	B2
1132:0105	11425	72
1132:0106	11426	89
1132:0107	11427	D1
1132:0108	11428	88
1132:0109	11429	C7
1132:010A	1142A	B3
1132:010B	1142B	9F
1132:010C	1142C	B4
1132:010D	1142D	20
1132:010E	1142E	01
1132:010F	1142F	D0
1132:0110	11430	01
1132:0111	11431	D9
1132:0112	11432	05
1132:0113	11433	35
1132:0114	11434	1F

قطعه داده

فرض کنید برنامه‌ای برای جمع 5 بایت داده مانند 2BH، 1FH، 15H، 12H، 25H، که هر یک مربوط به اضافه کاری یک شخص است. یک روش برای جمع آنها به شرح زیر است:

MOV	AL, 00H	; initialize AL
ADD	AL, 25H	; add 25H to AL
ADD	AL, 12H	; add 12H to AL
ADD	AL, 15H	; add 15H to AL
ADD	AL, 1FH	; add 1FH to AL
ADD	AL, 2BH	; add 2BH to AL

در این برنامه، داده و کد با هم در دستور مخلوط شده‌اند. مشکل نوشتن برنامه بروش فوق این است که اگر داده عوض شود، کد باید برای هر مکان که داده در آن است جستجو شود. و دوباره تایپ گردد به این دلیل، این ایده مطرح می‌شود که یک ناحیه خاص به داده اختصاص یابد. در ریزپردازنده‌های 80x86، ناحیه‌ای که برای داده کنار گذاشته شده قطعه داده خوانده می‌شود. همچون قطعه کد که با ثبات‌های CS و IP بعنوان قطعه کد و تفاوت مکان مرتبط است، قطعه داده ثبات DS و یک مقدار تفاوت مکان را بکار می‌برد.

در زیر چگونگی ذخیره شدن داده در قطعه داده نشان داده شده و برنامه طوری بازنویسی شده که قابل استفاده برای هر مجموعه داده باشد. فرض کنید که تفاوت مکان قطعه داده 200H است. داده در مکان‌های زیر قرار گرفته است:

DS:0200 = 25
DS:0201 = 12
DS:0202 = 15
DS:0203 = 1F
DS:0204 = 2B

و برنامه دوباره به شکل زیر بازنویسی می‌گردد:

```
MOV AL,0 ;clear AL
ADD AL,[0200] ;add the contents of DS:200 to AL
ADD AL,[0201] ;add the contents of DS:201 to AL
ADD AL,[0202] ;add the contents of DS:202 to AL
ADD AL,[0203] ;add the contents of DS:203 to AL
ADD AL,[0204] ;add the contents of DS:204 to AL
```

توجه داشته باشید که آدرس تفاوت مکان در داخل کروشه آورده شده است. کروشه‌ها بیان می‌دارند که آدرس داده، و نه خود داده، به وسیله عملوند نشان داده شده است. اگر از کروشه‌ها استفاده نشود، مثل "MOV AL,0200"، CPU در عوض محتوای آدرس 200 در AL سعی در بارکردن خود 200 خواهد کرد. بخاطر بسپارید که یک اختلاف مهم در قالب کد برای MASM و DEBUG وجود دارد و آن این است که DEBUG فرض می‌کند همه اعداد در مبنای شانزده هستند و بنابراین پسوند H را لازم ندارند، در صورتی که MASM فرض می‌کند آنها به دهدهی می‌باشند و بنابراین پسوند H در داده لازم است.

این برنامه با هر مجموعه داده‌ای کار می‌کند. تغییر داده هیچ تأثیری روی کد ندارد. گرچه این برنامه فرم اصلاح شده قبلی است، ولی باز هم قابلیت اصلاح دارد. اگر قرار باشد داده در آدرس با تفاوت مکان متفاوتی مثلاً 450H ذخیره شود، برنامه باید بازنویسی گردد. یک راه حل برای این مشکل استفاده از یک ثابت برای نگهداری آدرس تفاوت مکان است و بنابراین قبل از هر ADD، ثابت برای دستیابی به بایت بعدی افزایش می‌یابد. آنگاه درباره ثابت مورد استفاده تصمیم‌گیری می‌شود. 8088/86 تنها اجازه استفاده از BX، SI و DI را، بعنوان ثابت تفاوت مکان برای قطعه داده می‌دهد. به بیان دیگر، در حالی که CS فقط ثابت IP را بعنوان تفاوت مکان بکار می‌برد، DS فقط از BX، DI و SI برای نگهداری آدرس تفاوت مکان داده استفاده می‌کند. کلمه اشاره گر اغلب برای ثابتی که آدرس تفاوت مکان را نگهدارد بکار می‌رود. در مثال زیر BX بعنوان اشاره گر استفاده شده است.

```
MOV AL,0 ;initialize AL
MOV BX,0200H ;BX points to the offset addr of first byte
ADD AL,[BX] ;add the first byte to AL
INC BX ;increment BX to point to the next byte
ADD AL,[BX] ;add the next byte to AL
INC BX ;increment the pointer
ADD AL,[BX] ;add the next byte to AL
INC BX ;increment the pointer
ADD AL,[BX] ;add the last byte to AL
```

دستور "INC" 1 را به عملوند اضافه می‌کند. "INC BX" نتیجه یکسانی را با "ADD BX, 1" بدست می‌دهد. برای برنامه فوق، اگر آدرس تفاوت مکان داده تغییر یابد تنها لازم است یک دستور اصلاح شود و بقیه برنامه تغییر نخواهد کرد. بررسی برنامه فوق نشان می‌دهد که الگویی از دو دستور تکرار شده است. اینگونه برنامه‌ها منجر به استفاده از حلقه برای تکرار دستورات معینی می‌گردد. پیاده‌سازی حلقه نیاز به آشنایی با ثابت پرچم دارد که بعداً در این فصل در مورد آن بحث خواهد شد.

آدرس منطقی و آدرس فیزیکی در قطعه داده

آدرس فیزیکی برای داده با استفاده از همان قوانین قطعه کد محاسبه می‌شود. یعنی آدرس فیزیکی داده با شیفت DS به چپ به میزان یک رقم مبنای شانزده و جمع آن با مقدار تفاوت مکان بدست می‌آید. مثال‌های ۱-۲، ۱-۳ و ۱-۴ ملاحظه شوند.

مثال ۱-۲: فرض کنید DS برابر 5000 و مقدار تفاوت مکان 1950 است. آدرس فیزیکی بایت را حساب کنید.
حل:

تغییر مکان : DS

 :

آدرس فیزیکی برابر است با
۱- با DS شروع کنید

۲- شیفت DS به چپ

۳- تفاوت مکان را اضافه کنید

مثال ۱-۳: اگر $DS = 7FA2H$ و تفاوت مکان 438EH باشد،

(الف) آدرس فیزیکی را محاسبه کنید

(پ) محدوده پایین را محاسبه نمایید

(ب) محدوده بالای قطعه داده را حساب کنید

(ت) آدرس منطقی را نشان دهید

حل:

$$7FA20 = (7FA20 + 0000) \text{ (ب)}$$

$$7FA2:438E \text{ (ت)}$$

$$83DAE = (7FA20 + 438E) \text{ (الف)}$$

$$8FA1F = (7FA20 + FFFF) \text{ (پ)}$$

مثال ۱-۴: فرض کنید که ثبات DS برابر 578C باشد. برای دستیابی به یک بایت مفروض از داده در آدرس فیزیکی 67F66، آیا قطعه داده محدوده‌ای را که داده در آن است می‌پوشاند.

اگر خیر، چه تغییراتی لازم است؟

حل:

خیر، چون محدوده از 578C0 تا 678BF است و مکان 67F66 در این محدوده نیست. برای دستیابی به این بایت DS باید طوری تغییر کند که بایت مذکور را شامل شود.

قرارداد مبتنی بر کوچ‌تر

در مثال‌های قبلی داده 8 بیتی یا یک بایتی بکار رفت. در این حال بایت‌ها یکی پس از دیگری در حافظه ذخیره می‌شوند. وقتی که داده 16 بیت بکار بریم چه خواهد شد؟ مثلاً

MOV AX,35F3H ;load 35F3H into AX
MOV [1500],AX ;copy the contents of AX to offset 1500H

در حالانی این چنین، بایت پایین به مکان حافظه پایین می‌رود و بایت بالاتر به آدرس بالاتر خواهد رفت. در مثال فوق حافظه 1500 : DS حاوی F3H و مکان حافظه DS:1501 حاوی 35H خواهد بود.

$$DS:1500 = F3 \quad DS:1501 = 35$$

این قرارداد را مبتنی بر کوچ‌تر می‌گویند. مأخذ مبتنی بر کوچ‌تر یا مبتنی بر بزرگ‌تر از داستان سفرهای گالیور می‌باشد که در آن چگونگی باز شدن تخم مرغ بحث شده بود: یعنی از سمت کوچ‌تر باز می‌شود یا سمت بزرگ‌تر.

در روش مبتنی بر بزرگتر، بایت بالاتر به آدرس کوچکتر می‌رود، در صورتی که در روش مبتنی بر کوچکتر بایت بالاتر به آدرس بالا و بایت پایین به آدرس پایین می‌رود. مثال ۵-۱ ملاحظه شود. همه ریز پردازنده‌های اینتل و بسیاری از مینی کامپیوترها از جمله Digital VAX از قرارداد مبتنی بر کوچکتر استفاده می‌کنند. بنظر می‌رسد که تفاوت این دو روش به اهمیت شکستن یک تخم مرغ از سر یا ته باشد ولی تبدیل نرم افزار از یک گروه و اجرای آن بر روی کامپیوتری دیگر مایه دردسر است.

مثال ۵-۱: مکان‌هایی از حافظه را با محتوای زیر در نظر بگیرید: DS:6826=48 و DS:6827=22 محتوای ثبات BX را در دستور "MOV BX, [6826]" پیدا کنید.

حل:

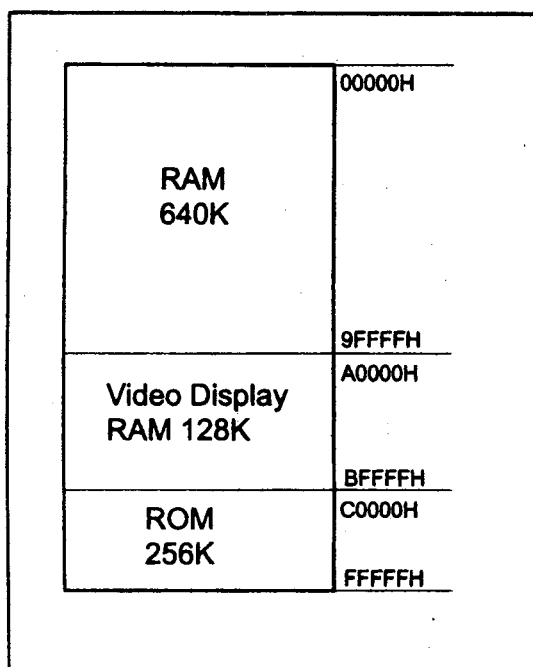
طبق قرارداد مبتنی بر کوچکتر که در تمام ریزپردازنده‌های 80x86 بکار رفته است، ثبات BX باید حاوی مقداری از آدرس تفاوت مکان پایین در 6826 و BH باید مقدار آدرس تفاوت مکان بالا یعنی 6827 باشد و بنابراین BL = 48H و BH = 22H خواهد بود.

	BH	BL
DS:6826 = 48	22	48
DS:6827 = 22		

قطعه اضافی (ES)

ES ثبات قطعه‌ای است که با نام قطعه اضافی مورد استفاده قرار می‌گیرد. گرچه در بسیاری از برنامه‌های معمولی این قطعه بکار نرفته است، ولی استفاده از آن برای عملیات رشته‌ای اساسی بوده و در فصل ۶ بحث خواهد شد.

نقشه حافظه IBM PC



شکل ۳-۱. تخصیص حافظه در PC

برای اینکه برنامه‌ای روی PC اجرا شود، DOS باید ابتدا آن را در RAM بار کند. سؤال این است که در کدام قسمت از RAM بار می‌شود؟ برای پاسخ به این سؤال باید ابتدا بعضی از مفاهیم مهم را درباره حافظه PC بدانیم. 20 بیت آدرس 8088/86 اجازه دستیابی به 1 مگابایت حافظه (1024K بایت) با آدرس 00000 الی FFFFF را می‌دهد. در فاز طراحی اولین IBM PC، باید در مورد تخصیص فضای حافظه یک مگابایتی به بخش‌های مختلف PC تصمیم می‌گرفتند. این تخصیص حافظه را طرح یا نقشه حافظه می‌گویند. نقشه حافظه IBM PC در شکل ۳-۱ دیده می‌شود. از این یک مگابایت، 640K بایت، از آدرس 00000-9FFFFH برای RAM کنار گذاشته شد. به میزان 128K بایت، از A0000-BFFFFH برای حافظه تصویر در نظر گرفته شد. 256K بایت باقیمانده، از C0000H-FFFFFH هم برای ROM تخصیص یافت.

مطالبی بیشتر درباره RAM

در اوایل دهه ۱۹۸۰، اغلب PCها فقط با 64K تا 256K بایت حافظه RAM تولید می‌شدند و در آن زمان این مقدار کاملاً مناسب بنظر می‌رسید. در صورت نیاز به حافظه اضافی، کاربران مجبور بودند تا پورت‌های

گسترش حافظه را تا 640K خریداری نمایند. نیازی به توسعه به ویرایش DOS و نرم افزار کاربردی کاربر بستگی داشت. سیستم عامل DOS ابتدا بخشی را برای استفاده خود کنار گذاشته و سپس اجازه می دهد تا باقیمانده آن بوسیله کاربردهایی مثل پردازش های کلمات مورد استفاده قرار گیرد. وظیفه پیچیده مدیریت حافظه RAM به DOS واگذار شد زیرا مقدار حافظه بکار رفته بوسیله DOS به ویرایش آن بستگی داشت و کامپیوترهای مختلف هم مقادیر متفاوتی RAM داشتند، علاوه حافظه مورد نیاز برنامه های کاربردی هم متفاوت بود. به این دلیل هیچ مقداری به ثبات های CS، DS و SS تخصیص نیافت زیرا تخصیص هر مقدار حافظه فیزیکی بمعنی یک تخصیص در فاصله 00000 الی 9FFFFH بوده و این ورا دانش کاربر است. دلیل دیگر آن است که هر تخصیصی از حافظه فیزیکی ممکن است روی یک PC کار کند ولی روی PC دیگری یا DOS متفاوت و سایر RAM مختلف کار نکند. به بیان دیگر برنامه قابلیت انتقال بر روی PC دیگری را ندارد. بنابراین مدیریت حافظه یکی از مهمترین وظایف سیستم عامل DOS است و باید به DOS واگذار شود. این مطلب اهمیت بسزایی دارد و باید بخاطر سپرده شود زیرا در بسیاری از مثال های این کتاب مقداری را برای ثبات های CS، DS و SS داریم که با آنچه در PC دیگر کاربران وجود دارد متفاوت است. بنابراین سعی در تخصیص مقدار به ثبات های قطعه و یکسان کردن آنها با این کتاب نکنید.

RAM تصویر

از A0000H تا BFFFFH برای تصویر کنار گذاشته شده است. مقدار بکار رفته و محل آن به مورد تصویر نصب شده در PC بستگی دارد. جدول ۲ - (ث) از پیوست (ث) آدرس های شروع بوردهای تصویر را لیست کرده است.

مطالبی بیشتر در مورد ROM

از C0000H تا FFFFFH به ROM منتسب شده است. البته همه محدوده حافظه PC در این محدوده بوسیله ROM بکار نرفته است. از این 256K، فقط 64K بایت از F0000H تا FFFFFH بوسیله BIOS ROM (سیستم ورودی / خروجی اصلی) استفاده شده است. بخشی از فضای باقیمانده بوسیله کارت های آداپتور (مانند آداپتور دیسک سخت) استفاده شده است، و بقیه آزاد است. در سال های اخیر ویرایش های جدید DOS که قابلیت مدیریت حافظه قوی تری دارند از فضای حافظه ورا 640K استفاده های مفیدی بعمل آورده اند. 640K بایت حافظه که از فضای 00000 تا 9FFFFH را اشغال می نماید را حافظه متعارف می گویند، در DOS 5، 384K بایت از A0000H تا FFFFFH را بلوک حافظه فوقانی (UMP) می خوانند. بحث کاملی از اصطلاحات و آرایش های این حافظه ها در جلد دوم آمده است.

وظیفه BIOS ROM

چون CPU تنها برنامه هایی را که در حافظه ذخیره شده اجرا می کند، باید مقداری حافظه دائمی یا غیر فزّار وجود داشته باشد تا برنامه ای را نگه دارد که به هنگام روشن شدن به CPU کارهایی را که باید انجام دهد گوشزد کند. این مجموعه برنامه ها که بوسیله ROM نگهداری می شود BIOS خوانده می شود که بمعنی سیستم ورود-خروج اصلی می باشد و حاوی برنامه های تست RAM و دیگر اجزاء متصل به CPU است. برنامه های ارتباط با وسایل جانبی مانند صفحه کلید، ویدئو، چاپگر و دیسک با DOS نیز در آن قرار دارد. تست همه وسایل متصل به PC به هنگام روشن شدن و گزارش هر خطا هم بعهد BIOS است. مثلاً اگر صفحه کلید قبل از روشن شدن کامپیوتر از PC جدا شود، BIOS پیام خطایی را روی صفحه نمایش فرستاده و وضعیت را اعلام خواهد کرد. تنها پس از تست و نصب وسایل جانبی است که BIOS، DOS را از دیسک به RAM انتقال داده و سپس کنترل PC را به DOS می سپارد.

- ۱- یک قطعه ناحیه‌ای از حافظه است که تا بایت را شامل می‌شود.
- ۲- در 8086، بزرگی قطعه چقدر است؟ آیا آدرس فیزیکی 346E0 می‌تواند آدرس شروع یک قطعه باشد؟ چرا بله و چرا خیر؟
- ۳- تفاوت بین آدرس‌های فیزیکی و منطقی را بیان کنید.
- ۴- آدرس فیزیکی یک آدرس بیت است؛ یک آدرس تفاوت مکان ... بیت دارد.
- ۵- کدام ثبات بعنوان ثبات تفاوت مکان برای ثبات قطعه CS بکار می‌رود؟
- ۶- اگر BX=1234H بوده و دستور BX [2400]، MOV اجرا شود محتوای مکان‌های حافظه در تفاوت مکان 2400 و 2401 چیست؟

بخش ۵-۱: مطالبی بیشتر در مورد قطعات در 80x86

در این بخش ما مفهوم پشته را بررسی می‌کنیم، کاربرد آن را در ریزپردازنده‌های 80x86 می‌بینیم و پیاده‌سازی آن را در قطعه پشته خواهیم دید. سپس مفاهیم پیشرفته‌تر مربوط به قطعات مانند همپوشی قطعات بحث می‌شوند.

پشته چیست و چرا لازم است؟

پشته بخشی از حافظه نوشتن/خواندن (RAM) است که بوسیله CPU برای ذخیره موقت اطلاعات بکار می‌رود. CPU به این ناحیه ذخیره‌سازی نیاز دارد زیرا تعداد ثبات‌ها محدود است. بنابراین برای ذخیره موقت و امن اطلاعات باید محلی برای CPU در نظر گرفته شود. سئوالی که ممکن است در اینجا مطرح شود این است که چرا CPU ها با ثبات‌های بیشتر طراحی نمی‌شوند؟ دلیل این است که در طراحی CPU، ترانزیستور گرانقیمت بوده و به تعداد کافی برای ساخت صدها ثبات در اختیار نیستند. بعلاوه، هر CPU چند ثبات داشته باشد تا هر برنامه یا کاربردی را جوابگو باشد؟ همه تکنیک‌های بکار رفته در برنامه‌نویسی و کاربردهای مختلف یکسان نیستند. می‌توان این مطلب را با خرید یک خانه مقایسه کرد که در آن خرید و ساخت خانه‌ای با ۵۰ اتاق برای کسی که احتمالاً بخواهد چیزی را نگه دارد بسیار پرهزینه می‌باشد. در عوض شخص دیگری برای نگهداری آن شیشی اتاقکی را می‌سازد و یا می‌خرد. مسلماً هزینه مورد دوم کمتر است.

با قبول مزایای داشتن پشته، سئوال دیگر این است که معایب چیست؟ مهمترین عیب پشته زمان دستیابی آن است. چون پشته در RAM قرار دارد، زمان دستیابی آن بسیار طولانی‌تر از ثبات است. گذشته از اینها، ثبات‌ها در داخل CPU هستند ولی RAM در خارج آن است. به این دلیل بعضی از کامپیوترهای قدرتمند فاقد پشته هستند و در عوض CPU دارای تعداد زیادی ثبات برای کار کردن است.

چگونه پشته‌ها دستیابی می‌شوند

پشته بخشی از RAM است و باید بوسیله ثبات‌هایی در داخل CPU به آن اشاره شود. دو ثبات اصلی برای دستیابی به پشته عبارتند از ثبات SS (قطعه پشته) و ثبات SP (اشاره‌گر پشته). این ثبات‌ها باید قبل از دستیابی دستورات به پشته بکار رفته باشند.

هر ثبات درون 80x86 (بجز ثبات‌های قطعه و SP) قابل ذخیره‌سازی در پشته و بازیابی آن به CPU است. ذخیره یک ثبات CPU در پشته را درج و بارکردن محتوای پشته را در CPU بازیافت می‌گویند. به بیان دیگر، یک ثبات برای ذخیره شدن در پشته درج می‌شود و برای بازیابی آن بازیافت می‌گردد. وظیفه SP هنگام اجرای درج و بازیافت خیلی حساس است. در 80x86، ثبات اشاره‌گر پشته SP به مکان حافظه جاری بکار رفته در بالای پشته

اشاره می‌کند و به محض درج داده کاهش می‌یابد. برعکس هنگام بازیافت پشته به CPU، این اشاره‌گر افزایش می‌یابد. وقتی که دستوری یک ثبات همه منظوره را در پشته درج و یا از آن بازیافت می‌کند روی همه 16 بیت ثبات عمل می‌نماید. به بیان دیگر باید "PUSH AX" بکار رود و دستوری مانند PUSH AL یا PUSH AH وجود ندارد. دلیل کاهش SP پس از هر درج این است که مطمئن شویم پشته از حافظه بالا بطرف آدرس‌های پایین‌تر رشد می‌کند. این برخلاف IP می‌باشد (اشاره‌گر دستور). همانطور که در بخش قبل دیدیم، IP به دستور بعدی برای اجرا اشاره می‌کند و پس از اجرای هر دستور اضافه می‌شود. برای اینکه مطمئن شویم که قطعه کد و قطعه پشته هرگز روی یکدیگر چیزی نمی‌نویسند آنها را در دو انتها مخالف هم در حافظه RAM قرار داده‌اند و سمت یکدیگر رشد می‌کنند ولی نباید به هم برسند اگر به هم برسند برنامه از هم می‌پاشد. برای مشاهده چگونگی رشد پشته به مثال‌های زیر توجه کنید.

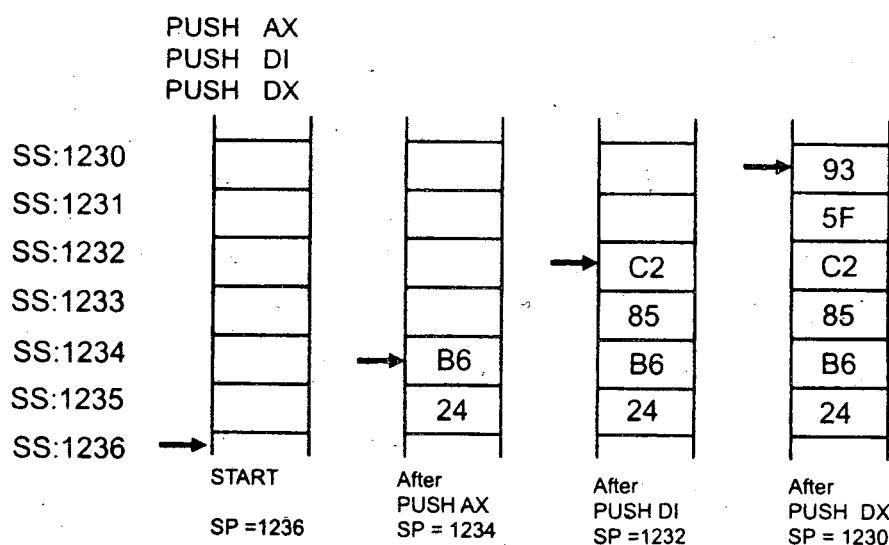
درج در پشته

در مثال ۶-۱ توجه کنید که بازاء اجرای هر درج، محتوای ثبات در پشته ذخیره شده و SP دو واحد کم می‌شود. برای هر بایت از داده که در پشته ذخیره شود، SP یکواحد کم می‌گردد و چون دستور PUSH محتوای ثبات 16 بیتی را ذخیره می‌کند باید دوبار کاهش یابد. همچنین به چگونگی ذخیره شدن داده توجه نمایید. در 80x86، بایت پایین‌تر به همیشه در آدرس پایین‌تر ذخیره می‌گردد. به این دلیل محتوای AH یعنی 24H در مکان 1235 و AL در مکان 1234 ذخیره می‌شود.

بازیافت پشته

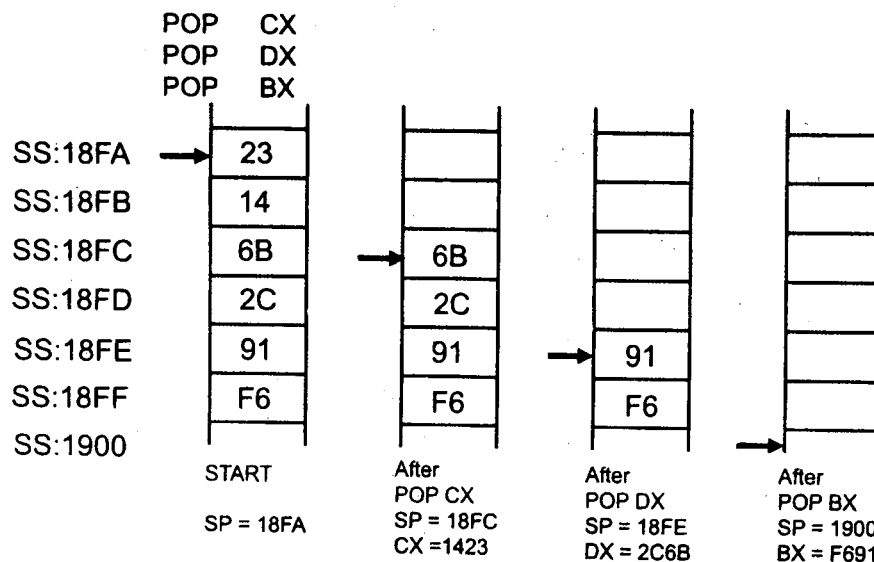
بازیافت محتوای پشته به CPU 80x80، برعکس درج است. با هر بازیافت دو بایت بالای پشته در ثباتی که بوسیله دستور مشخص شده کپی می‌گردد و آنگاه اشاره‌گر پشته دوبار افزایش می‌یابد. گرچه داده در واقع در حافظه باقی می‌ماند، ولی قابل دستیابی نیست زیرا اشاره‌گر پشته واری آن نقطه است. مثال ۷-۱ دستور POP را نمایش می‌دهد.

مثال ۶-۱: با فرض $SP=1236$ ، $AX=24B6$ ، $DI=85C2$ و $DX=5F93$ ، محتوای پشته را پس از اجرای هر یک از دستورات زیر نشان دهید.



حل:

مثال ۷-۱: با فرض پشته زیر و ثبات‌ها را پس از اجرای دستورات زیر مشخص کنید.



آدرس منطقی در برابر آدرس فیزیکی برای پشته

اکنون این سؤال مطرح است که مکان فیزیکی دقیق پشته کجاست؟ این مکان به مقدار قطعه داده SS و SP اشاره گر پشته است، بستگی دارد. برای محاسبه آدرس‌های فیزیکی پشته همان اصولی که برای کد و داده بیان شد، اعمال می‌شود. روش این است که SS به چپ شیفت داده شده و سپس با تفاوت مکان ثبات اشاره گر پشته SP جمع می‌گردد. این کار در مثال ۸-۱ نشان داده شده است.

چه مقادیری به SP و SS منتسب می‌شود و چه کسی آن را اختصاص می‌دهد؟ سیستم عامل DOS وظیفه دارد تا به SP و SS مقدار بدهد زیرا مدیریت حافظه از جمله وظایف سیستم عامل است. قبل از پایان بحث پشته، دو نکته ذکر می‌شود. یکی اینکه در بروشورهای 80x86، بالای پشته آخرین مکان اشغال شده است. این با دیگر CPUها فرق دارد. دوم اینکه BP ثبات دیگری است که می‌تواند بعنوان تفاوت مکان در پشته بکار رود ولی این ثبات کاربرد بسیار خاصی دارد و در دستیابی پارامترهای تحویلی بین برنامه‌های زبان اسمبلی و برنامه‌های زبان سطح بالا مثل C کاربرد فراوان دارد. این مطلب در فصل ۷ بحث شده است.

مثال ۸-۱: اگر SS=3500H و SP=FFFEH باشد،

- (الف) آدرس فیزیکی پشته را حساب کنید (ب) محدوده پایین‌تر را محاسبه نمایید
(پ) محدوده بالای قطعه پشته را حساب کنید (ت) آدرس منطقی پشته را نشان دهید

حل:

$$(ب) \quad 35000 = (35000 + 0000)$$

$$(ت) \quad 3500:FFFF$$

$$(الف) \quad 44FFE = (35000 + FFFE)$$

$$(پ) \quad 44FFF = (35000 + FFFF)$$

جمله‌ای چند درباره قطعات در 80x86

آیا یک آدرس فیزیکی می‌تواند متعلق به چند آدرس منطقی باشد؟ بله، به آدرس فیزیکی 15020H توجه کنید. چندین آدرس منطقی وجود دارند که می‌توانند این تک آدرس فیزیکی را نمایش دهند:

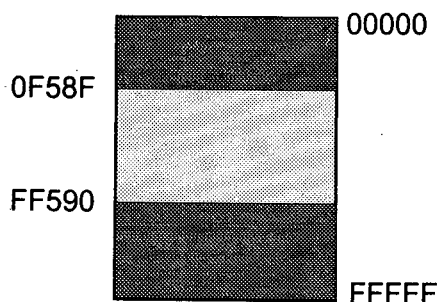
آدرس منطقی (hex)	آدرس فیزیکی (hex)
1000:5020	15020
1500:0020	15020
1502:0000	15020
1400:1020	15020
1302:2000	15020

این مطلب رفتار دینامیکی قطعه و مفهوم تفاوت مکان را در CPU 8086 نشان می‌دهد. آخرین نکته که باید روشن شود این است که وقتی تفاوت مکان به ثبات قطعه نقل مکان داده شده اضافه می‌شود، آدرس، ورای محدوده مجاز ماکزیمم FFFFFH خواهد بود. در چنین حالتی آدرس به آغاز باز می‌گردد. این مطلب در مثال زیر نشان داده شده است.

مثال ۹-۱: محدوده آدرس فیزیکی اگر CS=FF59 باشد چیست؟

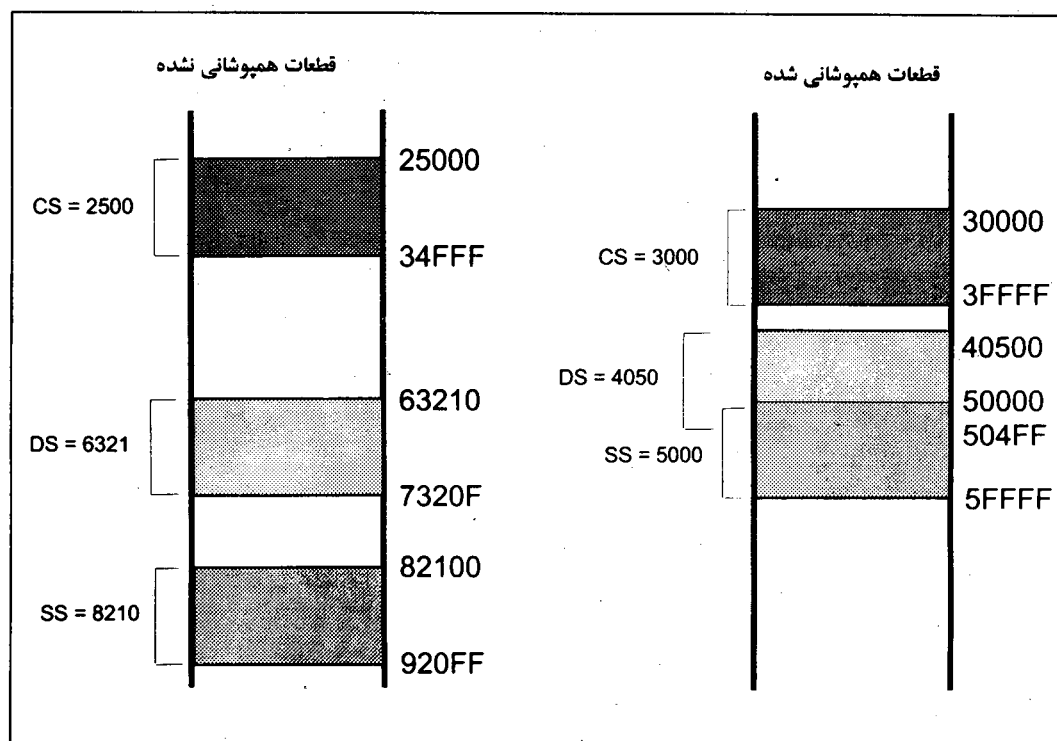
حل:

محدوده پایین (FF590+0000)FF590 است. محدوده به FFFFFH رفته و دوباره از آدرس 00000 به (FF590+FFFF=0F58F)0F58F خواهد رفت.



همپوشی

در محاسبه آدرس فیزیک امکان دارد که دو قطعه یکدیگر را بپوشانند که در بعضی مواقع موردنظر هم هست. مثلاً همپوشی در فایل‌های COM بکار می‌رود و ما نیز در فصل ۲ آن را خواهیم دید. شکل ۱-۴ قطعات همپوشی و غیر همپوشی را نشان می‌دهد.



شکل ۱-۴ قطعات غیر همپوش در مقابل پوشش

ثبات پرچم 16 بیتی :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

R = رزرو

U = تعریف نشده

OF = پرچم سرریز

DF = پرچم جهت

IF = پرچم وقفه

TF = پرچم تله

SF = پرچم علامت

ZF = پرچم صفر

AF = پرچم نقلی کمکی

PF = پرچم توازن

CF = پرچم نقلی

شکل ۵-۱ ثبات پرچم

ثبات پرچم

ثبات پرچم یک ثبات 16 بیتی است و آن را ثبات وضعیت هم می‌گویند. گرچه ثبات 16 بیت عرض دارد ولی فقط تعدادی از این بیت‌ها بکار رفته‌اند. بقیه یا تعریف نشده‌اند و یا بوسیله اینتل رزرو مانده‌اند. شش پرچم به پرچم‌های شرطی معروفند و برخی شرایط حاصل را پس از اجرای دستور مشخص می‌کند. این شش ثبات عبارتند از CF، PF، AF، ZF، SF و OF. سه پرچم باقیمانده، پرچم‌های کنترل خوانده می‌شوند زیرا از آنها برای کنترل عملکرد دستورات، قبل از اجرا استفاده می‌شود. نمودار ثبات پرچم در شکل ۵-۱ دیده می‌شود.

بیت‌های ثبات پرچم

در زیر بیت‌های ثبات پرچم بکار رفته در برنامه‌نویسی زبان اسمبلی 80x86 لیست شده‌اند. توضیح مختصری در مورد هر بیت داده شده است. چگونگی استفاده از این پرچم‌ها در مثال‌های برنامه‌نویسی در سرتاسر کتاب ملاحظه خواهد شد.

پرچم نقلی، CF. هر وقت پس از یک عمل هشت بیتی از بیت 7 (d7) و یا پس از عمل 16 بیتی از بیت 15 (d15) یک رقم نقلی تولید شود این پرچم 1 می‌گردد.

پرچم توازن، PF. پس از عملیات خاصی، توازن بایت پایین رتبه نتایج چک می‌شود. اگر بایت مذکور تعداد زوجی 1 داشته باشد پرچم توازن 1 می‌شود؛ در غیر اینصورت پاک می‌گردد.

پرچم نقلی کمکی AF. اگر در عملی از بیت d3 به d4 رقم نقلی وجود داشته باشد، این بیت 1 می‌گردد؛ در غیر اینصورت پاک می‌شود (صفر می‌گردد). این پرچم بوسیله دستوراتی که محاسبه BCD انجام می‌دهند بکار می‌رود.

پرچم صفر، ZF. این پرچم بشرطی 1 می‌شود که نتیجه یک عمل حسابی یا منطقی صفر باشد. در غیر اینصورت پاک می‌شود.

پرچم علامت، SF. نمایش دودویی اعداد علامت‌دار از باارزشترین بیت بعنوان بیت علامت استفاده می‌کند. پس از عملیات حسابی یا منطقی وضعیت این بیت علامت در SF کپی می‌شود و لذا علامت نتیجه را مشخص می‌کند.

پرچم تله، TF. وقتی که این پرچم 1 شود برنامه می‌تواند به صورت تک مرحله اجرا شود. به این معنی که هربار یک دستور اجرا می‌گردد. اجرای تک مرحله برای عیب‌یابی مورد استفاده قرار می‌گیرد.

پرچم فعال‌ساز وقفه، IF. این بیت به منظور فعال‌کردن و غیر فعال کردن تقاضاهای وقفه قابل پیشگیری خارجی بترتیب 1 یا 0 می‌شود.

پرچم جهت، DF. این بیت برای کنترل عملیات رشته بکار می‌رود و در فصل ۶ در مورد آن بحث شده است.

پرچم سرریز، OF. هر وقت نتیجه عملیات اعداد علامت‌دار خیلی بزرگ باشد و بنابراین موجب سرریز با ارزشترین بیت، به بیت علامت گردد. این بیت 1 می‌گردد. بطور کلی رقم نقلی برای تشخیص خطا در محاسبات بی‌علامت بکار می‌رود. پرچم سرریز فقط هنگام تشخیص خطا در عملیات محاسباتی علامت‌دار استفاده می‌شود.

ثبات پرچم و دستور ADD

در این بخش تأثیر دستور ADD روی ثبات پرچم را بعنوان مثالی از کاربرد بیت‌های پرچم خواهیم دید. بیت‌های پرچمی که بوسیله دستور ADD متأثر می‌شوند عبارتند از CF (پرچم نقلی)، PF (پرچم توازن)، AF (پرچم نقلی کمکی)، ZF (پرچم صفر)، SF (پرچم علامت)، و OF (پرچم سرریز). پرچم سرریز در فصل ۶ بحث شده است، زیرا تنها مربوط به اعداد حسابی علامت‌دار است. برای درک چگونگی متأثر شدن این بیت‌ها به مثال‌های ۱-۱۰ و ۱-۱۱ توجه کنید.

مثال ۱-۱۰: نشان دهید که چگونه ثبات پرچم بوسیله جمع 38H و 2FH متأثر می‌شود.
حل:

```
MOV BH,38H ;BH= 38H
ADD BH,2FH ;add 2F to BH, now BH=67H
```

	38	0011	1000
+	2F	0010	1111
	67	0110	0111

CF = 0، زیرا رقم نقلی بعد از d7 وجود ندارد.
PF = 0، زیرا تعداد 1ها در نتیجه فرد است.
AF = 1، زیرا از d3 به d4 رقم نقلی وجود دارد.
ZF = 0، زیرا نتیجه غیر صفر است.
SF = 0، زیرا d7 در نتیجه صفر است.

مثال ۱-۱۱: نشان دهید که چگونه ثبات پرچم بوسیله برنامه زیر متأثر می‌شود.

```
MOV AL,9CH ;AL=9CH
MOV DH,64H ;DH=64H
ADD AL,DH ;now AL=0
```

	9C	1001	1100
+	64	0110	0100
	00	0000	0000

حل:
CF = 1، زیرا پس از d7 رقم نقلی وجود دارد.
PF = 1، زیرا تعداد 1ها در نتیجه زوج است.
AF = 1، زیرا از d3 به d4 رقم نقلی وجود دارد.
ZF = 1، زیرا نتیجه صفر است.
SF = 0، زیرا d7 در نتیجه صفر است.

برای جمع 16 بیتی هم طبق مثال‌های ۱-۱۲ و ۱-۱۳ مفاهیم مشابهی وجود دارد. به تفاوت بین عملیات 8 بیت و 16 بیت و تأثیرشان روی بیت‌های پرچم توجه کنید. بیت توازن فقط برای 8 بیت پایین رتبه نتیجه به حساب می‌آید و برطبق آن به 1 تنظیم می‌شود. و نیز به بیت CF توجه کنید. رقم نقلی به شرطی 1 می‌شود که در عوض d7، از d15، نقلی تولید گردد.

مثال ۱-۱۲: نشان دهید که چگونه ثبات پرچم بوسیله برنامه زیر متأثر می‌شود.

```
MOV AX,34F5H ;AX= 34F5H
ADD AX,95EBH ;now AX= CAE0H
```

	34F5	0011	0100	1111	0101
+	95EB	1001	0101	1110	1011
	CAE0	1100	1010	1110	0000

حل:

CF = 0 ، زیرا بعد از d15 رقم نقلی وجود ندارد
 PF = 0 ، زیرا تعداد 1 های بایت پایین فرد است
 AF = 1 ، زیرا از d3 به d4 رقم نقلی وجود دارد
 ZF = 0 ، زیرا نتیجه غیر صفر است
 SF = 1 ، زیرا d15 در نتیجه، 1 است

مثال ۱۳-۱. چگونه متاثر شدن ثبات پرچم را با برنامه زیر نشان دهید.

```
MOV BX,AAAAH ;BX=AAAAH
ADD BX,5556H ;now BX= 0000H
```

حل:

	AAAA	1010	1010	1010	1010
+	5556	0101	0101	0101	0110
	0000	0000	0000	0000	0000

CF = 1 ، زیرا بعد از d15 ، نقلی وجود دارد
 PF = 1 ، زیرا در بایت پایین تعداد زوجی 1 وجود دارد.
 AF = 1 ، زیرا از بیت d3 به d4 رقم نقلی موجود است
 ZF = 1 ، زیرا نتیجه صفر است
 SF = 0 ، زیرا d15 در نتیجه، صفر است

به وضعیت پرچم صفر (ZF) پس از اجرای دستور ADD توجه کنید. چون نتیجه عمل 16 بیتی صفر است (یعنی BX) پس ZF در 1 نشانده می شود. آیا همه دستورات بیت های پرچم را متاثر می سازند؟ پاسخ منفی است، و در واقع دستوراتی همچون دستورات انتقال داده (MOV) هیچ پرچمی را تحت تأثیر قرار نمی دهند. بعنوان تمرین، این سه مثال را با DEBUG اجرا کرده و اثر دستورات مختلف را روی ثبات پرچم ملاحظه کنید.

مثال ۱۴-۱: نشان دهید که چگونه ثبات پرچم با برنامه زیر متاثر می شود.

```
MOV AX,94C2H ;AX=94C2H
MOV BX,323EH ;BX=323EH
ADD AX,BX ;now AX=C700H
MOV DX,AX ;now DX=C700H
MOV CX,DX ;now CX=C700H
```

حل:

	94C2	1001	0100	1100	0010
+	323E	0011	0010	0011	1110
	C700	1100	0111	0000	0000

پس از عمل ADD، بیت های پرچم بقرار زیراند
 CF = 0 ، زیرا بعد از d15 ، نقلی وجود ندارد
 PF = 1 ، زیرا تعداد 1 های بایت پایین تر زوج است
 AF = 1 ، زیرا از d3 به d4 رقم نقلی وجود دارد
 ZF = 0 ، زیرا نتیجه صفر نیست
 SF = 1 ، زیرا d15 در نتیجه، 1 است

اجرای دستورات مثال ۱۴-۱ در DEBUG ثابت می کند که دستورات MOV تأثیری روی پرچم ها ندارند. چگونگی استفاده از این بیت های پرچم در فصل های بعدی بحث خواهد شد. در پیوست (ب) اطلاعات اضافی دیگری درباره تأثیر انواع دستورات روی پرچم ها آورده شده است.

استفاده از پرچم صفر در حلقه

یکی از کاربردهای گسترده ثبات پرچم استفاده از پرچم صفر برای پیاده سازی حلقه های برنامه است. کلمه

حلقه به مجموعه‌ای از دستورات گفته می‌شود که بدفعات معینی تکرار گردند. مثلاً برای جمع 5 بایت داده، می‌توان از شمارنده‌ای برای شمارش دفعاتی که حلقه باید تکرار گردد استفاده کرد. هر بار که جمع انجام شود، شمارنده یک واحد کم و بیت صفر چک می‌گردد. وقتی که شمارنده صفر شود پرچم صفر 1 خواهد شد ($ZF=1$) و حلقه متوقف می‌گردد. برنامه زیر مفهوم پیاده‌سازی حلقه در برنامه را نشان می‌دهد، که در آن 5 بایت داده با هم جمع شده‌اند. ثبات CX برای نگهداشتن شمارش و BX برای اشاره‌گر تفاوت مکان بکار می‌روند (SI و DI هم می‌توانند بکار روند). AL قبل از شروع حلقه صفر می‌شود. در هر تکرار، ZF بوسیله دستور JNZ چک می‌گردد. JNZ بمعنی "پرش اگر نتیجه صفر نیست" می‌باشد و به بیان دیگر اگر $ZF=0$ است به آدرس جدید پرش شود. اگر $ZF=1$ باشد، پرش صورت نمی‌گیرد و دستور بعد از پرش اجرا خواهد شد. توجه کنید که دستور JNZ باید بلافاصله بعد از دستور کاهش CX آورده شود زیرا JNZ می‌باید اثر "DEX CX" را روی پرچم Z تست نماید. اگر هر دستوری بین آنها واقع شود، آن دستور ممکن است پرچم صفر را متأثر کند.

MOV	CX,05	;CX holds the loop count
MOV	BX,0200H	;BX holds the offset data address
MOV	AL,00	;initialize AL
ADD_LP: ADD	AL,[BX]	;add the next byte to AL
INC	BX	;increment the data pointer
DEC	CX	;decrement the loop counter
JNZ	ADD_LP	;jump to next iteration if counter not zero

سؤالات مروری

- ۱- کدام ثبات‌ها برای دستیابی به پشته استفاده می‌شوند؟
- ۲- با هر دستور PUSH، ثبات اشاره‌گر پشته SP، 2 بار افزایش / کاهش می‌یابد.
- ۳- با هر دستور POP، SP دوباره افزایش / کاهش می‌یابد.
- ۴- سه آدرس منطقی ممکن را برای آدرس فیزیکی 143F0 بنویسید.
- ۵- دستور ADD کدام بیت ثبات پرچم را متأثر می‌کند؟
- ۶- پرچم نقلی در یک دستور ADD هشت بیتی بشرطی 1 می‌شود که از بیت ... رقم نقلی وجود داشته باشد.
- ۷- پرچم نقلی در یک دستور ADD شانزده بیتی به شرطی 1 می‌شود که از بیت ... رقم نقلی وجود داشته باشد.

بخش ۶-۱: روش‌های آدرس‌دهی 80x86

CPU به روش‌های مختلف می‌تواند عملوندها (داده) را دستیابی کند. تعداد روش‌های آدرس‌دهی هنگام طراحی CPU مشخص می‌گردد و تغییر هم نمی‌کند. 80x86 جمعاً هفت روش آدرس‌دهی جدا از هم دارد.

- ۱- ثباتی
- ۲- فوری
- ۳- مستقیم
- ۴- غیرمستقیم ثباتی
- ۵- نسبی پایه
- ۶- نسبی اندیسی
- ۷- نسبی اندیس دار پایه

هر روش آدرس‌دهی در زیر توضیح داده شده است و مثال‌های کاربردی هم در فصل‌های بعدی هنگامی که خواننده زبان اسمبلی را بهتر بداند ارائه می‌گردد. چون خواننده در حال حاضر با دستورات MOV و ADD آشناست، از آنها برای توضیح روش‌های آدرس‌دهی استفاده می‌شود.

روش آدرس دهی ثابتی

در روش آدرس دهی ثابتی، از ثبات ها برای نگهداری داده مورد دستکاری استفاده می شود. در هنگام اجرای این دستور حافظه دستیابی نمی شود؛ بنابراین نسبتاً سریع است، مثال های روش آدرس دهی ثابتی در زیر آمده است:

```
MOV  BX,DX      ;copy the contents of DX into BX
MOV  ES,AX      ;copy the contents of AX into ES
ADD  AL,BH      ;add the contents of BH to contents of AL
```

باید متذکر شد که ثبات های مبدأ و مقصد باید از لحاظ سائز همخوانی داشته باشند. به بیان دیگر نوشتن "MOV CL,AX" خطا تولید می کند، زیرا مبدأ ثبات 16 بیتی و مقصد ثبات 8 بیتی است.

روش آدرس دهی فوری

در روش آدرس دهی فوری، عملوند مبدأ یک مقدار ثابت است. همانطور که از نام روش مشخص است، وقتی دستور اسمبل شود، عملوند بلافاصله بعد از کد عمل خواهد آمد. به این دلیل این دستور خیلی سریع اجرا می شود. با این وجود برنامه نویسی با آن کاربرد محدودی دارد. روش آدرس دهی فوری برای بار کردن اطلاعات در هر یک از ثبات ها بجز ثبات های قطعه و ثبات پرچم بکار می رود.

```
MOV  AX,2550H   ;move 2550H into AX
MOV  CX,625     ;load the decimal value 625 into CX
MOV  BL,40H     ;load 40H into BL
```

برای انتقال اطلاعات به ثبات قطعه، داده ابتدا به یک ثبات همه منظوره و سپس به ثبات قطعه منتقل می شود. مثلاً:

```
MOV  AX,2550H
MOV  DS,AX
```

به بیان دیگر کد زیر خطا تولید می نماید.

```
MOV  DS,0123H   ;illegal!!
```

در دو روش آدرس دهی اول، عملوند در داخل ریزپردازنده یا همراه دستور است. در بسیاری از برنامه ها، داده مورد پردازش اغلب جایی خارج از CPU و در واقع در مکانی از حافظه قرار دارد. روش های متعددی برای دستیابی به داده در قطعه داده موجود است. در زیر این روش ها معرفی می شوند.

روش آدرس دهی مستقیم

در روش آدرس دهی مستقیم، داده در مکانی از حافظه است و آدرس داده در حافظه بلافاصله بعد از دستور می آید. توجه کنید که در روش فوری، خود عملوند همراه دستور آورده شده در صورتی که در روش آدرس دهی مستقیم، آدرس عملوند با دستور آمده است. این آدرس، آدرس تفاوت مکان است و می توان آدرس فیزیکی را با نقل مکان به چپ ثبات DS و جمع آن با تفاوت مکان بطریق زیر محاسبه کرد.

```
MOV  DL,[2400]  ;move contents of DS:2400H into DL
```

در این حالت، آدرس فیزیکی با ترکیب محتوای تفاوت مکان 2400 با DS یعنی ثبات قطعه داده محاسبه می گردد. به کروش دور آدرس توجه کنید. در غیاب این کروش خطا تولید می شود زیرا تفسیر چنین خواهد بود که 2400 شانزده بیتی بداخل ثبات DL که 8 بیتی است برود. مثال ۱۵-۱ را برای تأکید بر این روش ملاحظه کنید.

مثال ۱۵-۱: آدرس فیزیکی مکان حافظه و محتوای آن را پس از اجرای کدهای زیر با فرض $DS = 1512H$ پیدا کنید.

```
MOV AL,99H
MOV [3518],AL
```

حل:

ابتدا AL با 99 مقداردهی اولیه می‌شود، سپس در خط دوم محتوای AL به آدرس منطقی 3518 : DS که 1512 است انتقال می‌یابد. با شیفت به چپ DS و جمع آن با تفاوت مکان آدرس فیزیکی $18638H (15120H + 3518H)$ بدست می‌آید. این بدان معنی است که پس از اجرای دستور دوم، مکان حافظه‌ای با آدرس 18638H حاوی 99H خواهد بود.

روش آدرس دهی غیر مستقیم ثباتی

در روش آدرس دهی غیر مستقیم ثباتی، آدرس مکان حافظه‌ای که عملوند در آن است بوسیله ثبات نگهداشته می‌شود. ثبات هایی که برای این هدف بکار می‌روند عبارتند از SI، DI، BX. اگر این سه ثبات بعنوان اشاره گر بکار روند، یعنی اگر آنها تفاوت مکان حافظه را نگهدارند، باید با DS ترکیب شوند تا آدرس فیزیکی 20 بیتی را تولید نمایند. مثلاً:

```
MOV AL,[BX] ;moves into AL the contents of the memory location
               pointed to by DS:BX.
```

توجه کنید که BX در کروش است. در غیاب کروش چنین تفسیر می‌شود که دستوری محتوای ثبات BX را به AL منتقل می‌نماید (که خطایی تولید می‌کند زیرا مبدأ و مقصد مطابقت ندارند)، در صورتی که محتوای حافظه‌ای که آدرس تفاوت مکانش در BX است باید انتقال یابد. قانون مشابهی به هنگام استفاده از SI و DI وجود دارد.

```
MOV CL,[SI] ;move contents of DS:SI into CL
MOV [DI],AH ;move contents of AH into DS:DI
```

در مثال فوق، داده انتقالی سایز بایت دارد. مثال ۱۶-۱ عملوند 16 بیتی را نشان می‌دهد.

مثال ۱۶-۱: فرض کنید که $DS=1120$ ، $SI=2498$ و $AX=17FE$ باشد. پس از اجرای دستور زیر محتوای مکانهای حافظه را نشان دهید.

```
MOV [SI],AX
```

حل:

محتوای AX به مکانهای حافظه‌ای با آدرس منطقی DS:SI و $DS:SI+1$ منتقل می‌شود؛ بنابراین، آدرس فیزیکی از جمع DS شیفت یافته با SI که برابر با 13698 است، حاصل می‌گردد. بر طبق قرارداد مبتنی بر کوچکتر، آدرس پایین 13698H حاوی FE، یعنی بایت پایین، و آدرس بالا یعنی 13699 حاوی 17 در بایت بالاتر خواهد بود.

روش آدرس دهی نسبی پایه

در این روش، ثبات‌های پایه BX و BP و نیز مقدار جابجایی برای محاسبه آنچه که آدرس مؤثر خواننده می‌شود، بکار می‌روند، قطعات پیش فرض که برای محاسبه آدرس فیزیکی (PA) بکار می‌روند عبارتند از DS برای BX و SS برای BP. مثلاً:

```
MOV CX,[BX]+10 ;move DS:BX+10 and DS:BX+10+1 into CX
                 ;PA = DS (shifted left) + BX + 10
```

کد جایگزین، "MOV CX,[BX+10]" یا "MOV CX,10[BX]" می‌باشد. در مرتبه محتوای آدرس پایین به CL و محتوای آدرس بالا به CH خواهد رفت. در حالت ثبات BP،

```
MOV AL,[BP]+5 ;PA = SS (shifted left) + BP + 5
```

دوباره کدهای جایگزین "MOV AL,[BP+5]" یا "MOV AL,5[BP]" خواهند بود. بهتر است مختصری

در رابطه با اصطلاح آدرس مؤثر که در بروشورهای اینتل بکار رفته است صورت گیرد. در "MOV BP+5, AL, [BP]+5" آدرس مؤثر خوانده می‌شود، زیرا بایت پنجم از شروع تفاوت مکان BP به دنبال AL انتقال پیدا کرده است. بطور مشابه در MOV CX, [BX]+10، آدرس مؤثر نام گرفته است.

روش آدرس دهی نسبی اندیس دار

این روش مثل نسبی پایه‌ای کار می‌کند غیر اینکه ثبات‌های DI و SI آدرس تفاوت مکان را نگه می‌دارند. مثلاً:

```
MOV DX, [SI]+5 ;PA = DS (shifted left) + SI + 5
MOV CL, [DI]+20 ;PA = DS (shifted left) + DI + 20
```

مثال ۱۷-۱۶ مثالهای بیشتری از این روش آدرس دهی را ارائه می‌دهد.

مثال ۱۷-۱۶: فرض کنید که DS=4500، SS=2000، BX=2100، DI=8500، BP=7814 و AX=2512 باشد. آدرس فیزیکی مکانی که AX در آن ذخیره می‌شود را برای هر یک از موارد زیر نشان دهید، همه مقادیر به مبنای شانزده هستند.

MOV [SI] + 10, AX (ب) MOV [BX] + 20, AX (الف)
MOV [BP] + 12, AX (ت) MOV [DI] + 4, AX (پ)

حل:

در هر حالت داریم: جابجایی + ثبات تفاوت مکان + (شیفت یافته به چپ) ثبات قطعه = PA

DS:BX+20 location 47120 = (12) and 47121 = (25) (الف)
DS:SI+10 location 46496 = (12) and 46497 = (25) (ب)
DS:DI + 4 location 4D504 = (12) and 4D505 = (25) (پ)
SS:BP+12 location 27826 = (12) and 27827 = (25) (ت)

روش آدرس دهی نسبی اندیس دار پایه

از ترکیب دو روش پایه و اندیس دار روش آدرس دهی جدید به نام روش اندیس دار پایه حاصل می‌شود. در این روش یک ثبات پایه و یک ثبات اندیس بکار می‌رود. مثال:

```
MOV CL, [BX][DI]+8 ;PA = DS (shifted left) + BX + DI + 8
MOV CH, [BX][SI]+20 ;PA = DS (shifted left) + BX + SI + 20
MOV AH, [BP][DI]+12 ;PA = SS (shifted left) + BP + DI + 12
MOV AH, [BP][SI]+29 ;PA = SS (shifted left) + BP + SI + 29
```

دستورات فوق ممکن است تغییر نمایند؛ مثلاً آخرین مثال را می‌توان چنین نوشت:

```
MOV AH, [BP+SI+29]
or
MOV AH, [SI+BP+29] ;the register order does not matter.
```

توجه کنید که جابجایی [SI][DI]+ MOV AX, غیر مجاز است. در بسیاری از مثال‌های فوق، دستور MOV به منظور واضح تر بودن بکار رفته است، با این وجود می‌توان هر دستور دیگری را بکار برد. مثلاً دستور ADD DL, [BX] محتوای مکانی که با DS:BX مشخص می‌شود را با محتوای ثبات DL جمع می‌کند.

نام ثبات :	CS	DS	ES	SS
ثبات های تغییر مکان :	IP	SI, DI, BX	SI, DI, BX	SP, BP

جدول ۱-۳. ثبات های تفاوت مکان برای قطعات مختلف

دستورالعمل	قطعه بکار رفته	قطعه پیش فرض
MOV AX,CS:[BP]	CS:BP	SS:BP
MOV DX,SS:[SI]	SS:SI	DS:SI
MOV AX,DS:[BP]	DS:BP	SS:BP
MOV CX,ES:[BX]+12	ES:BX+12	DS:BX+12
MOV SS:[BX][DI]+32,AX	SS:BX+DI+32	DS:BX+DI+32

جدول ۱-۴. تعدادی مثال لغو (حذف)

لغو قطعه

جدول ۱-۳ خلاصه ای از ثبات های قطعه را که بوسیله چهار ثبات قطعه 80x86 استفاده می شوند نشان می دهد. پردازنده 80x86 به برنامه اجازه می دهد تا قطعه پیش فرض را لغو (حذف) نموده و هر ثبات قطعه دیگری را بکار گیرد. برای انجام آن قطعه را در کد برنامه مشخص کنید.

مثلاً در "MOV AL, [BX]"، آدرس فیزیکی عملوندی که به AL می رود در DS:BX است، که در آن همانطور که قبلاً دیدیم DS قطعه پیش فرض برای اشاره گر BX می باشد. برای حذف آن، قطعه مورد نظر خود را در دستور بصورت "MOV AL, ES:[BX]" مشخص نمایید. اکنون آدرس عملوندی که قرار است به AL برود بعوض DS:BX، در ES:BX می باشد. در فصل های بعدی و در مثال های برنامه نویسی به میزان قابل توجهی از این گونه روش های آدرس دهی ملاحظه خواهد شد. جدول ۱-۴ مثال های بیشتری از قطعات حذف شده ای را در کنار آدرس های پیش فرض نشان می دهد. جدول ۱-۵ روش های آدرس دهی 8088/86 را نشان می دهد.

روش آدرس دهی	عملوند	قطعه پیش فرض
ثباتی	reg	none
فوری	data	none
مستقیم	[offset]	DS
غیر مستقیم ثباتی	[BX]	DS
	[SI]	DS
	[DI]	DS
نسبی پایه	[BX]+disp	DS
	[BP]+disp	SS
نسبی اندیس دار	[DI]+disp	DS
	[SI]+disp	DS
نسبی اندیس دار پایه	[BX][SI]+disp	DS
	[BX][DI]+disp	DS
	[BP][SI]+ disp	SS
	[BP][DI]+ disp	SS

جدول ۱-۵ خلاصه ای از روش های آدرس دهی 80x86

ریزپردازنده‌های خانواده اینتل در همه کامپیوترهای IBM PC، PS و سازگار با آنها بکار رفته‌اند. در PCهای اولیه، 8088 بوسیله IBM بکار گرفته شد که صنعت کامپیوتر را در اول دهه ۱۹۸۰ متحول کرد. در هر نسل از ریزپردازنده‌های اینتل اصلاحاتی از نظر سرعت و قدرت پردازش بعمل آمد.

یک برنامه زبان اسمبلی نمونه متشکل از حداقل سه قطعه است. قطعه کد حاوی دستورات زبان اسمبلی است. قطعه داده هم برای ذخیره داده لازم برای برنامه بکار می‌رود. قطعه پشته برای ذخیره موقت داده بکار گرفته می‌شود. حافظه درون هر قطعه با ترکیب یک ثبات قطعه و ثبات تفاوت مکان بدست می‌آید. از ثبات پرچم شناسایی شرایطی معین، پس از اجرای یک دستور استفاده می‌شود. این شناسایی شامل پرچم‌های نقلی، سرریز و نتیجه صفر است.

دستورات زبان اسمبلی یکی از هفت روش آدرس‌دهی را بکار می‌برد. یک روش آدرس‌دهی روشی است که برنامه‌نویس به CPU محل برداشتن عملوند را نشان می‌دهد.

مسائل

- ۱- کدام یک از دو ریزپردازنده 8088 یا 8086، اول معرفی شد؟
 - ۲- اگر هر دو ریز پردازنده 80286 و SX 80386 دارای گذرگاه بیرونی 16 بیت باشند، تفاوت میان آنها چیست؟
 - ۳- ریزپردازنده 16 بیتی یا 32 بیتی بچه معنی است؟ آیا اختلاف در گذرگاه داده درونی یا بیرونی است؟
 - ۴- آیا برنامه‌های نوشته شده برای 88/86 روی CPUهای 80286، 80386 و 80486 اجرا می‌شوند؟
 - ۵- سازگاری رو به بالا چه معنی می‌دهد؟
 - ۶- تفاوت‌های عمده بین 8088 و 8086 چیست؟
 - ۷- کدام یک از پردازنده‌های 8088 یا 8086 صف طولانی‌تری دارند؟
 - ۸- بجز افزایش فرکانس، چه راه دیگری برای بالا بردن سرعت می‌شناسید؟
 - ۹- "BIU" و "EU" بچه معنی هستند و کار آنها چیست؟
 - ۱۰- ثبات‌های همه منظوره 8086/86 را نام ببرید.
- (الف) 8 بیت (ب) 16 بیت
- ۱۱- کدام یک از ثبات‌های زیر نمی‌تواند به دو بایت بالا و پایین تقسیم شود؟
- | | | |
|----------|--------|--------|
| (الف) CS | (ب) AX | (پ) DS |
| (ت) SS | (ث) BX | (ج) DX |
| (ج) CX | (ح) SI | (خ) DI |
- ۱۲- کدام یک از دستورات زیر نمی‌توانند در زبان اسمبلی 8088/86 کد شوند. همه اعداد را در مبنای 16 تصور کنید.
- | | | |
|------------------|-----------------|-----------------|
| (الف) MOV AX,27 | (ب) MOV AL,97F | (پ) MOV DS,9BF2 |
| (ت) MOV CX,397 | (ث) MOV SI,9516 | (ج) MOV CS,3490 |
| (چ) MOV DS,BX | (ح) MOV BX,CS | (خ) MOV CH,AX |
| (د) MOV AX,23FB9 | (ذ) MOV CS,BH | (ز) MOV AX,DL |
- ۱۳- نام ثبات‌های قطعه و وظیفه آنها را در 8088/86 تعریف کنید.
 - ۱۴- اگر $CS = 3499H$ و $IP = 2500H$ باشند، پیدا کنید:
- (الف) آدرس منطقی (ب) آدرس فیزیکی (پ) محدوده پایین و بالای قطعه کد.
- ۱۷- مسئله ۱۶ را با استفاده از $DS = 1298H$ و تفاوت مکان $7CC8H$ تکرار کنید.
 - ۱۸- فرض کنید که آدرس فیزیکی برای مکانی، 0046CH باشد. یک آدرس منطقی ممکن را پیشنهاد کنید.

- ۱۹- اگر دستورالعملی از آدرس فیزیکی 389F2 با CS = 2700 برداشت بشود، آیا محدوده قطعه در آن منظور شده است یا خیر؟ اگر نه، چه مقداری باید به CS نسبت داده شود اگر IP = 1282 باشد؟
- ۲۰- با استفاده از DEBUG برنامه زیر را اسمبل و عکس اسمبل کرده و آدرس منطقی، آدرس فیزیکی و محتوای هر مکان را فراهم سازید. مقدار CS بوسیله DOS معین می شود، ولی IP = 170H را بکار ببرید.

```
MOV AL,76H
MOV BH,8FH
ADD BH,AL
ADD BH,7BH
MOV BL,BH
ADD BL,AL
```

- ۲۱- مسئله ۲۰ را برای برنامه زیر از صفحه 29 تکرار کنید.

```
;clear AL
;add the contents of DS:200 to AL
;add the contents of DS:201 to AL
;add the contents of DS:202 to AL
;add the contents of DS:203 to AL
;add the contents of DS:204 to AL
```

- ۲۲- پشته عبارتست از:

(الف) بخشی از ROM
(ب) بخشی از RAM برای ذخیره موقت
(پ) یک ثابت 16 بیتی درون CPU
(ت) مقداری حافظه در داخل CPU.

- ۲۳- در مسئله ۲۲، پاسخ صحیح را برای اشاره گر پشته انتخاب نمایید.

- ۲۴- هنگام درج داده در پشته اشاره گر پشته ... می شود، ولی وقتی داده از پشته بازخوانی شود اشاره گر پشته ... می گردد.

- ۲۵- پاسخ صحیح را انتخاب کنید:

(الف) قطعه پشته و قطعه کد از یک نقطه حافظه خواندن/نوشتن شروع شده و هر دو بسمت بالا افزایش می یابند.

(ب) قطعه پشته و قطعه کد از نقاط مقابلی در حافظه نوشتن/خواندن شروع شده و بسمت یکدیگر تغییر می نمایند.

(پ) اگر قطعه کد و قطعه پشته بهم برسند مشکلی پیش نمی آید.

- ۲۶- عیب اصلی پشته بعنوان ذخیره ساز موقت در مقایسه با تعداد زیادی ثابت در درون CPU چیست؟

- ۲۷- اگر SS = 2000 و SP = 4578 باشد، پیدا کنید:

(الف) آدرس فیزیکی

(ب) آدرس منطقی

(پ) محدوده پایین قطعه پشته

(ت) محدوده بالای قطعه پشته

- ۲۸- اگر SP = 24FC باشد، آدرس تفاوت مکان اولیه مکان پشته برای درج داده چیست؟

- ۲۹- فرض کنید که SP = FF2EH، AX = 3291H، BX = F43CH و CX = 09 باشد. محتوای پشته و اشاره گر پشته پس از اجرای هر دستور زیر را پیدا کنید.

```
PUSH AX
PUSH BX
PUSH CX
```

- ۳۰- برای اینکه هر ثابت در مسئله ۲۹ مقدار اولیه خود را بگیرد، رشته دستورات لازم را که باید اجرا شوند بنویسید. محتوای SP در هر نقطه چیست؟

۳۱- ثبات‌های زیر بعنوان تفاوت مکان بکار می‌روند. با فرض اینکه قطعه پیش فرض برای بدست آوردن آدرس منطقی بکار رفته باشد، ثبات قطعه مربوط به هر تفاوت مکان را معین کنید.

BP (الف) DI (ب) IP (پ)
SI (ت) SP (ث) BX (ج)

۳۲- ثبات قطعه لغو و ثبات قطعه پیش فرض (اگر لغو وجود نداشته باشد) را در هر حالت نشان دهید.

MOV SS:[DI],BX (ب) MOV SS:[BX],AX (الف)
MOV DX,DS:[BP+6] (پ)

۳۳- وضعیت CF، PF، AF، ZF و SF را برای اعمال زیر مشخص کنید.

MOV DX,10FFH (پ) MOV AL,23H (ب) MOV BL,9FH (الف)
ADD DX,1 ADD AL,97H ADD BL,61H

۳۴- فرض کنید که ثبات‌ها دارای مقادیر زیر هستند (به مبنای 16): CS = 1000، DS = 2000، SS = 3000، DI = 5000، SI = 4000، BX = 6080، BP = 7000، AX = 25FF، CX = 8791 و DX = 1299. آدرس فیزیکی حافظه‌ای که عملوند در آن ذخیره شده و محتوای مکان‌های حافظه در هر یک از مثال‌های آدرس‌دهی زیر را محاسبه کنید.

MOV [SI+BX+8],AH (ب)	MOV [SI],AL (الف)
MOV [DI+6],BX (ت)	MOV [BX],AX (پ)
MOV [BP][SI]+10,DX (ج)	MOV [DI][BX]+28,CX (ث)
MOV [BX]+30,DX (ح)	MOV [3600],AX (چ)
MOV [BP+SI+100],BX (د)	MOV [BP]+200,AX (خ)
MOV [DI+BP+100],AX (ر)	MOV [SI]+50,AH (ذ)

۳۵- روش آدرس‌دهی را برای هر یک از موارد زیر معین کنید.

MOV BX,5678H (ب)	MOV AX,DS (الف)
MOV AL,CH (ت)	MOV CX,[3000] (پ)
MOV AL,[BX] (ج)	MOV [DI],BX (ث)
MOV CX,DS (ح)	MOV DX,[BP+DI+4] (چ)
MOV AH,[BX+SI+50] (د)	MOV [BP+6],AL (خ)
MOV [BP][SI]+12,AX (ر)	MOV BL,[SI]+10 (ذ)

۳۶- محتوای مکان‌های حافظه پس از اجرای هر دستور را نشان دهید.

MOV DX,8C63H (ب)	MOV BX,129FH (الف)
MOV [2348],DX	MOV [1450],BX
DS:2348	DS:1450
DS:2349	DS:1451

✓ پاسخ به سؤالات مروری

بخش ۱-۱: تاریخچه خلاصه‌ای از خانواده 80x86

۱- (۱) افزایش ظرفیت حافظه از 64K به 1M بایت؛ (۲) 8086 در عوض 8 بیت، یک پردازنده 16 بیت است؛ (۳) 8086 یک پردازنده خط لوله است.

۲- 8088 دارای گذرگاه داده 8 بیت است در صورتی که 8086، دارای گذرگاه داده 16 بیتی می‌باشد.

۳- (الف) 20 بیت، 1 مگابایت (ب) 24 بیت، 16 مگابایت (پ) 32 بیت، 4 گیگابایت
۴- 16، 32

۵- 80386 دارای گذرگاه‌های آدرس و داده 32 بیتی است، در صورتی که 80386SX دارای گذرگاه آدرس 24 بیت و گذرگاه داده بیرونی 16 بیت است.

۶- حافظه مجازی، مد حفاظت شده

۷- پردازنده کمکی محاسباتی در تراشه CPU، حافظه نهان (کش) و کنترل‌گر

بخش ۲-۱، درون 8088/86

- ۱- واحد اجرا دستورات را اجرا می‌کند، واحد ارتباط (اتصال) گذرگاه، دستورات را برداشت می‌کند.
- ۲- خط لوله ریز پردازنده را به دو بخش تقسیم می‌کند: واحد اجرا و واحد ارتباط گذرگاه؛ این ساختار به CPU اجازه می‌دهد تا این دو عمل را بطور همزمان انجام دهد؛ یعنی، BIU می‌تواند دستورات را برداشت کند در حالی که EU دستورات برداشت شده قبلی را اجرا می‌نماید.

۳- 8، 16

۴- FR، IP، ES، SS، DS، CS، DI، SI، BP، SP، DX، CX، BX، AX

بخش ۳-۱: مقدمه‌ای بر برنامه‌نویسی اسمبلی

۱- MOV BX, 1234H

۲- MOV AX, 16H

ADD, AX, ABH

۳- ثبات‌های قطعه CS، DS، ES و SS

۴- FF = 25510، FFFFH = 6553510

بخش ۴-۱: مقدمه‌ای بر قطعات برنامه

۱- 64K

۲- یک قطعه حاوی 64K بایت است؛ به زیر 346E0H قابل تقسیم بر 16 است.

۳- آدرس فیزیکی یک آدرس 20 بیت است که روی گذرگاه آدرس قرار می‌گیرد تا یک بایت را تخصیص دهد؛ آدرس منطقی آدرسی بفرم yyyy : xxxx است که در آن xxxx آدرس قطعه و yyyy تفاوت مکان در قطعه است.

۴- 16، 20

۵- IP

۶- 2400 حاوی 34 و 2401 حاوی 12 است.

بخش ۵-۱: مطالبی بیشتر درباره قطعات در 80x86

۱- SS ثبات قطعه است، SP و BP بعنوان اشاره گر در پشته بکار می‌روند.

۲- کاهش

۳- افزایش

۴- 1410:02F0، 1000 : 43F0، 143F : 0000

۵- OF، SF، ZF، AF، PF، CF

۶- 7

۷- 15



برنامه‌نویسی به زبان اسمبلی

اهداف

در پایان این فصل قادر خواهید بود:

- « تفاوت میان دستورات زبان اسمبلی و شبه دستورات را بدانید.
- « قطعات یک برنامه به زبان اسمبلی را مشخص کنید.
- « دستورات ساده زبان اسمبلی را کد کنید.
- « یک برنامه زبان اسمبلی ساده را اسمبل، پیوند و اجرا نمایید.
- « دستورات کنترلی مانند پرش‌های شرطی و غیرشرطی و دستورات فراخوانی را کد نویسی نمایید.
- « داده‌های رهنمون زبان اسمبلی را به دودویی، مبنای شانزده و یا اسکی کد نویسی کنید.
- « یک برنامه زبان اسمبلی را با استفاده از تعریف قطعه کامل یا قطعه ساده بنویسید.
- « تفاوت میان فایل‌های COM و EXE را بیان کرده و مزایای هر یک را لیست کنید.

این فصل مقدمه‌ای بر برنامه‌نویسی زبان اسمبلی با 80x86 می‌باشد. ابتدا فرم مبنایی برای یک برنامه توضیح داده شده است، و بدنبال آن مراحل ویرایش، اسمبل کردن، پیوند و اجرای یک برنامه آمده است. سپس دستورات انتقال کنترل مانند پرش و فراخوانی بحث شده‌اند و آنگاه انواع داده‌ها و رهنمون‌های داده در زبان اسمبلی توضیح داده شده‌اند. پس از آن تعریف قطعه ساده بحث شده است. نهایتاً اختلاف بین فایل "exe" و "com" توضیح داده شده است. برنامه‌های این فصل و فصل بعد قابل اسمبل شدن و اجرا بر روی هر کامپیوتر PS، IBM PC و سازگار با پردازنده‌های 8088، 8086، 80286، 80386 و 80486 می‌باشند.

بخش ۱-۲: رهنمون‌ها و یک برنامه نمونه

در این بخش اجزاء یک برنامه ساده به زبان اسمبلی که قرار است با اسمبلر، اسمبل گردد بحث خواهد شد. یک برنامه مفروض (شکل ۱-۲) مجموعه‌ای از عبارات، یا خطوط حاوی دستورات زبان اسمبلی همچون MOV، ADD، یا عباراتی که رهنمون‌ها خوانده می‌شوند می‌باشد. رهنمون‌ها که شبه دستور هم خوانده می‌شوند در مورد ترجمه دستورات زبان اسمبلی به کد ماشین، به اسمبلر جهت می‌دهند. یک دستور العمل زبان اسمبلی دارای چهار میدان است:

[توضیحات]: [عملوند] نماد رمزی [برچسب]

گروه‌ها بمعنی اختیاری بودن آن است.

- ۱- میدان برچسب اجازه می‌دهد تا با یک نام به خطی ارجاع داده شود. میدان برچسب از 31 کاراکتر تجاوز نمی‌کند. برچسب برای رهنمون‌ها نیازی ندارند تا با دو نقطه ختم شوند. یک برچسب وقتی که به کد عملی که تولید دستور می‌کند اشاره نماید با دو نقطه ختم می‌شود. دو نقطه به اسمبلر می‌فهماند که به کدی در این قطعه کد اشاره می‌شود. پیوست (پ) بخش ۲ اطلاعات بیشتری در مورد برچسب‌ها در اختیار می‌گذارد.
- ۲ و ۳- میدان نماد رمزی زبان اسمبلی (دستور) و عملوندها کار واقعی برنامه را اجرا کرده وظایفی که برنامه بخاطر آن‌ها نوشته شده است را انجام می‌دهند. در عبارت زبان اسمبلی مانند

```
ADD AL,BL
MOV AX,6764
```

ADD و MOV نماد رمزی کدهای عملیات و "AL, BL" و "AX, 6764" عملوندها هستند. این دو میدان می‌توانند در عوض نماد رمزی یا عملوند، حاوی شبه دستورات اسمبلر باشند. این شبه دستورات توسط اسمبلر برای سازمان‌دهی برنامه و دیگر فایل‌های خروجی استفاده می‌شوند. رهنمون‌ها هیچگونه کد ماشینی تولید نمی‌کنند و فقط بوسیله اسمبلر استفاده می‌شوند. این برخلاف دستورالعمل‌هاست که برای اجرا در CPU به کد ماشین ترجمه می‌شوند. در شکل ۱-۲ فرمان‌های SEGMENT، DB، ENDS، ASSUME، END و ENDP مثال‌هایی از رهنمون هستند.

- ۴- میدان توضیحات با ";" شروع می‌شود. توضیحات ممکن است در انتهای یک خط بوده و یا یک خط را تشکیل دهند. اسمبلر توضیحات را چشم‌پوشی می‌کند. ولی برای برنامه‌نویس مفیدند. گرچه توضیحات اختیاری‌اند ولی استفاده از آنها برای خواندن و درک برنامه پیشنهاد می‌شود.

قطعه یک برنامه

گرچه هر کس می‌تواند برنامه‌ای به زبان اسمبلی بنویسد بطوری که فقط یک قطعه را بکار برد، ولی معمولاً یک برنامه شامل حداقل سه قطعه بحث شده در فصل ۱ می‌باشد که عبارتند از: قطعه پشته، قطعه داده و قطعه کد. رهنمون‌های "SEGMENTS" و "ENDS" به اسمبلر شروع و پایان یک قطعه را اعلام داشته و قالب زیر را دارند:

```
label SEGMENT [options]
;place the statements belonging to this segment here
label ENDS
```

برچسب یا نام، باید از مقررات نام‌گذاری تبعیت نماید (انتهای بخش ۴-۲ ملاحظه شود) و نیز باید منحصر به فرد باشد. میدان [اختیاری] اطلاعات مهمی را به اسمبلر برای سازمان‌دهی می‌دهد ولی الزامی نیست. برچسب ENDS باید با برچسب رهنمون SEGMENT یکسان باشد. عبارات زبان اسمبلی بصورت گروهی در قطعات می‌آیند تا بوسیله اسمبلر و در نتیجه بوسیله CPU شناسایی شوند. قطعه پشته محل ذخیره‌سازی را برای پشته تعریف می‌کند، قطعه داده هم داده‌ای را که برنامه بکار خواهد برد تعریف می‌نماید و قطعه کد نیز حاوی دستورات زبان اسمبلی است. در فصل ۱ مروری بر این قطعات در حافظه نمودیم. در صفحات بعد قطعات پشته، داده و کد را آن‌طور که در برنامه‌نویسی زبان اسمبلی تعریف شده‌اند، توصیف می‌نمائیم.

تعریف قطعه پشته

قطعه پشته‌ای که در زیر نشان داده شده حاوی یک خط "DB 64 DUP(?)" است. این رهنمون ۶۴ بایت از حافظه را رزرو می‌نماید. رهنمون DB و DUP بعداً بطور مشروح توصیف شده‌اند.

STSEG	SEGMENT	;the "SEGMENT" directive begins the segment
	DB 64 DUP (?)	;this segment contains only one line
STSEG	ENDS	;the "ENDS" segment ends the segment

تعریف قطعه داده

قطعه داده در برنامه شکل ۱-۲ سه قلم داده را تعریف می‌کند: DATA1، DATA2 و SUM تعریف بصورت DB تعریف شده است. رهنمون DB بوسیله اسمبلر برای استقرار مجموعه‌های بایتی در حافظه بکار می‌رود. حافظه می‌تواند در سایزهای مختلف مانند ۲ بایت ذخیره شود. رهنمون مربوطه بصورت DW می‌باشد. بسیاری از این شبه دستورات مفصلاً در بخش ۵-۲ تشریح شده‌اند. اقلام داده تعریف شده در قطعه داده بوسیله برچسبشان در قطعه کد دستیابی می‌شوند. به DATA1 و DATA2 در بخش داده مقادیر اولیه تخصیص یافته است. به SUM مقدار اولیه داده نمی‌شود، ولی این مکان برای استفاده بعدی برنامه کنار گذاشته شده است.

THE FORM OF AN ASSEMBLY LANGUAGE PROGRAM

```

STSEG      SEGMENT
            DB 64 DUP (?)
STSEG      ENDS

DTSEG      SEGMENT
DATA1      DB 52H
DATA2      DB 29H
SUM        DB ?
DTSEG      ENDS

CDSEG      SEGMENT
MAIN       PROC FAR           ;this is the program entry point
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG      ;load the data segment address
            MOV DS,AX         ;assign value to DS
            MOV AL,DATA1      ;get the first operand
            MOV BL,DATA2      ;get the second operand
            ADD AL,BL          ;add the operands
            MOV SUM,AL        ;store result in location SUM
            MOV AH,4CH        ;set up to
            INT 21H           ;return to DOS

MAIN       ENDP
CDSEG      ENDS
            END MAIN         ;this is the program exit point

```

شکل ۱-۲. یک برنامه ساده زبان اسمبلی

تعریف قطعه کد

آخرین بخش برنامه شکل ۱-۲ قطعه کد است. اولین خط قطعه پس از رهنمون SEGMENT رهنمون PROC می‌باشد. یک رویه (procedure) گروهی از دستورات است که برای انجام عمل خاصی در نظر گرفته شده‌اند. یک قطعه کد ممکن است فقط از یک رویه سازمان یافته باشد و یا معمولاً از چندین رویه کوچکتر تشکیل بشود تا ساختار بهتری داشته باشد. هر رویه باید دارای یک نام باشد که بوسیله رهنمون PROC تعریف می‌شود. بدنبال آن دستورات زبان اسمبلی و نهایتاً رهنمون ENDP آمده است. عبارات PROC و ENDP دارای برجسب یکسان هستند. رهنمون PROC می‌تواند انتخاب نوع FAR (دور) یا NEAR (نزدیک) را داشته باشد. سیستم عاملی که کامپیوتر را کنترل می‌کند برای اجرای برنامه باید به آغاز آن هدایت شود. در DOS نقطه ورود به برنامه کاربر از نوع رویه FAR است. از آن پس، هر یک از دو نوع FAR یا NEAR می‌تواند بکار رود. تفاوت بین رویه FAR و NEAR، و اینکه چرا و کجا هر یک از آنها بکار می‌روند، بعداً در این فصل توضیح داده می‌شوند. فعلاً فقط بخاطر بسپارید که برای اجرای یک برنامه، در نقطه ورودی باید از FAR استفاده کرد.

در شکل ۱-۲، بلافاصله پس از رهنمون PROC، رهنمون ASSUME آمده است که ثبات‌های قطعه را، با فرض یکی بودن ثبات قطعه با برجسب قطعه بکار رفته در برنامه، مرتبط می‌سازد. اگر از قطعه اضافی استفاده شود، ES هم در عبارت ASSUME استفاده خواهد شد. عبارت ASSUME به این دلیل لازم است که در یک برنامه به زبان اسمبلی ممکن است چندین قطعه کد، یک، دو، سه، و یا چند قطعه داده و بیش از یک قطعه پشته وجود داشته باشد ولی هر بار فقط یکی از آنها بوسیله CPU آدرس‌دهی می‌شود زیرا هر بار فقط یکی از هر نوع ثبات قطعه در CPU قابل دستیابی است. بنابراین ASSUME به اسمبلی می‌گوید که کدام قطعات بوسیله رهنمون‌های تعریف شده بوسیله SEGMENT باید استفاده شوند. این شبه دستور، اسمبلی را هم برای محاسبه آدرس‌های تفاوت مکان از شروع قطعه کمک می‌کند. مثلاً در دستور MOV AL,[BX]، ثبات BX تفاوت مکان قطعه داده است. هنگامی که بیش از یک قطعه داده در برنامه وجود دارد اسمبلی باید به وضوح بداند که کدام قطعه داده باید بکار رود. این دقیقاً هدف استفاده از رهنمون ASSUME است.

سؤال مناسبی که در اینجا پیش می‌آید این است که اگر ASSUME فقط مربوط به اسمبلی است، در واقع چه مقادیری به ثبات‌های CS، DS و SS برای اجرای برنامه تخصیص یافته است؟ سیستم عامل DOS باید کنترل را به برنامه طوری انتقال دهد که آن را اجرا کند ولی قبل از اجرا، مقادیری به ثبات‌های قطعه تخصیص می‌دهد. سیستم عامل باید این کار را انجام دهد زیرا می‌داند که چقدر حافظه در کامپیوتر نصب شده است، چقدر از آن به وسیله سیستم مورد استفاده قرار گرفته است و چقدر باقی مانده است. در IBM PC، سیستم عامل ابتدا درمی‌یابد که چند کیلوبایت RAM نصب شده است، مقداری از آن را برای خود نگه می‌دارد، و سپس اجازه می‌دهد تا کاربر برنامه، بخشی از آن را که لازم دارد استفاده کند. انواع ویرایش‌های DOS مقادیر مختلفی از حافظه را نیاز دارند. البته باید متذکر شد که کسی نمی‌تواند به DOS بگوید که به برنامه، یک ناحیه خاصی از حافظه را اختصاص دهد (مثلاً از 25FFF تا 289E2). بنابراین، تخصیص مقدار دقیق برای ثبات‌های قطعه وظیفه DOS می‌باشد. پس از تحویل کنترل از DOS، فقط CS و SS دارای مقادیر صحیحی هستند. مقدار DS (و ES اگر بکار رفته باشد) باید بوسیله برنامه مقداردهی اولیه شود. این کار در زیر انجام شده است:

```
MOV AX,DTSEG ;DTSEG is the label for the data segment
MOV DS,AX
```

از فصل ۱ بخاطر می‌آورید که هیچ ثبات قطعه‌ای مستقیماً بار نمی‌شود. باین دلیل دو خط دستور فوق لازم است. شما نمی‌توانید کد "MOV DS,DTSEG" را بنویسید.

پس از این کارهای پرمشقت، دستورات برنامه زبان اسمبلی را می‌توان برای کار مورد نظر نوشت. در شکل ۱-۲، برنامه AL و BL را به ترتیب با DATA1 و DATA2 پر می‌کند، آنها را با هم جمع کرده و نتیجه را در

```
SUM ذخیره می‌نماید.
MOV AL,DATA1
MOV BL,DATA2
ADD AL,BL
MOV SUM,AL
```


THE FORM OF AN ASSEMBLY LANGUAGE PROGRAM

```

STSEG      SEGMENT
            DB 64 DUP (?)
STSEG      ENDS

DTSEG      SEGMENT
            ;place data here
DTSEG      ENDS

CDSEG      SEGMENT
MAIN       PROC FAR                ;this is the program entry point
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG           ;bring in the segment for data
            MOV DS,AX              ;assign the DS value

            ;place code here

            MOV AH,4CH             ;set up to
            INT 21H                ;return to DOS

MAIN       ENDP
CDSEG      ENDS
END        MAIN                    ;this is the program exit point
    
```

شکل ۲-۲. پوسته یک برنامه زبان اسمبلی

همه فرامین بالا و پائین چهار خط فوق در قطعه کد را می توان بعنوان پوسته برنامه زبان اسمبلی تصور کرد. هنگامی که چند برنامه اول را می نویسد، بهتر است یک کپی از این پوسته را در دیسک خود نگهداشته و داخل آن را با دستورات و داده های برنامه خود پر کنید. شکل ۲-۲ یک پوسته نمونه برنامه به زبان اسمبلی را نشان می دهد. دو دستور آخر در پوسته عبارتند از:

```

MOV  AH,4CH
INT  21H
    
```

هدف آنها بازگشت کنترل به سیستم عامل است. سه خط آخر رویه را پایان می دهند، که بترتیب عبارتند از رویه، قطعه و برنامه. توجه کنید که (MAIN)ENDP با برچسب PROC و ENDS با SEGMENT تطابق دارند.

سؤالات مروری

- ۱- هدف از شبه دستورات چیست؟
- ۲- ... بوسیله اسمبلر به کد ماشین ترجمه می شوند، در صورتی که ... اینطور نیستند.

۳- بامشخصات زیر یک برنامه به زبان اسمبلی بنویسید.

```

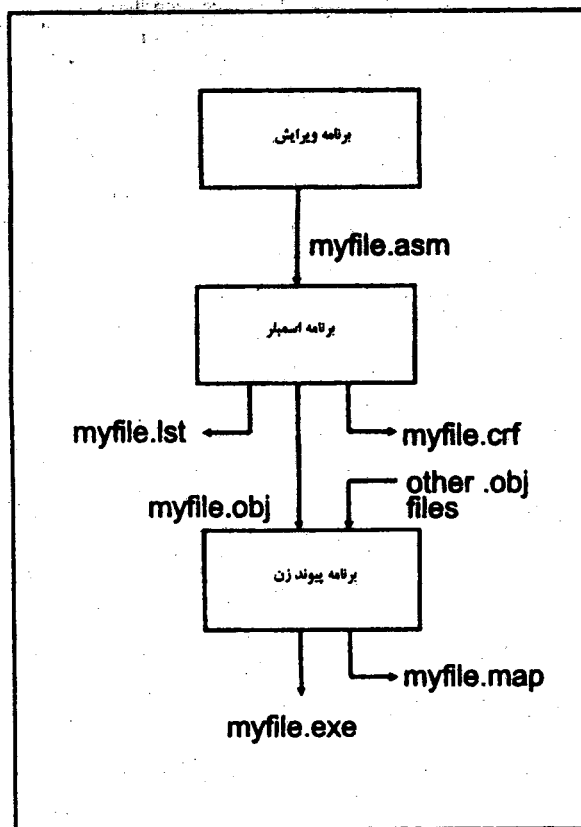
STAKSG     SEGMENT
            DB 100 DUP (?)
STA SG     ENDS
DTSEG      SEGMENT
DATA1      DB ?
DTSEG      ENDS
CDSEG      SEGMENT
MAIN       PROC FAR
            MOV AX,DATSEG
            MOV DS,AX
            MOV AL,34H
            ADD AL,4FH
            MOV DATA1,AL
            ENDP
START      ENDS
CDSEG      ENDS
    
```

- (الف) یک قطعه کد که COD-SG نام گذاری شود.
 - (ب) یک قطعه داده که DAT-SG نام گذاری شود.
 - (پ) یک قطعه پشته که STK-SG نام گذاری شود.
 - (ت) یک قلم داده که با نام HIGH DAT که حاوی 95 است.
 - (ث) دستوراتی که HIGH DAT را به ثبات های AH، BH و DL انتقال دهند.
 - (ج) نام نقطه ورودی برنامه START باشد.
- ۴- در برنامه مقابل پنج خطا پیدا کنید:

بخش ۲-۲: اسمبل کردن، پیوند و اجرای یک برنامه

اکنون فرم اساسی یک برنامه بزبان اسمبلی داده شد. سؤال بعدی این است که چگونه این برنامه ایجاد و اسمبل می‌شود؟ سه مرحله مربوط به ایجاد یک برنامه قابل اجرا در زیر آمده است.

مرحله	ورودی	برنامه	خروجی
۱- ویرایش برنامه	صفحه کلید	ادیتور	myfile.asm
۲- اسمبل کردن برنامه	myfile.asm	MASM یا TASM	myfile.obj
۳- پیوند برنامه	myfile.obj	LINK یا TLINK	myfile.exe



شکل ۲-۳. مراحل تولید فایل قابل اجرا

برنامه‌های MASM و LINK برنامه‌های اسمبلر و پیوند برای اسمبلی MASM میکروسافت هستند. اگر اسمبلی دیگری مثل بورلند TASM بکار می‌برید کتابچه آن را برای رویه اسمبل و پیوند زدن مطالعه کنید. ویرایشگرها و یا پردازشگرهای کلمات بسیاری وجود دارند که برای ایجاد یا اصلاح یک برنامه می‌توانند بکار روند. ویرایشگر یا ادیتور باید قادر باشد تا یک فایل اسکی تولید نماید. گرچه نام فایل روال معمول DOS رادنبال می‌کند، فایل مبدأ باید برای اسمبلی بکار رفته در این کتاب به asm. ختم شود. این نام فایل "asm". بوسیله یک اسمبلر، اسمبل می‌شود. گرچه اسمبلر بکار رفته در این کتاب MASM (اسمبلر میکروسافت) است، می‌توان اسمبلرهای دیگر کمپانی‌ها را هم بکار برد. اسمبلر فایل مقصود، فایل لیست و دیگر فایل‌های مفید ممکن برای برنامه‌نویس را تولید می‌کند. توسعه فایل مقصود باید "obj" باشد. این فایل مقصد ورودی به برنامه LINK است که برنامه قابل اجرا را تولید می‌کند که این یکی با "exe" ختم می‌شود. فایل "exe". بوسیله ریزپردازنده اجرا می‌گردد. قبل از تزریق فایل "obj" به LINK، همه خطاهای نحوی که بوسیله اسمبلر تولید شده است باید اصلاح شود. البته اصلاح این خطاها به این معنی نیست که برنامه آن طور که باید کار کند زیرا برنامه ممکن است خطاهای مفهومی داشته باشد. شکل ۲-۳ مراحل تولید فایل قابل اجرا را نشان می‌دهد.

شکل ۲-۴ چگونگی ایجاد یک برنامه قابل اجرا را با مراحل فوق‌الذکر و اجرای آن را تحت BEBUG نشان می‌دهد. خطوط ایتالیک آنچه که کاربر باید تایپ کند تا این مراحل اجرا شود، را مشخص می‌کند بخش پررنگ‌تر پاسخ کامپیوتر را نشان می‌دهد. در شکل ۲-۴ فرض بر این است که برنامه‌های MASM LINK و DEBUG در دیسک گردان A و برنامه‌زبان اسمبلی در دیسک گردان B باشند. دیسک گردان‌ها ممکن است براساس نصب سیستم تغییر کنند.

```
A>MASM B:MYFILE.ASM <CR>
```

Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

```
Object filename [B:MYFILE.OBJ]: B:<CR>  
Source listing [NUL.LST]:B:MYFILE.LST <CR>  
Cross-reference [NUL.CRF]: <CR>
```

47962 + 413345 Bytes symbol space free

0 Warning Errors
0 Severe Errors

```
A>LINK B:MYFILE.OBJ <CR>
```

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

```
Run File [B:MYFILE.EXE]:B:<CR>  
List File [NUL.MAP]: <CR>  
Libraries [.LIB]:<CR>  
LINK : warning L4021: no stack segment
```

```
A>DEBUG B:MYFILE.EXE <CR>  
-U CS:0 1 <CR>  
1064:0000 B86610      MOV    AX,1066  
-D 1066:0 F <CR>  
1066:0000 52 29 00 00 00 00 00 00 00 00 00 00 00 00 00 R).....  
-G <CR>  
Program terminated normally  
-D 1066:0 F <CR>  
1066:0000 52 29 7B 00 00 00 00 00 00 00 00 00 00 00 00 R){.....  
-Q <CR>  
A>
```

شکل ۴-۲. ایجاد و اجرای فایل exe

فایل های asm و obj.

فایل asm . (فایل مبدأ) فایلی است که با پردازنده کلمه یا ویرایشگر خط ایجاد می شود. اسمبلر MASM دستورات زبان اسمبلی فایل asm را به زبان ماشین تبدیل می کند (فایل مقصود، obj). علاوه بر ایجاد برنامه مقصود، MASM فایل لیست "lst". را هم ایجاد می کند.

فایل lst.

فایل "lst". که یک فایل اختیاری است ، برای برنامه نویس بسیار مفید است، زیرا همه کدهای عملیات و آدرس های تفاوت مکان همراه با خطاهای شناسایی شده بوسیله MASM را لیست می کند. MASM فرض می کند که فایل lst. خواسته نشده است (NUL.LST بمعنی نبودن لیست است). برای بدست آوردن فایل lst.، پس از نشانه DOS ، نام فایل را تایپ کنید. این فایل قابل نمایش بر روی مانیتور و ارسال به چاپگر است. برنامه نویس از آن برای کمک به برنامه debug استفاده می کند. فقط پس از اصلاح همه خطاها در فایل lst.، فایل obj. قادر خواهد بود وارد برنامه LINK شده و برنامه قابل اجرا را تولید کند. قبل از ورود به این بحث چند نکته درباره چاپ لیست ذکر می شود. برای لیست کردن فایل "lst". روی صفحه نمایش، چنین تایپ کنید:

```
C>type B:myfile.lst <return>
```

عیب این است که، سرعت حرکت طوماری صفحه نمایش زیاد است و نمی توان آن را خواند. Control-S و

Control-Q برای توقف و راه اندازی قابل استفاده اند. اگر شما از آخرین ویرایش DOS استفاده می کنید، می توانید از فرمان "more" برای چاپ یک صفحه در هر بار استفاده نمایید.

```
C>type B:myfile.lst | more <return>
```

راه دیگری برای نگاه به فایل .lst، ارسال آن به چاپگر و استفاده از دو فرمان زیر است:

```
C>mode lpt1:132,80 <return>
```

```
C>copy B:myfile.lst prn <return>
```

فرمان mode قادر است چاپگر را برای چاپ 80 خط در هر صفحه و 132 ستون در آن انتخاب کند. فرمان mode باید در دایرکتوری DOS باشد ولی فقط بر روی چاپگر اپسون یا سازگار با اپسون کار می کند. در واقع این فرمان mode لازم هم نیست؛ بلکه فقط موجب می شود تا مطالب چاپ شده بهتر جلوه کنند. فرمان دوم فایل .lst را در چاپگر کپی می کند.

برای خوانا تر شدن فایل ".lst" دو رهنمون اسمبلر PAGE و TITLE وجود دارد.

رهنمون های PAGE و TITLE

قالب رهنمون PAGE بشکل زیر است.

```
PAGE [lines],[columns]
```

و نقش آن اطلاع به چاپگر در مورد نحوه چاپ است. در مد پیش فرض، فرض بر این است که رهنمون PAGE بدون هر کدی پس از آن آمده است، و خروجی 66 خط در هر صفحه با 80 کاراکتر در هر خط دارد. در این کتاب، برنامه ها مقادیر فوق را به 60 و 132 بطریق زیر عوض می کنند.

```
PAGE 60,132
```

PAGE 60 , 132

محدوده تعداد خطوط از 10 تا 255 و برای ستون ها از 60 تا 132 است. وقتی که لیست چاپ شود و تعداد آن بیش از یک صفحه باشد، می توان به اسمبلر دستور داد تا عنوان برنامه، را در بالای هر صفحه چاپ کند. اینکه پس از شبه دستور TITLE چه چیزی آورده شود بعهد برنامه نویس است، ولی رسم بر این است تا نام برنامه همانطور که در دیسک پس از شبه دستور TITLE آمده در بالای هر صفحه چاپ شود و سپس شرح مختصری از برنامه داده شود. هر آنچه که پس از شبه دستور آورده می شود، نباید بیش از 60 کاراکتر اسکی (حروف، اعداد، فاصله ها و دیگر علائم) باشد.

فایل .crf

MASM فایل اختیاری دیگری بنام ارجاع متقابل که توسعه "crf" دارد را تولید می نماید. این فایل لیستی الفبایی از همه نمادها و برجسب های بکار رفته در برنامه و تعداد خطوط برنامه ای که به آنها ارجاع شده را تهیه می کند. در برنامه های بزرگ با تعداد زیادی قطعات کد و قطعات داده، این فایل بسیار مفید خواهد بود. این فایل بوسیله برنامه CREF در MASM قابل تبدیل به فایل اسکی است. در زیر فایل "crf" به فایل اسکی "CREF.ASC" تبدیل شده است.

```
C>cref prog1.crf cref.asc <return>
```

پیوند برنامه

اسمبلر کدهای عملیات، عملوندها و آدرس های تفاوت مکان را تحت فایل ".obj" ایجاد می نماید. نوع آماده برای اجرا از یک برنامه، بوسیله برنامه LINK تولید می شود که توسعه "exe" را داراست. برنامه LINK فایل را طوری ایجاد می کند که بوسیله DOS قابل بار شدن و اجرا باشد. در شکل ۴-۲، از DEBUG برای اجرای برنامه شکل ۱-۲ استفاده کرده و نتیجه را تحلیل نمودیم. در برنامه

شکل ۲-۱، سه قلم داده در قطعه داده تعریف شده است. قبل از اجرای برنامه با تخلیه محتوای «تفاوت مکان: DS»، طبق شکل ۲-۴ می‌توان به داده در قطعه داده نگاه کرد. سؤال این است که در ثبات DS چیست؟ این مقدار می‌تواند از یک کامپیوتر به کامپیوتر دیگر و از یک DOS به DOS دیگر فرق کند. باین دلیل نظاره بر مقدار در MOV AX,xxxx با اهمیت است. نتیجه برنامه پس از اجراش می‌تواند تحقیق شود، شکل ۲-۴. وقتی که برنامه با موفقیت کار کند، در سطح DOS قابل اجرا است. برای اجرای myfile.exe فقط عبارت زیر را تایپ کنید.

B>myfile <return>

با این وجود چون این برنامه خروجی تولید نمی‌کند، راهی برای تأیید نتایج وجود ندارد. وقتی که نام برنامه در سطح DOS طبق شکل فوق تایپ شود، DOS برنامه را در حافظه بار می‌کند. گاهی به این عمل نگاشت هم می‌گویند که بمعنی نگاشته شدن برنامه در حافظه فیزیکی PC است. قبل از ترک این بخش لازم است تا نقش فایل "map" که بوسیله برنامه LINK تولید شده است، مشخص گردد.

فایل map

وقتی که برای کد یا داده تعدادی قطعات وجود دارد، لازم است بدانیم که هر یک در کجا مستقر است و برای هر کدام چند بایت بکار برده شده است. این عمل بوسیله فایل map صورت می‌گیرد. این فایل که یک فایل اختیاری است، نام هر قطعه، نقطه شروع، نقطه پایان و سائز بایت‌ها را مشخص می‌نماید. در فصل ۷ وقتی که چندین زیر روال بطور جداگانه (مدول) اسمبل و سپس بهم پیوند شدند اهمیت نگاشت روشن‌تر خواهد شد.

سؤالات مروری

- ۱- (الف) فایل ورودی به اسمبلر MASM دارای توسعه ... است.
- (ب) فایل ورودی برنامه LINK دارای توسعه ... است.
- ۲- انواع فایل‌های ستون دوم را که خروجی برنامه در ستون اول هستند انتخاب نمایید.

ستون اول	ستون دوم	
(الف) obj	(ب) .asm	_____ ویرایشگر
(پ) .exe	(ت) .lst	_____ اسمبلر
(ث) .crf	(ج) .map	_____ پیوند زن

بخش ۲-۳: چند برنامه نمونه دیگر

اکنون که با برنامه نویسی به زبان اسمبلی در IBM PC تا حدودی آشنا شدید، در این بخش نگاهی به مثال‌های پیچیده‌تری در برنامه‌نویسی می‌اندازیم تا خواننده بر ویژگی‌های اصلی برنامه‌نویسی اسمبلی تسلط داشته باشد. در صفحات بعد برنامه ۲-۱ و فایل لیست (IST) که هنگام اسمبل کردن برنامه تولید می‌شود، آمده است. پس از اسمبل و پیوند برنامه، DEBUG برای مشاهده مقدار متناسب به ثبات DS در قطعه کد بکار رفته است. اینکه DOS دقیقاً یک برنامه را در کجای حافظه RAM بار می‌کند به فاکتورهای زیادی بستگی دارد. منجمله از مقدار RAM سیستم و ویرایش DOS می‌توان نام برد. بنابراین مقداری که برای "MOV AX, xxxx" بدست می‌آورد و نیز CS در مثال‌های برنامه می‌تواند کاملاً متفاوت باشد. سعی در اصلاح محتوای ثبات‌ها در مثال‌ها نکنید در غیراینصورت سیستم احتمال فروپاشی را دارد!

برنامه‌ای بنویسید که 5 بایت داده را جمع و سپس ذخیره کند. آنگاه آن را اجرا و تحلیل نمایید. داده‌ها عبارتند از 1F، 15، 12، 25 و 2B.

```

PAGE      60,132
TITLE     PROG2-1 (EXE) PURPOSE: ADDS 5 BYTES OF DATA
STSEG     SEGMENT
          DB 32 DUP (?)
STSEG     ENDS

DTSEG     SEGMENT
DATA_IN   DB 25H,12H,15H,1FH,2BH
SUM        DB ?
DTSEG     ENDS

CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          MOV CX,05 ;set up loop counter CX=5
          MOV BX,OFFSET DATA_IN ;set up data pointer BX
          MOV AL,0 ;initialize AL
AGAIN:    ADD AL,[BX] ;add next data item to AL
          INC BX ;make BX point to next data item
          DEC CX ;decrement loop counter
          JNZ AGAIN ;jump if loop counter not zero
          MOV SUM,AL ;load result into sum
          MOV AH,4CH ;set up return
          INT 21H ;return to DOS
MAIN      ENDP
CDSEG     ENDS
          END MAIN

```

پس از اسمبل شدن برنامه و پیوند آن، با استفاده از DEBUG اجرا می‌شود.

```

C>debug prog2-1.exe
-u cs:0 19
1067:0000 B86610 MOV AX,1066
1067:0003 8ED8 MOV DS,AX
1067:0005 B90500 MOV CX,0005
1067:0008 BB0000 MOV BX,0000
1067:000D 0207 ADD AL,[BX]
1067:000F 43 INC BX
1067:0010 49 DEC CX
1067:0011 75FA JNZ 000D
1067:0013 A20500 MOV [0005],AL
1067:0016 B44C MOV AH,4C
1067:0018 CD21 INT 21
-d 1066:0 f
1066:0000 25 12 15 1F 2B 00 00 00-00 00 00 00 00 00 00 00 %...+.....
-g

Program terminated normally
-d 1066:0 f
1066:0000 25 12 15 1F 2B 96 00 00-00 00 00 00 00 00 00 00 %...+.....
-q
C>

```

برنامه ۲-۱

تحلیل برنامه ۲-۱

برنامه DEBUG بطور کامل در پیوست (الف) توضیح داده شده است. فرمان‌های بکار رفته در اجرای برنامه ۲-۱ عبارتند از (۱) u، که برای اسمبل کردن معکوس کد، یا عکس اسمبل کد، از 0: CS برای 19 بایت بکار رفته است؛ (۲) d، برای تخلیه محتوای حافظه از 1066:0 برای F بایت بعدی؛ و (۳) g، برای اجرای برنامه.

```

                                PAGE 60,132
                                TITLE PROG2-1 (EXE) PURPOSE: ADDS 5 BYTES OF DATA
0000                                STSEG SEGMENT
0000 0020[                        DB 32 DUP (?)
??                                ]
0020                                STSEG ENDS

                                -----
0000                                DTSEG SEGMENT
0000 25 12 15 1F 2B DATA_IN DB 25H,12H,15H,1FH,2BH
0005 00 SUM DB ?
0006                                DTSEG ENDS

                                -----
0000                                CDSEG SEGMENT
0000 MAIN PROC FAR
                                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
0000 B8 ---- R MOV AX,DTSEG
0003 8E D8 MOV DS,AX
0005 B9 0005 MOV CX,05 ;set up loop counter CX=5
0008 BB 0000 R MOV BX,OFFSET DATA_IN ;set up data pointer BX
000B B0 00 MOV AL,0 ;initialize AL
000D 02 07 AGAIN: ADD AL,[BX] ;add next data item to AL
000F 43 INC BX ;make BX point to next data item
0010 49 DEC CX ;decrement loop counter
0011 75 FA JNZ AGAIN ;jump if loop counter not zero
0013 A2 0005 R MOV SUM,AL ;load result into sum
0016 B4 4C MOV AH,4CH ;set up return
0018 CD 21 INT 21H ;return to DOS
001A MAIN ENDP
001A CDSEG ENDS
                                END MAIN
  
```

Microsoft (R) Macro Assembler Version 5.10 4/28/92
 09:25:49 PROG2-1 (EXE) PURPOSE: ADDS 5 BYTES OF DATA Symbols-1

Segments and Groups:

Name	Length	Align	Combine Class
CDSEG	001A	PARA	NONE
DTSEG	0006	PARA	NONE
STSEG	0020	PARA	NONE

Symbols:

Name	Type	Value	Attr
AGAIN	L NEAR	000D	CDSEG
DATA_IN	L BYTE	0000	DTSEG
MAIN	F PROC	0000	CDSEG
SUM	L BYTE	0005	DTSEG

Length = 001A

```

@CPU..... TEXT 0101h
@FILENAME..... TEXT PROG2-1
@VERSION..... TEXT 510
  
```

30 Source Lines 30 Total Lines 12 Symbols
 47900 + 403391 Bytes symbol space free

0 Warning Errors 0 Severe Errors

دقت داشته باشید که وقتی برنامه ۱-۲ در DEBUG اجرا شود، محتوای قطعه حافظه داده قبل از اجرای برنامه تخلیه شده تا مشخص شود که برنامه طبق نقشه طرح شده کار می‌کند. معمولاً لازم نیست این تعداد کد، عکس اسمبل شود، ولی در اینجا باین دلیل صورت گرفته است که در بخش‌های بعدی این فصل، ما دستور پرش را در این برنامه بررسی خواهیم کرد. و نیز توجه کنید که 5 بایت اول تخلیه شده فوق اقلامی از داده هستند که در محدوده قطعه داده برنامه تعریف شده‌اند و ششمین قلم حاصل جمع آن پنج قلم است، بنابراین به نظر می‌رسد که برنامه صحیح کار کرده است.

در زیر برنامه ۱-۲ دستور به دستور توضیح داده شده است.

"MOV CX, 05" مقدار 05 را در ثبات CX بار می‌کند. این ثبات بوسیله برنامه بعنوان شمارنده در تکرار (حلقه) بکار می‌رود "MOV BX, OFFSET DATA_IN" مقدار آدرس تفاوت مکان را به DATA تخصیص می‌دهد. اسمبلر از تفاوت مکان 0000 شروع کرده و حافظه را برای داده بکار می‌گیرد و سپس حافظه بعدی تفاوت مکان را برای SUM بکار می‌برد (در این حالت 0005).
"ADD AL, [BX]" محتوای حافظه اشاره شده یا BX را در AL بار می‌کند. توجه کنید که [BX] اشاره‌گری به یک مکان از حافظه است.

"INC BX" اشاره‌گر را با افزایش ثبات BX به میزان یک واحد، به مکان بعدی اصلاح می‌کند.
"DEX CX" شمارنده CX را یک واحد کاهش داده و اگر CX صفر شود پرچم صفر را بالا خواهد برد.
"JNZ AGAIN" این دستور مادامی که CX مخالف صفر باشد پرشی به پرچسب AGAIN را انجام خواهد داد. اگر پرچم صفر با دستور DEC CX برابر 1 شود، دستور JNZ AGAIN هیچ پرشی را انجام نداده و دستور بعد از آن اجرا خواهد شد. وقتی که CX صفر گردد، مفهوم این است که حلقه کامل شده و هر پنج عدد با هم جمع و حاصل در AL قرار گرفته است.

روش‌های مختلف طراحی برنامه ۱-۲

در نوشتن هر برنامه‌ای روش‌های مختلفی وجود دارد. در روشی که در برنامه ۱-۲ دیدید، یک میدان داده تعریف شده و اشاره‌گر [BX] برای دستیابی به عناصر حافظه بکار رفت. در روش زیر، به هر یک از اقلام داده یک نام اختصاص می‌یابد و این داده‌ها در برنامه دستیابی خواهند شد. تغییرات برنامه ۱-۲ به منظور روشن کردن استفاده از روش‌های آدرس‌دهی در یک برنامه واقعی است و نیز استفاده از هر ثبات همه منظوره برای انجام عملیات محاسباتی و منطقی در 80x86 هم نشان داده شده است. در فصل‌های اولیه CPU، انباره، مقصد همه اعمال محاسباتی و منطقی بود، ولی در 80x86 این وضع برقرار نیست. چون هدف از این مثال‌ها نمایش روش‌های مختلف دستیابی به عملوندها است، اجرا و تحلیل برنامه‌ها به خواننده واگذار می‌شود.

؛ از قطعه داده:

DATA1	DB 25H
DATA2	DB 12H
DATA3	DB 15H
DATA4	DB 1FH
DATA5	DB 2BH
SUM	DB ?

؛ از قطعه کد:

MOV	AL, DATA1	; MOVE DATA1 INTO AL
ADD	AL, DATA2	; ADD DATA2 TO AL
ADD	AL, DATA3	
ADD	AL, DATA4	
ADD	AL, DATA5	
MOV	SUM, AL	; SAVE AL IN SUM

تفاوت‌های چشمگیری در نوشتن یک برنامه با بکارگیری دو روش وجود دارد. در حالی که در اولی روش آدرس‌دهی غیرمستقیم ثباتی بکار رفته است، در روش دوم روش آدرس‌دهی مستقیم برای دستیابی به حافظه استفاده شده است. در مثال فوق، هربایت داده قابل انتقال به ثبات‌های مختلف و سپس در یک زمان قابل جمع شدن هستند:

```
MOV AL,DATA1
MOV AH,DATA2
MOV BL,DATA3
MOV BH,DATA4
MOV DH,DATA5
ADD DH,AL           ;Add to DH the contents of AL
ADD DH,AH           ;etc.
ADD DH,BL
ADD DH,BH
MOV SUM,DH          ;save DH in SUM
```

برنامه‌ای بنویسید که چهار کلمه داده را جمع و سپس نتیجه را ذخیره کند. مقادیر عبارتند از 3BC7H، 1DE6H، 234DH و 566AH. از DEBUG برای تأیید جمع که D364 است استفاده کنید.

```
TITLE      PROG2-2 (EXE) PURPOSE: ADDS 4 WORDS OF DATA
PAGE      60,132
STSEG     SEGMENT
DB 32 DUP (?)
STSEG     ENDS

;-----
DTSEG     SEGMENT
DATA_IN   DW 234DH,1DE6H,3BC7H,566AH
ORG       10H
SUM        DW ?
DTSEG     ENDS

;-----
CDSEG     SEGMENT
MAIN      PROC FAR
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
MOV       AX,DTSEG
MOV       DS,AX
MOV       CX,04           ;set up loop counter CX=4
MOV       DI,OFFSET DATA_IN ;set up data pointer DI
MOV       BX,00           ;initialize BX
ADD_LP:   ADD       BX,[DI] ;add contents pointed at by [DI] to BX
INC       DI             ;increment DI twice
INC       DI             ;to point to next word
DEC       CX             ;decrement loop counter
JNZ       ADD_LP         ;jump if loop counter not zero
MOV       SI,OFFSET SUM  ;load pointer for sum
MOV       [SI],BX        ;store in data segment
MOV       AH,4CH         ;set up return
INT       21H            ;return to DOS
MAIN      ENDP
CDSEG     ENDS
END       MAIN
```

پس از اسمبل شدن برنامه و پیوند آن، با استفاده از DEBUG اجرا می‌شود.

```
-u cs:0 1
1068:0000 B86610      MOV     AX,1066
-D 1066:0 1F
1066:0000 4D 23 E6 1D C7 3B 6A 56-00 00 00 00 00 00 00 M#f.G;jV.....
1066:0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-G

Program terminated normally
-D 1066:0 1F
1066:0000 4D 23 E6 1D C7 3B 6A 56-00 00 00 00 00 00 00 M#f.G;jV.....
1066:0010 64 D3 00 00 00 00 00 00-00 00 00 00 00 00 00 dS.....
-Q
A>
```

تحلیل برنامه ۲-۲

توجه کنید که هنگام ذخیره داده 16 بیتی، ابتدا بایت کم ارزشتر ذخیره می شود. مثلاً "234D" که در قطعه داده تعریف شده بصورت "4D23" ذخیره شده است. این بدان معنی است که آدرس پائین تر 0000، کم ارزشترین بایت، 4D، و آدرس بالاتر 0001، با ارزشترین بایت، 23، را داراست. این مطلب در نمایش قطعه داده بوسیله DEBUG نشان داده شده است. بطور مشابه D364 بصورت 64D3 ذخیره می شود. همانطور که در فصل ۱ گفته شد، این روش جایگزینی بایت پائین تر در حافظه پائین تر و بایت بالاتر در حافظه بالاتر به ذخیره سازی مبتنی بر کوچکتر معروف است.

دومین نکته این است که اشاره گر آدرس دوبار افزایش می یابد، زیرا عملوند مورد نظر یک کلمه (دو بایت) است. برنامه می تواند از "ADD DI, 2" بجای دوبار استفاده کند. وقتی که حاصل جمع کلمه ای بکار می رود، از "MOV DI, OFFSET SUM" برای بارکردن اشاره گر (در این حالت 0010 که بوسیله 0010H ORG مشخص شده) استفاده می شود تا حافظه مربوط به برجسب SUM تخصیص یابد، و سپس "MOV [DI],BX" برای انتقال محتوای BX به حافظه های 0010 و 0011 بکار برده می شود. مجدداً، همانطور که قبلاً ذکر شد با استفاده از روش آدرس دهی مستقیم بصورت "MOV SUM,BX" نوشته خواهد شد.

برنامه ۲-۲ از رهنمون ORG استفاده کرده است. در برنامه های قبل که ORG بکار نرفت، اسمبلر از تفاوت مکان 0000 شروع و برای هر قلم داده حافظه ای را بکار برد. رهنمون ORG می تواند برای تنظیم آدرس های تفاوت مکان اقلام داده مورد استفاده قرار گیرد. گرچه برنامه نویس نمی تواند آدرس های فیزیکی دقیقی را معین کند، ولی به وی اجازه تخصیص آدرس های تفاوت مکان داده شده است. رهنمون ORG در برنامه 2-2 موجب می شود تا SUM در مکان 0010 : DS ذخیره شود. این مطلب در نمایش DEBUG قطعه داده دیده می شود.

تحلیل برنامه ۲-۳

مثال DEBUG تخیله قطعه داده را قبل از اجرای برنامه و بعد از آن برای تأیید کپی شدن موفقیت آمیز داده، و اجرای موفق برنامه، نشان می دهد. دقت کنید که C4 در قطعه داده بصورت 0C4 کد شده است. این کار باین دلیل انجام می شود تا به اسمبلر مبنای 16 بودن C تفهیم شود و آن را با حرف C اشتباه نکند. هر گاه اولین رقم ارقام A تا F باشد این عمل الزامی است.

این برنامه دو ثابت SI و DI را بعنوان اشاره گر برای اقلام داده به هنگام دستکاری بکار می برد. اولین ثابت بعنوان اشاره گر اقلام داده کپی شونده و دومین ثابت بعنوان اشاره گر مکانی که اقلام داده در آن ذخیره می شوند استفاده می شود. در هر بار تکرار حلقه، هر دو اشاره گر یک واحد اضافه می شوند تا به بایت بعدی اشاره گردد.

بررسی مجدد تعریف قطعه پشته

یکی از کارهای اولیه سیستم عامل DOS تعیین جمع کل RAM نصب شده در PC و سپس مدیریت صحیح آن است. DOS از بخشی که متعلق بخودش است استفاده کرده و بقیه را واگذار می کند. چون نیازهای حافظه برای انواع ویرایش DOS متفاوت است، یک برنامه محل واقعی حافظه را برای هر قطعه نمی تواند دقیقاً معین نماید. چون مدیریت حافظه از جمله وظایف DOS است، برنامه های اسمبلی را بکمک LINK در حافظه PC خواهد نگاشت.

در مثال های قبل هنگام اسمبل کردن و پیوند برنامه پیام "Warning : no stack segment" تولید شد. چرا این پیام ظاهر گردید؟ با این وجود برنامه قطعه را تعریف کرد. چون خطا از نوع مخرب نبود، اجرای برنامه ها تحت تأثیر قرار نگرفت. دلیل آشکار شدن پیام هم این بود که علیرغم تعریف پشته، با پشته سیستم پیوند نخورده بود. گرچه در محیط DOS یک برنامه دارای چندین قطعه کد و داده است ولی قویاً تأکید می شود تا فقط یک قطعه پشته وجود داشته باشد، تا RAM بوسیله پشته تفکیک نگردد. بعلاوه قطعه پشته باید طوری نام گذاری شود که بوسیله فرآیند پیوند زنی قابل شناسایی باشد. ترکیب همه قطعات کد، داده برای ایجاد یک برنامه واحد

قابل اجرا به همراه یک قطعه پشته که همان پشته سیستم است، بعهده برنامه LINK است. در تفهیم به LINK برای ترکیب پشته تعریف شده بوسیله کاربر با پشته سیستم (تا باین ترتیب از پیام اعلام خطای فوق راحت شویم)، پشته باید بطریق زیر تعریف شود:

```
name SEGMENT STACK 'stack'
```

این پیام به DOS دستور می دهد تا پشته تعریف شده در برنامه را با پشته سیستم ترکیب کند. برای رهنمون SEGMENT انتخاب های مختلفی وجود دارد که در فصل مورد بحث قرار خواهند گرفت و بسیاری از این مفاهیم در آنجا توضیح داده خواهند شد.

برنامه ای بنوسید که 6 بایت داده را از مکان های حافظه ای با تفاوت مکان 0010H به مکان های حافظه دیگری با تفاوت مکان 0028H جابجا کند.

```
TITLE      PROG2-3 (EXE) PURPOSE: TRANSFERS 6 BYTES OF DATA
PAGE      60,132
STSEG     SEGMENT
DB        32 DUP (?)
STSEG     ENDS

DTSEG     SEGMENT
ORG       10H
DATA_IN   DB 25H,4FH,85H,1FH,2BH,0C4H
ORG       28H
COPY      DB 6 DUP(?)
DTSEG     ENDS

CDSEG     SEGMENT
MAIN      PROC FAR
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
MOV       AX,DTSEG
MOV       DS,AX
MOV       SI,OFFSET DATA_IN      ;SI points to data to be copied
MOV       DI,OFFSET COPY         ;DI points to copy of data
MOV       CX,06H                 ;loop counter = 6
MOV_LOOP: MOV AL,[SI]            ;move the next byte from DATA area to AL
MOV       [DI],AL                ;move the next byte to COPY area
INC       SI                     ;increment DATA pointer
INC       DI                     ;increment COPY pointer
DEC       CX                     ;decrement LOOP counter
JNZ       MOV_LOOP               ;jump if loop counter not zero
MOV       AH,4CH                 ;set up to return
INT       21H                    ;return to DOS
MAIN      ENDP
CDSEG     ENDS
END       MAIN
```

پس از اسمبل و پیوند زدن برنامه با استفاده از DEBUG آن را اجرا نمایید.

```
C>debug prog2-3.exe
-u cs:0 1
1069:0000 B86610      MOV     AX,1066
-d 1066:0 2f
1066:0000 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1066:0010 25 4F 85 1F 2B C4 00 00-00 00 00 00 00 00 00 00 %O...+D.....
1066:0020 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-g

Program terminated normally
-d 1066:0 2f
1066:0000 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1066:0010 25 4F 85 1F 2B C4 00 00-00 00 00 00 00 00 00 00 %O...+D.....
1066:0020 00 00 00 00 00 00 00 00 00-25 4F 85 1F 2B C4 00 00 %O...+D.....
-q
C>
```

۱- کار دستور INC چیست؟

۲- کار دستور DEC چیست؟

۳- در برنامه ۲-۱ چرا پس از برجسب AGAIN دو نقطه آمده است، در صورتی که MAIN آن را ندارد؟

۴- تفاوت بین دو دستور زیر چیست؟

```
MOV BX, DATA1
MOV BX, OFFSET DATA1
```

۵- تفاوت بین دو دستور زیر را بیان کنید.

```
ADD AX, BX
ADD AX, [BX]
```

بخش ۲-۴: دستورات انتقال کنترل

در رشته دستوراتی که باید اجرا شوند، اغلب لازم است تا کنترل برنامه به مکان دیگری منتقل شود. برای حصول به این کار دستورات بسیاری در 80x86 وجود دارد. این بخش دستورات انتقال کنترل موجود در زمان اسمبلی 8086 را مورد بحث قرار می دهد. با این وجود قبل از شروع، لازم است مفاهیم FAR و NEAR که به دستورات JUMP و CALL (پرش و فراخوانی) اختصاص می یابند توضیح داده شوند.

FAR (دور) و NEAR (نزدیک)

اگر کنترل برنامه به مکانی در داخل قطعه کد جاری انتقال یابد، به آن پرش NEAR گوئیم، گاهی اوقات به آن درون قطعه ای هم می گویند. چون ثبات های CS:IP همیشه به آدرس دستور بعدی مورد اجرا اشاره می کنند، وقتی دستور انتقال کنترل اجرا می شود باید آن ها را اصلاح کرد. در پرش NEAR، IP تصحیح و CS دست نخورده باقی می ماند، زیرا هنوز کنترل در داخل قطعه کد جاری است. در پرش FAR، چون کنترل به خارج قطعه کد جاری تحویل می گردد، هر دو قسمت CS و IP با مقادیر جدید اصلاح می شوند. به بیان دیگر در هر دستور انتقال کنترلی مانند پرش یا فراخوانی، IP باید تغییر نماید ولی در حالت FAR، CS هم تغییر می کند.

پرش های شرطی

پرش های شرطی که در جدول ۲-۱ خلاصه شده اند دارای نمادهای رمزی همچون JNZ (پرش اگر غیر صفر) و JC (پرش اگر نقلی) می باشند. در پرش شرطی، کنترل به مکان جدید براساس برقراری شرط میسر است. ثبات پرچم یکی از شاخص های شرایط جاری است. مثلاً با "JNZ Label" پردازنده به پرچم صفر نگاه می کند تا ببیند آیا بالا رفته است یا خیر. اگر جواب منفی است پردازنده شروع به برداشت و اجرای دستورات از آدرس Label خواهد کرد. اگر $ZF = 1$ است پرش صورت نگرفته و دستور بعد که در زیر JNZ قرار گرفته اجرا خواهد شد.

پرش های کوتاه

همه پرش های شرطی از نوع پرش کوتاه هستند. در پرش کوتاه آدرس بایت هدف در محدوده 128- تا 127+ بایت از IP باشد. به بیان دیگر، پرش شرطی یک دستور دو بایتی است: یک بایت متعلق به کد عمل پرش و دومی مقداری بین 00 تا FF می باشد. محدوده تفاوت مکان 00 تا FF، 256 آدرس را ممکن می سازد، که بین پرش به عقب تا 128- و پرش به جلو 127+ تقسیم شده است.

در پرش به عقب بایت دوم متمم 2 مقدار جابجایی است. برای محاسبه آدرس هدف بایت دوم به IP دستور بعد از پرش اضافه می شود. برای درک این مطلب، به کد اسمبل نشده دستور JNZ AGAIN در برنامه ۲-۱ که در زیر تکرار شده است توجه کنید.

1067:0000 B86610	MOV AX,1066
1067:0003 8ED8	MOV DS,AX
1067:0005 B90500	MOV CX,0005
1067:0008 BB0000	MOV BX,0000
1067:000D 0207	ADD AL,[BX]
1067:000F 43	INC BX
1067:0010 49	DEC CX
1067:0011 75FA	JNZ 000D
1067:0013 A20500	MOV [0005],AL
1067:0016 B44C	MOV AH,4C
1067:0018 CD21	INT 21

دستور "JNZ AGAIN" بصورت JNZ 000D اسمبل شده است که در آن 000D آدرس دستور با پرچسب AGAIN است. دستور "JNZ 000D" دارای کد 75 و آدرس هدف FA است، که در تفاوت مکان‌های 0011 و 0012 قرار گرفته است. بدنبال آن "MOV SUM,AL" آمده است که از تفاوت مکان 0013 شروع می‌شود. مقدار IP مربوط به 0013، MOV به FA اضافه می‌شود تا آدرس برچسب AGAIN (0013 + FA = 000D) حاصل شود که در آن رقم نقلی حذف شده است. در عمل FA متمم 2 عدد 6- است که IP دستور بعدی هدف را به میزان 6- مشخص می‌کند.

جدول ۱-۲ دستور العمل JMP شرطی 8086

نماد خلاصه	شرط تست شده	"Jump IF ..."
JA/JNBE	(CF = 0) and (ZF = 0)	above/not below nor equal
JAE/JNB	CF = 0	above or equal/not below
JB/JNAE	CF = 1	below/not above nor equal
JBE/JNA	(CF or ZF) = 1	below or equal/not above
JC	CF = 1	carry
JE/JZ	ZF = 1	equal/zero
JG/JNLE	((SF xor OF) or ZF) = 0	greater/not less nor equal
JGE/JNL	(SF xor OF) = 0	greater or equal/not less
JL/JNGE	(SF xor OF) = 1	less/not greater nor equal
JLE/JNG	((SF xor OF) or ZF) = 1	less or equal/not greater
JNC	CF = 0	not carry
JNE/JNZ	ZF = 0	not equal/not zero
JNO	OF = 0	not overflow
JNP/JPO	PF = 0	not parity/parity odd
JNS	SF = 0	not sign
JO	OF = 1	overflow
JP/JPE	PF = 1	parity/parity equal
JS	SF = 1	sign

توجه: "Above" و "below" به رابطه بین دو مقدار بدون علامت اشاره می‌کند، و "greater" و "Less" به رابطه دو عدد علامت‌دار اشاره می‌کند.

بظور مشابه آدرس هدف در پرش به جلو از جمع IP و عملوند دستور حاصل می‌گردد. در این حال جابجایی مثبت است. در زیر بخشی از فایل لیست که کدهای عمل را برای چند پرش شرطی نشان می‌دهد، آمده است.

0005	8A 47 02	AGAIN: MOV	AL,[BX]+2
0008	3C 61	CMP	AL,61H
000A	72 06	JB	NEXT
000C	3C 7A	CMP	AL,7AH
000E	77 02	JA	NEXT
0010	24 DF	AND	AL,ODFH
0012	88 04	NEXT: MOV	[SI],AL

در برنامه فوق "JB NEXT" دارای کد عمل 72 و آدرس هدف 06 بوده و در مکان 000A = IP و 000B واقع است. پرش به 6 بایت بعد از دستور خواهد بود که 000C = IP می‌باشد. جمع این دو مقدار برابرست با 000CH + 0006H = 0012H، که دقیقاً آدرس پرچسب NEXT است. به دستور "JA NEXT" هم نگاه کنید که دارای کد 77 و جابجایی 02 است، IP دستور بعدی 0010 به 02 اضافه شده و 0012 را که آدرس هدف است تولید می‌کند.

باید تأکید کرد که علیرغم رو به جلو یا عقب بودن پرش، در دستورات شرطی آدرس هدف هرگز 128- و یا 127+ بایت دورتر از IP دستور بعد از پرش نیست. هر تلاشی در راستای سرپیچی از این قانون موجب تولید پیام "relative jump out of range" می‌گردد. این پرش‌های شرطی گاهی پرش‌های کوتاه (SHORT) هم خوانده می‌شوند.

پرش‌های بدون شرط

"JMP label" یک پرش بدون شرط است که کنترل را بدون هر شرطی به هدفی که با label مشخص شده محول می‌کند. پرش بدون شرط فرم‌های زیر را داراست.

۱- SHORT JUMP با قالب "JMP SHORT label" مشخص می‌گردد. در این نوع پرش آدرس هدف در محدوده 128- تا 127+ بایتی نسبت به آدرس IP جاری واقع است. در این حال، کد عمل EB و عملوند یک بایتی و در محدوده 00 تا FF است. بایت عملوند به منظور محاسبه آدرس هدف با IP جمع می‌شود. اگر پرش رو به عقب باشد عملوند متمم 2 است. این دقیقاً مثل حالت پرش شرطی است. نوشتن رهنمون "short" پرش را بسیار کارآتر می‌نماید زیرا در عوض دستور 3 بایتی، دستور دوبایتی خواهیم داشت.

۲- NEAR JUMP، که پیش فرض است دارای قالب "JMP Label" می‌باشد. این یک دستور پرش نزدیک (در قطعه کد جاری) با کد عمل E9 است. آدرس هدف می‌تواند هر یک از روش‌های مستقیم، ثباتی، غیرمستقیم ثباتی یا غیرمستقیم حافظه‌ای باشد:

(الف) JUMP مستقیم دقیقاً مثل پرش کوتاه است که قبلاً توضیح داده شد، بجز اینکه آدرس هدف هر جایی در قطعه و در محدوده 32767+ تا 32768- از IP باشد.

(ب) JUMP غیرمستقیم ثباتی؛ در این نوع پرش آدرس هدف در یک ثبات است. مثلاً در JMP BX، مقدار BX را اختیار می‌کند.

(پ) JUMP غیرمستقیم حافظه‌ای؛ که در آن آدرس هدف محتوای دو مکان از حافظه است که بوسیله ثبات به آنها اشاره می‌شود. مثلاً در "JUMP [DI]"، IP را با محتوای حافظه‌های اشاره شده با DI و DI+1 جایگزین می‌کند.

۳- FAR JUMP که دارای قالب "JMP FAR PTR Label" است. این پرش به خارج قطعه کد جاری صورت می‌گیرد و به این معنی است که نه فقط IP بلکه CS هم با مقدار جدید جایگزین می‌شود.

عبارات فراخوانی (CALL)

دستورالعمل کنترل دیگر، دستور CALL است، که برای فراخوانی یک رویه مورد استفاده قرار می‌گیرد.

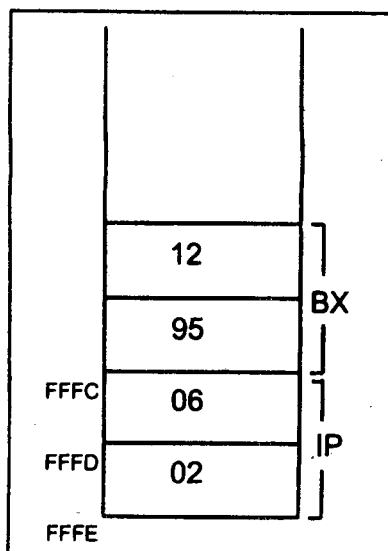
CALLها برای رویه‌هایی مورد استفاده قرار می‌گیرند که نیاز به اجرای متعدد دارند. این موجب می‌شود تا برنامه ساخت یافته‌تر باشد. آدرس هدف می‌تواند در قطعه جاری باشد که در اینصورت یک فراخوانی **NEAR** و یا خارج قطعه **CS** جاری باشد که در این حال یک فراخوانی **FAR** است. برای اطمینان از اجرای زیر روال‌های فرا خوانده شده، ریزپردازنده می‌داند که چه موقعی به عقب بازگردد، زیرا پس از اجرای دستور فراخوانی، بطور خودکار آدرس دستور بعد از فراخوانی را در پشته ذخیره می‌کند. باید توجه کرد که در فراخوانی **NEAR** فقط **IP** در پشته ذخیره می‌شود و در فراخوانی **FAR** هر دو مقدار **CS** و **IP** ذخیره می‌شوند. وقتی که زیر روالی فراخوانی شود، کنترل به زیر روال انتقال می‌یابد و **IP** (و نیز **CS** بهنگام فراخوانی **FAR**) ذخیره می‌شود، سپس شروع به برداشت دستورات از مکان جدید می‌نماید. پس از پایان اجرای زیر روال، برای انتقال کنترل به محل فراخوان، آخرین دستور زیر روال باید **RET** باشد. به همان ترتیبی که اسمبلر کدهای مختلفی را برای حالت **FAR** و **NEAR** در فراخوانی تولید می‌کرد، کد عمل **RET** در حالت **NEAR** و **FAR** نیز متفاوت است. برای فراخوانی **NEAR**، **IP** ذخیره می‌شود، در فراخوانی **FAR**، **CS** و **IP** هر دو ذخیره می‌گردند. این خود بازگشت به فراخوان را تضمین می‌کند. مثلاً فرض کنید که $SP = FFFEH$ بوده و کد زیر بخشی از برنامه اسمبل نشده در **DEBUG** باشد:

```
12B0:0200 BB1295
12B0:0203 E8FA00
12B0:0206 B82F14
```

چون دستور **CALL** یک فراخوانی **NEAR** است، فقط **IP** در پشته ذخیره می‌شود. در این حال، آدرس **IP** دستور بعد از فراخوانی در پشته طبق شکل ۵-۲ ذخیره می‌گردد. این **IP**، **0206** است که متعلق به دستور **"MOV AX, 142F"** می‌باشد.

آخرین دستور زیر روال فرا خوانده شده باید یک دستور **RET** باشد که **CPU** را برای **POP** (بازیافت) دو بایت بالای پشته به **IP** و ادامه اجرا در آدرس **0206** وامی‌دارد. باین دلیل تعداد دستورات **PUSH** و **POP** باید با هم مطابقت داشته باشند (این دستورات **SP** را تغییر می‌دهند). به بیان دیگر بازاء هر **PUSH** باید یک **POP** وجود داشته باشد.

12B0:0300 53	PUSH BX
12B0:0301 ...	
..... ..	
12B0:0309 5B	POP BX
12B0:030A C3	RET



شکل ۵-۲. **IP** در پشته

زیر روال‌های زبان اسمبلی

در برنامه‌نویسی با زبان اسمبلی، معمول این است که یک برنامه اصلی و چند زیر روال وجود داشته باشد که از برنامه اصلی فراخوانی می‌شوند. این شما اجازه می‌دهد تا هر زیر روال را در یک مدول جداگانه بنویسید. هر مدول قابل تست جداگانه بوده و سپس طبق آنچه در فصل ۷ آمده است بیکدیگر متصل می‌گردند.

برنامه اصلی نقطه ورود از **DOS** و همانطور که قبلاً عنوان شد از نوع **FAR** می‌باشد ولی زیر روال‌های درون برنامه اصلی می‌توانند از نوع **FAR** یا **NEAR** باشند. مجدداً بیاد می‌آورید که روال‌های **NEAR** در همان قطعه کد می‌باشد در حالی که روال‌های **FAR** خارج از قطعه کد جاری است. اگر پس از **PROC** ذکر از **FAR** نشود، پیش فرض **NEAR** است، شکل ۶-۲. از این پس همه قطعات کد در این قالب نوشته می‌شوند.

CODSEG	SEGMENT		
MAIN	PROC FAR		;THIS IS THE ENTRY POINT FOR DOS
	ASSUME		
	MOV	AX,....	
	MOV	DS,AX	
	CALL	SUBR1	
	CALL	SUBR2	
	CALL	SUBR3	
	MOV	AH,4CH	
	INT	21H	
MAIN	ENDP		
SUBR1	PROC		
	...		
	RET		
SUBR1	ENDP		
SUBR2	PROC		
	...		
	RET		
SUBR2	ENDP		
SUBR3	PROC		
	...		
	RET		
SUBR3	ENDP		
CODSEG	ENDS		
	END	MAIN	;THIS IS THE EXIT POINT

شکل ۶-۲ پوسته زیر روال‌های زبان اسمبلی

قوانین نام‌گذاری در زبان اسمبلی

با انتخاب نام برای برچسب، برنامه‌نویس می‌تواند برنامه را راحت‌تر خوانده و نگهداری کند. در اینجا چند قانون که باید رعایت گردند ذکر می‌شود. اول اینکه هر نام باید منحصر بفرد باشد. نام‌های بکار رفته برای برچسب برنامه‌های زبان اسمبلی متشکل از حروف الفباء به هر دو صورت بزرگ و کوچک، ارقام 0 الی 9 و کاراکترهای خاص مثل علامت سؤال (?)، دوره (.)، در (@)، زیرنویس (_) و علامت دلار (\$) هستند. اولین کاراکتر نام باید یک کاراکتر الفبایی یا کاراکتر خاص باشد. این مکان نمی‌تواند به یک رقم اختصاص یابد. دوره فقط بعنوان اولین کاراکتر بکار گرفته می‌شود، ولی چنین کاری پیشنهاد نمی‌شود زیرا ویرایش‌های اخیر MASM دارای کلمات رزروی است که با آن شروع می‌گردد. نام‌ها می‌توانند تا 31 کاراکتر طول داشته باشند. لیستی از کلمات رزو در پایان پیوست (پ) آمده است.

سؤالات مروری

- ۱- اگر کنترل به خارج از قطعه کد جاری محول شود از نوع NEAR است یا FAR؟
- ۲- اگر پرش شرطی صورت نگیرد، دستور بعدی که اجرا می‌شود چیست؟
- ۳- در محاسبه آدرس هدف برای پرش، جابجایی به محتوای ثبات افزوده می‌شود.
- ۴- مزیت استفاده از عملگر "SHORT" در پرش غیر شرطی چیست؟
- ۵- یک پرش ... در محدوده 128- تا 127+ بایت از IP جدید قرار دارد. یک پرش ... در قطعه کد جاری است، در صورتی که یک پرش ... به خارج از قطعه کد جاری صورت می‌گیرد.
- ۶- CPU چگونه می‌فهمد که بعد از دستور RET به کجا باز گردد؟
- ۷- بطور خلاصه نقش دستور RET را بیان نمایید؟
- ۸- بگوئید که چرا نام‌های زیر نامعتبرند
 (الف) GET.DATA (ب) 1_NUM (پ) TESET-DATA (ت) RET

بخش ۵-۲: انواع داده‌ها و تعریف داده

اسمبلر به کمک رهنمون‌های داده که انواع داده را تعریف می‌کند و کنار نهادن حافظه برای آنها همه نوع داده‌های ریزپردازنده 80x86 را پشتیبانی می‌نماید. در این بخش این رهنمون‌ها را مطالعه می‌کنیم و چگونگی استفاده از آنها را برای نمایش انواع داده‌ها در 80x86 خواهیم دید.

انواع داده‌های 80x86

ریزپردازنده‌های 80x86 بسیاری از انواع داده‌ها را پشتیبانی می‌کنند، ولی نباید عرض آنها بیشتر از 16 بیت باشد زیرا ساینز ثابت‌ها 16 است. این وظیفه برنامه نویس است تا داده‌های بزرگتر از 16 بیت (0000 تا FFFFH یا 0 تا 65535 به دهمی) را برای پردازش بوسیله CPU بشکند. نوع داده‌های بکار رفته با 8088/86 می‌تواند 8 بیت یا 16 بیت و یا مثبت یا منفی باشد. اگر عددی کوچکتر از 8 بیت باشد، باز هم باید بصورت کد 8 بیت با ارقام بالاتر صفر در آید. بطور مشابه، اگر عدد کمتر از 16 بیت باشد باید کلاً 16 بیتی ولی بقیه 0 باشند، مثلاً عدد 5 فقط سه بیت عرض دارد (101) ولی 8088/86 آن را بصورت 05 یا بصورت دودویی "0000 0101" می‌پذیرد. عدد 514 در دودویی "10 0000 0010" می‌باشد. بحث اعداد علامت‌دار تا فصل بعد بتعویق می‌افتد زیرا نمایش و کاربرد آنها منحصر به فرد است.

رهنمون‌های داده اسمبلر

همه اسمبلرهای طراحی شده برای پردازنده‌های 80x86 (8088، 8086، 80186، 80188، 80286، 80386، 80386SX و 80486) برای نمایش داده دارای رهنمون استاندارد هستند. در زیر بعضی از رهنمون‌های داده بکار رفته بوسیله 80x86 و پشتیبانی شده بوسیله همه فروشندگان نرم افزار و سخت افزارهای IBM PC و سازگار با آنها آورده شده‌اند.

ORG (مبدأ)

ORG برای مشخص کردن شروع آدرس تفاوت مکان بکار می‌رود. عددی که بعد از ORG می‌آید می‌تواند به شانزده شانزدهمی یا دهدهی باشد. اگر عدد با H دنبال نشود، دهدهی تصور شده و اسمبلر آن را به مبنای شانزده تبدیل می‌کند. گرچه رهنمون ORG بدفعات برای جدا سازی میدان‌های داده در این کتاب بکار رفته است، می‌توان از آن برای تفاوت مکان قطعه کد (IP) هم استفاده کرد.

DB (تعریف بایت)

رهنمون DB یکی از رهنمون‌هایی است که در اسمبلر بسیار مورد استفاده قرار می‌گیرد. این رهنمون اجازه می‌دهد تا حافظه بصورت گروه‌هایی با ساینز بایت در اختیار قرار گیرند. ضمناً این کوچک‌ترین تخصیص مجاز است. DB می‌تواند برای تعریف اعداد به دهدهی، دودویی، شانزده شانزدهمی و اسکی هم بکار رود. برای دهدهی، D بعد از عدد دهدهی اختیاری است ولی ذکر B برای دودویی و H برای شانزده شانزدهمی الزامی است. بدون توجه به اینکه کدام نوع بکار می‌رود آن را با علامت تک نقل قول ('مثل این') نشان می‌دهیم. اسمبلر کد اسکی را بطور خودکار به اعداد یا کاراکترها اختصاص می‌دهد. DB تنها رهنمونی است که می‌تواند برای تعریف رشته اسکی بزرگتر از دو کاراکتر مورد استفاده قرار گیرد. بنابراین برای همه تعاریف داده اسکی باید بکار رود. در زیر بعضی از مثال‌های DB آمده است:

DATA1	DB 25	;DECIMAL
DATA2	DB 10001001B	;BINARY
DATA3	DB 12H	;HEX
	ORG 0010H	
DATA4	DB '2591'	;ASCII NUMBERS
	ORG 0018H	
DATA5	DB ?	;SET ASIDE A BYTE

```

                ORG 0020H
DATA6          DB 'My name is Joe' ;ASCII CHARACTERS

```

0000 19	DATA1 DB 25	;DECIMAL
0001 89	DATA2 DB 10001001B	;BINARY
0002 12	DATA3 DB 12H	;HEX
0010	ORG 0010H	
0010 32 35 39 31	DATA4 DB '2591'	;ASCII NUMBERS
0018	ORG 0018H	
0018 00	DATA5 DB ?	;SET ASIDE A BYTE
0020	ORG 0020H	
0020 4D 79 20 6E 61 6D 65 20 69 73 20 4A 6F 65	DATA6 DB 'My name is Joe'	;ASCII CHARACTERS

فایل لیست برای مثال های DB

در جدول رشته های اسکی می توان از تک یا جفت علامت نقل قول استفاده کرد. این برای رشته هایی که حاوی یک علامت نقل قول هستند مثل "O' Leary" مفید است.

DUP (کپی)

DUP برای کپی کردن تعداد مفروضی کاراکتر بکار می رود. مثلاً دو روش زیر را برای پر کردن شش حافظه با FFH با هم مقایسه کنید.

```

                ORG 0030H
DATA7          DB 0FFH,0FFH,0FFH,0FFH,0FFH,0FFH ;FILL 6 BYTES WITH FF
                ORG 38H
DATA8          DB 6 DUP(0FFH) ;FILL 6 BYTES WITH FF

```

; the following reserves 32 bytes of memory with no initial value given

```

                ORG 40H
DATA9          DB 32 DUP (?) ;SET ASIDE 32 BYTES

```

;DUP can be used inside another DUP

; the following fills 10 bytes with 99

```

DATA10         DB 5 DUP (2 DUP (99)) ;FILL 10 BYTES WITH 99

```

قطعه زیر 32 بایت را بدون داشتن مقدار اولیه آنها، معکوس می کند

0030	FF FF FF FF FF FF	DATA7	DB	0FFH,0FFH,0FFH,0FFH,0FFH,0FFH	;FILL 6 BYTES WITH FF
0038		DATA8	DB	6 DUP(0FFH)	;FILL 6 BYTES WITH FF
0038 0006[	FF				
	]				
0040		DATA9	DB	32 DUP (?)	;SET ASIDE 32 BYTES
0040 0020[					
??					
	]				
0060		DATA10	DB	5 DUP (2 DUP (99))	;FILL 10 BYTES WITH 99
0060 0005[					
	0002[				
	63				
	]				
	]				

لیست مثال های DUP

DW (تعریف کلمه)

DW برای اختصاص دو بایت از حافظه در هر زمان می‌باشد. DW بطور گسترده‌ای در ریز پردازنده‌های 80286 بکار رفته است زیرا ثبات‌ها 16 بیت عرض دارند. در زیر مثال‌هایی از DW آورده شده است:

	ORG	70H	
DATA11	DW	954	:DECIMAL
DATA12	DW	100101010100B	:BINARY
DATA13	DW	253FH	:HEX
	ORG	78H	
DATA14	DW	9,2,7,0CH,00100000B,5,'HI'	:MISCELLANEOUS DATA
DATA15	DW	8 DUP (?)	:SET ASIDE 8 WORDS

```
0070                                ORG 70H
0070 03BA      DATA11 DW   954                      ;DECIMAL
0072 0954      DATA12 DW  100101010100B            ;BINARY
0074 253F      DATA13 DW  253FH                    ;HEX
0078                                ORG 78H
0078 0009 0002 0007 000C    DATA14 DW  9,2,7,0CH,00100000B,5,'HI' ;MISCELLANEOUS DATA
                               0020 0005 4849
0086 0008[      DATA15 DW  8 DUP (?)                ;SET ASIDE 8 WORDS
                              ]
```

فایل لیست برای مثال‌های DW

EQU (برابر پنداشتن)

این رهنمون برای تعریف یک ثابت بدون اشغال مکان حافظه بکار برده می‌شود. EQU هیچ محلی را برای ذخیره یک قلم داده کنار نمی‌گذارد بلکه یک مقدار ثابت را به پرچسب داده نسبت می‌دهد بطوری که وقتی پرچسب در برنامه ظاهر می‌شود، مقدار ثابت آن بجای پرچسب بکار برده می‌شود. EQU می‌تواند در خارج قطعه داده و حتی در وسط قطعه کد هم بکار رود. از EQU می‌توان برای ثابت شمارنده در روش آدرس‌دهی فوری هم استفاده کرد:

COUNT EQU 25

وقتی که دستور "MOV CX , COUNT" اجرا می شود ثبات CX با 25 بار می گردد. این برخلاف استفاده از DB است:

COUNT DB 25

که در آن وقتی که همان دستور "MOV CX,COUNT" دوباره اجرا شود، روش آدرس دهی مستقیم خواهد بود. سؤال این است که مزیت EQU چیست؟ ابتدا توجه کنید که EQU می تواند در قطعه داده هم باشد.

COUNT	EQU	25
COUNTER1	DB	COUNT
COUNTER2	DB	COUNT

فرض کنید که مقدار ثابتی در چند محل از قطعه داده یا کد مورد استفاده قرار گیرد. با بکارگیری EQU، می‌توان یکبار آن را عوض کرد و سپس اسمبلر همه آنها را عوض خواهد کرد و نیازی نیست که برنامه‌نویس در جهت یافتن آنها تلاش کند.

DD (تعریف جفت کلمه)

رهنمون DD برای استقرار داده‌های 4 بایتی (دو کلمه) بکار می‌رود. مجدداً، داده می‌تواند به دهدهی، دودویی، شانزده شانزدهی باشد. در هر حال داده به مبنای شانزده تبدیل و در مکان‌های حافظه طبق قانون بایت یائین به آدرس یائین و بایت بالا به آدرس بالا قرار می‌گیرد. مثال‌های DD عبارتند از:

```

ORG 00A0H
DATA16 DD 1023 ;DECIMAL
DATA17 DD 10001001011001011100B ;BINARY
DATA18 DD 5C2A57F2H ;HEX
DATA19 DD 23H,34789H,65533

```

```

00A0 ORG 00A0H
00A0 000003FF DATA16 DD 1023 ;DECIMAL
00A4 0008965C DATA17 DD 10001001011001011100B ;BINARY
00A8 5C2A57F2 DATA18 DD 5C2A57F2H ;HEX
00AC 00000023 00034789 DATA19 DD 23H,34789H,65533
      0000FFFF

```

فایل لیست برای مثال های DD

DQ (تعریف چهار کلمه)

DQ برای تخصیص 8 بایت از حافظه (چهار کلمه) بکار می رود. از این رهنمون می توان برای نمایش هر متغیری تا عرض 64 بیت استفاده کرد.

```

ORG 00C0H
DATA20 DQ 4523C2H ;HEX
DATA21 DQ 'HI' ;ASCII CHARACTERS
DATA22 DQ ? ;NOTHING

```

```

00C0 ORG 00C0H
00C0 C223450000000000 DATA20 DQ 4523C2H ;HEX
00C8 4948000000000000 DATA21 DQ 'HI' ;ASCII CHARACTERS
00D0 0000000000000000 DATA22 DQ ? ;NOTHING

```

فایل لیست برای مثال های DQ

DT (تعریف ده بایت)

DT برای تخصیص حافظه به اعداد بسته بندی شده بصورت BCD اختصاص یافته است. کاربرد DT در جمع چند بایتی اعداد BCD در فصل ۳ دیده خواهد شد. فعلاً چگونگی استقرار آنها را در حافظه ملاحظه کنید. به "H" پس از داده نیازی نیست. این رهنمون تا 10 بایت را جای می دهد ولی می توان تا 18 رقم را وارد کرد.

```

ORG 00E0H
DATA23 DT 867943569829 ;BCD
DATA24 DT ? ;NOTHING

```

```

00E0 ORG 00E0H
00E0 299856437986000000 DATA23 DT 867943569829 ;BCD
      00
00EA 000000000000000000 DATA24 DT ? ;NOTHING
      00

```

فایل لیست برای مثال های DT

از DT می توان برای 10 بایت عدد صحیح با بکارگیری "D" استفاده کرد:

```

DEC DT 65535d ;the assembler will convert the decimal
               ;number to hex and store it

```

شکل ۷-۲ تخلیه حافظه بخش داده، از جمله مثال های این بخش را نشان می دهد. درک نحوه ذخیره شدن عملوندها در حافظه اهمیت دارد. با مشاهده رونوشت حافظه ملاحظه می شود که تمام رهنمون های داده از قالب مبتنی بر کوچکتر استفاده می کنند، و این بدان معنی است که بایت کم ارزشتر در حافظه پائین تر و بایت با ارزشتر

```

-D 1066:0 100
1066:0000 19 89 12 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
1066:0010 32 35 39 31 00 00 00 00-00 00 00 00 00 00 00 00 2591.....
1066:0020 4D 79 20 6E 61 6D 65 20-69 73 20 4A 6F 65 00 00 My name is Joe..
1066:0030 FF FF FF FF FF FF 00 00-FF FF FF FF FF FF 00 00 .....
1066:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
1066:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
1066:0060 63 63 63 63 63 63 63 63-63 63 00 00 00 00 00 00 cccccccccc.....
1066:0070 BA 03 54 09 3F 25 00 00-09 00 02 00 07 00 0C 00 :.T.??%.....
1066:0080 20 00 05 00 4F 48 00 00-00 00 00 00 00 00 00 00 ...OH.....
1066:0090 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
1066:00A0 FF 03 00 00 5C 96 08 00-F2 57 2A 5C 23 00 00 00 ....\....rW*\#...
1066:00B0 89 47 03 00 FD FF 00 00-00 00 00 00 00 00 00 00 B#E.....IH.....
1066:00C0 C2 23 45 00 00 00 00 00-49 48 00 00 00 00 00 00 .....
1066:00D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
1066:00E0 29 98 56 43 79 86 00 00-00 00 00 00 00 00 00 00 9.VCy6.....

```

شکل ۷-۲. تخلیه DEBUG

در حافظه بالاتر حافظه ذخیره می‌گردد. مثلاً به حالت "DATA 20 DQ 4523C2" توجه کنید، که در حافظه‌ای که از تفاوت مکان 00C0H شروع می‌شود قرار دارد. C2 یعنی کم ارزشترین بایت در محل 00C0، 23 در 00C1 و 45 که با ارزشترین بایت است در 00C2 واقع است. و نیز باید توجه کرد که برای داده اسکی، فقط رهنمون DB قابل استفاده است که برای تعریف داده با هر طول بکار می‌رود و استفاده از رهنمون‌های DD، DQ یا DT برای رشته‌های اسکی بیش از 2 بایت خطای اسمبلی تولید خواهد کرد. وقتی که DB برای اعداد اسکی بکار می‌رود به چگونگی چیدن معکوس آنها در حافظه توجه نمایید. مثلاً به "DATA4 DB '2591'" در مبداء 10H توجه کنید: 32، یعنی اسکی عدد 2، در محل 10H واقع است؛ 35، اسکی 5، در 11H قرار دارد و الی آخر.

سؤالات مروری

- ۱- رهنمون ... همیشه برای رشته‌های اسکی طولانی‌تر از 2 بایت بکار می‌رود.
- ۲- چند بایت بوسیله موارد زیر تعریف می‌شود.
- ۳- دو قطعه داده زیر را بصورت بایت در تفاوت مکان 10H و 11H تعریف کنید؟ اگر امکان ندارد بگوئید چرا؟

DTSEG	SEGMENT	DTSEG	SEGMENT
	ORG 10H		ORG 10H
DATA_1	DB 72H	DATA_1	DW 7204H
DATA_2	DB 04H	DTSEG	ENDS
DTSEG	ENDS		

- ۴- رهنمون DD برای تخصیص حافظه به داده‌های ... بایستی بکار می‌رود.
- رهنمون DQ برای تخصیص حافظه به داده‌های ... بایستی مورد استفاده قرار می‌گیرد.
- ۵- هدف از رهنمون ORG را بطور خلاصه بگوئید.
- ۶- مزیت استفاده از رهنمون EQU برای تعریف مقدار ثابت چیست؟
- ۷- برای هر یک از رهنمون‌های زیر چند بایت کنار گذاشته می‌شود؟
- HEX_DATA DW 1234H (ب) ASC_DATA DB '1234' (الف)
- ۸- آیا قرارداد ذخیره‌سازی مبتنی بر کوچکتر برای ذخیره داده اسکی معتبر است.

بخش ۶-۲: تعریف قطعه ساده شده

روشی که بوسیله آن قطعات در برنامه فوق تعریف شدند روشی سنتی بنام تعریف قطعه کامل است. در

سال‌های اخیر روش جدیدی برای تعریف قطعه معرفی شده است، که تعریف قطعه ساده شده، یا قالب ساده شده نام دارد. این قالب ساده بوسیله ویرایش MASM 5.0 میکروسافت و بالاتر و ویرایش TASM 1 بورلند و بالاتر و بسیاری از سازگارهای دیگر پشتیبانی می‌شود. گرچه تعریف قطعه ساده شده برای درک بسیار ساده‌تر است، ولی برای کاربرد برنامه‌های قدیمی دانستن تعریف قطعه کامل اساسی است.

مدل حافظه

قبل از اینکه از تعریف قطعه ساده شده استفاده کنیم، مدل حافظه برای برنامه باید انتخاب شود. در میان این انتخاب‌ها برای مدل حافظه می‌توان از SMALL، MEDIUM، COMPACT و LARGE استفاده کرد. مدل SMALL: این مدل یکی از پرمصرف‌ترین مدل‌های حافظه برای برنامه‌های زبان اسمبلی است و برای برنامه‌های ذکر شده در این کتاب کافی است. مدل SMALL حداکثر 64K بایت از حافظه را برای کد و به همان مقدار هم برای داده بکار می‌برد.

مدل MEDIUM: در این مدل، داده باید در همان 64K قرار گیرد ولی کد می‌تواند از 64K تجاوز نماید. مدل COMPACT: این برخلاف مدل MEDIUM است. در حالی که داده می‌تواند بیش از 64K بایت باشد، کد نمی‌تواند.

مدل LARGE: از ترکیب دو مدل قبل، مدل LARGE بدست می‌آید. گرچه این مدل اجازه می‌دهد تا کد و داده از 64K تجاوز کنند، هیچ مجموعه به تنهایی (مثل آرایه) نباید از 64K بیشتر شود. مدل HUGE: هر دو کد و داده می‌توانند از 64K بایت حافظه بیشتر باشند. یک مجموعه تک داده مانند آرایه هم می‌تواند از 64K بایت بیشتر باشد.

مدل دیگری برای حافظه بنام TINY وجود دارد. این مدل همراه با فایل‌های COM که در آن جمع کل حافظه برای کد و داده باید در 64K بایت جای گیرد بکار می‌رود. این مدل نمی‌تواند با تعریف قطعه ساده شده مورد استفاده قرار گیرد.

تعریف قطعه

همانطور که در فصل ۱ ذکر شد، CPU 80x86 دارای چهار ثبات قطعه با نام‌های CS (قطعه کد) DS (قطعه داده)، SS (قطعه پشته)، و ES (قطعه اضافی) می‌باشد. هر خط از برنامه زبان اسمبلی باید به یکی از این قطعات تعلق داشته باشد. قالب تعریف قطعه ساده شده سه رهنمون ساده را استفاده می‌کند: ".CODE"، ".DATA" و ".STACK" که بترتیب مربوط به ثبات‌های CS، DS و SS می‌باشند. با استفاده از این روش، بکارگیری رهنمون‌های SEGMENT و END دیگر لازم نیست. شکل ۸-۲ تعریف قطعه کامل و نوع ساده شده را در کنار هم نشان می‌دهد. بدنبال این شکل برنامه‌های ۲-۲ و ۲-۳ با استفاده از تعریف قطعه ساده شده بازنویسی شده‌اند.

؛FULL SEGMENT DEFINITION

```

;---- stack segment ----
name1 SEGMENT
    DB 64 DUP (?)
name1 ENDS
;---- data segment ----
name2 SEGMENT
DATA1 DW 2345H
DATA2 DW 98F4H
RESULT DW ?
name2 ENDS
;---- code segment ----
name3 SEGMENT
MAIN PROC FAR
    ASSUME ...
    MOV AX,name2
    MOV DS,AX
    ...
MAIN ENDP
name3 ENDS
END MAIN

```

؛SIMPLIFIED FORMAT

```

MODEL SMALL
    STACK 64

;-----
    .DATA
DATA1 DW 2345H
DATA2 DW 98F4H
RESULT DW ?

;-----
    .CODE
MAIN: MOV AX,@DATA
      MOV DS,AX
      ...
      ...
      ...
      ...
      ...
END MAIN

```

شکل ۸-۲. تعریف قطعه کامل در برابر ساده شده

TITLE	PROG2-2 (EXE) Rewritten with simple segment definition		
PAGE	60,132		
	.MODEL SMALL		
	.STACK 32		
	.DATA		
DATA1	DW	234DH,1DE6H,3BC7H,566AH	
	ORG	10H	
SUM	DW	?	
	.CODE		
START:	MOV	AX,@DATA	
	MOV	DS,AX	
	MOV	CX,04	;set up loop counter CX=4
	MOV	DI,OFFSET DATA1	;set up data pointer DI
	SUB	BX,BX	;initialize sum BX
ADD_LP:	ADD	BX,[DI]	;add next word to BX
	INC	DI	;increment data pointer twice
	INC	DI	; to point to next word
	DEC	CX	;decrement loop counter
	JNZ	ADD_LP	;jump if loop counter not zero
	MOV	SI,OFFSET SUM	;set up pointer for sum
	MOV	[SI],BX	;store sum
	MOV	AH,4CH	;set up to return
	INT	21H	;return to DOS
	END	START	

برنامه ۲-۲. با تعریف قطعه ساده شده

برنامه ۲-۳ که در زیر با تعریف قطعه ساده نوشته شده است نشان می دهد که ترتیب قطعات اهمیت ندارد.
برنامه فوق بترتیب زیر کد شده است:

```
.STACK
.DATA
.CODE
```

در صورتی که برنامه زیر بترتیب معکوس کد شده است:

```
.CODE
.DATA
.STACK
```

TITLE	PROG2-3 (EXE) Rewritten with simple segment definition		
PAGE	60,132		
	.MODEL SMALL		
	.CODE		
MAIN:	MOV	AX,@DATA	
	MOV	DS,AX	
	MOV	SI,OFFSET DATA1	;SI points to original DATA
	MOV	DI,OFFSET COPY	;DI points to area DATA to be copied to
	MOV	CX,06H	;CX is loop counter
MOV_LOOP:	MOV	AL,[SI]	;move next byte from DATA to AL
	MOV	[DI],AL	;move AL to copy area
	INC	SI	;increment DATA pointer
	INC	DI	;increment COPY pointer
	DEC	CX	;decrement LOOP counter
	JNZ	MOV_LOOP	;jump if loop counter not zero
	MOV	AH,4CH	;set up to return
	INT	21H	;return to DOS
		.DATA	;notice segments can be in any order
	ORG	10H	
DATA1	DB	25H,4FH,85H,1FH,2BH,0C4H	
	ORG	28H	
COPY	DB	6 DUP (?)	
	.STACK	32	
	END	MAIN	

برنامه ۲-۳. که مجدداً با تعریف قطعه ساده نوشته شده است

بخش ۷-۲: فایل‌های EXE در برابر COM

همه مثال‌های برنامه‌ای که تاکنون طراحی شدند باید در فایل EXE اسمبل و پیوند زده شوند. این بخش به فایل COM خواهد پرداخت که مانند فایل EXE حاوی کدهای ماشین قابل اجرا بوده و می‌تواند در سطح DOS اجرا شود. در پایان این بخش فرآیند تبدیل یک فایل به فایل دیگر نشان داده شده است.

چرا فایل‌های COM؟

با توجه به محدودیت حافظه مواردی وجود دارد که مجبوریم تا کد خیلی فشرده‌ای داشته باشیم. در این موقع فایل COM مفید است. این واقعیت که فایل EXE می‌تواند به هر سائیزی باشد دلیل اصلی کاربرد گسترده فایل‌های EXE است. از طرف دیگر فایل COM بدلیل فشرده بودنش بکار گرفته می‌شود، زیرا این نوع فایل‌ها نمی‌توانند بیش از 64K باشند. دلیل محدودیت به 64K بایت این است که فایل COM بیش از 64K نخواهد بود. برای محدود کردن سائز فایل به 64K بایت لازم است تا داده را در داخل قطعه کد تعریف کنیم و نیز با استفاده از ناحیه انتهایی قطعه کد، پشته تعریف شود. یکی از ویژگی‌های بارز برنامه فایل COM این است که برخلاف فایل EXE، این نوع فایل فاقد قطعه داده جداگانه است. اختلافات بین فایل‌های COM و EXE را می‌توان بصورت جدول ۲-۲ خلاصه کرد.

جدول ۲-۲. قالب فایل EXE در برابر COM

فایل EXE	فایل COM
سائز نامحدود	حداکثر اندازه 64K بایت
تعریف قطعه پشته	قطعه پشته تعریف نمی‌شود
تعریف قطعه داده	قطعه داده در قطعه کد تعریف می‌شود
کد و داده می‌توانند در هر تفاوت مکانی باشند	کد و داده باید از تفاوت مکان 0100 شروع شوند
فایل بزرگ (حافظه زیادی می‌برد)	فایل کوچکتر (حافظه کمتری می‌برد)

دلیل دیگر تفاوت سائز فایل EXE و COM این است که فایل COM فاقد بلوک سرصفحه می‌باشد. بلوک سرصفحه که 512 بایت از حافظه را اشغال می‌کند و مقدم بر هر فایل EXE قرار می‌گیرد حاوی اطلاعاتی مانند سائز، محل آدرس در حافظه، آدرس پشته مدل EXE و غیره است.

```

TITLE      PROG2-4 COM PROGRAM TO ADD TWO WORDS
PAGE
CODSG
            SEGMENT
            ORG     100H
            ASSUME  CS:CODSG,DS:CODSG,ES:CODSG
;-----THIS IS THE CODE AREA
PROGCODE PROC NEAR
            MOV     AX,DATA1      ;move the first word into AX
            ADD     AX,DATA2      ;add the second word
            MOV     SUM,AX        ;move the sum
            MOV     AH,4CH        ;return to DOS
            INT     21H
PROGCODE ENDP
;-----THIS IS THE DATA AREA
DATA1      DW      2390
DATA2      DW      3456
SUM         DW      ?
;-----
CODSG      ENDS
            END     PROGCODE
    
```


برنامه ۲-۴ که در قالب COM نوشته شده است دو کلمه داده را جمع کرده و نتیجه را ذخیره می‌کند. این قالب خیلی شبیه بسیاری از برنامه‌های نوشته شده برای ریزپردازنده‌های 8080/85، یعنی نسل قبل از 8088/86 می‌باشند. در این قالب اگر ابتدا کد و سپس داده وارد شود زمان اسمبل شدن طولانی‌تری را خواهد داشت، بنابراین پیشنهاد می‌شود تا داده اول و سپس کد وارد شود، و برنامه باید بوسیله دستور JUMP از ناحیه داده پرش کند، برنامه ۲-۵.

```

TITLE      PROG2-5 COM PROGRAM TO ADD TWO WORDS
PAGE       60,132
CODSG      SEGMENT
            ASSUME CS:CODSG,DS:CODSG,ES:CODSG
            ORG     100H
START:     JMP     PROGCODE      ;go around the data area
;---THIS IS THE DATA AREA
DATA1     DW      2390
DATA2     DW      3456
SUM        DW      ?
;---THIS IS THE CODE AREA
PROGCODE:  MOV     AX,DATA1      ;move the first word into AX
            ADD     AX,DATA2     ;add the second word
            MOV     SUM,AX       ;move the sum
            MOV     AH,4CH
            INT     21H          ;return to DOS
;-----
CODSG      ENDS
            END      START

```

برنامه ۲-۵

تبدیل از EXE به COM

به منظور کارایی مفیدتر حافظه، اغلب تبدیل یک فایل EXE به COM مورد توجه است. فایل مبدأ طبق آنچه در بالا دیدیم باید به قالب COM درآید و سپس طبق معمول اسمبل و پیوند شود. آنگاه باید وارد یک برنامه کمکی بنام EXE2BIN که همراه DOS است گردد. نقش آن تبدیل فایل EXE به فایل COM است. مثلاً برای تبدیل یک فایل EXE بنام PROG1.EXE در دیسک‌گردان B، با فرض قرار داشتن EXE2 BIN در دیسک‌گردان A، عمل زیر را انجام دهید:

A>EXE2BIN B:PROG1,B:PROG1.COM <return>

توجه داشته باشید که هیچ توسعه EXE برای PROG1 وجود ندارد، زیرا فرض بر این است فایل EXE تبدیل خواهد شد. به یاد داشته باشید که برای تبدیل یک فایل به فایل COM قالب برنامه‌های ۲-۴ و ۲-۵ باید رعایت گردد.

خلاصه

یک برنامه به زبان اسمبلی از یک سری عبارت که دستور یا شبه‌دستورند (و یا رهنمون) تشکیل شده است. دستورات بوسیله اسمبلی به صورت کد ماشین درمی‌آیند. شبه‌دستورات به زبان ماشین ترجمه نمی‌شوند. آنها اسمبلی را در چگونگی ترجمه به کد ماشین راهنمایی می‌کنند. عبارت یک برنامه اسمبلی در قطعات گروه‌بندی شده‌اند. نقاط شروع و ختم قطعات بوسیله شبه‌دستورات SEGMENT و ENDS مشخص می‌شود. دیگر شبه دستورات، که اغلب به آنها رهنمون داده گفته می‌شود، برای تعریف داده در قطعه داده بکار می‌روند. داده می‌تواند دودویی، دهدهی، شانزده شانزدهی یا اسکی باشد.

روند یک برنامه بطور متوالی از یک دستور به دستور دیگر به پیش می‌رود، مگر اینکه دستور انتقال کنترل اجرا شود. انواع دستورات انتقال کنترل در زبان اسمبلی شامل پرش‌های شرطی و غیرشرطی و دستورات فراخوانی است.

۱- در برنامه زیر جاهای خالی را پر کنید. برچسب‌ها منطبق بر اصول نیستند، ولی این فقط یک تمرین است. برنامه در واقع کاری انجام نمی‌دهد.

```

CATSG          SEGMENT
                DB      64 DUP(?)
                ENDS

DOGS
MYDATA         DB      25H
                ENDS

MSG
START_HERE     SEGMENT
                PROC    FAR
                ASSUME  _____:MSG,DS:DOGS;SS:_____
                MOV     _____,DOGS
                MOV     DS,_____
                CALL    JOE
                CALL    TOM
                MOV     AH,4CH
                INT     21H
                ENDP

                PROC    NEAR
                NOP
                NOP
                RET
                ENDP

JOE
TOM            NEAR
                NOP
                NOP
                RET

                ENDS
                END
    
```

۲- مراحل لازم برای رسیدن به یک برنامه آماده به اجرا را لیست کنید

۳- کدام برنامه فایل ".exe" تولید می‌کند؟

۴- کدام برنامه فایل ".obj" تولید می‌نماید.

۵- صحیح یا غلط: فایل ".lst" بوسیله اسمبلر بدون توجه به اینکه برنامه‌نویس بخواهد یا نخواهد تولید می‌شود.

۶- فایل برنامه میداد در بعضی اسمبلرها باید توسعه ".asm" داشته باشد. مثل MASM. آیا این نکته در مورد اسمبلری که شما کار می‌کنید صادق است؟

۷- کدام یک صحیح است دور آن دایره بکشید: عمل پیوند (قبل از، بعد از) اسمبل کردن است.

۸- در بعضی کاربردها ذخیره همه ثبات‌ها در آغاز زیرروال معمول است. فرض کنید که قبل از فراخوانی زیر روال SP = 1288H باشد. محتوای اشاره‌گر پشته و محتوای دقیق حافظه پشته پس از PUSHF برنامه‌ریز چیست

```

1132:0450  CALL  PROC1
1132:0453  INC   BX

PROC1      PROC
            PUSH  AX
            PUSH  BX
            PUSH  CX
            PUSH  DX
            PUSH  SI
            PUSH  DI
            PUSHF
            ....
            PROC1  ENDP
    
```

۹- برای بازگرداندن اطلاعات اولیه به CPU در انتهای فراخوانی یک زیر روال، رشته دستورات POP باید ترتیب خاصی را دنبال کند. یک رشته دستورات POP بنویسید که اطلاعات برنامه 8 را باز گرداند. در هر نقطه محتوای SP را نشان دهید.

۱۰- وقتی CALL اجرا شود، CPU چگونه می فهمد به کجا باز گردد؟

۱۱- در یک FAR CALL ... و ... در پشته ذخیره می شوند در صورتی که در یک NEAR CALL، ... در پشته ذخیره می گردد.

۱۲- تعداد بایت های پشته را که بوسیله NEAR CALL و FAR CALL اشغال می شوند مقایسه کنید.

۱۳- محتوای پشته و اشاره گر پشته را پس از اجرای دستور CALL که در زیر آمده اند پیدا کنید. SUM یک رویه NEAR است. فرض کنید که قبل از اجرای CALL، SS : 1296 می باشد.

```
CS : IP
2450:673A CALL SUM
2450:673D DEC AH
```

۱۴- در زیر بخشی از BIOS در IBM PC که بطور مشروح در فصل ۳ راجع به آن صحبت شده، آمده است. همه پرش ها از نوع کوتاه (short) هستند، یعنی برچسب ها در محدوده 128- تا 127+ می باشند.

```
IP Code
E06C 733F JNC ERROR1
...
E072 7139 JNO ERROR1
...
E08C 8ED8 C8: MOV DS,AX
...
E0A7 EBE3 JMP C8
...
E0AD F4 ERROR1: HLT
```

محاسبات آدرس را برای موارد زیر تحقیق کنید.

JMP C8 (پ) JNO ERROR1 (ب) JNC ERROR1 (الف)

۱۵- تفاوت مکان دقیق هر کاراکتر اسکی یا داده را در زیر بدست آورید.

```
DATA1 ORG 20H
DB '1-800-555-1234'
DATA2 ORG 40H
DB 'Name: John Jones'
DATA3 ORG 60H
DB '5956342'
DATA4 ORG 70H
DW 2560H,1000000000110B
DATA5 ORG 80H
DW 49
DATA6 ORG 80H
DD 25697F6EH
DATA7 ORG 90H
DQ 9E7BA21C99F2H
DATA8 ORG 90H
DT 439997924999828
DATA9 DB 6 DUP (0EEH)
```

۱۶- در برنامه زیر مقداری خطا وجود دارد. خطاها را اصلاح کرده و برنامه را اجرا کنید. صحت آن را از طریق برنامه DEBUG تحقیق کنید. این برنامه چهار کلمه را جمع کرده و نتیجه را ذخیره می کند.

```
TITLE PROBLEM (EXE) PROBLEM 16 PROGRAM
PAGE 60,132
STSEG SEGMENT
DB 32 DUP(?)
STSEG
DTSEG SEGMENT
DATA DW 234DH,DE6H,3BC7H,566AH
SUM ORG 10H
DTSG DW ?
ENDS
```

```

CDSEG      SEGMENT
START:     PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            MOV CX,04                ;SET UP LOOP COUNTER CX=4
            MOV BX,0                ;INITIALIZE BX TO ZERO
            MOV DI,OFFSET DATA     ;SET UP DATA POINTER BX
LOOP1:      ADD BX,[DI]              ;ADD CONTENTS POINTED AT BY [DI] TO BX
            INC DI                  ;INCREMENT DI
            DEC BX                  ;DECREMENT LOOP COUNTER
            JNZ LOOP1              ;JUMP IF LOOP COUNTER NOT ZERO
            MOV SI,OFFSET RESULT    ;LOAD POINTER FOR THE RESULT
            MOV [SI],BX             ;STORE THE SUM
            MOV AH,4CH
            INT 21H
CDSEG:     ENDS
START      ENDP
            END STRT

```

✓ پاسخ به سؤالات مروری

بخش ۱-۲ رهنمون ها و یک برنامه نمونه

۱- شبه دستورات برنامه را برای چگونگی اسمبل کردن راهنمایی می کنند

۲- دستورات، شبه دستورات یا رهنمون ها

۳-

```

STK_SEG    SEGMENT
            DB 64 DUP (?)
STK_SEG    ENDS

DAT_SEG    SEGMENT
HIGH_DAT   DB 95
DAT_SEG    ENDS

COD_SEG    SEGMENT
START:     PROC FAR
            ASSUME CS:COD_SEG,DS:DAT_SEG,SS:STK_SEG
            MOV AX,DAT_SEG
            MOV DS,AX
            MOV AH,HIGH_DAT
            MOV BH,AH
            MOV DL,BH
            MOV AH,4CH
            INT 21H
START      ENDP
COD_SEG    ENDS
            END START

```

۴- (۱) برچسب های قطعه کد برای رهنمون های SEGMENT و ENDS مطابقت ندارند.

(۲) برچسب ENDP با رهنمون PROC مطابقت ندارد

(۳) قطعه داده به رهنمون "ENDS" نیاز دارد نه به "END"

(۴) "MOV AX, DATA SEG" باید "MOV AX, DTSEG" باشد

(۵) END باید برچسب نقطه "MAIN" داشته باشد.

بخش ۲-۲ اسمبل، پیوند و اجرای یک برنامه

۱- (الف) MASM باید فایل "asm" را بعنوان ورودی داشته باشد.

(ب) LINK باید "obj" را بعنوان فایل ورودی داشته باشد.

۲- خروجی های ویرایشگر: (پ) asm

خروجی های اسمبلر: (الف) obj

خروجی های پیوندزن: (پ) exe و (ج) فایل های map

بخش ۳-۲ چند برنامه نمونه دیگر

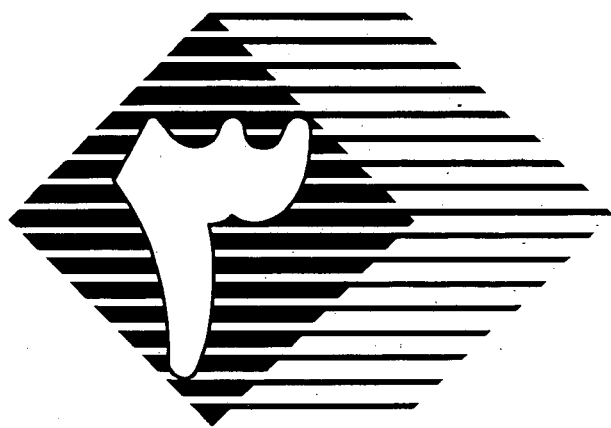
- ۱- عملوند را افزایش می‌دهد، یعنی 1 را با عملوند جمع می‌کند.
- ۲- عملوند را کاهش می‌دهد، یعنی یک واحد از عملوند کم می‌نماید.
- ۳- یک جفت نقطه پس از برچسب‌هایی که به دستورات اشاره می‌کنند؛ جفت نقطه پس از برچسب رهنمون‌ها قرار نمی‌گیرد.
- ۴- اولی محتوای کلمه شروع در تفاوت مکان DATA1 را انتقال می‌دهد، دومی آدرس تفاوت مکان DATA1 را منتقل می‌سازد
- ۵- اولی محتوای BX را به AX می‌افزاید، دومی محتوای حافظه واقع در تفاوت مکان BX را جمع می‌کند.

بخش ۴-۲ دستورات انتقال کنترل

- ۱- دور (FAR)
- ۲- دستور واقع در زیر پرش
- ۳- IP
- ۴- کد ماشین برای دستور یک بایت کمتر است
- ۵- کوتاه (short) نزدیک (near) دور (far)
- ۶- محتوای CS و IP به‌نگام اجرای دستور فراخوانی در پشت‌ذخیره می‌شوند
- ۷- محتوای IP : CS را می‌چرخاند و کنترل را بدستور بعد از CALL باز می‌گرداند.
- ۸- (الف) GET.DAT نامعتبر است زیرا " فقط بعنوان کاراکتر اول مورد قبول است. (ب) 1_NUM باین دلیل نامعتبر است که کاراکتر اول نمی‌تواند یک عدد باشد. (پ) TEST - DATA باین دلیل نامعتبر است که "-" مجاز نیست. (ت) RET یک کلمه رزو شده است و بنابراین نامعتبر است

بخش ۵-۲ انواع داده‌ها و تعریف داده

- ۱- DB
- ۲- 24
- ۳- خیر، زیرا قرارداد ذخیره کردن مبتنی بر کوچکتر که موجب می‌شود 7204H بصورت بایت پائین‌تر (04) در تفاوت مکان 10H و بایت بالاتر در تفاوت مکان 11H قرار گیرد؛ DB هر بایت را همانطور که تعریف شده مستقر می‌نماید.
- ۴- 4 و 8
- ۵- برای تخصیص تفاوت مکان بکار می‌رود.
- ۶- اگر قرار باشد که مقدار بعداً عوض شود، فقط در یک جا عوض می‌شود.
- ۷- (الف) 4 (ب) 2
- ۸- خیر



دستورات محاسباتی، منطقی و برنامه‌ها

اهداف

در پایان این فصل قادر خواهید بود:

- » چگونگی جمع اعداد ۸ بیتی و ۱۶ بیتی بدون علامت در 80x86 را شرح دهید.
- » داده‌ها را به هر یک از فرمهای اسکي، BCD فشرده، یا BCD غير فشرده تبديل نماييد.
- » تأثير دستورات محاسباتي بدون علامت را روی ثبات پرچم توضيح دهيد.
- » دستورات محاسباتي بدون علامت زیر را به زبان اسمبلي کد نماييد:
 - » دستورالعمل‌های جمع ADD و ADC
 - » دستورالعمل‌های تفریق SUB و SBB
 - » دستورالعمل‌های ضرب و تقسیم MUL و DIV
 - » دستورات محاسباتي BCD و اسکي زیر را کد نماييد:
 - » DAA, DAS, AAA, AAS, AAM و AAD
 - » دستورات منطقي زیر را به زبان اسمبلي کد نماييد:
 - » AND, OR و XOR
 - » دستورات نقل مکان منطقي SHR و SHL
 - » دستور مقایسه CMP
 - » دستورات محاسباتي اسکي و BCD را کد نماييد.
 - » دستورات چرخشی بيتي، ROR, ROL, RCR و RCL را کد نماييد.
 - » قابليت استفاده همه دستورات فوق را در برنامه‌های زبان اسمبلي نشان دهيد.
 - » عمليات بيتي را در زبان C انجام دهيد.

در این فصل، اغلب دستورات محاسباتی و منطقی مورد بحث قرار گرفته‌اند و مثالهایی از برنامه‌نویسی نیز آورده شده‌اند تا کاربرد این دستورات نشان داده شود. در این بحث دستورات محاسباتی و منطقی اعداد بدون علامت مورد استفاده قرار گرفته‌اند. اعداد علامت‌دار بطور جداگانه در فصل ۶ بحث شده‌اند. در اعداد بدون علامت همه بیت‌ها از نوع داده می‌باشند و هیچ بیتی به عنوان علامت مثبت و یا منفی در نظر گرفته نمی‌شود. این بدان معنی است که عملوند برای داده‌های ۸ بیتی می‌تواند از 00 تا FFH (0 تا 256) و برای داده‌های ۱۶ بیتی از 0000 تا FFFFH (0 تا 65535) باشد. آخرین بخش این فصل عملیات بیتی در زبان C را شرح می‌دهد.

بخش ۱-۳: جمع و تفریق بدون علامت

جمع اعداد بدون علامت

فرم دستور ADD به صورت زیر است.

ADD destination,source ;dest. operand = dest. operand + source operand

دستورات ADD و ADC برای جمع دو عملوند استفاده می‌شوند. عملوند مقصد می‌تواند یک ثبات یا در حافظه باشد. عملوند مبدأ هم می‌تواند یک ثبات، در حافظه و یا عملوند فوری باشد. بخاطر داشته باشید که دستورات حافظه به حافظه هرگز در زبان اسمبلی 80x86 مجاز نمی‌باشد. دستور می‌تواند هر یک از بیت‌های ZF، SF، AF، CF و PF از ثبات پرچم را با توجه به عملوندها، تغییر دهد. اثر دستور جمع بر روی بیت سرزیر در فصل ۶ بحث شده است زیرا در عملیات اعداد علامت‌دار مورد استفاده قرار می‌گیرد. به مثال ۱-۳ توجه نمایید.

مثال ۱-۳: چگونگی تأثیر دستورات زیر را بر روی ثبات پرچم نشان دهید.

```
MOV    AL,0F5H
ADD    AL,0BH

F5H    1111 0101
+ 0BH  + 0000 1011
100H   0000 0000
```

حل:

بعد از دستور جمع، ثبات AL (مقصد) حاوی 00 و پرچم‌ها، نیز به صورت زیر می‌باشند:

- CF = 1 چون یک رقم نقلی از بیت D7 خارج می‌شود
- SF = 0 وضعیت بیت D7 نتیجه را نشان می‌دهد
- PF = 1 تعداد 1 ها صفر است و صفر یک عدد زوج است
- AF = 1 یک رقم نقلی از بیت D3 به D4 انجام شده است.
- ZF = 1 نتیجه عمل صفر است (برای ۸ بیت)

در بحث جمع، دو حالت زیر بررسی خواهند شد:

- ۱- جمع داده‌های به صورت بایتی یا کلمه‌ای
- ۲- جمع داده‌های چند بایتی

مورد ۱: جمع داده‌های بایتی یا کلمه‌ای

در فصل ۲ برنامه‌ای را دیدیم که پنج بایت داده را با هم جمع می‌کرد. جمع نهایی از FFH، که حداکثر مقدار قابل نگهداری در ۸ بیت است، پایین تر گرفته شد. برای محاسبه جمع کل هر تعداد عملوند، پرچم نقلی باید بعد از جمع هر عملوند واریسی گردد. برنامه ۱-۳ (الف) برای نگهداری رقم نقلی پس از جمع عملوندها با ثبات AL از ثبات AH استفاده کرده است.

برنامه‌ای بنویسید که مجموع پنج عدد یک بایتی را محاسبه نماید. هر بایت دستمزد روزانه یک کارگر می‌باشد. این کارگر بیش از 255 دلار (FFH) در روز دستمزد دریافت نمی‌نماید. داده‌های دهمی به صورت زیر است:

```

TITLE      PROG3-1A (EXE)  ADDING 5 BYTES
PAGE       60,132
STSEG      SEGMENT
            DB 64 DUP (?)
STSEG      ENDS

DTSEG      SEGMENT
COUNT     EQU    05
DATA       DB 125,235,197,91,48
            ORG    0008H
SUM        DW     ?
DTSEG      ENDS

CDSEG      SEGMENT
MAIN       PROC  FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV   AX,DTSEG
            MOV   DS,AX
            MOV   CX,COUNT      ;CX is the loop counter
            MOV   SI,OFFSET DATA;SI is the data pointer
            MOV   AX,00         ;AX will hold the sum
BACK:      ADD   AL,[SI]        ;add the next byte to AL
            JNC   OVER         ;If no carry, continue
            INC   AH            ;else accumulate carry in AH
OVER:      INC   SI            ;increment data pointer
            DEC   CX            ;decrement loop counter
            JNZ   BACK         ;if not finished, go add next byte
            MOV   SUM,AX       ;store sum
            MOV   AH,4CH
            INT   21H          ;go back to DOS
MAIN       ENDP
CDSEG      ENDS
            END    MAIN

```

برنامه ۱-۳ (الف)

تحلیل برنامه ۱-۳ (الف)

این اعداد توسط اسمبلی به صورت زیر به مبنای شانزده تبدیل می‌گردند: 235=0EBH, 125=7DH, 197=0C5H, 91=5BH, 48=30H. سه تکرار حلقه به صورت زیر نشان داده شده‌اند. دنبال نمودن برنامه به عنوان یک تمرین به خواننده واگذار شده است.

۱- در اولین نوبت از تکرار حلقه، 7DH به ثبات AL افزوده می‌شود و CF=0 و نیز AH=00 خواهد شد و ضمناً CX=04 و ZF=0 است.

۲- در دومین تکرار حلقه، EBH به AL افزوده می‌شود، بنابراین نتیجه AL=68H و CF=1 خواهد بود. چون یک رقم نقلی رخ می‌دهد محتوای ثبات AH افزایش خواهد یافت.

۳- در سومین تکرار حلقه، C5H به محتوای ثبات AL افزوده می‌شود، که در نتیجه AL=2DH خواهد شد. دوباره یک رقم نقلی رخ می‌دهد، بنابراین AH مجدداً افزایش می‌یابد و در نتیجه CX=02 و ZF=0 می‌گردد.

این فرآیند تا CX=00 ادامه می‌یابد و آنگاه پرچم صفر 1 خواهد شد. این موجب می‌شود تا برنامه در دستور JNZ از حلقه خارج شود. سپس نتیجه در حافظه کلمه‌ای که در قطعه داده‌ها قرار دارد ذخیره می‌گردد. هرچند این برنامه به درستی کار می‌کند، اما به سبب پردازش لوله‌ای قویاً توصیه می‌گردد که سطرهای زیر در برنامه جایگزین گردند:

سطرهای اولیه		سطرهای جایگزین	
BACK: ADD	AL,[SI]	BACK: ADD	AL,[SI]
JNC	OVER	ADC	AH,00 ;add 1 to AH if CF=1
INC	AH	INC	SI
OVER: INC	SI		

دستور "ADC AH,00" در واقع به معنی جمع $00+AH+CF$ است که نتیجه هم در AH قرار می‌گیرد. این روش کارآتر است، زیرا دستور "JNC OVER" باعث می‌شود که صف دستورات پردازش لوله‌ای خالی گردد و هر بار که رقم نقلی صفر شود ($CF=0$) دستورات از هدف OVER برداشت شوند.

جمع عملوندهای چند کلمه‌ای به همین صورت عمل می‌نماید. ثبات AX (یا CX یا DX یا BX) می‌تواند به عنوان انباره استفاده گردد و ثبات BX (یا هر ثبات همه منظوره 16 بیتی دیگر) برای نگهداری رقم نقلی استفاده می‌شود. برنامه ۳-۱ (ب) همانند برنامه ۳-۱ (الف) است و برای جمع کلمه‌ای دوباره‌نویسی شده است.

برنامه‌ای بنویسید که جمع کل پنج کلمه داده را محاسبه نماید. هر مقدار داده دستمزد سالیانه یک کارگر را نشان می‌دهد. این شخص سالانه بیش از \$ 65535 (FFFFH) دریافت نمی‌نماید. داده‌های دهمی به صورت زیر می‌باشند: 32375، 30105، 23533، 28521، 27345.

```

TITLE      PROG3-1B (EXE)  ADDING 5 WORDS
PAGE       60,132
STSEG      SEGMENT
DB         64 DUP (?)
STSEG      ENDS

DTSEG      SEGMENT
COUNT     EQU      05
DATA       DW        27345,28521,29533,30105,32375
           ORG        0010H
SUM        DW        2 DUP(?)
DTSEG      ENDS

CDSEG      SEGMENT
MAIN       PROC      FAR
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
MOV        AX,DTSEG
MOV        DS,AX
MOV        CX,COUNT      ;CX is the loop counter
MOV        SI,OFFSET DATA;SI is the data pointer
MOV        AX,00          ;AX will hold the sum
MOV        BX,AX          ;BX will hold the carries
BACK:      ADD        AX,[SI] ;add the next word to AX
           ADC        BX,0    ;add carry to BX
           INC        SI      ;increment data pointer twice
           SI          ; to point to next word
           DEC        CX      ;decrement loop counter
           JNZ        BACK    ;if not finished, continue adding
           MOV        SUM,AX  ;store the sum
           MOV        SUM+2,BX ;store the carries
           MOV        AH,4CH
           INT        21H      ;go back to DOS
MAIN       ENDP
CDSEG      ENDS
END        MAIN

```

برنامه ۳-۱ (ب)

مورد ۲: جمع اعداد چند کلمه‌ای

فرض کنید برنامه‌ای برای بودجه کشور ایالات متحده آمریکا در یکصد سال گذشته یا جرم کلیه سیارات منظومه شمسی مورد نیاز باشد. در چنین مواردی اعداد باید 8 بایت و یا بیشتر باشند. چون ثباتها 16 بیتی (دوبایت) هستند وظیفه برنامه‌نویس این است که این اعداد بزرگ را به بخش‌های کوچکتری چنان تقسیم نماید که قابل پردازش توسط CPU باشد. اگر ثبات 16 بیتی استفاده شود و عملوند 8 بیتی باشد، در اینصورت 4 مرتبه تکرار لازم خواهد بود. با این حال، اگر از ثبات 8 بیتی استفاده گردد، همان عملوند نیاز به 8 مرتبه تکرار خواهد شد. این کار وقت بیشتری از CPU را خواهد گرفت. این یکی از دلایل استفاده از ثبات‌های عریض‌تر در CPU می‌باشد. کامپیوترهای بزرگ و قدرتمندی چون CRAY دارای ثباتهای 64 بیتی و بزرگتر می‌باشند.

برنامه‌ای بنویسید که دو عدد چند کلمه را با هم جمع نماید و نتیجه را ذخیره کند:

DATA1=54FB9963CE7H و DATA2=3FCD4FA23B8DH

```

TITLE    PROG3-2 (EXE)  MULTIWORD ADDITION
PAGE     60,132
STSEG    SEGMENT
          DB 64 DUP (?)
STSEG    ENDS

-----
DTSEG    SEGMENT
DATA1    DQ    548FB9963CE7H
          ORG   0010H
DATA2    DQ    3FCD4FA23B8DH
          ORG   0020H
DATA3    DQ    ?
DTSEG    ENDS

-----
CDSEG    SEGMENT
MAIN     PROC    FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV     AX,DTSEG
          MOV     DS,AX
          CLC                     ;clear carry before first addition
          MOV     SI,OFFSET DATA1 ;SI is pointer for operand1
          MOV     DI,OFFSET DATA2 ;DI is pointer for operand2
          MOV     BX,OFFSET DATA3 ;BX is pointer for the sum
          MOV     CX,04            ;CX is the loop counter
BACK:     MOV     AX,[SI]           ;move the first operand to AX
          ADC     AX,[DI]           ;add the second operand to AX
          MOV     [BX],AX          ;store the sum
          INC     SI               ;point to next word of operand1
          INC     SI
          INC     DI               ;point to next word of operand2
          INC     DI
          INC     BX               ;point to next word of sum
          INC     BX
          LOOP    BACK             ;if not finished, continue adding
          MOV     AH,4CH
          INT     21H              ;go back to DOS
MAIN     ENDP
CDSEG    ENDS
END       MAIN

```

برنامه ۲-۳

تحلیل برنامه ۲-۳

در نوشتن این برنامه، استفاده از رهنمون از زمره تصمیمات اولیه برای کد کردن داده در قطعه داده بوده است. در اینجا از DQ استفاده شده است زیرا می‌تواند داده را به بزرگی ۸ بایت نشان دهد. سؤال این است که از چه دستور جمعی باید استفاده نماییم؟ در جمع اعداد چند بایتی (چند کلمه‌ای) دستور ADC همیشه استفاده می‌شود زیرا رقم نقلی باید به بایت بالاتر بعدی (کلمه) در تکرار بعدی اضافه گردد. قبل از اجرای دستور ADC پرچم نقلی باید صفر شود ($CF = 0$) بطوریکه در اولین تکرار، رقم نقلی نباید اضافه گردد. صفر نمودن پرچم نقلی توسط دستور CLC (صفر نمودن بیت C) انجام می‌گیرد. سه اشاره گر استفاده شده است: SI برای DATA1، DI برای DATA2 و BX برای DATA3 که نتایج در آن ذخیره می‌گردد. دستورالعمل جدیدی در این برنامه قرار دارد، LOOP xxxx، که جایگزین دستوراتی که زیاد مورد استفاده قرار می‌گیرند یعنی "DEC CX" و "JNZ xxxx" می‌شود. به عبارت دیگر:

هنگامی که "LOOP xxxx" اجرا می‌شود، CX به طور خودکار یک واحد کاهش می‌یابد، اگر CX صفر نباشد، ریزپردازنده به آدرس xxxx جهش خواهد نمود. اگر CX مساوی صفر باشد دستورالعمل بعدی (دستور زیر "LOOP xxxx") اجرا خواهد شد.

تفریق اعداد بدون علامت

SUB dest,source ;dest = dest - source

در تفریق، ریزپردازنده‌های 80x86 (در حقیقت، تقریباً همه CPU های جدید) از روش متمم دو استفاده می‌نمایند. چون CPU مدار جمع دارد، طراحی مدار تفریق‌گر کار بی‌هوده و پرهزینه خواهد بود (و تعداد زیادی ترانزیستور نیاز می‌باشد). به همین دلیل، پردازنده‌های 80x86 از مدار جمع داخلی برای انجام فرمان تفریق استفاده می‌نمایند. فرض کنید که، 80x86 در حال اجرای دستورات تفریق ساده می‌باشد. می‌توان مراحل سخت‌افزاری CPU را در اجرای دستورات ساده تفریق برای اعداد بدون علامت به صورت زیر در نظر گرفت.

- ۱- متمم دو مفروق را به دست می‌آورد (عملوند مبدأ).
- ۲- حاصل مرحله یک را با مفروق منه جمع می‌نماید (عملوند مقصد)
- ۳- رقم نقلی معکوس می‌شود.

این سه مرحله برای هر دستور SUB توسط سخت‌افزار داخلی CPU 80x86 مادامی‌که بوسیله روش آدرس‌دهی پشتیبانی شود بدون توجه به عملوندهای مبدأ و مقصد اجرا می‌شود. بعد از اجرای این سه مرحله نتیجه حاصل و پرچم‌ها مقدار می‌گیرند مثال ۲-۳ این سه مرحله را تشریح می‌کند

مثال ۲-۳: مراحل استفاده شده در دستورات زیر را نشان دهید

```
MOV AL,3FH ;load AL=3FH
MOV BH,23H ;load BH=23H
SUB AL,BH ;subtract BH from AL. Place result in AL
```

حل:

AL	3F	0011 1111	0011 1111	
- BH	- 23	- 0010 0011	+1101 1101	(2's complement)
	1C		1 0001 1100	CF=0 (step 3)

پرچم‌ها به صورت زیر مقدار می‌گیرند CF=0, ZF=0, AF=0, PF=0 و SF=0. برنامه‌نویس باید با توجه به پرچم نقلی (و نه پرچم علامت) مثبت یا منفی بودن نتیجه را مشخص کند.

بعد از اجرای دستور SUB، اگر CF=0 باشد نتیجه عملیات مثبت است؛ اگر CF=1 باشد نتیجه منفی است و متمم 2 نتیجه در مقصد قرار می‌گیرد. معمولاً، نتیجه به صورت متمم 2 باقی می‌ماند ولی از دستورات NOT و INC برای تغییر آن می‌توان استفاده نمود. دستور NOT متمم 1 عملوند را انجام می‌دهد؛ سپس یک واحد به عملوند افزوده می‌شود تا متمم دو حاصل گردد. مثال ۳-۳ را ملاحظه نمایید.

مثال ۳-۳: برنامه زیر را تحلیل نمایید:

از قطعه داده:

```
DATA1 DB 4CH
DATA2 DB 6EH
DATA3 DB ?
```

از قطعه کد:

```
MOV DH,DATA1 ;load DH with DATA1 value (4CH)
SUB DH,DATA2 ;subtract DATA2 (6E) from DH (4CH)
JNC NEXT ;if CF=0 jump to NEXT target
NOT DH ;if CF=1 then take 1's complement
INC DH ;and increment to get 2's complement
NEXT: MOV DATA3,DH ;save DH in DATA3
```

حل:

با پیگیری سه مرحله برای "SUB DH,DATA2"

4C	0100 1100	0100 1100	
- 6E	0110 1110	2's comp +1001 0010	
- 22		0 1101 1110	CF=1 نتیجه منفی است (مرحله ۳)

تفریق با قرض، SBB

این دستور برای اعداد چند بایتی (چند کلمه‌ای) استفاده می‌شود و رقم قرضی عملوند پایین رتبه را بکار می‌گیرد. اگر پرچم نقلی 0 باشد، دستور SBB مانند دستور SUB کار می‌کند. اگر پرچم نقلی 1 باشد، دستور SBB یک واحد از نتیجه کم می‌نماید. به عملوند "PTR" در مثال ۳-۴ توجه کنید. رهنمون توصیف‌گر داده PTR برای مشخص کردن ساینز عملوند هنگامی که با ساینز تعریف شده تفاوت داشته باشد بطور گسترده‌ای مورد استفاده قرار می‌گیرد. در مثال ۳-۴ "WORD PTR" به اسمبلی می‌گوید تا در تعریف داده، از عملوند کلمه‌ای استفاده کند، هرچند که داده به صورت جفت کلمه تعریف شده باشد.

مثال ۳-۴: برنامه زیر را تحلیل نمایید:

DATA_A	DD 62562FAH		
DATA_B	DD 412963BH		
RESULT	DD ?		
...	...		
MOV	AX,WORD PTR DATA_A		;AX=62FA
SUB	AX,WORD PTR DATA_B		;SUB 963B from AX
MOV	WORD PTR RESULT,AX		;save the result
MOV	AX,WORD PTR DATA_A+2		;AX=0625
SBB	AX,WORD PTR DATA_B+2		;SUB 0412 with borrow
MOV	WORD PTR RESULT+2,AX		;save the result

حل:

بعد از انجام دستور SUB، $AX=62FA-963B=CCBF$ و پرچم نقلی مقدار 1 می‌گیرد. چون $CF=1$ است، هنگامی که دستور SBB اجرا می‌شود $AX=625-412-1=212$ خواهد شد. بنابراین مقداری که در RESULT قرار می‌گیرد مساوی با 0212CCBF است.

سؤالات مروری

- دستور جمع دارای شکل نحوی "ADD destination, source" می‌باشد که عملوند را با جمع دو عملوند جایگزین می‌کند.
- چرا دستور ADD زیر غیر مجاز است؟
- دستور فوق را به شکل درست بازنویسی نمایید.
- دستور ADC که دارای شکل نحوی "ADC destination, source" می‌باشد عملوند را با مجموع جایگزین می‌نماید.
- در اجرای بخش (الف) زیر $ZF=1$ خواهد شد، در صورتیکه در اجرای بخش (ب) $ZF=0$ می‌گردد. توضیح دهید چرا؟

MOV BX,04FH (ب)	MOV BL,04FH (الف)
ADD BX,0B1H	ADD BL,0B1H

- دستورالعمل "LOOP ADD_LOOP" معادل کدام دو دستور است؟
- نشان دهید که چگونه CPU عدد 05H را از 43H کم می‌نماید؟
- اگر قبل از اجرای دستور BL و AL داشته باشیم $CF=1$ و $AL=95$ و $BL=4F$ محتوی AL بعد از اجرای این دستور تفریق چقدر خواهد بود.

بخش ۲-۳: ضرب و تقسیم بدون علامت

یکی از تغییرات عمده از ریزپردازنده 8080/85 به 8086 افزودن دستورات ضرب و تقسیم می‌باشد. در این بخش هر یک از این دستورات را با یک مثال توضیح می‌دهیم. این دستورات ضرب و تقسیم اعداد بدون علامت می‌باشند، اعداد علامت‌دار در فصل ۶ مطالعه خواهند شد.

در دستورات ضرب و تقسیم دو عدد در ریزپردازنده 80x86، استفاده از ثباتهای AX، AL، AH و DX

الزامی می باشد زیرا دستورات به طور ضمنی از این ثباتها استفاده می نمایند.

ضرب اعداد بدون علامت

در مبحث ضرب، موارد زیر مطالعه خواهند شد (۱) بایت ضرب در بایت (۲) کلمه ضرب در کلمه و (۳) بایت ضرب در کلمه.

بایت x بایت : در ضرب بایت در بایت، یکی از عملوندها باید در ثبات AL قرار داشته باشد، و دومین عملوند می تواند در ثبات و یا حافظه باشد. در هر صورت روش آدرس دهی آن باید یکی از روش های بحث شده در فصل ۱ باشد. بعد از انجام عمل ضرب نتیجه در ثبات AX قرار می گیرد. به مثال زیر توجه نمایید:

```

RESULT DW ? ;result is defined in the data segment

...
MOV AL,25H ;a byte is moved to AL
MOV BL,65H ;immediate data must be in a register
MUL BL ;AL = 25 x 65H
MOV RESULT,AX ;the result is saved

```

در برنامه بالا، 25H در 65H ضرب می شود و نتیجه در حافظه RESULT به طول یک کلمه قرار می گیرد. در آن مثال از روش آدرس دهی ثباتی استفاده شده است. سه مثال بعدی نحوه آدرس دهی ثبات، مستقیم و غیر مستقیم ثباتی را نشان می دهد.

؛از قطعه داده:

```

DATA1 DB 25H
DATA2 DB 65H
RESULT DW ?

```

؛از قطعه کد:

```

MOV AL,DATA1
MOV BL,DATA2
MUL BL ;register addressing mode
MOV RESULT,AX

-or

MOV AL,DATA1
MUL DATA2 ;direct addressing mode
MOV RESULT,AX

or

MOV AL,DATA1
MOV SI,OFFSET DATA2
MUL BYTE PTR [SI] ;register indirect addressing mode
MOV RESULT,AX

```

در مثال آدرس دهی با استفاده از ثبات، هر ثبات ۸ بیتی می تواند بجای BL بکار برده شود. به همین ترتیب، در مثال آدرس دهی غیرمستقیم از ثباتهای BX یا DI می توان به عنوان اشاره گر استفاده نمود.

اگر روش آدرس دهی غیرمستقیم ثباتی بکار برده شود، با استفاده از شبه دستور PTR اندازه عملوند باید مشخص گردد. در غیاب رهنمون "BYTE PTR" در مثال بالا اسمبلی نمی تواند تعیین کند که آیا عملوندی که توسط SI آدرس داده می شود یک بایتی است یا یک کلمه. این سردرگمی باعث بروز خطا می گردد.

کلمه x کلمه: در ضرب کلمه در کلمه، یک عملوند باید در ثبات AX قرار گیرد و دومین عملوند می تواند در ثبات قرار گیرد یا در حافظه. بعد از انجام عمل ضرب نتیجه در ثباتهای AX و DX قرار خواهد داشت. چون ضرب کلمه در کلمه می تواند یک نتیجه 32 بیتی تولید نماید، بنابراین کلمه پایین تر حاصل ضرب در ثبات AX و کلمه بالاتر حاصل ضرب در ثبات DX قرار خواهد داشت. مثال:

DATA3 DW 2378H
DATA4 DW 2F79H
RESULT1 DW 2 DUP(?)

```

.....
MOV AX,DATA3      ;load first operand into AX
MUL DATA4         ;multiply it by the second operand
MOV RESULT1,AX     ;store the lower word result
MOV RESULT1+2,DX   ;store the higher word result

```

بایت × کلمه: این ضرب شبیه ضرب کلمه در کلمه می باشد بجز اینکه AL حاوی بایت عملوند بوده و AH باید صفر شود. مثال:

DATA5 DB 6BH
DATA6 DW 12C3H
RESULT3 DW 2 DUP(?)

از قطعه داده :

```

... ..
MOV AL,DATA5      ;AL holds byte operand
SUB AH,AH         ;AH must be cleared
MUL DATA6        ;byte in AL multiplied by word operand
MOV BX,OFFSET RESULT3 ;BX points to storage for product
MOV [BX],AX       ;AX holds lower word
MOV [BX]+2,DX     ;DX holds higher word

```

از قطعه کد :

جدول ۱-۳ خلاصه ای از ضرب اعداد بدون علامت را نشان می دهد. گرچه در اجرای ضرب با عملوندهای بزرگتر از ۱۶ بیت از پردازنده کمکی 8087 استفاده می شود ولی بکارگیری ریزپردازنده 80x86 بجای آن نیاز به مقداری دستکاری دارد.

جدول ۱-۳. خلاصه ای از ضرب اعداد بدون علامت

نتیجه	عملوند دوم	عملوند اول	ضرب
AX	ثبات یا حافظه	AL	بایت × بایت
DXAX	ثبات یا حافظه	AX	کلمه × کلمه
DXAX	ثبات یا حافظه	بایت AL = 0 و AH	بایت × کلمه

تقسیم اعداد بدون علامت

در تقسیم اعداد بدون علامت، موارد زیر بحث می گردند:

- ۱- بایت تقسیم بر بایت
- ۲- کلمه تقسیم بر کلمه
- ۳- کلمه تقسیم بر بایت
- ۴- کلمه مضاعف تقسیم بر کلمه

در تقسیم، مواردی پیش می آید که CPU قادر به انجام عمل تقسیم نیست. در این موارد یک وقفه فعال می گردد که به آن استثناء گویند. در چه مواردی ریزپردازنده نمی تواند عمل تقسیم انجام دهد و باید وقفه را فراخوانی نماید؟ موارد به صورت زیر می باشند:

- ۱- اگر مقسوم علیه صفر باشد (هر عددی را بر صفر تقسیم نماییم)، و
- ۲- اگر خارج قسمت خیلی بزرگ باشد که در ثبات اختصاص یافته جای نگیرد.

در PC های IBM و سازگار با آنها اگر هر یک از موارد فوق اتفاق افتاد PC پیام "divide error" را نمایش خواهد داد.

بایت بر بایت: در تقسیم بایت بر بایت، مقسوم باید در ثبات AL قرار گیرد و ثبات AH صفر شود. مقسوم علیه نمی تواند یک عملوند فوری باشد ولی می تواند در ثبات یا حافظه قرار بگیرد. چنانچه مقسوم علیه در حافظه قرار گرفت آدرس دهی آن باید قابل قبول باشد. بعد از اجرای دستور DIV، خارج قسمت در AL و باقیمانده در ثبات AH قرار خواهد گرفت. در زیر روشهای مختلف آدرس دهی که مقسوم علیه می تواند داشته باشد نشان داده شده است.

```
DATA7    DB    95
DATA8    DB    10
QOUT1    DB    ?
REMAIN1  DB    ?
```

;using immediate addressing mode will give an error

```
MOV     AL,DATA7    ;move data into AL
SUB     AH,AH       ;clear AH
DIV     10          ;immed. mode not allowed!!
```

;allowable modes include:

;using direct mode

```
MOV     AL,DATA7    ;AL holds numerator
SUB     AH,AH       ;AH must be cleared
DIV     DATA8      ;divide AX by DATA8
MOV     QOUT1,AL    ;quotient = AL = 09
MOV     REMAIN1,AH  ;remainder = AH = 05
```

;using register addressing mode

```
MOV     AL,DATA7    ;AL holds numerator
SUB     AH,AH       ;AH must be cleared
MOV     BH,DATA8    ;move denom. to register
DIV     BH           ;divide AX by BH
MOV     QOUT1,AL    ;quotient = AL = 09
MOV     REMAIN1,AH  ;remainder = AH = 05
```

;using register indirect addressing mode

```
MOV     AL,DATA7    ;AL holds numerator
SUB     AH,AH       ;AH must be cleared
MOV     BX,OFFSET DATA8 ;BX holds offset of DATA8
DIV     BYTE PTR [BX] ;divide AX by DATA8
MOV     QOUT2,AX
MOV     REMAIND2,DX
```

کلمه بر کلمه: در این حالت مقسوم در AX قرار می گیرد و DX باید صفر گردد. مقسوم علیه می تواند در یک ثبات یا حافظه باشد. بعد از اجرای دستور تقسیم، خارج قسمت در AX و باقیمانده در DX قرار می گیرند.

```
MOV     AX,10050    ;AX holds numerator
SUB     DX,DX       ;DX must be cleared
MOV     BX,100      ;BX used for denominator
DIV     BX
MOV     QOUT2,AX    ;quotient = AX = 64H = 100
MOV     REMAIND2,DX ;remainder = DX = 32H = 50
```

کلمه بر بایت: دوباره، مقسوم در AX و مقسوم علیه در ثبات یا حافظه قرار می گیرند. بعد از اجرای دستور DIV، خارج قسمت در ثبات AL و باقیمانده در ثبات AH قرار خواهد داشت. حداکثر مقدار خارج قسمت FFH می باشد. برنامه زیر مقدار AX=2055 را بر مقدار CL=100 تقسیم می نماید. خارج قسمت (20 دهمی) AL=14H و باقیمانده (55 دهمی) AH=37H خواهد بود.

```
MOV    AX,2055    ;AX holds numerator
MOV    CL,100     ;CL used for denominator
DIV    CL
MOV    QUO,AL      ;AL holds quotient
MOV    REMI,AH     ;AH holds remainder
```

کلمه مضاعف بر کلمه: مقسوم در ثبات AX و DX می باشد به نحوی که با ارزش ترین کلمه در ثبات DX و کم ارزش ترین کلمه در ثبات AX قرار می گیرد. مقسوم علیه می تواند در یک ثبات و یا حافظه باشد. بعد از انجام دستور DIV خارج قسمت در ثبات AX و باقیمانده در ثبات DX خواهد بود: حداکثر مقدار خارج قسمت FFFFH می باشد.

```
DATA1  DD    105432
DATA2  DW    10000
QUOT   DW    ?
REMAIN DW    ?
```

از قطعه داده :

```
MOV    AX,WORD PTR DATA1    ;AX holds lower word
MOV    DX,WORD PTR DATA1+2  ;DX higher word of numerator
DIV    DATA2
MOV    QUOT,AX                ;AX holds quotient
MOV    REMAIN,DX              ;DX holds remainder
```

از قطعه کد :

در برنامه فوق محتویات DX:AX بر داده ای به اندازه یک کلمه که مقدار آن 10000 می باشد تقسیم می گردد. اکنون ممکن است این سؤال پیش آید که: چگونه CPU می داند که مقسوم را باید از یک کلمه مضاعف که در ثباتهای DX:AX قرار دارد، بردارد؟

هنگامی که سائز مقسوم علیه یک کلمه باشد، 8086/88 بطور خودکار DX:AX را بعنوان مقسوم در نظر می گیرد. این مطلب را قبلاً در مورد تقسیم کلمه بر کلمه دیده ایم. بهمین دلیل DX باید قبلاً صفر گردد. در مثال فوق توجه کنید که DATA1 قبلاً بصورت DD تعریف شده است ولی با کمک WORD PTR به ثباتی با سائز کلمه برده شده است. در غیاب WORD PTR، اسمبلر خطایی تولید خواهد کرد. خلاصه ای از نتایج تقسیم اعداد بی علامت در جدول ۲-۳ آمده است.

جدول ۲-۳. خلاصه ای از تقسیم بدون علامت

تقسیم	مقسوم	مقسوم علیه	خارج قسمت	باقیمانده
بایت بر بایت	AH=0 و بایت AX	ثبات یا حافظه	AL ¹	AH
کلمه بر کلمه	DX=0 و کلمه AX	ثبات یا حافظه	AX ²	DX
کلمه بر بایت	کلمه AX	ثبات یا حافظه	AL ¹	AH
کلمه مضاعف بر کلمه	کلمه مضاعف DX:AX	ثبات یا حافظه	AX ²	DX

! توجه:

۱- وقفه خطای تقسیم اگر AL > FFH

۲- وقفه خطای تقسیم اگر AX > FFFFH

- ۱- در ضرب بدون علامت یک بایت در DATA1 با یک بایت در AL، حاصلضرب در ثبات (یا ثباتهای) قرار می‌گیرد.
- ۲- در ضرب بدون علامت AX در BX، حاصلضرب در ثبات (ثباتهای) قرار می‌گیرد.
- ۳- در ضرب بدون علامت CX، با یک بایت در AL، حاصلضرب در ثبات (ثباتهای) قرار می‌گیرد.
- ۴- در تقسیم بدون علامت یک بایت در AL بر یک بایت در DH، خارج قسمت در و باقیمانده در می‌باشد.
- ۵- در تقسیم بدون علامت یک کلمه در AX بر یک کلمه در DATA1، خارج قسمت در و باقیمانده در قرار می‌گیرند.
- ۶- در تقسیم بدون علامت یک کلمه در AX بر یک بایت در DATA2، خارج قسمت در و باقیمانده در قرار می‌گیرند.
- ۷- در تقسیم بدون علامت یک کلمه مضاعف در DXAX، بر یک کلمه که در CX قرار دارد خارج قسمت در و باقیمانده در قرار می‌گیرند.

بخش ۳-۳: دستورات منطقی و برنامه‌های نمونه

در این بخش دستورات منطقی AND، OR، XOR، SHIFT و COMPARE مطالعه می‌شوند. دستورات در قالب مثالها توضیح داده شده‌اند.

AND destination, source

AND

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

این دستور عمل AND منطقی را بر روی عملوندها انجام داده و نتیجه را در عملوند مقصد قرار می‌دهد. عملوند مقصد می‌تواند ثبات یا حافظه باشد. عملوند مبدا هم می‌تواند ثبات، یا در حافظه و یا فوری باشد.

مثال ۳-۵: نتایج دستورات زیر را نشان دهید
حل:

MOV BL, 35H
AND BL, 0FH

;AND BL with 0FH. Place the result in BL.

35H 00110101
0FH 00001111
05H 00000101

مقادیر قرار گرفته در پرچم:

SF = 0, ZF = 0, PF = 1, CF = OF = 0.

دستور AND بطور خودکار CF و OF را به صفر تبدیل می‌نماید و بیت‌های SF و ZF را بر طبق نتیجه عملیات مقدار می‌دهد. بقیه بیت‌های پرچم در اثر اجرای این دستور تغییر نمی‌کنند. همانطوریکه در مثال ۳-۵ مشاهده می‌شود، از دستور AND برای پوشش بیت‌های عملوند استفاده می‌گردد. از این دستور برای آزمون صفر بودن یک عملوند هم استفاده می‌شود.

AND DH, DH
JZ XXXX

XXXX: ...

دستورات فوق ثبات DH را با خودش AND می نماید و اگر نتیجه صفر شد ZF را مساوی 1 می نماید و سپس CPU دستوری که در آدرس xxxx قرار دارد را برداشت می کند. در غیر اینصورت دستور زیر JZ اجرا می گردد. از دستور AND برای آزمون صفر بودن محتوی یک ثبات استفاده می شود.

OR destination, source

OR

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

عملوندهای مقصد و مبداء با هم OR می شوند و حاصل در عملوند مقصد قرار می گیرد. از این دستور برای 1 نمودن بعضی بیت های یک عملوند استفاده می شود. عملوند مبداء می تواند یک ثبات، حافظه و یا یک داده فوری باشد.

پرچمها مثل دستور AND مقدار می گیرند. CF و OF صفر می شوند و ZF، CF و PF بر طبق نتیجه مقدار می گیرند. این دستور بر روی پرچمهای دیگر تأثیری ندارد. مثال ۶-۳ را ملاحظه نمایید.

از دستور OR برای آزمون صفر بودن یک عملوند می توان استفاده نمود. برای مثال "OR BL,0" ثبات BL را با مقدار صفر OR می نماید و اگر BL صفر باشد $ZF = 1$ خواهد شد. دستور "OR BL,BL" همان نتیجه را بدست می دهد.

مثال ۶-۳: نتیجه دستورات زیر را نشان دهید:

MOV AX,0504
OR AX,0DA68H

;AX = 0504
;AX = DF6C

حل:

0504H 0000 0101 0000 0100.
DA68H 1101 1010 0110 1000
DF6C 1101 1111 0110 1100

پرچم ها عبارت خواهند بود از : $SF = 1, ZF = 0, PF = 1, CF = OF = 0$.
توجه داشته باشید که بیت توازن فقط برای ۸ بیت پایین تر بررسی می گردد

XOR dest,src

XOR

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

دستور XOR عملوندها را با هم "OR-انحصاری" می نماید و حاصل را در عملوند مقصد قرار می دهد. دستور XOR بیت های نتیجه را 1 می کند، بشرطی که با هم یکسان نباشند؛ در غیر اینصورت، آنها را صفر می نماید. پرچمها همانند دستور AND تنظیم می شوند. $CF = 0$ و $OF = 0$ در درون مقدار می گیرند و بقیه تغییر می کنند. بیتها بر طبق نتیجه عملیات مقدار می گیرند. قوانین مربوط به عملوندها، همانند دستورات AND و OR می باشد. به مثالهای ۷-۳ و ۸-۳ توجه نمایید.

مثال ۷-۳: نتایج دستورات زیر را نشان دهید:

MOV DH,54H
XOR DH,78H

حل:

54H 01010100
78H 01111000
2C 00101100

پرچم ها عبارت خواهند بود از : $SF = 0, ZF = 0, PF = 0, CF = OF = 0$.

مثال ۸-۳. از دستور XOR می‌توان برای پاک کردن محتوای یک ثبات استفاده کرد. به این ترتیب که آن را با خودش XOR نماییم. اگر AH=45 فرض شود نشان دهید که چگونه AH, XOR AH, AH را پاک می‌کند.

حل:

```
45H 01000101
45H 01000101
00  00000000
```

پرچم‌ها عبارت خواهند بود از: SF = 0, ZF = 1, PF = 1, CF = OF = 0.

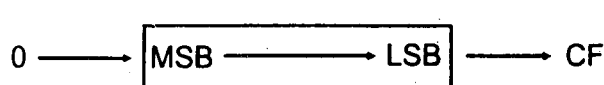
چنانچه بخواهیم ببینیم که دو ثبات دارای مقدار یکسانی هستند از دستور XOR استفاده می‌گردد. دستور "XOR BX, CX" باعث می‌شود تا اگر دو ثبات دارای مقدار یکسانی باشند ZF = 1 گردد. در این صورت، نتیجه (0000) در مقصد BX قرار می‌گیرد. کاربرد متداول دیگر از دستور XOR معکوس کردن بیتیهای یک عملوند است. برای مثال، به منظور معکوس نمودن بیت دوم ثبات AL:

```
XOR AL,04H ;XOR AL with 0000 0100
```

دستور فوق باعث می‌گردد که مقدار بیت دوم AL معکوس شود، و بیتیهای دیگر آن دست نخورده باقی بماند.

نقل مکان

دو نوع دستور نقل مکان (شیفت) وجود دارد: محاسباتی و منطقی. نقل مکان منطقی برای عملوندهای بدون علامت استفاده می‌شود، و نقل مکان محاسباتی برای عملوندهای علامت‌دار می‌باشد. نقل مکان منطقی در این بخش بحث خواهد شد و بحث نقل مکان محاسباتی به بخش ۶ موكول گردیده است. با استفاده از دستورات نقل مکان، محتوی یک ثبات یا یک مکان از حافظه به راست یا چپ نقل مکان داده می‌شود. تعداد دفعاتی (یا بیتیهای) که عملوند نقل مکان داده می‌شود، اگر نقل مکان یکبار باشد مستقیماً در دستور ذکر می‌گردد و اگر دفعات بیش از یکبار باشد از ثبات CL استفاده می‌شود.



SHR: این دستور، نقل مکان منطقی

به راست می‌باشد. عملوند بیت به بیت

به راست نقل مکان می‌یابد، و در هر نقل

مکان، LSB (کم ارزش‌ترین بیت) به پرچم نقلی منتقل می‌شود. همچنین MSB (با ارزش‌ترین بیت) با صفر پر می‌گردد. مثالهای ۹-۳ و ۱۰-۳ به روشن شدن SHR کمک می‌نماید.

مثال ۹-۳: نتیجه SHR را در زیر نشان دهید:

```
MOV AL,9AH
MOV CL,3 ;set number of times to shift
SHR AL,CL
```

حل:

```
9AH = 10011010
01001101 CF=0 (یک بار نقل مکان)
00100110 CF=1 (دو بار نقل مکان)
00010011 CF=0 (سه بار نقل مکان)
```

بعد از سه مرتبه نقل مکان به راست AL = 13H و CF = 0 خواهد شد.

اگر عملوند فقط یکبار نقل مکان یابد، بجای قراردادن 1 در CL، 1 در خوددستور قرار می‌گیرد. این کار یک دستور صرفه‌جویی می‌نماید.

```
MOV BX,0FFFFH ;BX=FFFFH
SHR BX,1 ;shift right BX once only
```

بعد از نقل مکان فوق، $CF = 1$ ، $BX = 7FFFH$ خواهد شد. گرچه SHR بر روی پرچمهای SF ، OF و PF تأثیر می‌گذارد، ولی در این مورد چندان مهم نیستند. عملوند نقل مکان یافته می‌تواند در ثبات یا در حافظه باشد، اما استفاده از روش آدرس‌دهی فوری در دستورات نقل مکان مجاز نمی‌باشد. برای مثال `"SHR-25,CL"` باعث بروز خطا می‌گردد.

مثال ۳-۱۰: نتایج SHR را در زیر نشان دهید.

از قطعه داده:

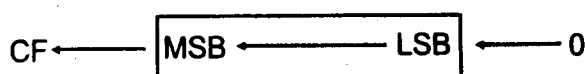
`DATA1 DW 7777H`

از قطعه کد:

```
TIMES EQU 4
MOV     CL,TIMES      ;CL=04
SHR     DATA1,CL     ;shift DATA1 CL times
```

حل:

بعد از چهار نقل مکان، کلمه موجود در مکان `DATA1` حافظه حاوی `0777` خواهد بود. چهار بیت `LSB` از طریق بیت نقل مکان یک به یک به بیرون می‌ریزد و چهار بیت `MSB` با صفر پر می‌شود.



SHL : نقل مکان به چپ نیز یک نقل

مکان منطقی است. این عمل عکس SHR

است. پس از هر نقل مکان، `LSB` با 0 پر

می‌شود و `MSB` به `CF` می‌رود. همه قوانین همانند SHR است.

مثال ۳-۱۱: تأثیر SHL را در زیر نشان دهید

```
MOV     DH,6
MOV     CL,4
SHL     DH,CL
```

حل:

	00000110	
$CF=0$	00001100	(نقل مکان به چپ یک بار)
$CF=0$	00011000	
$CF=0$	00110000	
$CF=0$	01100000	(چهار بار نقل مکان)

پس از چهار مرتبه نقل مکان، ثبات `DH` دارای `60H` بوده و $CF = 0$ است.

مثال ۳-۱۱ بطریق زیر می‌توانست کد شود.

```
MOV     DH,6
SHL     DH,1
SHL     DH,1
SHL     DH,1
SHL     DH,1
```

COMPARE برای اعداد بدون علامت

`CMP destination,source ;compare dest and src`

دستور `CMP` دو عملوند را مقایسه کرده و برطبق نتیجه آن پرچم‌ها را تغییر می‌دهد. خود عملوندها بدون تغییر باقی می‌مانند. عملوند مقصد در یک ثبات، در حافظه یا فوری است. گرچه همه پرچمهای SF ، AF ، CF ، PF و OF نتیجه مقایسه را منعکس می‌سازند، ولی فقط CF و ZF بکار می‌روند، جدول ۳-۳ ملاحظه شود.

مقایسه عملوندها	CF	ZF
destination > source	0	0
destination = source	0	1
destination < source	1	0

جدول ۳-۳. تنظیم پرچم‌ها برای دستور مقایسه

در زیر چگونگی استفاده از CMP نمایش داده شده است:

```
DATA1    DW    235FH
        ....
        MOV    AX,0CCCCCH
        CMP    AX,DATA1    ;compare CCCC with 235F
        JNC    OVER        ;jump if CF=0
        SUB    AX,AX
OVER:     INC    DATA1
```

در برنامه فوق، AX بزرگتر از محتوای مکان حافظه DATA1 (OCCCCCH > 235FH) است بنابراین CF = 0 و JNC (پرش اگر نقلی نیست) به هدف OVER خواهد رفت. بالعکس به برنامه زیر نگاه کنید:

```
TEMP     DB ?
        ...
        MOV    AL,TEMP      ;move the TEMP variable into AL
        CMP    AL,99        ;compare AL with 99
        JZ     HOT_HOT      ;if ZF=1 (TEMP = 99) jump to HOT_HOT
        INC    BX           ;otherwise (ZF=0) increment BX
        ...
HOT_HOT: HLT                ;halt the system
```

در برنامه فوق BX کوچک‌تر از CX است (7888H < 9FFFH)، و بنابراین CF = 1 شده و سبب می‌شود "JNC NEXT" به دستور بعد رفته "ADD BX, 4000H" اجرا شود. در مثال فوق توجه کنید که BX و CX همان مقادیر اولیه خود را (BX=7888H, CX=9FFFH) پس از اجرای "CMP BX, CX" دارند. توجه کنید که CX همیشه برای حالات بزرگتر یا کوچکتر تست می‌شود، ولی برای مساوی، ZF باید چک شود. برنامه بعدی تغییری با نام TEMP دارد و برای وجود 99 در آن چک می‌شود:

```
        MOV    BX,7888H
        MOV    CX,9FFFH
        CMP    BX,CX        ;compare 7888 with 9FFF
        JNC    NEXT
        ADD    BX,4000H
NEXT:     ADD    CX,250H
```

دستور مقایسه در واقع یک تفریق است با این تفاوت که در آن مقادیر عملوندها تغییر نمی‌کنند. پرچم‌ها مثل اجرای SUB تغییر می‌کنند. گرچه پرچم‌ها متأثر می‌شوند ولی فقط ZF و CF موردنظر هستند. باید تأکید کرد که در دستورات CMP، عملوندها بدون توجه به نتیجه مقایسه، هیچ تغییری نمی‌کنند و فقط پرچم‌ها عوض می‌شود. این عملکرد علیرغم این حقیقت که CMP از عملیات SUB استفاده می‌کند رخ می‌دهد. برنامه ۳-۳ دستور CMP را بکار می‌برد تا بالاترین بایت را در سری ۵ بایتی تعریف شده در قطعه داده جستجو کند. دستور [BX], CMP AL، بطریق زیرکار می‌کند، و در آن [BX] محتوای حافظه اشاره شده بوسیله ثبات BX است.

اگر [BX] < AL باشد، آنگاه CF = 1 و [BX] مبنای مقایسه جدید خواهد بود.
اگر [BX] > AL باشد، سپس CF = 0 و AL بزرگتر بوده و مبنای مقایسه است.

گرچه JC (پرش بشرط نقلي) و JNC (پرش اگر نقلي نيست) پرچم نقلي را پس از يك دستور مقايسه بكار مي‌زنند، پيشنهاد مي‌شود كه بدو دليل از JA (پرش بالاتر) و JB (پرش پايين تر) استفاده شود. يكي از دليل اين است كه DEBUG دستور JC را مثل JB و JNC را مثل JA عكس اسمبل مي‌كند و البته ممكن است موجبات اشتباه را براي برنامه‌نويسان مبتدي فراهم كند. دليل ديگر اين است كه JA و JB براي درك از JC و JNC ساده‌ترند، زيرا سريع تر مي‌توان فهميد كه يكي از اعداد بزرگ‌تر از ديگري است و يا رقم نقلي بعد از تفريق دو عدد رخ داده است.

برنامه ۳-۳ دستور CMP را براي جستجوي 5 بايت داده بكار مي‌برد تا بزرگ‌ترين را يابد. برنامه، ثبات AL را براي نگهداري بزرگ‌ترين عددي كه تا آن لحظه پيدا شده است بكار مي‌برد. به AL مقدار اوليه 0 داده شده است. از حلقه براي مقايسه هر يك از پنج بايت با مقدار داخل AL استفاده شده است. اگر AL حاوي مقدار بزرگ‌ترى باشد، حلقه بايت بعدي را چك مي‌كند. اگر AL كوچكتر از بايت چك شده باشد، محتوای AL با بايت مزبور جايگزين شده و حلقه ادامه مي‌يابد.

فرض كنيد كه يك گروه پنج نفری با نمره‌های 69، 87، 96، 45 و 75 وجود دارند. بالاترين نمره را پيدا كنيد.

```

TITLE      PROG3-3 (EXE)  CMP EXAMPLE
PAGE
STSEG      SEGMENT
            DB      64 DUP (?)
STSEG      ENDS

DTSEG      SEGMENT
GRADES     DB      69,87,96,45,75
            ORG     0008
HIGHEST    DB      ?
DTSEG      ENDS

CDSEG      SEGMENT
MAIN       PROC   FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV     AX,DTSEG
            MOV     DS,AX
            MOV     CX,5
            MOV     BX,OFFSET GRADES
AGAIN:     CMP     AL,[BX]
            JA      NEXT
            MOV     AL,[BX]
NEXT:     INC     BX
            LOOP    AGAIN
            MOV     HIGHEST,AL
            MOV     AH,4CH
            INT     21H
MAIN       ENDP
CDSEG      ENDS
            END     MAIN

```

;set up loop counter
;BX points to GRADE data
;AL holds highest grade found so far
;compare next grade to highest
;jump if AL still highest
;else AL holds new highest
;point to next grade
;continue search
;store highest grade
;go back to dos

برنامه ۳-۴ از دستور CMP براي تغيير بزرگ يا كوچك بودن كاراكثر اسكي (ASCII) استفاده مي‌كند. توجه كنيد كه حروف بزرگ و كوچك در اسكي داراي مقادير زيرند:

دودویی	مبنای شانزده	حرف	دودویی	مبنای شانزده	حرف
01100001	61	a	01000001	41	A
01100010	62	b	01000010	42	B
01100011	63	c	01000011	43	C
...	...	...	...	...	...
01111001	79	y	01011001	59	Y
01111010	7A	z	01011010	5A	Z

همانطور که دیده می شود رابطه ای بین حروف کوچک و بزرگ وجود دارد که در زیر هم می توان برای A و a ملاحظه کرد:

A 0100 0001 41H
a 0110 0001 61H

تنها بیتی که تغییر می کند d5 است. برای تبدیل از حرف کوچک به بزرگ، باید d5 ماسک (پوشش) شود. برنامه ۳-۴ ابتدا کوچک بودن حرف را تشخیص می دهد، و اگر صحت داشته باشد آن را با 1101 1111B=DFH AND می کند. در غیر این صورت رها می شود. برای تعیین کوچک بودن، آن را با 61H و 7AH مقایسه می کند تا ببیند در محدوده a تا z قرار داد یا خیر. هر چیز در بالا یا پایین محدوده باشد باید رها شود.

TITLE	PROG3-4 (EXE) LOWERCASE TO UPPERCASE CONVERSION
PAGE	60,132
STSEG	SEGMENT
	DB 64 DUP (?)
STSEG	ENDS
DTSEG	SEGMENT
DATA1	DB 'my NAME is jOe'
	ORG 0020H
DATA2	DB 14 DUP(?)
DTSEG	ENDS
CDSEG	SEGMENT
MAIN	PROC FAR
	ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
	MOV AX,DTSEG
	MOV DS,AX
	MOV SI,OFFSET DATA1 ;SI points to original data
	MOV BX,OFFSET DATA2 ;BX points to uppercase data
	MOV CX,14 ;CX is loop counter
BACK:	MOV AL,[SI] ;get next character
	CMP AL,61H ;if less than 'a'
	JB OVER ;then no need to convert
	CMP AL,7AH ;if greater than 'z'
	JA OVER ;then no need to convert
OVER:	AND AL,11011111B ;mask d5 to convert to uppercase
	MOV [BX],AL ;store uppercase character
	INC SI ;increment pointer to original
	INC BX ;increment pointer to uppercase data
	LOOP BACK ;continue looping if CX > 0
	MOV AH,4CH
	INT 21H ;go back to dos
MAIN	ENDP
CDSEG	ENDS
	END MAIN

برنامه ۳-۴

در برنامه ۳-۴، 20H بجای عمل AND با 1101 1111B، می توانست از حروف کوچک تفریق شود. این همان کاری است که IBM مطابق آنچه در زیر آمده، انجام داده است.

تبدیل حروف کوچک به بزرگ به روش IBM BIOS

	2357	;--- CONVERT ANY LOWER CASE TO UPPER CASE	
	2358		
EBFB	2359	K60:	;LOWER TO UPPER
EBFB 3C61	2360	CMP AL,'a'	;FIND OUT IF ALPHABETIC
EBFD 7206	2361	JB K61	;NOT_CAPS_STATE
EBFF 3C7A	2362	CMP AL,'z'	
EC01 7702	2363	JA K61	;NOT_CAPS_STATE
EC03 2C20	2364	SUB AL,'a'-'A'	;CONVERT TO UPPER CASE
EC05	2365	K61:	
EC05 8B1E1C00	2366	MOV BX,BUFFER_TAIL	;GET THE END POINTER
			;TO THE BUFFER

سوالات مروری

۱- عملوندهای 4FCAH و C237H را برای انجام اعمال زیربکار ببرید

(الف) AND (ب) OR (پ) XOR

۲- AND یک کلمه عملوند با FFFFH چه مقداری را برای عملوند تولید خواهد کرد؟

برای 0 کردن همه بیت‌های عملوند باید آن را با ... AND کرد.

۳- برای 1 کردن همه بیت‌های یک عملوند، باید آن را با ... OR کرد.

۴- XOR کردن یک عملوند با خودش چه تغییری در عملوند می‌دهد؟

۵- مرحله‌ای را که A0F2H سه بار به چپ نقل مکان می‌کند نشان دهید. سپس همین عمل را برای سه بار نقل مکان به راست تکرار کنید.

۶- دستورات CMP با اجرای یک عمل ... روی عملوندها کار می‌کند و برطبق آن پرچم‌ها را تنظیم می‌نماید.

۷- صحیح یا غلط، دستور CMP محتوای عملوندش را عوض می‌کند.

مثالهایی از BIOS برای دستورات منطقی

در این بخش بعضی مثال‌های واقعی را از برنامه‌های IBM PC BIOS بررسی می‌کنیم. هدف از این کار دیدن

	306	ASSUME CS:CODE,DS:NOTHING,ES:NOTHING,SS:NOTHING		
E05B	307	ORG	0E05BH	
E05B		...	...	
E05B	309	START:		
E05B FA	310	CLI		; DISABLE INTERRUPTS
E05C B4D5	311	MOV	AH,0D5H	; SET SF, CF, ZF, AND AF FLAGS ON
E05E 9E	312	SAHF		
E05F 734C	313	JNC	ERR01	; GO TO ERR ROUTINE IF CF NOT SET
E061 754A	314	JNZ	ERR01	; GO TO ERR ROUTINE IF ZF NOT SET
E063 7B48	315	JNP	ERR01	; GO TO ERR ROUTINE IF PF NOT SET
E065 7946	316	JNS	ERR01	; GO TO ERR ROUTINE IF SF NOT SET
E067 9F	317	LAHF		; LOAD FLAG IMAGE TO AH
E068 B105	318	MOV	CL,5	; LOAD CNT REG WITH SHIFT CNT
E06A D2EC	319	SHR	AH,CL	; SHIFT AF INTO CARRY BIT POS
E06C 733F	320	JNC	ERR01	; GO TO ERR ROUTINE IF AF NOT SET
E06E B040	321	MOV	AL,40H	; SET THE OF FLAG ON
E070 D0E0	322	SHL	AL,1	; SETUP FOR TESTING
E072 7139	323	JNO	ERR01	; GO TO ERR ROUTINE IF OF NOT SET
E074 32E4	324	XOR	AH,AH	; SET AH = 0
E076 9E	325	SAHF		; CLEAR SF, CF, ZF, AND PF
E077 7634	326	JBE	ERR01	; GO TO ERR ROUTINE IF CF ON
	327			; OR GO TO ERR ROUTINE IF ZF ON
E079 7832	328	JS	ERR01	; GO TO ERR ROUTINE IF SF ON
E07B 7A30	329	JP	ERR01	; GO TO ERR ROUTINE IF PF ON
E07D 9F	330	LAHF		; LOAD FLAG IMAGE TO AH
E07E B105	331	MOV	CL,5	; LOAD CNT REG WITH SHIFT CNT
E080 D2EC	332	SHR	AH,CL	; SHIFT 'AF' INTO CARRY BIT POS
E082 7229	333	JC	ERR01	; GO TO ERR ROUTINE IF ON
E084 D0E4	334	SHL	AH,1	; CHECK THAT 'OF' IS CLEAR
E086 7025	335	JO	ERR01	; GO TO ERR ROUTINE IF ON
	336			
	337	;--- READ/WRITE THE 8088 GENERAL AND SEGMENTATION REGISTERS		
	338	; WITH ALL ONES AND ZEROES.		
	339			
E088 B8FFFF	340	MOV	AX,0FFFFH	; SET UP ONE'S PATTERN IN AX
E08B F9	341	STC		
E08C	342	C8:		
E08C 8ED8	343	MOV	DS,AX	; WRITE PATTERN TO ALL REGS
E08E 8CDB	344	MOV	BX,DS	
E090 8EC3	345	MOV	ES,BX	
E092 8CC1	346	MOV	CX,ES	
E094 8ED1	347	MOV	SS,CX	
E096 8CD2	348	MOV	DX,SS	
E098 8BE2	349	MOV	SP,DX	
E09A 8BEC	350	MOV	BP,SP	
E09C 8BF5	351	MOV	SI,BP	
E09E 8BFE	352	MOV	DI,SI	
E0A0 7307	353	JNC	C9	; TST1A
E0A2 33C7	354	XOR	AX,DI	; PATTERN MAKE IT THRU ALL REGS
E0A4 7507	355	JNZ	ERR01	; NO - GO TO ERR ROUTINE
E0A6 F8	356	CLC		
E0A7 EBE3	357	JMP	C8	
E0A9	358	C9:		
E0A9 0BC7	359	OR	AX,DI	; ZERO PATTERN MAKE IT THRU?
E0AB 7401	360	JZ	C10	; YES - GO TO NEXT TEST
E0AD F4	361	ERR01: HLT		; HALT SYSTEM

دستوراتی است که تاکنون در این متن بحث شده‌اند و در کاربردهای واقعی بکار گرفته شده‌اند. وقتی کامپیوتری روشن می‌شود، CPU شروع به اجرای برنامه‌های ذخیره شده در BIOS می‌کند تا کامپیوتر را برای DOS آماده کند. اگر برای برنامه‌های BIOS مشکلی پیش آمده باشد، کامپیوتر کاری نمی‌تواند انجام دهد. اولین زیر روال BIOS تست CPU است. این تست شامل تست تک تک بیت‌های ثبات پرچم و تست همه ثبات‌های دیگر است. برنامه BIOS برای تست پرچم‌ها و ثبات‌ها در صفحه قبل و توضیحات آنها در زیر آمده است.

توضیح خط به خط:

شماره خط	توضیح
310	CLI ممانعت از وقوع هر وقفه را ضمن تست تضمین می‌کند
311	MOV AH,0D5H: flag S Z — AC — P — C D5H 1 1 0 1 0 1 0 1
312	SAHF (AH) را در بایت پایین تر ثبات پرچم ذخیره می‌کند (یکی از راه‌های انتقال داده به پرچم‌هاست. راه دیگر استفاده از پشته است. MOV AX,00D5H PUSH AX POPF
313-316	با این وجود هنوز RAM برای پشته تعریف نشده است زیرا CPU در حال تست شدن است. اگر پرچمی کار نکند CPU را به HLT می‌پراند.
317	LAHF (بار کردن AH با بایت پایین تر ثبات پرچم) برعکس SAHF است.
318	CL را برای پنج نقل مکان بار می‌کند.
319	"SHR AH,CL" با پنج بار نقل مکان AH، AF (نقل کمکی) در محل CF خواهد بود
320	اگر AF صفر باشد، خطایی وجود دارد، خطوط 317 تا 320 باین علت آمده است که دستور پرش شرطی برای AF در دسترس نیست.
321-323	پرچم OF را چک می‌کند. این مطلب در فصل ۶ به همراه بحث اعداد علامت‌دار مطرح شده است.
324-335	همان پرچم‌ها را برای صفر تست می‌کند. متذکر می‌شود که JNZ مثل JBE است.
340	AX را با FFFFH بار می‌کند
341	STC (1 کردن نقلی) موجب می‌شود تا CF=1 گردد
343-352	مقدار AX (FFFFH) را به داخل هر ثبات می‌برد و اگر همه سالم باشند کار را با DI=FFFFH پایان می‌دهد.
353	چون CF = 1 است (با توجه به STC فوق) پس اجرا به پایین منتقل می‌گردد.
354	OR انحصاری AX و DI با یکدیگر، که دارای مقادیر یکسان FFFFH هستند موجب AX = 0000 شده در اینصورت اگر ثبات‌ها سالم باشند نتیجه 1 ZF = خواهد بود. اگر ZF = 0 حاصل شود یکی از ثبات‌ها خراب و لذا CPU خراب است.
355	اگر ZF = 0 باشد خطایی وجود دارد.
356	CLC پرچم نقلی را پاک می‌کند. برعکس STC است.
357	به C8 پریده و پردازش مشابهی را این بار با 0000 انجام می‌دهد. محتوای AX به هر ثبات منتقل می‌شود تا به DI = 0000 برسد و در 353، JNC C9 پرش خواهد کرد زیرا قبل از ورود به حلقه بوسیله CLC، CF = 0 شده است.
359	در C9، روی AX و DI عمل OR انجام می‌شود. اگر حاصل 0000 شود محتوای AX در تمام ثبات‌ها کپی می‌شود و DI = 0000 است؛ بنابراین OR موجب ZF = 1 می‌گردد.
360	اگر ZF = 1 باشد، CPU سالم است و سیستم می‌تواند تست بعدی را انجام دهد در غیر اینصورت CPU خراب بوده و سیستم متوقف می‌گردد. در غیر اینصورت ZF=0 بوده و CPU خراب است و سیستم باید متوقف گردد.

بخش ۳-۴: دستورات و عملوندهای BCD و اسکی

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

شکل ۳-۱ کد BCD

در پردازنده‌های 80x86 دستورات بسیاری وجود دارد که اعداد اسکی و BCD را بکار می‌برند. این بخش این دستورات را همراه با مثال‌هایی پوشش می‌دهد.

سیستم اعداد BCD

BCD بمعنی دهدهی کد شده به دودویی است. اعداد BCD باین دلیل لازمند که انسان برای اعداد از 0 تا 9 از آن‌ها استفاده می‌کند (شکل ۳-۱). در کامپیوتر معمولاً با دو نوع اعداد BCD مواجه می‌شویم.
(الف) BCD فشرده نشده (ب) BCD فشرده شده

BCD فشرده نشده

در BCD فشرده نشده، چهار بیت پایین عدد BCD را نشان می‌دهد و بقیه صفر است. مثلاً "0000 1001" و "0000 0101" مقادیر BCD برای 9 و 5 هستند. در حالت BCD فشرده نشده، برای ذخیره BCD فشرده نشده، یک بایت از مکان حافظه یا یک ثبات 8 بیتی لازم است.

BCD فشرده شده

در BCD فشرده شده، یک بایت دارای دو عدد BCD است، کد کلی در 4 بیت پایین تر و دیگری در 4 بیت بالاتر است. مثلاً 0101 10001، فرم BCD فشرده شده برای 59 است. عملوندهای BCD فشرده شده تنها 1 بایت حافظه را اشغال می‌کنند. به این دلیل BCD فشرده شده بکار می‌رود و در آن کارایی ذخیره‌سازی داده دو برابر است.

اعداد اسکی

در صفحه کلیدهای اسکی، وقتی مثلاً کلید "0" فعال شود "0110000" (30H) برای کامپیوتر فراهم می‌شود. بطریقی مشابه، برای کلید "1"، 31H تهیه می‌گردد. به همین ترتیب. این مقادیر برای کلیدهای 0 تا 9 در زیر لیست شده است.

اسکی (hex) کلید	دودویی	BCD (فشرده نشده)
0 30	011 0000	0000 0000
1 31	011 0001	0000 0001
2 32	011 0010	0000 0010
3 33	011 0011	0000 0011
4 34	011 0100	0000 0100
5 35	011 0101	0000 0101
6 36	011 0110	0000 0110
7 37	011 0111	0000 0111
8 38	011 1000	0000 1000
9 39	011 1001	0000 1001

باید توجه داشت که هر چند اسکی در ایالات متحده و بسیاری از کشورهای دیگر استاندارد است، ولی اعداد BCD کاربرد جهانی دارد. اکنون که صفحه کلید، چاپگرها و مانیتورها همگی به اسکی هستند، چگونه اسکی به BCD و بالعکس تبدیل می‌شود؟ این‌ها مطالبی هستند که در زیر بحث شده‌اند.

تبدیل اسکی به BCD

برای پردازش داده در BCD، ابتدا داده اسکی تولید شده بوسیله صفحه کلید باید به BCD تبدیل شود. اینکه بفرم فشرده و یا غیر فشرده در آید به دستور بستگی دارد. دستوراتی وجود دارند که برای آنها داده BCD فشرده شده لازم است و نیز دستورات دیگری موجودند که باید داده آنها BCD فشرده نشده باشد. هر یک از آنها جداگانه بحث می‌شوند.

تبدیل اسکی به BCD فشرده نشده

برای تبدیل داده اسکی به BCD، برنامه نویس باید از شر برجسب "011" در چهار بیت بالا رتبه اسکی خلاص شود. برای انجام آن، روی هر عدد اسکی با 0000 1111 (0FH) عمل AND انجام می‌شود. این عمل در مثال بعدی دیده خواهد شد. این مثال با استفاده از سه روش آدرس‌دهی به سه طریق مختلف نوشته شده است. سه برنامه زیر سه روش را برای تبدیل 10 رقم اسکی به BCD نشان می‌دهد. همه آنها قطعه داده یکسانی را بکار برده‌اند:

```

ASC      DB      '9562481273'
          ORG     0010H
UNPACK   DB      10 DUP(?)

```

در برنامه ۳-۵ (الف) دقت کنید که هر چند داده بصورت DB، یعنی رهنمون بآی، تعریف شده است، ولی بصورت سائزهای کلمه‌ای برداشته می‌شود. با این وجود، این یک روش عملی است و در آن رهنمون PTR طبق برنامه ۳-۵ (ب) بکار رفته است تا برای برنامه نویس‌ها خواناتر باشد.

```

AGAIN:    MOV     CX,5                ;BX points to ASCII data
          MOV     BX,OFFSET ASC       ;DI points to unpacked BCD data
          MOV     DI,OFFSET UNPACK
          MOV     AX,[BX]             ;move next 2 ASCII numbers to AX
          AND     AX,0F0FH            ;remove ASCII 3s
          MOV     [DI],AX             ;store unpacked BCD
          ADD     DI,2                 ;point to next unpacked BCD data
          ADD     BX,2                 ;point to next ASCII data
          LOOP    AGAIN

```

برنامه ۳-۵ (الف)

```

AGAIN:    MOV     CX,5                ;CX is loop counter
          MOV     BX,OFFSET ASC       ;BX points to ASCII data
          MOV     DI,OFFSET UNPACK   ;DI points to unpacked BCD data
          MOV     AX,WORD PTR [BX]    ;move next 2 ASCII numbers to AX
          AND     AX,0F0FH            ;remove ASCII 3s
          MOV     WORD PTR [DI],AX    ;store unpacked BCD
          ADD     DI,2                 ;point to next unpacked BCD data
          ADD     BX,2                 ;point to next ASCII data
          LOOP    AGAIN

```

برنامه ۳-۵ (ب)

در هر دو حل فوق، ثبات‌های BX و DI بعنوان اشاره‌گرها برای هر آرایه‌ای از داده بکار رفتند. هر آرایه در واقع مجموعه‌ای از داده در مکان‌های متوالی است. اکنون سؤال این است که اگر چهار، پنج و یا شش آرایه وجود داشته باشند چه می‌شود؟ چگونه می‌توان به همه آنها با فقط سه ثبات BX، DI و SI بعنوان اشاره‌گر دسترسی پیدا کرد؟ برنامه ۳-۵ (پ) چگونگی انجام این عمل را نشان می‌دهد و در آن با یک ثبات بعنوان اشاره‌گر به دو آرایه دسترسی می‌شود. با این وجود برای انجام آن، آرایه‌ها باید سائز یکسانی داشته و بصورت مشابهی تعریف شوند.

```

AGAIN:    MOV     CX,10                ;load the counter
          SUB     BX,BX                ;clear BX
          MOV     AL,ASC[BX]           ;move to AL content of mem [BX+ASC]
          AND     AL,0FH               ;mask the upper nibble
          MOV     UNPACK[BX],AL        ;move to mem [BX+UNPACK] the AL
          INC     BX                   ;make the pointer to point at next byte
          LOOP    AGAIN                ;loop until it is finished

```

برنامه ۳-۵ (پ)

برنامه ۳-۵ (پ) روش آدرس‌دهی پایه را بکار می‌برد زیرا $BX + ASC$ بعنوان اشاره‌گر بکار رفته است. ASC یک جابجایی اضافه شده به BX است. برای این منظور هر یک از ثبات‌های DI یا SI می‌تواند بکار گرفته شود. برای عملوندهای ساینز کلمه‌ای، "WORD PTR" بکار می‌رود زیرا داده بصورت DB تعریف شده است. این مطلب در زیر نشان داده شده است.

```
MOV    AX,WORD PTR ASC[BX]
AND    AX,0F0FH
MOV    WORD PTR UNPACKED[BX],AX
```

تبدیل اسکی به BCD فشرده

برای تبدیل اسکی به BCD فشرده، ابتدا آن را به BCD غیرفشرده تبدیل می‌نمایند و سپس از ترکیب آنها BCD فشرده بدست می‌آید. مثلاً، برای ۵ و ۹، صفحه کلید به ترتیب ۳۹ و ۳۵ را تولید می‌کند. هدف تولید ۹۵H برای "1001 0101" است که BCD فشرده می‌باشد. این پردازش در زیر تشریح شده است.

کلید	ASCII	BCD فشرده نشده	BCD فشرده شده
4	34	00000100	
7	37	00000111	01000111 or 47H

```
MOV    AX,WORD PTR VAL_ASC    ;AH=37,AL=34
AND    AX,0F0FH                ;mask 3 to get unpacked BCD
XCHG   AH,AL                   ;swap AH and AL
MOV    CL,4                     ;CL=04 to shift 4 times
SHL    AH,CL                   ;shift left AH to get AH=40H
OR     AL,AH                    ;OR them to get packed BCD
MOV    VAL_BCD,AL              ;save the result
```

پس از این تبدیل، اعداد BCD فشرده پردازش شده و نتیجه به صورت قالب BCD فشرده خواهد بود. همانطور که بعداً در این فصل خواهیم دید، دستورات خاصی همچون DAA و DAS وجود دارند که برای آنها باید داده بصورت BCD تحویل و حاصل نیز بصورت BCD دریافت می‌شود. برای نمایش نتیجه روی مانیتور یا چاپ بوسیله چاپگر، باید قالب داده اسکی باشد تبدیل از BCD فشرده به اسکی در زیر بحث شده است.

تبدیل BCD فشرده به اسکی

برای تبدیل BCD فشرده به اسکی، ابتدا باید آن را بصورت غیرفشرده در آورد و سپس به BCD غیرفشرده برچسب 0110000 (30H) را ملحق نمود. در زیر روش تبدیل BCD فشرده نشده به اسکی دیده می‌شود.

```
ORG    0010H
VAL_ASC DB '47'
VAL_BCD DB ?
```

BCD فشرده شده	BCD فشرده نشده	اسکی
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 & 0000 1001	011 0010 & 011 1001

```
VAL1_BCD DB 29H
VAL3_ASC DW ?

...
MOV    AL,VAL1_BCD
MOV    AH,AL                    ;copy AL to AH. now AH=29,AL=29H
AND    AX,0F00FH                ;mask 9 from AH and 2 from AL
MOV    CL,4                     ;CL=04 for shift
SHR    AH,CL                    ;shift right AH to get unpacked BCD
OR     AX,3030H                 ;combine with 30 to get ASCII
XCHG   AH,AL                    ;swap for ASCII storage convention
MOV    VAL3_ASC,AX              ;store the ASCII
```

جمع و تفریق BCD

پس از آموختن تبدیل اسکی به BCD، کاربرد اعداد BCD در مرحله بعدی مطالعه می شود. دو دستور وجود دارند که خصوصاً با اعداد BCD بکار برده می شوند: DAA و DAS. هر یک جداگانه بحث شده است.

BCD فشرده شده	BCD فشرده نشده	اسکی
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 & 0000 1001	011 0010 & 011 1001

VAL1_BCD DB 29H
VAL3-ASC DW ?

```

...
MOV     AL,VAL1_BCD
MOV     AH,AL           ;copy AL to AH. now AH=29,AL=29H
AND     AX,0F00FH       ;mask 9 from AH and 2 from AL
MOV     CL,4            ;CL=04 for shift
SHR     AH,CL           ;shift right AH to get unpacked BCD
OR      AX,3030H        ;combine with 30 to get ASCII
XCHG    AH,AL           ;swap for ASCII storage convention
MOV     VAL3-ASC,AX     ;store the ASCII
    
```

جمع و تصحیح BCD

در جمع اعداد BCD مشکلی وجود دارد که باید اصلاح شود. مشکل این است که پس از جمع اعداد BCD فشرده، نتیجه بهیچوجه BCD نیست. به این مثال توجه کنید:

```

MOV AL,17H
ADD AL,28H
    
```

از جمع آن‌ها 1111B (0011 3FH) را خواهیم داشت که BCD نیست. یک عدد BCD فقط می تواند بین 0000 تا 1001 (یا 0 تا 9) باشد. به بیان دیگر، جمع دو عدد BCD باید نتیجه BCD هم تولید کند. نتیجه فوق می باید 45=17+28 (0101 0100) می بود. برای اصلاح این اشکال، برنامه نویس باید 6 (0110) را به رقم پایین اضافه کند: 45H=3F+06. مشکل مشابهی می تواند در رقم بالا رخ دهد (مثلاً در 52H+87H=D9H) مجدداً برای حل این مشکل، باید آن را با 6 جمع کرد (D9H+60H=139H) تا مطمئن شویم که نتیجه BCD (139=52+87) است. این مطلب بقدری اهمیت دارد که همه ریزپردازنده های CISC تک تراشه ای مانند 80x86 اینتل و 680x0 موتورولا دستوراتی برای برخورد با آن دارند. پردازنده های RISC این دستور را حذف کرده اند.

DAA

دستور DAA (تنظیم دهمی برای جمع) در ریزپردازنده های 80x86 دقیقاً برای اصلاح جمع BCD تهیه شده است. هرگاه لازم باشد DAA، 6 واحد را به نیل پایین یا بالا اضافه می کند، در غیر این صورت نتیجه را تغییر نخواهد داد. مثال زیر این نکات را روشن می کند:

```

DATA1 DB 47H
DATA2 DB 25H
DATA3 DB ?

MOV     AL,DATA1        ;AL holds first BCD operand
MOV     BL,DATA2        ;BL holds second BCD operand
ADD     AL,BL           ;BCD addition
DAA                     ;adjust for BCD addition
MOV     DATA3,AL       ;store result in correct BCD form
    
```

پس از اجرای برنامه، DATA3 حاوی $72H (25+47=72)$ خواهد بود. توجه کنید که DAA فقط روی AL عمل می‌کند. به بیان دیگر در حالی که مبدأ می‌تواند عملوندی با هر روش آدرس‌دهی باشد، مقصد همواره AL است تا DAA بتواند کار کند. باید تأکید کرد که DAA باید بعد از جمع BCD قرار گیرد و عملوندهای BCD هم نباید ارقام بزرگتر از 9 باشد، یعنی ارقام A-F مجاز نیستند. همچنین توجه کنید که DAA پس از یک دستور ADD کار می‌کند و پس از دستور INC هیچ عملی انجام نمی‌دهد.

خلاصه عمل DAA

۱- اگر پس از یک دستور ADD یا ADC، نیل پائینی بزرگتر از 9 باشد یا اگر $AF=1$ باشد به این چهار بیت پایین تر، 0110 اضافه می‌شود.

۲- اگر نیل بالایی بزرگتر از 9 باشد یا اگر $CF=1$ بشود، 0110 را با نیل بالایی جمع می‌کند. در واقع کاربرد دیگری برای AF (بزرگ کمکی) بجز در جمع BCD و اصلاح آن وجود ندارد. مثلاً جمع $29H$ و $18H$ برابر $47H$ خواهد شد که از نظر عملیات BCD غلط است.

Hex	BCD
29	0010 1001
+ 18	+ 0001 1000
41	0100 0001 $AF=1$
+ 6	+ 0110
47	0100 0111

چون $AF=1$ است، DAA مقدار 6 را به نیل پائینی می‌افزاید.
نتیجه نهایی BCD است.

برنامه ۳-۶ کاربرد DAA را پس از جمع چند بایتی اعداد BCD فشرده نشان می‌دهد.

تفریق BCD و اصلاح آن

مسئله مرتبط با جمع اعداد BCD فشرده، در تفریق هم وجود دارد. مجدداً دستوری خاص (DAS) برای حل این مشکل طراحی شده است. بنابراین، وقتی که عملوندهای BCD فشرده را تفریق می‌کنیم دستور DAS پس از SUB یا SBB قرار می‌گیرد. AL باید بعنوان ثبات مقصد بکار رود تا DOS کار کند.

خلاصه عمل DAS

۱- اگر پس از هر دستور SUB و SBB نیل پایین تر بزرگتر از 9 باشد، و یا اگر $AF=1$ باشد 0110 را از چهار بیت پایین کم می‌کند.

۲- اگر نیل بالایی بزرگتر از 9 باشد و یا اگر $CF=1$ باشد، 0110 را از نیل بالایی کسر می‌نماید. بدلیل کاربرد وسیع اعداد BCD، رهنمون داده خاصی بنام DT تعریف شده است، DT می‌تواند برای ارائه اعداد BCD از 0 تا $10^{20}-1$ (تقریباً بیست عدد 9 است) بکار رود. فرض کنید که عملوندهای زیر در آمد، خرج و بالانس را نشان دهند که در آن بالانس برابر با درآمد منهای خرج است.

BUDGET DT 87965141012
EXPENSES DT 31610640392
BALANCE DT ?

;balance = budget - expenses

MOV	CX,10	;counter=10
MOV	BX,00	;pointer=0
CLC		;clear carry for the 1st iteration
BACK: MOV	AL,BYTE PTR BUDGET[BX]	;get a byte of the BUDGET
SBB	AL,BYTE PTR EXPENSES[BX]	;subtract a byte from it
DAS		;correct the result for BCD
MOV	BYTE PTR BALANCE[BX],AL	;save it in BALANCE
INC	BX	;increment for the next byte
LOOP	BACK	;continue until CX=0

دو مجموعه داده اسکی از صفحه کلید آمده‌اند. برنامه‌ای برای موارد زیر نوشته و اجرا کنید.

۱- اسکی را به BCD فشرده تبدیل نمایید.

۲- BCD فشرده چند بایتی را جمع و آن را ذخیره کنید.

۳- BCD فشرده به اسکی تبدیل شود.

TITLE PROG3-6 (EXE) ASCII TO BCD CONVERSION AND ADDITION

PAGE 60,132

STSEG SEGMENT

DB 64 DUP(?)

STSEG ENDS

DTSEG SEGMENT

DATA1_ASC DB '0649147816'

ORG 0010H

DATA2_ASC DB '0072687188'

ORG 0020H

DATA3_BCD DB 5 DUP (?)

ORG 0028H

DATA4_BCD DB 5 DUP (?)

ORG 0030H

DATA5_ADD DB 5 DUP (?)

ORG 0040H

DATA6_ASC DB 10 DUP (?)

DTSEG ENDS

CDSEG SEGMENT

MAIN PROC FAR

ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG

MOV AX,DTSEG

MOV DS,AX

MOV BX,OFFSET DATA1_ASC ;BX points to first ASCII data

MOV DI,OFFSET DATA3_BCD ;DI points to first BCD data

MOV CX,10 ;CX holds number bytes to convert

CALL CONV_BCD ;convert ASCII to BCD

MOV BX,OFFSET DATA2_ASC ;BX points to second ASCII data

MOV DI,OFFSET DATA4_BCD ;DI points to second BCD data

MOV CX,10 ;CX holds number bytes to convert

CALL CONV_BCD ;convert ASCII to BCD

CALL BCD_ADD ;add the BCD operands

MOV SI,OFFSET DATA5_ADD ;SI points to BCD result

MOV DI,OFFSET DATA6_ASC ;DI points to ASCII result

MOV CX,05 ;CX holds count for convert

CALL CONV_ASC ;convert result to ASCII

MOV AH,4CH

INT 21H ;go back to DOS

MAIN ENDP

THIS SUBROUTINE CONVERTS ASCII TO PACKED BCD

CONV_BCD PROC

AGAIN: MOV AX,[BX] ;BX=pointer for ASCII data

XCHG AH,AL

AND AX,0F0FH ;mask ASCII 3s

PUSH CX ;save the counter

MOV CL,4 ;shift AH left 4 bits

SHL AH,CL ; to get ready for packing

OR AL,AH ;combine to make packed BCD

MOV [DI],AL ;DI=pointer for BCD data

ADD BX,2 ;point to next 2 ASCII bytes

INC DI ;point to next BCD data

POP CX ;restore loop counter

LOOP AGAIN

RET

CONV_BCD ENDP

THIS SUBROUTINE ADDS TWO MULTIBYTE PACKED BCD OPERANDS

```
BCD_ADD PROC
    MOV     BX,OFFSET DATA3_BCD ;BX=pointer for operand 1
    MOV     DI,OFFSET DATA4_BCD ;DI=pointer for operand 2
    MOV     SI,OFFSET DATA5_ADD ;SI=pointer for sum
    MOV     CX,05
    CLC
BACK:  MOV     AL,[BX]+4           ;get next byte of operand 1
        ADC     AL,[DI]+4         ;add next byte of operand 2
        DAA                     ;correct for BCD addition
        MOV     [SI]+4,AL        ;save sum
        DEC     BX               ;point to next byte of operand 1
        DEC     DI               ;point to next byte of operand 2
        DEC     SI               ;point to next byte of sum
        LOOP    BACK
    RET
BCD_ADD ENDP
```

THIS SUBROUTINE CONVERTS FROM PACKED BCD TO ASCII

```
CONV_ASC PROC
AGAIN2: MOV     AL,[SI]           ;SI=pointer for BCD data
        MOV     AH,AL            ;duplicate to unpack
        AND     AX,0F00FH        ;unpack
        PUSH    CX               ;save counter
        MOV     CL,04            ;shift right 4 bits to unpack
        SHR     AH,CL            ;the upper nibble
        OR      AX,3030H         ;make it ASCII
        XCHG    AH,AL            ;swap for ASCII storage convention
        MOV     [DI],AX          ;store ASCII data
        INC     SI               ;point to next BCD data
        ADD     DI,2              ;point to next ASCII data
        POP     CX               ;restore loop counter
        LOOP    AGAIN2
    RET
CONV_ASC ENDP
CDSEG  ENDS
END     MAIN
```

ادامه برنامه ۶-۳

در بخش کد فوق توجه کنید که (۱) هیچ شاخص مبنای شانزده H برای اعداد BCD بهنگام استفاده از رهنمون DT لازم نیست و (۲)، استفاده از روش آدرس دهی نسبی پایه (جابجایی + BX) دستیابی به هر سه آرایه را با یک ثبات BX ممکن می سازد.

در مسئله ۷-۳، رهنمون DB برای تعریف مقادیر اسکی بکار رفته است. این موجب می شود تا LSD (کم ارزشترین رقم) در بالاترین مکان آرایه قرار گیرد. در VALUE1، 37، اسکی برای 7 در مکان حافظه 0009 است، بنابراین BX باید به آن اشاره کند و سپس کاهش یابد. برنامه ۷-۳ با تعریف قطعه ساده تکرار و بازنویسی شده است.

جمع و تفریق اسکی

اعداد اسکی می توانند بعنوان عملوند در جمع و تفریق به همان ترتیبی که هستند و بدون حذف برچسب 011 استفاده شوند. دستورات AAA و AAS خصوصاً برای این کار طراحی شده اند. به حالات زیر توجه کنید. در برنامه فوق:

```
MOV     AL,'5'           ;AL=35
ADD     AL,'2'           ;add to AL 32 the ASCII for 2
AAA                     ;changes 67H to 07H
OR      AL,30            ;OR AL with 30H to get ASCII
```

0011 0101
0011 0010
0110 0111

برنامه ای بنویسید که دو عدد اسکی چند رقمی را با هم جمع کند . نتیجه باید به اسکی باشد .

```

TITLE      PROG3-7 (EXE) ADDING ASCII NUMBERS
PAGE      60,132
STSEG     SEGMENT
          DB      64 DUP (?)
STSEG     ENDS

;-----
DTSEG     SEGMENT
VALUE1    DB      '0659478127'
          ORG      0010H
VALUE2    DB      '0779563678'
          ORG      0020H
RESULT1   DB      10 DUP (?)
          ORG      0030H
RESULT2   DB      10 DUP (?)
DTSEG     ENDS

;-----
CDSEG     SEGMENT
MAIN      PROC    FAR
          ASSUME  CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV     AX,DTSEG
          MOV     DS,AX
          CALL    ASC_ADD           ;call ASCII addition subroutine
          CALL    CONVERT          ;call convert to ascii subroutine
          MOV     AH,4CH
          INT     21H               ;go back to DOS
MAIN      ENDP

;THIS SUBROUTINE ADDS THE ASCII NUMBERS AND MAKES THE RESULT UNPACKED.
ASC_ADD PROC
          CLC                       ;clear the carry
          MOV     CX,10             ;set up loop counter
          MOV     BX,9              ;point to LSD
BACK:     MOV     AL,VALUE1[BX]     ;move next byte of operand 1
          ADC     AL,VALUE2[BX]     ;add next byte of operand 2
          AAA                      ;adjust to make it unpacked BCD
          MOV     RESULT1[BX],AL    ;store BCD sum
          DEC     BX               ;point to next byte
          LOOP    BACK
          RET
ASC_ADD ENDP

;THIS SUBROUTINE CONVERTS UNPACKED BCD TO ASCII
CONVERT PROC
          MOV     BX,OFFSET RESULT1 ;BX points to unpacked BCD data
          MOV     SI,OFFSET RESULT2 ;SI points to ASCII data
          MOV     CX,05             ;CX is loop counter
BACK2:    MOV     AX,WORD PTR [BX]  ;get next 2 ASCII bytes
          OR      AX,3030H          ;insert ASCII 3s
          MOV     WORD PTR [SI],AX  ;store ASCII
          ADD     BX,2              ;increment BCD pointer
          ADD     SI,2              ;increment ASCII pointer
          LOOP    BACK2
          RET
CONVERT ENDP
CDSEG     ENDS
          END      MAIN

```

برنامه ۳-۷

اگر نتیجه بزرگتر از 9 باشد، AAA آن را اصلاح کرده و بیت اضافی را به نقلی منتقل کرده و 1 را به ثبات AH اضافه می‌کند.

```

SUB      AH,AH      ;AH=00
MOV      AL,'7'     ;AL=37H
MOV      BL,'5'     ;BL=35H
ADD      AL,BL      ;37H+35H=6CH therefore AL=6C.
AAA      ;changes 6CH to 02 in AL and AH=CF=1
OR       AX,3030H   ;AX=3132 which is the ASCII for 12H.

```

```

TITLE    PROG3-7 (EXE) ADDING ASCII NUMBERS
;rewritten with simplified segment definition
PAGE     60,132
.MODEL   SMALL
.STACK   64

;-----
.DATA
VALUE1   DB    '0659478127'
          ORG    0010H
VALUE2   DB    '0779563678'
          ORG    0020H
RESULT1  DB    10 DUP (?)
          ORG    0030H
RESULT2  DB    10 DUP (?)
;-----
.CODE
MAIN:    MOV     AX,@DATA
          MOV     DS,AX
          CALL    ASC_ADD      ;call ASCII addition subroutine
          CALL    CONVERT      ;call convert to ascii subroutine
          MOV     AH,4CH
          INT     21H           ;go back to DOS
;-----
;THIS SUBROUTINE ADDS THE ASCII NUMBERS AND MAKES THE RESULT UNPACKED.
ASC_ADD  PROC
          CLC                    ;clear the carry
          MOV     CX,10          ;set up loop counter
          MOV     BX,9           ;point to LSD
BACK:    MOV     AL,VALUE1[BX]   ;move next byte of operand 1
          ADC     AL,VALUE2[BX]  ;add next byte of operand 2
          AAA                    ;adjust to make it ASCII
          MOV     RESULT1[BX],AL ;store ASCII sum
          DEC     BX             ;point to next byte
          LOOP    BACK
          RET
ASC_ADD  ENDP
;-----
;THIS SUBROUTINE CONVERTS UNPACKED BCD TO ASCII
CONVERT  PROC
          MOV     BX,OFFSET RESULT1 ;BX points to BCD data
          MOV     SI,OFFSET RESULT2 ;SI points to ASCII data
          MOV     CX,05           ;CX is loop counter
BACK2:   MOV     AX,WORD PTR [BX]  ;get next 2 ASCII bytes
          OR      AX,3030H         ;insert ASCII 3s
          MOV     WORD PTR [SI],AX ;store ASCII
          ADD     BX,2             ;increment BCD pointer
          ADD     SI,2             ;increment ASCII pointer
          LOOP    BACK2
          RET
CONVERT  ENDP
END      MAIN

```

برنامه ۷-۳. بازنویسی شده با تعریف قطعه ساده شده

در اینجا دو واقعیت را باید تذکر داد. اول اینکه AAA و AAS فقط روی ثبات AL عمل می‌کنند و دوم اینکه داده جمع شده می‌تواند بجای اسکی، BCD فشرده نشده باشد که در این حال نیز AAA و AAS بخوبی کار می‌کنند. در زیر چگونگی عملکرد AAS روی تفریق BCD غیر فشرده برای اصلاح نتیجه به BCD فشرده نشده دیده می‌شود:

```

MOV     AX,105H      ;AX=0105H unpacked BCD for 15
MOV     CL,06        ;CL=06H
SUB     AL,CL         ;5 - 6 = -1 (FFH)
AAS                      ;FFH in AL is adjusted to 09, and
                        ;AH is decremented, leaving AX = 0009

```

ضرب و تقسیم BCD غیر فشرده

برای عملیات ضرب و تقسیم اعداد BCD غیر فشرده دو دستور خاص طراحی شده است. این دستورات نتیجه را به BCD غیر فشرده باز می گردانند.

AAM

در کتابچه های راهنمای اینتل این نماد رمزی بمعنی "تنظیم ضرب اسکی" است، ولی در واقع اصلاح ضرب فشرده نشده می باشد. اگر دو عدد BCD غیر فشرده ضرب شوند، نتیجه با AAM به BCD باز می گردد.

MOV	AL,'7'	;AL=37H
AND	AL,0FH	;AL=07 unpacked BCD
MOV	DL,'6'	;DL=36H
AND	DL,0FH	;DL=06 unpacked BCD
MUL	DL	;AX=ALxDL. =07x06=002AH=42
AAM		;AX=0402 (7x6=42 unpacked BCD)
OR	AX,3030H	;AX=3432 result in ASCII

ضرب فوق بایت به بایت و نتیجه نیز در مبنای شانزده است. استفاده از AAM تبدیل آن به BCD غیر فشرده است که با افزایش برچسب 30H آن را به اسکی تبدیل نماید.

AAD

دوباره اینتل AAD را بمعنی "تنظیم تقسیم اسکی" تعریف می کند ولی در واقع داده باید به فرم BCD غیر فشرده در آید تا دستورالعمل بتواند کار کند. قبل از تقسیم BCD غیر فشرده به BCD غیر فشرده دیگر، از ADD برای تبدیل آن به HEX استفاده می شود. با این عمل هر دو خارج قسمت و باقیمانده بصورت BCD غیر فشرده اند.

MOV	AX,3539H	;AX=3539. ASCII for 59
AND	AX,0F0FH	;AH=05,AL=09 unpacked BCD data
AAD		;AX=003BH hex equivalent of 59
MOV	BH,08H	;divide by 08
DIV	BH	;3B / 08 gives AL=07, AH=03
OR	AX,3030H	;AL=37H (quotient) AH=33H (rem)

همانطور که در مثال فوق دیده می شود، تقسیم 59 بر 8، خارج قسمت 7 و باقیمانده 3 را تولید می نماید. با کمک AAD، نتیجه به BCD غیر فشرده تبدیل می شود و بنابراین می توان به آن برچسب 30H را تخصیص داد تا نتیجه اسکی حاصل گردد. باید متذکر شد که AAM و نیز AAD فقط روی AX عمل می نمایند.

سؤالات مروری

۱- برای اعداد دهدهی زیر، فرم فشرده BCD و غیر فشرده BCD را نشان دهید.
(الف) 15 (ب) 99

۲- نمادهای رمزی زیر را با عملکردشان مطابقت دهید.

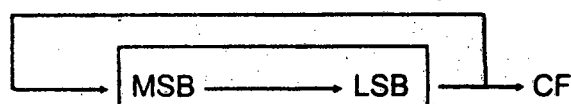
DAA	_____	(الف) جمع اسکی
DAS	_____	(ب) تفریق اسکی
AAS	_____	(پ) تفریق BCD
AAA	_____	(ت) جمع BCD

بخش ۵-۳: دستورات چرخش

در بسیاری از کاربردها نیاز به چرخش بیتی یک عملوند وجود دارد. دستور چرخش ROR، ROL و RCR و RCL خصوصاً به همین منظور طراحی شده‌اند. این دستورات می‌توانند یک عملوند را به راست یا به چپ بچرخانند. در این بخش ما دستورات چرخش که کاربردهای خاصی دارند را مورد تجزیه و تحلیل قرار می‌دهیم. در دستورات چرخش، عملوند می‌تواند در یک ثابت یا حافظه باشد. اگر تعداد چرخش عملوند بیش از یکبار باشد، از CL استفاده می‌شود. این مطلب مشابه دستورات نقل مکان (شیفت) است. دو نوع دستور چرخش وجود دارد. یکی چرخش ساده عملوند و دیگری چرخش از طریق رقم نقلی است. هر یک در زیر توضیح داده شده است.

چرخش بیت‌های یک عملوند به راست و به چپ

چرخش به راست ROR



در چرخش به راست، بانقل مکان بیت‌ها از چپ به راست، از انتهای سمت راست (LSB) خارج می‌شوند و از سمت چپ وارد می‌گردند.

بعلاوه، با خروج هر بیت از LSB، یک کپی به پرچم نقلی انتقال می‌یابد. به بیان دیگر در ROR، LSB به MSB انتقال یافته و در CF هم، طبق نمودار، کپی می‌شود. اگر عملوند یکبار بچرخد، کد یکبار بکار می‌رود ولی اگر بیش از یکبار باشد، ثابت CL برای نگهداری دفعات چرخش بکار خواهد رفت.

```
MOV AL,36H      ;AL=0011 0110
ROR AL,1         ;AL=0001 1011 CF=0
ROR AL,1         ;AL=1000 1101 CF=1
ROR AL,1         ;AL=1100 0110 CF=1
```

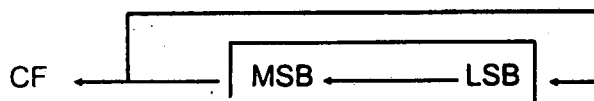
یا

```
MOV AL,36H      ;AL=0011 0110
MOV CL,3        ;CL=3 number of times to rotate
ROR AL,CL       ;AL=1100 0110 CF=1
```

؛ عملوند می‌تواند کلمه باشد:

```
MOV BX,0C7E5H   ;BX=1100 0111 1110 0101
MOV CL,6        ;CL=6 number of times to rotate
ROR BX,CL       ;BX=1001 0111 0001 1111 CF=1
```

چرخش به چپ ROL



در چرخش به چپ، با چرخش راست به چپ، بیت انتهای چپ (MSB) خارج و از سمت راست (LSB) وارد می‌شود.

بعلاوه، هر بیت که MSB را ترک کند به پرچم نقلی CF هم وارد می‌شود. اگر عملوند یکبار بچرخد، اجرای یک دستور کافی است. در غیر اینصورت تعداد دفعات در CL نگهداری می‌شود.

```
MOV BH,72H      ;BH=0111 0010
ROL BH,1        ;BH=1110 0100 CF=0
ROL BH,1        ;BH=1100 1001 CF=1
ROL BH,1        ;BH=1001 0011 CF=1
ROL BH,1        ;BH=0010 0111 CF=1
```

یا:
 MOV BH,72H ;BH=0111 0010
 MOV CL,4 ;CL=4 number of times to rotate
 ROL BH,CL ;BH=0010 0111 CF=1

عملوند می تواند یک کلمه باشد

MOV DX,672AH ;DX=0110 0111 0010 1010
 MOV CL,3 ;CL=3 number of times to rotate
 ROL DX,CL ;DX=0011 1001 0101 0011 CF=1

برنامه ای بنویسید که تعداد 1 ها را در یک بایت پیدا کند .

From the data segment:

DATA1 DB 97H
 COUNT DB ?

From the code segment:

SUB BL,BL ;clear BL to keep the number of 1s
 MOV DL,8 ;rotate total of 8 times
 MOV AL,DATA1
 AGAIN: ROL AL,1 ;rotate it once
 JNC NEXT ;check for 1
 INC DL ;if CF=1 then add one to count
 NEXT: DEC DL ;go through this 8 times
 JNZ AGAIN ;if not finished go back
 MOV COUNT,BL ;save the number of 1s

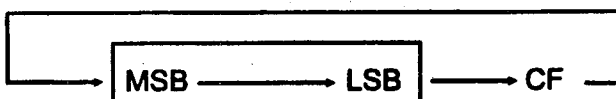
برنامه ۳-۸

برنامه ۳-۸ کاربردی از دستور چرخش را نشان می دهد، حداکثر شمارش در برنامه ۳-۸ برابر ۸ خواهد بود زیرا برنامه تعداد 1 ها را در یک بایت از داده می شمارد. اگر عملوند یک کلمه 16 بیتی باشد، تعداد 1 ها می تواند حداکثر 16 باشد. برنامه ۳-۹ همان بازنویسی شده برنامه ۳-۸ برای عملوند کلمه ای است. همچنین قادر است شمارش را در عوض مبنای 16، به BCD تحویل دهد.
 توجه: AL برای تهیه شمارنده BCD بکار می رود زیرا DAA فقط با AL کار می کند.

برنامه ای بنویسید که تعداد 1 ها را یک کلمه بشمارد. شمارش به BCD باشد .

DATAW1 DW 97F4H
 COUNT2 DB ?
 ...
 SUB AL,AL ;clear AL to keep the number of 1s in BCD
 MOV DL,16 ;rotate total of 16 times
 MOV BX,DATAW1 ;move the operand to BX
 AGAIN: ROL BX,1 ;rotate it once
 JNC NEXT ;check for 1. If CF=0 then jump
 ADD AL,1 ;if CF=1 then add one to count
 NEXT: DAA ;adjust the count for BCD
 DEC DL ;go through this 16 times
 JNZ AGAIN ;if not finished go back
 MOV COUNT2,AL ;save the number of 1s in COUNT2

برنامه ۳-۹



RCR چرخش به راست از طریق نقلی

در RCR، با نقل مکان در جهت چپ به راست، بیت ها از انتهای سمت راست

(LSB) به پرچم نقلی و از پرچم نقلی وارد انتهای چپ (MSB) می گردند. به بیان دیگر، در RCR، بیت LSB به رقم نقلی CF و CF هم به MSB می رود. در واقع، CF بخشی از عملوند عمل می کند. این نکته در نمودار دیده می شود. اگر عملوند یکبار بچرخد دستور یکبار نوشته می شود، ولی اگر بیش از یکبار بچرخد ثبات CL تعداد دفعات را نگهدارند.

```
CLC ;make CF=0
MOV AL,26H ;AL=0010 0110
RCR AL,1 ;AL=0001 0011 CF=0
RCR AL,1 ;AL=0000 1001 CF=1
RCR AL,1 ;AL=1000 0100 CF=1
```

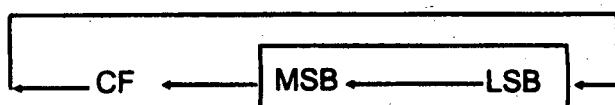
```
CLC ;make CF=0
MOV AL,26H ;AL=0010 0110
MOV CL,3 ;CL=3 number of times to rotate
RCR AL,CL ;AL=1000 0100 CF=1
```

یا

عملوند می تواند یک کلمه باشد:

```
STC ;make CF=1
MOV BX,37F1H ;BX=0011 0111 1111 0001
MOV CL,5 ;CL=5 number of times to rotate
RCR BX,CL ;BX=0001 1001 1011 1111 CF=0
```

چرخش به چپ از طریق نقلی RCL



در RCL، با نقل مکان بیت‌ها به راست، انتهای چپ (MSB) وارد پرچم نقلی می‌گردد، و پرچم نقلی وارد انتهای

سمت راست (LSB) می‌گردد. به بیان دیگر، در RCL، MSB به CF و CF هم به LSB می‌رود. در واقع، CF در نقش بخشی از عملوند عمل می‌کند. این مطلب در نمودار نشان داده شده است. اگر عملوند یکبار بچرخد، دستور یکبار بکار می‌رود ولی اگر چرخش بیش از یکبار باشد CL تعداد دفعات را نگه خواهد داشت

```
STC ;make CF=1
MOV BL,15H ;BL=0001 0101
RCL BL,1 ;0010 1011 CF=0
RCL BL,1 ;0101 0110 CF=0
```

یا:

```
STC ;make CF=1
MOV BL,15H ;BL=0001 0101
MOV CL,2 ;CL=2 number of times for rotation
RCL BL,CL ;BL=0101 0110 CF=0
```

عملوند می تواند یک کلمه باشد:

```
CLC ;make CF=0
MOV AX,191CH ;AX=0001 1001 0001 1100
MOV CL,5 ;CL=5 number of times to rotate
RCL AX,CL ;AX=0010 0011 1000 0001 CF=1
```

سؤالات مروری

```
MOV BL,25H
MOV CL,4
ROR BL,CL
```

۱- مقدار BL پس از دستورات زیر چیست؟

```
MOV DX,3FA2H
MOV CL,7
ROL DX,CL
```

۲- مقدار DX و CF پس از دستورات زیر چیست؟

۳- پس از دستورات زیر BH چیست؟

SUB BH, BH
STC
RCR BH, 1
STC
RCR BH, 1

۴- مقدار BX پس از دستورات زیر چیست؟

MOV BX, FFFFH
MOV CL, 5
CLC
RCL BX, CL

۵- چرا 4, ROR BX در 8086 خطا تولید می‌کند؟ چگونه آن را اصلاح می‌کنید؟

جدول ۳-۴ AND، OR و EX-OR در C

A	B	A & B	A B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

بخش ۳-۶: عملیات بیتی در زبان C

یکی از مهم‌ترین و قوی‌ترین ویژگی‌های زبان C توانایی آن در دستکاری‌های بیتی است. با توجه به اینکه بسیاری از کتب زبان C این عنوان مهم را پوشش نمی‌دهند، بحث آن در این بخش مناسب دیده شد. این بخش نقش عملگرها را توصیف کرده و مثال‌هایی را فراهم کرده است.

جدول ۳-۵ معکوس‌گر بیتی در C

A	~A
0	1
1	0

عملگرهای بیتی در C

در حالی که بسیاری از برنامه‌نویسان C با عملگرهای منطقی AND (&&) یا OR (||) و NOT (!) آشنا هستند، بسیاری دیگر با عملگرهای بیتی AND (&)، OR (|)، EX-OR (^)، معکوس‌گر (~)، نقل مکان به راست (>) و نقل مکان به چپ (<) کمتر آشنا می‌باشند. این عملگرهای بیتی یوفور در مهندسی نرم افزار و کنترل بکار می‌روند، در

نتیجه دانستن آنها در طراحی سیستم و ارتباط اهمیت دارد. جدول ۳-۴ و ۳-۵ ملاحظه شوند. قطعه کد زیر مثال‌های ۳-۵ تا ۳-۷ را با استفاده از عملگرهای منطقی نشان می‌دهد، توجه داشته باشید که "0X" در زبان C بمعنی قالب مبنای شانزده برای داده است.

0x35 & 0x0F = 0x05

/* ANDing: see Example 3-5 */

0x0504 | 0xDA68 = DF6C

/* ORing: see Example 3-6 */

0x54 ^ 0x78 = 0x2C

/* XORing: see Example 3-7 */

~0x37 = 0xC8

/* inverting 37H */

آخرین سطر مانند دستور NOT در ریزپردازنده x86 است:

MOV AL, 37H ; AH=37H

NOT AL ; AFTER INVERTING 37, AL=C8H

عملگرهای نقل مکان بیتی در C

دو عملگر نقل مکان بیتی در C وجود دارد: یکی نقل مکان به راست (>) و دیگری نقل مکان به چپ (<). این دو عملگر دقیقاً مثل SHR، SHL در زبان اسمبلی مورد بحث در بخش ۳-۳، کار می‌کنند. قالب آنها در C بشرح زیر است:

data >> number of bits to be shifted /* shifting right */

data << number of bits to be shifted /* shifting left */

در زیر مثال‌های ۳-۹ تا ۳-۱۱ با عملگرهای نقل مکان در C تکرار شده‌اند. مسئله ۱۰ همه این مثال‌ها را با ساختار C نشان می‌دهد.

```
0x9A >> 3 = 0x13 /* shifting right 3 times : see Example 3-9 */
0x7777 >> 4 = 0x0777 /* shifting right 4 times : see Example 3-10 */
0x6 << 4 = 0x60 /* shifting left 4 times : see Example 3-11 */
```

/*Program 3-10 Repeats Examples 3-5 through 3-11 in C */

```
#include <stdio.h>
main()
{
    // Notice the way data is defined in C for hex format using 0x

    unsigned char data_1 = 0x35;
    unsigned int data_2 = 0x504;
    unsigned int data_3 = 0xDA66;
    unsigned char data_4 = 0x54;
    unsigned char data_5 = 0x78;
    unsigned char data_6 = 0x37;
    unsigned char data_7 = 0x09A;
    unsigned char temp;
    unsigned int temp_2;

    temp = data_1 & 0x0F; /*ANDing
    printf("\nMasking the upper four bits of %X (hex) we get %X (hex)\n", data_1, temp);

    temp_2 = data_2 | data_3; /*ORing
    printf("The result of %X hex ORed with %X hex is %X hex\n", data_2, data_3, temp_2);

    temp = data_4 ^ data_5; /*EX-ORing
    printf("The results of %X hex EX-ORed with %X hex is %X hex\n", data_4, data_5, temp);

    temp = ~data_6; /*INVERTING
    printf("The result of %X hex inverted is %X hex\n", data_6, temp);

    temp = data_7 >> 3; /*SHIFTING Right
    printf("When %X hex is shifted right three times we get %X hex\n", data_7, temp);

    printf("When %X hex is shifted right four times we get %X hex\n", 0x7777, 0x7777 >> 4);
    temp = (0x6 << 4); /*SHIFTING Left

    printf("When %X hex is shifted left %d times we get %X hex\n", 0x6, 4, temp);
}
```

برنامه ۳-۱۰

برنامه ۳-۱۰ ساختار عملگرهای بیتی را در C نشان می‌دهد. در بخش بعد تعدادی مثال واقعی از کاربرد آنها ارائه شده است.

تبدیل BCD فشرده به اسکی در C

بخش ۳-۴ روشی را برای تبدیل عدد BCD به اسکی نشان داد. این تبدیل بطور گسترده‌ای به هنگام مواجه با تراشه ساعت زمان واقعی بکار می‌رود. بسیاری از تراشه‌ها، ساعت زمان واقعی و تاریخ بسیار دقیقی را تا ده سال بدون نیاز به منبع تغذیه خارجی فراهم می‌سازند. در هر کامپیوتر x86 IBM PC یا سازگار با آن یک

ساعت واقعی وجود دارد. با این وجود، این تراشه‌ها زمان و تاریخ را بصورت BCD فشرده فراهم می‌سازند. برای نمایش داده، لازم است که به اسکی بازگردند. برنامه ۳-۱۱ نوع زبان C مثال تبدیل BCD فشرده به اسکی است که در بخش ۳-۴ ملاحظه شد. برنامه ۳-۱۱ یک بایت داده BCD فشرده را به کاراکترهای اسکی و نمایش آنها را با عملگرهای بیتی C نشان می‌دهد.

/ Program 3-11 shows packed BCD-to-ASCII conversion using logical bitwise operators in C */*

```
#include <stdio.h>
main()
{
    unsigned char mybcd=0x29;          /* declare a BCD number in hex */
    unsigned char asci_1;
    unsigned char asci_2;
    asci_1=mybcd&0x0f;                 /* mask the upper four bits */
    asci_1=asci_1|0x30;                 /* make it an ASCII character */
    asci_2=mybcd&0xf0;                 /* mask the lower four bits */
    asci_2=asci_2>>4;                  /* shift it right 4 times */
    asci_2=asci_2|0x30;                 /* make it an ASCII character */
    printf("BCD data %X is %c , %c in ASCII\n",mybcd,asci_1,asci_2);
    printf("My BCD data is %c if not converted to ASCII\n",mybcd);
}
```

برنامه ۳-۱۱

در برنامه ۳-۱۱ توجه داشته باشید که اگر داده BCD فشرده بدون تبدیل به اسکی نمایش داده شود در اینصورت پرانتز "() را خواهیم داشت. پیوست ج ملاحظه شود.

تست بیت‌ها در C

در بسیاری از موارد از برنامه‌نویسی سیستم و ارتباط سخت افزار لازم است تا یک بیت تست شود و مشخص گردد که در سطح بالاست یا خیر. مثلاً بسیاری از وسایل یک سیگنال سطح بالا را ارسال می‌کنند تا برای یک عمل، اعلام آمادگی نمایند و مشخص نمایند که حاوی داده هستند. چگونه یک بیت تست می‌شود؟ در این موارد بیت‌های بکار نرفته پوشش یافته و بقیه داده تست می‌شود. برنامه ۳-۱۲ از کاربر برای یک بایت و تست اینکه آیا D0 از آن بایت در سطح بالاست یا خیر سؤال می‌کند.

/ Program 3-12 shows how to test bit D0 to see if it is high */*

```
#include <stdio.h>
main()
{
    unsigned char status;
    unsigned char temp;
    printf("\nType in a Hex value\n");
    scanf("%X",&status);              //get the data
    temp=status&0x01;                  //mask all bits except D0
    if (temp==0x01)                    //is it high?
        printf("D0 is high");          //if yes, say so
    else printf("D0 is low");           //if no, say no
}
```

برنامه ۳-۱۲

زبان اسمبلی برنامه ۱۲-۳ بشرح زیر است:

;assume AL=value (in hex)

AND AL,01

CMP AL,01

JNE BELOW

....

BELOW:

....

;MASK ALL BITS EXCEPT D0

;IS D0 HIGH

;MAKE A DECISION

;YES D0 IS HIGH

;D0 IS LOW

سؤالات مروری

۱- نتیجه $0x2F \& 0x27$ چیست؟

۲- نتیجه $0x2F | 0x27$ چیست؟

۳- نتیجه $0x2F \wedge 0x27$ چیست؟

۴- نتیجه $0x2F \sim 0$ چیست؟

۵- نتیجه $0x2F >> 3$ چیست؟

۶- نتیجه $0x27 << 4$ چیست؟

۷- در برنامه ۱۱-۳ اگر $mybcd = 0x32$ باشد، و به BCD تبدیل نشود چه چیز نشان داده می شود.

۸- برنامه ۱۲-۳ را برای تست D3 اصلاح کنید.

خلاصه

اقدام داده ۸ یا ۱۶ بیتی در کامپیوترهای ۸۰۸۶ را می توان بصورت علامت دار یا بی علامت بکار برد. داده بی علامت همه ۸ یا ۱۶ بیت را برای نمایش داده بکار می برد. داده علامت دار MSB را بعنوان بیت علامت بکار می برد و مابقی بیت ها برای نمایش داده است. این فصل دستورات حسابی و منطقی که داده بی علامت را بکار برد معرفی کرد. دستورات ADD و SUB جمع و تفریق را روی داده های بی علامت اجرا می کنند. دستورات ADD و SBB همان کار را انجام می دهند ولی رقم نقلی را هم به حساب می آورند. دستورات MUL و DIV ضرب و تقسیم را روی داده های بی علامت اجرا می کنند. دستورات منطقی AND، OR، XOR و CMP عملیات منطقی را روی همه بیت های عملوندشان انجام می دهند و بنابراین در این فصل گنجانده شدند. دستورات نقل مکان و چرخش برای داده های بی علامت شامل SHL، SHR، ROL، ROR، RCL و RCR می باشند. عملیات جمع و تفریق روی داده های BCD و اسکی نیز ارائه شدند. در پایان عملیات منطقی بیتی با استفاده از زبان C ارائه گردید.

مسائل

۱- برای هر یک از موارد زیر CF، ZF و AF را پیدا کنید. همچنین نتیجه جمع و محل ذخیره آن را معین کنید.

MOV AX,255 (پ) MOV DX,4599H (ب) MOV BH,3FH (الف)

STC MOV CX,3458H ADD BH,45H

ADC AX,00 ADD CX,DX

MOV AH,0FEH (ج) MOV CX,0FFFFH (ث) MOV BX,0FF01H (ت)

STC STC ADD BL,BH

ADC AH,00 ADC CX,00

۲- برنامه ای بنویسید که کل پرداخت های یک فروشنده را در هشت ماه محاسبه کند. مقادیر زیر پرداختی این

ماه هاست: \$2300, \$4300, \$1200, \$3700, \$1298, \$4323, \$5673, \$986

۳- برنامه ۳-۲ (در بخش ۳-۱) را با جمع بایتی دوباره بنویسید

۴- برنامه‌ای بنویسید که دو چند بایتی را از هم کسر کرده و نتیجه را ذخیره کند. تفریق بصورت یک بایت در هر نوبت انجام شود. داده‌های برنامه ۳-۲ را بکار برید.

۵- سه مرحله مربوط به یک SUB را بیان کرده و برای داده‌های زیر مراحل را نشان دهید.

(الف) 23H-12H (ب) 43H-51H (پ) 99-99

۶- برنامه‌ای برای اجرای عملیات زیر نوشته و پس از اجرا نتیجه را تحلیل کنید.

(۱) (الف) بایت ۱ × بایت ۲ (ب) بایت ۱ × کلمه ۱ (پ) کلمه ۱ × کلمه ۲

(۲) (الف) بایت ۱ تقسیم بر بایت ۲ (ب) کلمه ۱ تقسیم بر کلمه ۱ (پ) جفت کلمه بر بایت ۱

فرض کنید 230 = بایت ۱، 100 = بایت ۲، 9998 = کلمه ۱، 300 = کلمه ۲، 100000 = جفت کلمه باشند.

۷- فرض کنید که محتوای ثبات‌های زیر در مبنای ۱۶ باشد: AX=F000، BX=3456 و DX=E390. عملیات زیر را انجام دهید. نتیجه و محل ذخیره آن را مشخص نمایید.

در هر حالت ZF و CF را هم معین نمایید

توجه: عملیات مستقل از یکدیگرند

XOR AL,76H	(پ)	OR DH,BL	(ب)	AND DX,AX	(الف)
OR BX,DX	(ج)	XOR AX,AX	(ث)	AND DX,DX	(ت)
XOR DX,0EEEEH	(خ)	OR AX,9999H	(ح)	AND AH,OFF	(چ)
SHR DX,1	(ر)	MOV CL,04	(ذ)	XOR BX,BX	(د)
		SHL AL,CL			
MOV CL,6	(ش)	MOV CL,5	(س)	MOV CL,3	(ز)
SHL DX,CL		SHL BX,CL		SHR DL,CL	

۸- وضعیت ZF و CF را پس از اجرای CMP در هر یک از موارد زیر معین کنید.

MOV DL,34	(پ)	MOV AL,0FFH	(ب)	MOV BX,2500	(الف)
CMP DL,88		CMP AL,6FH		CMP BX,1400	
SUB CX,CX	(ج)	XOR DX,DX	(ث)	SUB AX,AX	(ت)
DEC CX		CMP DX,0FFFFH		CMP AX,0000	
CMP CX,0FFFFH					
		MOV AL,0AAH	(ح)	MOV BX,2378H	(چ)
		AND AL,55H		MOV DX,4000H	
		CMP AL,00		CMP DX,BX	

۹- معین کنید که آیا در هر یک از موارد زیر پرش رخ می‌دهد یا خیر.

MOV AH,55H	(پ)	MOV BH,65H	(ب)	MOV CL,5	(الف)
SUB DL,DL		MOV AL,48H		SUB AL,AL	
OR DL,AH		OR AL,BH		SHL AL,CL	
MOV CL,AH		SHL AL,1		JNC TRAGET	
AND CL,0FH		JC TARGET			
SHR DL,CL					
JNC TARGET					

۱۰- برنامه ۳-۳ را برای یافتن پایین‌ترین نمره در کلاس پیدا کنید.

۱۱- برنامه ۳-۴ را برای تبدیل حروف بزرگ به کوچک بازنویسی کنید.

۱۲- در برنامه IBM BIOS برای تست پرچم‌ها و ثبات‌ها صحت آدرس هر پرش (شرطی و غیرشرطی) را تحقیق

کنید. توجه: همانطور که در فصل ۲ اشاره شد، در پرش رو به جلو آدرس هدف از جمع جابجایی با IP بعد از دستور محاسبه می‌شود و در پرش رو به عقب از آن تفریق می‌گردد.

۱۳- در برنامه ۶-۳، BCD_ADD را برای انجام تفریق BCD چند بایتی بازنویسی کنید.

۱۴- برنامه ۷-۳ را برای تفریق DATA2 از DATA1 بازنویسی کنید. داده‌های زیر را بکار برید.

```
DATA1    DB '0999999999'  
DATA2    DB '0077777775'
```

۱۵- با بکارگیری رهنمون DT، برنامه‌ای برای جمع دو عدد 10 بایتی BCD بنویسید.

۱۶- می‌خواهیم شمارنده‌ای بسازیم تا از 0 تا 99 را در BCD بشمارد. در محل‌های نقطه‌چین چه دستوری قرار می‌دهید.

```
SUB AL,AL  
ADD AL,1  
....
```

۱۷- برنامه مسئله ۱۶ را برای شمارش از 99 تا 0 بنویسید

۱۸- فرمانده‌ای دارای سیاست طبقه زیر است: "نمودار نمرات با جمع هر نمره و تفاضل بین 99 و بالاترین نمره در کلاس حاصل می‌شود" اگر در زیر نمرات کلاس داده شود، برنامه‌ای برای محاسبه نمرات پس از ترسیم نمودار بنویسید: 81، 65، 77، 82، 73، 55، 88، 78، 51، 91، 86، 76. برنامه شما برای هر مجموعه نمره باید کار کند.

۱۹- اگر سعی بر تقسیم 1000000 بر 2 کنیم:

(الف) چه نوع مسئله‌ای با این عمل در CPUهای 8086/286 مرتبط می‌شود؟

(ب) CPU چگونه به ما اجازه می‌دهد تا وجود مشکل را بدانیم؟

۲۰- کدام یک از گروه‌های زیر همانند LOOP XXX عمل می‌کنند؟

DEC CX	(ت)	DEC BX	(پ)	DEC CH	(ب)	DEC CL	(الف)
JNZ XXX		JNZ		JNZ XXX		JNZ XXX	

۲۱- برنامه‌ای بنویسید که تعداد صفرها را در یک کلمه 16 بیتی پیدا کند

۲۲- در برنامه ۲-۳ که جمع چند کلمه‌ای نشان داده شده است اشاره‌گرها با دو دستور INC در عوض "ADD SI,2" بهنگام شده‌اند، چرا؟

۲۳- برنامه‌ای بزبان C با اجزاء زیر بنویسید:

(الف) دو مقدار مبنای شانزده دارد: data1 = 55H و data2 = AAH که بصورت کاراکتر بدون علامت تعریف شده‌اند.

(ب) چهار بیت مرتبه بالای data1 را حذف و سپس آن را به مبنای 16 نشان دهید.

(پ) عملیات AND، OR و EX - OR را بین دو قلم داده انجام دهید و سپس هر نتیجه را نمایش دهید.

(ت) یکبار معکوس کرده و آن را نمایش دهید

(ث) data1 را چهار بار به چپ نقل مکان کنید و data2 را دوباره به راست ببرید، آنگاه هر نتیجه را نشان دهید.

۲۴- مسئله فوق را با دو ورودی از کاربر تکرار کنید. از تابع scanf ("%X") برای بدست آوردن داده مبنای شانزده استفاده کنید.

۲۵- به همان طریق که تراشه ساعت زمان واقعی داده BCD را تولید می‌کند، هنگام مقدار دهی اولیه نیز داده را در BCD می‌پذیرد. با این وجود داده‌ای که از صفحه کلید می‌آید به اسکی است. برنامه‌ای بزبان C برای تبدیل دو بایت داده اسکی به BCD فشرده بنویسید.

۲۶- برنامه‌ای به C بنویسید که در آن کاربر برای یک مقدار مبنای شانزده به انتظار بنشیند سپس داده را برای دیدن اینکه آیا دو بیت کم ارزشتر در وضعیت بالا هستند تست کند. اگر وضعیت برقرار است اعلام کند "D0، D1 هر دو بالا هستند" در غیر اینصورت اعلام شود که بیت بالا نیست.

۲۷- مسئله فوق را برای D0 و D7 تکرار کنید.

بخش ۱-۳: جمع و تفریق بدون علامت

۱- مقصد

۲- در زبان اسمبلی 80x86، عملیات حافظه به حافظه وجود ندارد.

۳-

```
MOV AX, DATA_2
ADD DATA_1, AX
```

۴- مقصد، CF + مقصد + منبع

۵- در (الف)، جمع بایتی تولید نقلی به CF می‌کند، در (ب)، جمع کلمه‌ای تولید نقلی به بایت بالاتر BH می‌نماید.

۶- DEC CX

۷- JNZ ADD_LOOP

```
43H 0100 0011 0100 0011
-05H 0000 0101 2's complement = +1111 1011
3EH 0011 1110
```

CF=0، بنابراین نتیجه مثبت است

۸-

AL = 95 - 4F - 1 = 45

بخش ۲-۳: ضرب و تقسیم بدون علامت

۱- AX ۲- DX and AX ۳- AX ۴- AL, AH

۵- AX, DX ۶- AL, AH ۷- AX, DX

بخش ۳-۳: دستورات منطقی و برنامه‌های نمونه

۱- (الف) 4202 (ب) CFFF (پ) 8DFD

```
A0F2 = 1010 0000 1111 0010
shift left: 0100 0001 1110 0100 CF = 1
shift again: 1000 0011 1100 1000 CF = 0
shift again: 0000 0111 1001 0000 CF = 1
A0F2 shifted left three times = 0790.
A0F2 = 1010 0000 1111 0010
shift right: 0101 0000 0111 1001 CF = 0
shift again: 0010 1000 0011 1100 CF = 1
shift again: 0001 0100 0001 1110 CF = 0
A0F2 shifted right three times = 141E
```

۲- عملوند بدون تغییر می‌ماند؛ تمام صفر

۳- تمام ۱ ۴- تمام صفر ۵-

۶- SUB ۷- غلط

بخش ۴-۳: دستورات و عملوندهای BCD و اسکی

۱- (الف) 000010101 packed BCD = 15

0001 0000 0101 unpacked BCD

99 = 1001 1001 packed BCD = 0000 1001 0000 1001 unpacked BCD (ب)

۲- جمع BCD DAA

تفریق BCD DAS

تفریق اسکی ASS

جمع اسکی AAA

بخش ۵-۳: دستورات چرخش

۱- BX=52H, CF=0 ۲- DX=D11FH, CF=1

۳- BH=C0H ۴- BX=FFEFH

۵- عملوند مبدأ نمی‌تواند فوری باشد. برای اصلاح آن:

بخش ۶-۳: عملیات بیتی در زبان C

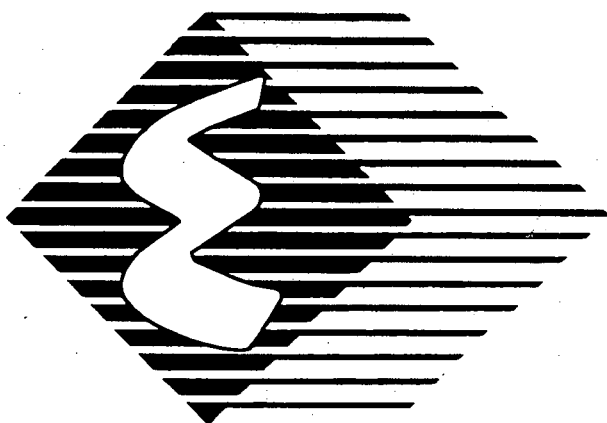
۱- 0x27 ۲- 0x2F ۳- 0x08

۴- 0xD0 ۵- 0x05 ۶- 0x70

۷- 2

۸- این برنامه نشان می‌دهد که چگونه بیت D3 برای دیدن سطح آن تست می‌شود.

```
#include <stdio.h>
main()
{
    unsigned char status;
    unsigned char temp;
    printf("\nType in a Hex value\n");
    scanf("%X",&status);
    temp=status&0x04;
    if (temp==0x04)
        printf("D3 is high");
    else printf("D3 is low");
}
```



برنامه‌نویسی BIOS و DOS در اسمبلی و C

اهداف

- » در پایان این فصل قادر خواهید بود:
- » با فراخوانی توابع INT 10H:
- » صفحه نمایش را با کد نمایش.
- » محل مکان نما را معین کنید.
- » کاراکترها را در روش متن روی صفحه نمایش بنویسید.
- » در روش گرافیک خطوطی را روی صفحه نمایش رسم کنید.
- » مد نمایش تصویر را تغییر دهید.
- » با فراخوانی توابع INT 21H می‌توانید:
- » کاراکترها را از صفحه کلید وارد کنید.
- » کاراکترها را به صفحه نمایش ارسال نمایید.
- » رشته‌ها را وارد یا خارج کنید.
- » برای ایجاد اقلام داده ساخت یافته از رهنمون LABEL استفاده نمایید.
- » INT 16H را برای دستیابی به صفحه کلید بکار ببرید.
- » از توابع INT 21H برای پردازش‌های فایل استفاده کنید: از جمله:
- » یک فایل را باز کنید و یا ببندید.
- » داده‌های را در فایل بنویسید.
- » برای اجرای وقفه‌های BIOS و DOS استفاده نمایید.
- » برای وقفه‌های BIOS و DOS از توابع int 86 و int dos در C استفاده نمایید.
- » در برنامه‌های C از اسمبلی استفاده کنید.

زیر روال‌های بسیار مفیدی در BIOS و DOS وجود دارند که بعضی از آنها از طریق دستورالعمل‌های INT (وقفه) در دسترس کاربر قرار می‌گیرند. در این فصل تعدادی از این دستورات مطالعه شده‌اند تا کاربرانشان در این باره ملاحظه شود. دستورالعمل INT از بعضی لحاظ شبیه فراخوانی دور (FAR call) است. هنگامی که این دستور اجرا شود، سیستم CS:IP و پرچم‌ها را در پشته ذخیره کرده و به زیر روال مربوط به وقفه می‌رود. دستورالعمل INT دارای قالب زیر است:

INT xx ;the interrupt number xx can be 00 - FFH

که در آن xx هر عدد بین 00-FFH است.

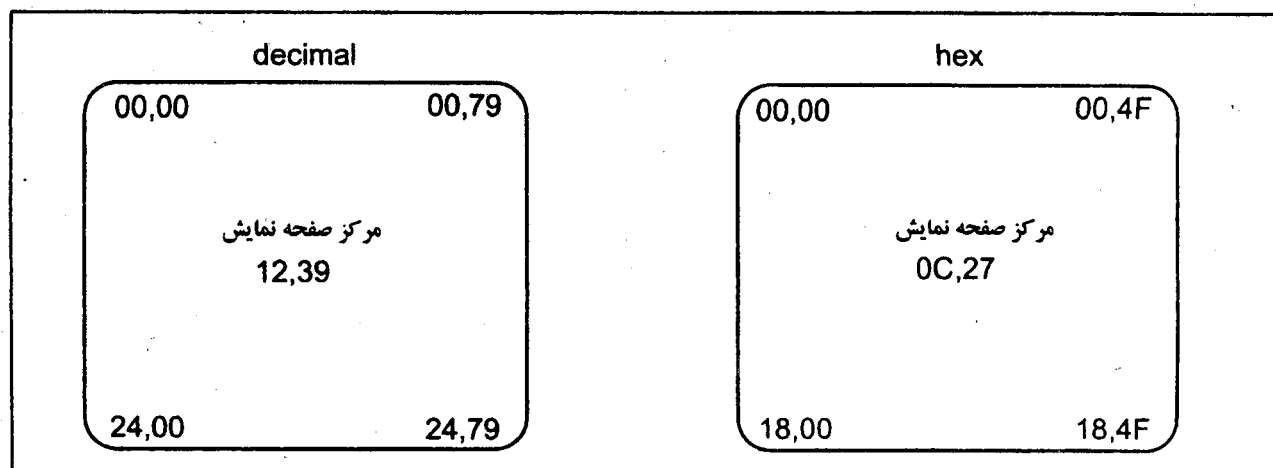
چون وقفه‌ها از 00 تا FF شماره خورده‌اند، در ریزپردازنده‌های 80x86 تعداد 256 وقفه وجود دارد. دو عدد از این وقفه‌ها بطور گسترده‌ای بکار می‌روند که عبارتند از: INT10H و INT21H. هر یک از آنها قادرند توابع (اعمال) بسیاری را انجام دهند. لیستی از این توابع در پیوست (ت) و (ث) آمده است. قبل از تقاضای سرویس بوسیله INT10H یا INT21H، بسته به تابع مورد تقاضا، باید در ثبات‌های معینی مقادیر خاصی وارد شود. تعدادی از توابع انتخابی INT10H, INT21H همراه با مقادیر قرار گرفته در ثبات AH در پیوست‌های (ت) و (ث) نشان داده شده‌اند. دستورالعمل‌های وقفه در پیوست (ب) بطور مشروح مورد بحث قرار گرفته‌اند.

بخش ۴-۱: برنامه‌نویسی INT10H از BIOS

زیر روال‌های INT10H در ROM BIOS کامپیوترهای IBM PC مبتنی بر 80x86 و سازگار با آنها سوزانده شده و برای ارسال اطلاعات به صفحه تصویر کامپیوتر مورد استفاده قرار می‌گیرند. بسیاری از دستکاری‌های متن یا گرافیک در صفحه نمایش از طریق INT10H صورت می‌گیرد. از جمله این دستکاری‌ها از تغییر رنگ کاراکترها و رنگ پس‌زمینه، پاک‌کردن صفحه نمایش و تغییر محل مکان‌نما می‌توان نام برد. این امکانات با قرار دادن یک مقدار خاص در ثبات AH انتخاب می‌گردند. در این بخش چگونگی استفاده از INT10H را در پاک‌کردن صفحه نمایش، تغییر محل مکان‌نما، تغییر رنگ صفحه و ترسیم خطوط را روی صفحه نشان خواهیم دید.

صفحه نمایش در روش متن

صفحه نمایش در روش متن در IBM PC به 80 ستون، 25 سطر تقسیم شده است (شکل ۴-۱). این تقسیم‌بندی مادامی که در روش متن باشد، بدون توجه به رنگی یا تک رنگ بودن صفحه نمایش، معتبر است.



شکل ۴-۱ موقعیت‌های مکان‌نما

وقتی که کامپیوتر روشن شود، مانیتور بصورت پیش فرض به روش متن می‌رود. بسته به نوع مانیتور می‌توان مد را تغییر داد. انواع مانیتورها عبارتند از VGA، EGA، CGA، MDA و SVGA. صفحه نمایش در مد متن دارای عرض 80 و طول 25 کاراکتر است. چون به هر مکان در روی صفحه نمایش اعداد سطر و ستونی تخصیص یافته است، کاربر می‌تواند بسادگی با تغییر مقادیر سطر و ستون مکان‌نما را در هر محلی از صفحه حرکت دهد. 80 ستون از 0 تا 79 و سطر از 0 تا 24 شماره گذاری شده‌اند. گوشه چپ بالا مربوط به 00,00 است (سطر = 00، ستون = 00). بنابراین گوشه راست بالا 79,00 (سطر = 00، ستون = 79) خواهد بود. بهمین ترتیب گوشه چپ پائین 00,24 (سطر = 24، ستون = 00) و گوشه راست پائین 79,24 (سطر = 24، ستون = 79) خواهد بود. شکل ۴-۱ هر یک از مکان‌ها را در هر دو فرم دهمی و شانزده شانه‌دهی نشان می‌دهد.

پاک کردن صفحه نمایش یکمک INT10H تابع 06H

در اغلب موارد لازم است صفحه نمایش قبل از نمایش داده پاک شود. برای پاک کردن صفحه نمایش یکمک INT10H، ثبات‌های زیر باید قبل از فراخوانی INT10H با مقادیر معینی بترتیب زیر بار شوند:

AH=06, AL=00, BH=07, CX=0000, DH=24, DL=79

کد برنامه به فرم زیر است:

MOV	AH,06	;AH=06 to select scroll function
MOV	AL,00	;AL=00 the entire page
MOV	BH,07	;BH=07 for normal attribute
MOV	CH,00	;CH=00 row value of start point
MOV	CL,00	;CL=00 column value of start point
MOV	DH,24	;DH=24 row value of ending point
MOV	DL,79	;DL=79 column value of ending point
INT	10H	;invoke the interrupt

بخاطر داشته باشید که DEBUG عملوندهای فوری را در مبنای 16 تصور می‌کند؛ بنابراین داده در DX بصورت 184F وارد می‌شود. با این وجود MASM عملوندهای فوری را در مبنای 10 تصور می‌نماید. در اینصورت DL=79, DH=24 خواهد بود.

در برنامه فوق یکی از انتخاب‌های INT 10H با قرار دادن 06 در AH انجام شده است. انتخاب AH=06، که تابع حرکت طوماری خوانده می‌شود، موجب می‌گردد تا صفحه نمایش رو به بالا بصورت طوماری حرکت کند. ثبات‌های CH و CL بترتیب سطر و ستون ابتدا را نگهدارند و DH و DL سطر و ستون انتها را نگهداری می‌نمایند. برای پاک کردن تمام صفحه نمایش باید از محل مکان‌نما در گوشه چپ بالا در 00,00 بعنوان نقطه ابتدا و نقطه سمت راست پائین در 24,79 بعنوان انتها استفاده نمود.

در واقع انتخاب AH=06 از INT 10H به معنی گزینش تابع «پنجره طوماری» می‌باشد و بنابراین انتخاب پنجره‌ای با هر سایز با تعیین سطر و ستون مناسب ابتدا و انتها صورت می‌گیرد. با این وجود برای پاک کردن صفحه نمایش، مقادیر چپ بالا و راست پائین بعنوان ابتدا و انتها بکار می‌روند تا تمام صفحه، حرکت طوماری داشته باشد. کدنویسی کارآتر در پاک کردن صفحه با ترکیب برخی از خطوط فوق مطابق زیر بدست می‌آید.

MOV	AX,0600H	;scroll entire screen
MOV	BH,07	;normal attribute
MOV	CX,0000	;start at 00,00
MOV	DX,184FH	;end at 24,79 (hex = 18,4F)
INT	10H	;invoke the interrupt

INT 10H تابع 02: استقرار مکان‌نما در یک محل خاص

INT 10H تابع AH=02 موقعیت مکان‌نما را به هر محل دیگر تغییر می‌دهد. محل موردنظر برای مکان‌نما توسط مقادیر موجود در DX شناسایی می‌شود، که در آن سطر=DH و ستون=DL می‌باشند. RAM تصویر

می‌تواند به بیش از یک صفحه از متن تخصیص یابد، ولی هر بار تنها یکی از آنها قابل مشاهده است. هنگام AH=02 برای استقرار محل مکان‌نما، صفحه صفر بوسیله BH=00 انتخاب می‌گردد. باید توجه داشت که پس از اجرای INT 10H (یا INT 21H)، ثبات‌های استفاده نشده بوسیله وقفه دست‌نخورده باقی می‌مانند. به بیان دیگر، این ثبات‌ها پس از اجرای وقفه همان مقادیر قبل از فراخوانی را دارند. مثال ۴-۱ و ۴-۲ استقرار مکان‌نما را در یک محل معین نشان می‌دهد.

مثال ۴-۱. برنامه‌ای برای استقرار مکان‌نما در سطر 15 (OFH) و ستون 25 (19H) بنویسید.
حل:

```
MOV AH,02      ;set cursor option
MOV BH,00      ;page 0
MOV DL,25      ;column position
MOV DH,15      ;row position
INT 10H         ;invoke interrupt 10H
```

مثال ۴-۲. برنامه‌ای بنویسید که (۱) صفحه نمایش را پاک‌کند و (۲) مکان‌نما را در مرکز صفحه نمایش مستقر سازد.
حل:

مرکز صفحه نمایش محل تلاقی سطر و ستون وسط می‌باشد. سطر 12 در وسط سطرها 0 تا 24 بوده و ستون 39 (یا 40) در وسط ستون‌های 0 تا 79 قرار دارد. بنابراین با قرار دادن DH=12 = سطر و DL=39 = ستون، مکان‌نما در وسط صفحه نمایش قرار خواهد گرفت.

؛ پاک‌کردن صفحه

```
MOV AX,0600H   ;scroll the entire page
MOV BH,07      ;normal attribute
MOV CX,0000    ;row and column of top left
MOV DX,184FH   ;row and column of bottom right
INT 10H        ;invoke the video BIOS service
```

؛ استقرار مکان‌نما در مرکز صفحه تصویر

```
MOV AH,02      ;set cursor option
MOV BH,00      ;page 0
MOV DL,39      ;center column position
MOV DH,12      ;center row position
INT 10H        ;invoke interrupt 10H
```

INT 10H تابع 03: تعیین محل فعلی مکان‌نما

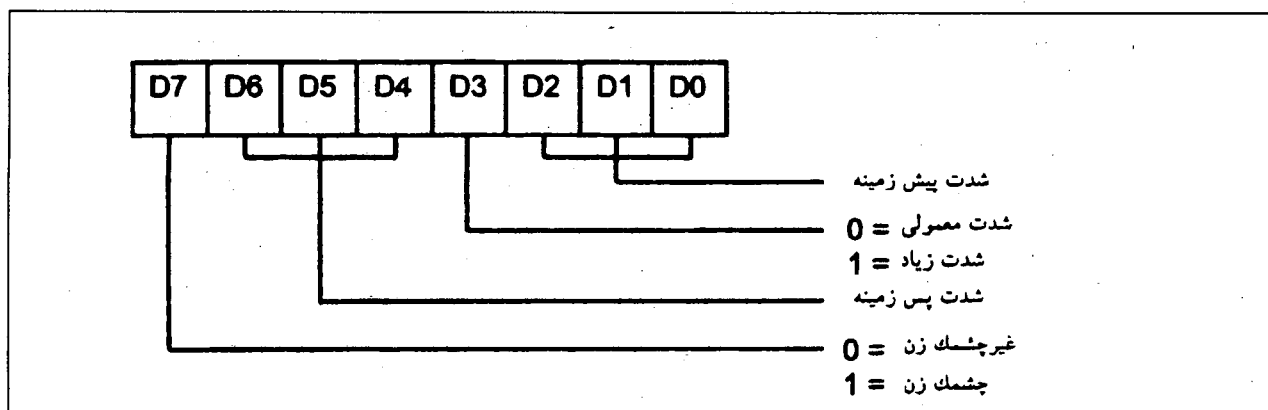
در مد متن، می‌توان در هر زمان محل مکان‌نما را با اجرای برنامه زیر مشخص کرد.

```
MOV AH,03      ;option 03 of BIOS INT 10H
MOV BH,00      ;page 00
INT 10H        ;interrupt 10H routine
```

پس از اجرای برنامه فوق، ثبات‌های DH و DL حاوی شماره سطر و ستون محل‌های جاری مکان‌نما، و CX اطلاعات مربوط به شکل آن را در اختیار می‌گذارد. دلیل انتخاب صفحه 00 این است که بسته به مورد نصب شده بر روی PC، حافظه تصویری می‌تواند بیش از یک صفحه داده داشته باشد. در مد متن، صفحه 00 بعنوان صفحه مشاهده شده جاری در نظر گرفته می‌شود.

تغییر مد تصویر

ابتدا باید متذکر شد که سواى نوع آداپتور بکار رفته (MDA، CGA، EGA، MCGA، یا VGA)، همه آنها بطور صعودی با هم سازگارند. مثلاً، VGA همه عملیات MCGA، EGA، CGA و MDA را تقلید می‌نماید. بهمین ترتیب، EGA همه عملیات CGA و MDA را تقلید می‌کند. بنابراین، باید راهی برای تغییر مد تصویر به مد موردنظر وجود داشته باشد. برای انجام آن، می‌توان INT 10H را با AH=00 و، مد تصویر AL استفاده کرد. لیستی از مدهای تصویر در پیوست (ث)، جدول ۲- (ث) داده شده است.



شکل ۲-۴. بایت خصیصه برای مانیتورهای تک رنگ

بایت خصیصه در مانیتورهای تک رنگ

در ارتباط با هر کاراکتر روی صفحه نمایش، یک خصیصه وجود دارد. خصیصه، اطلاعاتی را برای مدار تصویر فراهم می‌کند. از جمله این اطلاعات می‌توان از رنگ و شدت کاراکتر (پیش‌زمینه) و پس‌زمینه نام برد. بایت خصیصه برای هر کاراکتر در مانیتور تک رنگ محدود است. شکل ۲-۴ تعریف بیتی بایت خصیصه تک رنگ را نشان می‌دهد.

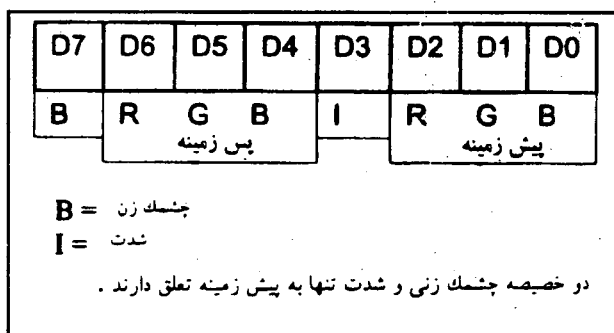
منظور از پیش‌زمینه، کاراکترهای واقعی نمایش داده شده بر روی صفحه است. شدت نور معمولی، متمایز و چشمک‌زنی مختص پیش‌زمینه است. در زیر برخی از تغییرات ممکن در خصیصه‌های فوق آورده شده است.

نتیجه	شانزده شانه‌دهی	دودویی
سفید روی سفید (چیزی نمایش داده نمی‌شود)	00	0000 0000
سفید روی سیاه معمولی	07	0000 0111
سفید روی سیاه متمایز	0F	0000 1111
سفید روی سیاه چشمک‌زن	87	1000 0111
سیاه روی سیاه (چیزی نمایش داده نمی‌شود)	77	0111 0111
سیاه روی سفید	70	0111 0000
سیاه روی سفید چشمک‌زن	F0	1111 0000

مثلاً 0000 0111 مد نمایش معمولی را تولید می‌کند که در آن پس‌زمینه سیاه و پیش‌زمینه سفید معمولی بدون چشمک‌زن است. 0000 1111 نیز همان مد را با پیش‌زمینه متمایز نسبت به حالت قبلی ایجاد می‌کند. 0111 0000 حالت نمایش تصویری معکوسی را با پیش‌زمینه سیاه و پس‌زمینه معمولی ارائه می‌نماید. مثال ۳-۴.

بایت خصیصه در مد متن CGA

چون نام مانیتورهای رنگی و مدارات تصویری بصورت رو به بالا سازگارند، در مثال‌هایی که در این فصل ملاحظه خواهیم کرد از مد CGA بعنوان مبنای مشترک استفاده می‌کنیم. تعریف بیتی بایت خصیصه در مد CGA در شکل ۳-۴ دیده می‌شود. از تعریف بیت می‌توان دید که



شکل ۳-۴. بایت خصیصه CGA

پس زمینه می تواند هشت رنگ مختلف را از ترکیب رنگ های اصلی سبز، آبی و قرمز دارا باشد. پیش زمینه می تواند هر یک از شانزده رنگ ممکن حاصل از سبز، آبی، قرمز و شدت آنها باشد. مثال ۴-۴ کاربرد بایت خصیصه را در مد CGA نشان می دهد. جدول ۴-۱ همه رنگ های ممکن را فهرست کرده است.

به منظور مشاهده برخی از این انواع رنگ به حالات زیر توجه کنید.

رنگ	شانزده شانزدهی	دودویی
سیاه روی سیاه	00	0000 0000
آبی روی سیاه	01	0000 0001
سبز روی آبی	12	0001 0010
قرمز روی آبی	14	0001 0100
سفید روشن روی آبی	1F	0001 1111

مثال ۴-۳. برنامه ای با INT 10H بنویسید تا:

(الف) مد تصویر را عوض کند

(ب) حرف "D" را در مکان "200H" با خصیصه سیاه روی سفید چشمک زن نشان دهد (حروف "D" چشمک زن، سیاه و صفحه پس زمینه سفید است.)

(پ) آنگاه از DEBUG برای اجرا و اطمینان از صحت آن استفاده کنید.

حل:

(الف) تابع 00=AH از INT 10H برای، مد تصویر =AL، جهت تغییر مد تصویر بکار می رود. 07=AL را برای مانیتورهای تک رنگ MDA، EGA یا VGA بکار ببرید؛ در غیر اینصورت هر یک از مدهای متن 80x25 را استفاده کنید، و یا برای CGA از 03=AL استفاده نمایید که قادرست همه مانیتورهای رنگی را تقلید کند.

```
MOV AH,00      ;SET MODE OPTION
MOV AL,07      ;7 FOR MONOCHROME OR 03 FOR CGA TEXT
INT 10H        ;MODE OF 80X25 FOR ANY COLOR MONITOR
```

(ب) با تابع 09=AL از INT 10H، می توان یک کاراکتر را به تعداد مورد نظر و خصیصه خاصی نمایش داد.

```
MOV AH,09      ;DISPLAY OPTION
MOV BH,00      ;PAGE 0
MOV AL,44H     ;THE ASCII FOR LETTER "D"
MOV CX,200H    ;REPEAT IT 200H TIMES
MOV BL,0F0H    ;BLACK ON WHITE BLINKING
INT 10H
```

(پ) یادآوری: DEBUG همه اعداد را به مبنای شانزده تصور می کند.

```
C>debug
-A
1131:0100 MOV AH,00
1131:0102 MOV AL,07      ;USE 03 IF MONITOR IS COLOR
1131:0104 INT 10
1131:0106 MOV AH,09
1131:0108 MOV BH,00
1131:010A MOV AL,44
1131:010C MOV CX,200
1131:010F MOV BL,F0
1131:0111 INT 10
1131:0113 INT 3
1131:0114
```

اکنون نتیجه را با تایپ فرمان G- ملاحظه کنید، قبل از اجرای آن مطمئن شوید که IP=100 است. بعنوان تمرین، ثبات BL را با دیگر خصیصه هایی که قبلاً دیدید عوض کنید. مثلاً، 07=BL که سفید روی سیاه است، یا 87H=BL که سفید روی سیاه چشمک زن می باشد.

مثال ۴-۴. برنامه‌ای بنویسید که 20H (فاصله در ASCII) را در تمام صفحه قرار دهد. از سفید روشن روی پس‌زمینه آبی برای هر کاراکتر مورد نمایش استفاده کنید.
حل:

جدول ۳-۴. رنگ ممکن

رنگ	I	R	G	B
سیاه	0	0	0	0
آبی	0	0	0	1
سبز	0	0	1	0
یشی	0	0	1	1
قرمز	0	1	0	0
بنفش	0	1	0	1
قهوه‌ای	0	1	1	0
سفید	0	1	1	1
خاکستری	1	0	0	0
آبی روشن	1	0	0	1
سبز روشن	1	0	1	0
یشی روشن	1	0	1	1
قرمز روشن	1	1	0	0
بنفش روشن	1	1	0	1
زرد	1	1	1	0
سفید روشن	1	1	1	1

```

MOV AH,00 ;SET MODE OPTION
MOV AL,03 ;CGA COLOR TEXT MODE OF 80X25
INT 10H
MOV AH,09 ;DISPLAY OPTION
MOV BH,00 ;PAGE 0
MOV AL,20H ;ASCII FOR SPACE
MOV CX,800H ;REPEAT IT 800H TIMES
MOV BL,1FH ;HIGH INTENSITY WHITE ON BLUE
INT 10H

```

گرافیک: وضوح پیکسل و رنگ

در مد متن، صفحه نمایش بعنوان ماتریسی از کاراکترهای سطری و ستونی تصور می‌شود. در مد گرافیک، صفحه نمایش بمثابة ماتریسی از پیکسل‌های عمودی و افقی است. تعداد پیکسل‌ها بسته به وضوح و کارت تصویر برای انواع مانیتورها متفاوت است. در این بخش نشان خواهیم داد که چگونه این پیکسل‌ها دستیابی و برنامه‌ریزی می‌شوند. قبل از پرداختن به برنامه‌ریزی پیکسل‌ها، رابطه بین وضوح پیکسل، تعداد رنگ‌های موجود، و میزان حافظه تصویر در یک کارت تصویر بررسی می‌گردد. دو نکته در مورد هر پیکسل وجود دارد. (۱) محل یا موقعیت پیکسل و (۲) خصوصیات آن، که رنگ و

شدت آن را شامل می‌شود. این دو مورد باید در RAM تصویر ذخیره گردند. بنابراین هر چقدر تعداد پیکسل و رنگ بیشتر باشد، حافظه بیشتری برای ذخیره آنها لازم است. به بیان دیگر نیاز به حافظه با افزایش وضوح و تعداد رنگ‌ها، بیشتر می‌شود. بورد CGA قادر است بسته به طراحی ذاتی‌اش حداکثر تا 16K بایت حافظه را دارا باشد. این 16K بایت حافظه به سه طریق مورد استفاده قرار می‌گیرد.

۱- مد متن 80×25 کاراکتری؛ در این مد 2K بایت (80×25=2000) برای کاراکترها و 2K بایت هم برای خصیصه‌های آنها مورد استفاده قرار می‌گیرد، زیرا هر کاراکتر یک بایت خصیصه دارد. این بدان معنی است که هر صفحه نمایش (قاب) 4K بایت حافظه را استفاده می‌کند و در نتیجه CGA قادر است 4 صفحه داده را پشتیبانی کند، که هر صفحه داده به یک صفحه نمایش متعلق است. در این مد، 16 رنگ پشتیبانی می‌شود. برای انتخاب این مد از AL = 03 و تابع AH = 00 از INT10H استفاده کنید.

۲- مد گرافیک با وضوح متوسط 320×200؛ در این مد جمعاً 64000 (64000 = سطر 200 × ستون 320) موجود است. از تقسیم کل حافظه RAM تصویر، یعنی 128K بیت (128 = 16K × 8)، دو بیت برای هر رنگ هر پیکسل در اختیار خواهد بود. این دو بیت چهار امکان را در اختیار می‌گذارد. بنابراین وضوح 320 × 200 در CGA نمی‌تواند بیش از 4 رنگ را پشتیبانی نماید. هنگام انتخاب این مد از AL = 04 استفاده کنید.

۳- وضوح گرافیک 640×200 (وضوح بالا)؛ در این مد جمعاً 128000 پیکسل (128000 = 640×200) وجود دارد. از تقسیم 16K بایت حافظه بر این عدد، برای هر رنگ 1 بیت بدست می‌آید (1 = 128000/128000). این بیت می‌تواند روشن (سفید) یا خاموش (سیاه) باشد. بنابراین، وضوح بالای 640×200 در CGA می‌تواند فقط سیاه و سفید را پشتیبانی کند. برای انتخاب این مد از AL = 06 استفاده نمایید.

مد وضوح پائین 160×100 که در سیستم‌های تلویزیونی رنگی بکار می‌رود در اینجا مورد بررسی قرار نگرفته است زیرا هیچ کامپیوتری از آن استفاده نمی‌کند. از بحث فوق می‌توان نتیجه گرفت که با مقدار معینی از RAM تصویر، افزایش دقت موجب کاهش تعداد رنگ می‌گردد. باین دلیل اگر بخواهیم رنگ‌های بیشتری را

روی بوردهای VGA داشته باشیم باید حافظه مورد تصویر را افزایش دهیم، زیرا باید برای رنگ‌های اضافی محل ذخیره‌ای در حافظه وجود داشته باشد. بسیاری از بوردهای VGA ظرفیت لازم برای توسعه RAM تصویر تا 1 مگا بایت حافظه و یا بیشتر را فراهم می‌نمایند.

INT 10H و برنامه‌ریزی پیکسل

برای دستیابی به یک پیکسل در صفحه نمایش از INT10H با AH = 0CH استفاده کنید. مختصات X و Y پیکسل موردنظر باید معلوم باشند. مقادیر X (ستون) و Y (سطر) بسته به وضوح مانیتور متغیرند. ثبات‌هایی که این مقادیر را نگهدارند عبارتند از CX برای ستون (مختصات X) و DX برای سطر (مختصات Y). اگر مد نمایش بیش از یک صفحه را پشتیبانی می‌کند، BH شماره صفحه خواهد بود، در غیر اینصورت از آن صرف نظر می‌شود. برای روشن و خاموش کردن پیکسل، بترتیب AL=1 و AL=0 برای سیاه و سفید است. مقدار AL را می‌توان برای انواع رنگ‌ها اصلاح کرد.

ترسیم خطوط افقی و عمودی در مد گرافیک

برای رسم یک خط افقی، مقادیری را برای سطر و ستون اختیار کنید بطوری که ابتدای خط را مشخص نماید و سپس ستون را اضافه کنید تا به انتهای خط برسید. مثال ۴-۵ ملاحظه شود.

مثال ۴-۵. برنامه‌ای بنویسید تا:

(الف) صفحه نمایش را پاک کند

(ب) مد CGA با وضوح 640x200 انتخاب شود.

(پ) یک خط افقی از ستون 100 سطر 50 انتخاب و آن را در ستون 200 سطر 50 ختم نماید.

حل:

MOV	AX,0600H	;SCROLL THE SCREEN
MOV	BH,07	;NORMAL ATTRIBUTE
MOV	CX,0000	;FROM ROW=00,COLUMN=00
MOV	DX,184FH	;TO ROW=18H,COLUMN=4FH
INT	10H	;INVOKE INTERRUPT TO CLEAR SCREEN
MOV	AH,00	;SET MODE
MOV	AL,06	;MODE = 06 (CGA HIGH RESOLUTION)
INT	10H	;INVOKE INTERRUPT TO CHANGE MODE
MOV	CX,100	;START LINE AT COLUMN = 100 AND
MOV	DX,50	;ROW = 50
BACK:	MOV	AH,0CH
	MOV	AL,01
	INT	10H
	INC	CX
	CMP	CX,200
	JNZ	BACK

بعنوان تمرین، INT3 را در پایان برنامه قرار دهید و آن را در DEBUG اجرا نمایید تا مفهوم را درک کنید. برای ترسیم یک خط عمودی، CX را ثابت نگهداشته و فقط ثبات DX را افزایش دهید. معادله خطی $y = mx + b$ برای رسم هر خط قابل استفاده است.

تغییر رنگ پس زمینه

وضوح متوسط گرافیکی در CGA، 16 رنگ را فراهم می‌کند. انتخاب AH = 0BH برای تغییر رنگ پس زمینه طبق آنچه در مثال ۴-۶ آمده می‌تواند بکار رود. با اجرای آن برنامه در DEBUG، تمام صفحه نمایش به رنگ آبی تغییر می‌کند و وقتی که INT3 فراخوانی شود، همه ثبات‌ها با حروف بزرگ نمایش داده می‌شوند. مکان‌نما هم محو می‌شود زیرا صفحه نمایش در مد گرافیک است. برای خروج از این مد، فقط Q را تایپ کنید و بدنبال آن کلید بازگشت را برای خروج از DEBUG به DOS بفشارید. آنگاه از فرمان DOS "MODE CO80" برای بازگشت به مد صفحه نمایش اولیه استفاده نمایید.

مثال ۴-۶. برنامه‌ای در DEBUG نوشته و اجرا کنید تا:
 (الف) مد تصویر را به گرافیکی 320x200 همراه با چهار رنگ (AL = 04) تغییر دهید.
 (ب) همه صفحه را آبی کنید.
حل:

```
MOV AH,00      ;SET MODE
MOV AL,4        ;CGA STANDARD 300X 200 WITH
INT 10H

MOV AH,0B       ;SET COLOR
MOV BH,0        ;SET BACKGROUND COLOR
MOV BL,1        ;BLUE (SEE TABLE 4-2)
INT 10H
INT 3           ;STOP
```

سؤالات مروری

- ۱ - وقفه 10H برای اجرای چه سرویس‌هایی است.
- ۲ - مانیتور در مد متن دارای ... ستون و ... سطراست. مکان بالای سمت چپ ...، ... و گوشه پائین سمت راست ...، ... است.
- ۳ - مکان‌های خالی را در برنامه زیر که صفحه را پاک می‌نماید پر کنید. در مقابل هر خط توضیحات کافی را برای بیان هدف هر کد بنویسید.

```
MOV AH,___
MOV AL,___
MOV BH,___
MOV CH,___
MOV CL,___
MOV DH,___
MOV DL,___
INT 10H
```

- ۴ - تابع AH=03 از وقفه 10 INT بکار رفته است. پس از آن DL=34, DH=05 بکار برده شده است. این به چه معنی است؟

- ۵ - هدف از بایت خصیصه برای مانیتورهای تک‌رنگ چیست؟
 - ۶ - در مد متن، یک بایت خصیصه برای هر ... روی صفحه نمایش وجود دارد.
 - ۷ - بایت خصیصه را طوری بنویسید که پس‌زمینه سبز و پیش‌زمینه چشمک‌زن باشد.
 - ۸ - هدف از برنامه مقابل که برای مانیتور تک رنگ است چیست؟
- ```
MOV AH,02
MOV BH,00
MOV DX,0000
INT 10H
MOV AH,09
MOV BH,00
MOV AL,2AH
MOV CX,80
MOV BL,0F0H
INT 10H
```

### بخش ۲-۴: وقفه 21H در DOS

INT21H که مبتنی بر BIOS - ROM می‌باشد بوسیله DOS در مقابل INT10H فراهم گشته است. وقتی که MS - DOS (یا نوع IBM آن به نام PC-DOS) در کامپیوتر بار شود می‌توان برخی از توابع مفید را با INT21H اجرا نمود. این توابع معمولاً به نام توابع DOS INT21H شناخته شده‌اند. لیست غیرکاملی از این انتخاب‌ها در پیوست (ت) آمده است. در این بخش مافقط انتخاب‌هایی را مورد بررسی قرار می‌دهیم که با اطلاعات ورودی از صفحه کلید سر و کار دارند و قرار است روی صفحه تصویر نمایش داده شوند. در فصل‌های قبل، مجموعه ثابتی از داده در قطعه داده تعریف شد و در تخلیه حافظه هم مشاهده گردید. در این فصل، داده‌ها از صفحه کلید داده شده و پس از پردازش، نتایج روی صفحه نمایش نشان داده خواهند شد. این شیوه روش پویاتری در پردازش اطلاعات است و به این دلیل این فصل از کتاب در اینجا مطرح شده است. گرچه داده از طریق صفحه کلید وارد و از صفحه نمایش خارج می‌شود، ولی به هنگام رفع اشکالات برنامه باید داده را از حافظه رونویسی نمود.

## INT21H انتخاب 09 : خروج رشته‌ای از داده روی مانیتور

INT21H را می‌توان برای ارسال یک رشته داده ASCII به مانیتور مورد استفاده قرار داد. برای انجام آن، ثبات‌های زیر باید تنظیم شوند:  $AH = 09$  و  $DX$  نیز برابر با تفاوت مکان آدرس داده ASCII مورد نمایش است. آنگاه INT21H اجرا می‌گردد. آدرس در ثبات  $DX$  یک آدرس تفاوت مکان بوده و  $DS$  هم قطعه داده تصور می‌شود. INT21H انتخاب 09 رشته داده ASCII اشاره شده با  $DX$  را نمایش می‌دهد تا اینکه به علامت دلار "\$" برخورد نماید. مثلاً برای نمایش پیام "The earth is but one country" قطعه داده و قطعه کد مطابق زیر است.

DATA\_ASC DB 'The earth is but one country','\$'

|     |                    |                                      |
|-----|--------------------|--------------------------------------|
| MOV | AH,09              | ;Option 09 to display string of data |
| MOV | DX,OFFSET DATA_ASC | ;DX= offset address of data          |
| INT | 21H                | ;invoke the interrupt                |

## INT21H انتخاب 02 : خروج یک کاراکتر به مانیتور

مواردی وجود دارد که در آن فقط یک کاراکتر به مانیتور ارسال می‌شود. برای انجام آن، 02 را در  $AH$  قرار می‌دهیم،  $DL$  با کاراکتر خروجی بار می‌شود و سپس INT21H اجرا می‌گردد. در زیر حرف "J" با برنامه زیر نمایش داده می‌شود.

|     |        |                                         |
|-----|--------|-----------------------------------------|
| MOV | AH,02  | ;option 02 displays one character       |
| MOV | DL,'J' | ;DL holds the character to be displayed |
| INT | 21H    | ;invoke the interrupt                   |

این انتخاب برای نمایش '\$' روی مانیتور هم بکار می‌رود، زیرا انتخاب نمایش رشته‌ای (انتخاب 09) '\$' را نمایش نمی‌دهد.

## INT21H انتخاب 01 : ورود یک کاراکتر همراه با انعکاس

این تابع تا ورود یک کاراکتر از صفحه کلید به انتظار می‌ماند، سپس آن را به مانیتور منعکس می‌سازد. پس از وقفه، کاراکتر ورودی در  $AL$  خواهد بود.

|     |       |                                             |
|-----|-------|---------------------------------------------|
| MOV | AH,01 | ;option 01 inputs one character             |
| INT | 21H   | ;after this interrupt, AL = input character |

برنامه ۱-۴ وقفه‌های INT10H و INT21H را با هم ترکیب می‌کند. برنامه اعمال زیر را انجام می‌دهد: (۱) صفحه نمایش را پاک می‌کند، (۲) مکان نما را در مرکز صفحه نمایش مستقر می‌سازد و (۳) با شروع از آن نقطه پیام "This is a test of the display routine" را نمایش می‌دهد.

## INT21H انتخاب 0AH : ورود یک رشته داده از صفحه کلید

انتخاب 0AH از INT21H مفهوم دریافت داده از صفحه کلید و ذخیره آن را در مکان حافظه‌ای که از قبل در قطعه داده تعریف شده، بیان می‌دارد. برای انجام آن ثبات‌ها بطریق زیر تنظیم می‌شوند:  $AH=0AH$  و  $DX$  نیز آدرس تفاوت مکانی است که رشته داده در آن ذخیره می‌گردد. این ناحیه را بافر داده می‌نامند. DOS به تعریف ناحیه بافری در قطعه داده نیاز دارد و اولین بایت ساین بافر را مشخص می‌نماید. DOS تعداد کاراکترهایی را که از طریق صفحه کلید وارد می‌شود در بایت دوم، و داده وارد شده نیز در سومین بایت ذخیره می‌کند. مثلاً برنامه زیر می‌تواند تا 6 کاراکتر منجمله کلید بازگشت را از صفحه کلید دریافت کند. 6 مکان برای بافر ذخیره، و با

```

TITLE PROG4-1 SIMPLE DISPLAY PROGRAM
PAGE 60,132
STSEG SEGMENT
 DB 64 DUP (?)
STSEG ENDS

;-----
DTSEG SEGMENT
MESSAGE DB 'This is a test of the display routine','$'
DTSEG ENDS

;-----
CDSEG SEGMENT
MAIN PROC FAR
 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
 MOV AX,DTSEG
 MOV DS,AX
 CALL CLEAR ;CLEAR THE SCREEN
 CALL CURSOR ;SET CURSOR POSITION
 CALL DISPLAY ;DISPLAY MESSAGE
 MOV AH,4CH
 INT 21H ;GO BACK TO DOS
MAIN ENDP

;-----
;THIS SUBROUTINE CLEARS THE SCREEN
CLEAR PROC
 MOV AX,0600H ;SCROLL SCREEN FUNCTION
 MOV BH,07 ;NORMAL ATTRIBUTE
 MOV CX,0000 ;SCROLL FROM ROW=00,COL=00
 MOV DX,184FH ;TO ROW=18H,COL=4FH
 INT 10H ;INVOKE INTERRUPT TO CLEAR SCREEN
 RET
CLEAR ENDP

;-----
;THIS SUBROUTINE SETS THE CURSOR AT THE CENTER OF THE SCREEN
CURSOR PROC
 MOV AH,02 ;SET CURSOR FUNCTION
 MOV BH,00 ;PAGE 00
 MOV DH,12 ;CENTER ROW
 MOV DL,39 ;CENTER COLUMN
 INT 10H ;INVOKE INTERRUPT TO SET CURSOR POSITION
 RET
CURSOR ENDP

;-----
;THIS SUBROUTINE DISPLAYS A STRING ON THE SCREEN
DISPLAY PROC
 MOV AH,09 ;DISPLAY FUNCTION
 MOV DX,OFFSET MESSAGE ;DX POINTS TO OUTPUT BUFFER
 INT 21H ;INVOKE INTERRUPT TO DISPLAY STRING
 RET
DISPLAY ENDP
CDSEG ENDS
END MAIN

```

برنامه ۱-۴.

FFH پر شده است. در زیر بخش‌هایی از قطعه کد و قطعه داده نشان داده شده است.

```

DATA1 ORG 0010H
 DB 6,?,6 DUP (FF) ;0010H=06, 0012H to 0017H = FF

 MOV AH,0AH ;string input option of INT 21H
 MOV DX,OFFSET DATA1 ;load the offset address of buffer
 INT 21H ;invoke interrupt 21H

```

در زیر محتوای حافظه با تفاوت مکان 0010H دیده می‌شود:

| 0010 | 0011 | 0012 | 0013 | 0014 | 0015 | 0016 | 0017 |
|------|------|------|------|------|------|------|------|
| 06   | 00   | FF   | FF   | FF   | FF   | FF   | FF   |



پس از اجرای این برنامه، کامپیوتر به انتظار ورود اطلاعات از صفحه کلید باقی می ماند. پس از ورود داده، IBM PC از روال INT21H خارج نمی شود تا با کلید بازگشت مواجه شود. فرض کنید که داده ورودی از صفحه کلید <RETURN> "USA" باشد، محتوای مکان های حافظه پس از 0010H بشکل زیر خواهد بود:

|      |      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|------|
| 0010 | 0011 | 0012 | 0013 | 0014 | 0015 | 0016 | 0017 |
| 06   | 03   | 55   | 53   | 41   | 0D   | FF   | FF   |
|      |      | U    | S    | A    | CR   |      |      |

در زیر تحلیل قدم به قدمی ارائه شده است.

DOS ساینز بافر را در اولین مکان نیاز دارد  
 0010H = 06  
 کلید سه بار فعال شده است (بغیر از کلید RETURN)  
 0011H = 03  
 تا U، S و A وارد شوند.  
 مقدار مبنای شانزده ASCII برای U است.  
 0012H = 55H  
 مقدار مبنای شانزده ASCII برای S است.  
 0013H = 53H  
 مقدار مبنای شانزده ASCII برای A است.  
 0014H = 41H  
 مقدار مبنای شانزده ASCII برای CR (بازگشت) است.  
 0015H = 0DH

ممکن است این سؤال پیش می آید که مقدار 03 در 0011H از کجا آمده است. DOS این مقدار را در آن مکان قرار می دهد تا نشان دهد سه کاراکتر وارد شده است. چگونه این شمارش کاراکتر صورت گرفته است؟ به آنچه در زیر آمده است توجه کنید.

```
MOV AH,0AH
MOV DX,OFFSET DATA1
INT 21H
```

؛ پس از کلید زنی داده مقدار شمارش برداشت می شود.

```
MOV BX,OFFSET DATA1
SUB CH,CH ;CH=00
MOV CL,[BX]+1 ;move count to CL
```

برای قرار دادن مقدار CR یعنی 0DH در رشته و جایگزینی آن با مثلاً 00، کدهای زیر را وارد کنید:  
 داده های واقعی وارده از مکان 2 + [BX] قرار دارد.

```
MOV SI,CX
MOV BYTE PTR[BX+SI]+2,00
```

### ورود بیش از ساینز بافر

اکنون ببینیم اگر بیش از 6 کاراکتر (حداکثر 5 کاراکتر + CR = 6) کلیدزنی شود چه پیش خواهد آمد؟ ورود پیامی مانند "USA a Country in North America" موجب می شود تا کامپیوتر بلندگو را بصدا در آورد و محتوای بافر بشکل زیر خواهد بود:

|      |      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|------|
| 0010 | 0011 | 0012 | 0013 | 0014 | 0015 | 0016 | 0017 |
| 06   | 05   | 55   | 53   | 41   | 20   | 61   | 0D   |
|      |      | U    | S    | A    | SP   | a    | CR   |

مکان 0015 حاوی ASCII 20H برای فضای خالی (فاصله) و 0016 دارای ASCII 61H برای "a" و بالاخره 0D برای RETURN در 0017 است. طول واقعی که 05 می باشد در تفاوت مکان 0011H است. سؤال دیگری

که مطرح است، اینست که: اگر فقط کلید بازگشت CR فعال شود و کاراکتر وارد نشود چه خواهد شد؟ مثلاً در زیر،

```

ORG 20H
DATA4 DB 10,?,10 DUP (FF)

```

که 0AH را در 0020H می‌گذارد، 0021H را برای تعداد و 0022H را بعنوان اولین مکانی که داده وارد می‌شود، اختیار می‌کند. بنابراین اگر فقط کلید بازگشت فعال شود، 0022H دارای 0DH، یعنی کد مبنای شانزده CR خواهد بود.

```

0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 002A 002B 002C
0A 00 0D FF FF FF FF FF FF FF FF FF FF

```

تعداد واقعی کاراکترهای وارده 0 بوده و در مکان 0021H دیده می‌شود. دقت کنید که CR شمرده نمی‌شود. باید توجه کرد که هر داده وارده، روی صفحه نمایش نشان داده می‌شود. به این عمل انعکاس یا عکس‌العمل گوییم. بنابراین انتخاب 0AH از INT21H رشته داده‌ها را از صفحه کلید دریافت و آنها را روی صفحه نمایش منعکس می‌نماید.

برنامه 4-2 اعمال زیر را انجام می‌دهد، (۱) صفحه را پاک می‌کند، (۲) مکان‌نما را در آغاز خط سوم از بالای صفحه نمایش قرار می‌دهد، (۳) پیام "IBM Personal Computer" را از صفحه کلید می‌پذیرد، (۴) حروف کوچک پیام را به بزرگ تبدیل می‌نماید، (۵) نتایج معکوس شده را روی خط بعدی نمایش می‌دهد.

```

TITLE PROG4-2
PAGE 60,132
STSEG SEGMENT
DB 64 DUP (?)
STSEG ENDS

DTSEG SEGMENT
BUFFER DB 22,?,22 DUP (?) ;BUFFER FOR KEYED-IN DATA
ORG 18H
DATAAREA DB CR,LF,22 DUP (?),'$' ;AREA TO PLACE DATA AFTER CONVERSION
DTSEG ENDS
CR EQU 0DH
LF EQU 0AH

CDSEG SEGMENT
MAIN PROC FAR
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
MOV AX,DTSEG
MOV DS,AX
CALL CLEAR ;CLEAR THE SCREEN
CALL CURSOR ;SET CURSOR POSITION
CALL GETDATA ;INPUT A STRING INTO BUFFER
CALL CONVERT ;CONVERT STRING TO UPPER CASE
CALL DISPLAY ;DISPLAY STRING DATAAREA
MOV AH,4CH
INT 21H ;GO BACK TO DOS
MAIN ENDP

;THIS SUBROUTINE CLEARS THE SCREEN
CLEAR PROC
MOV AX,0600H ;SCROLL SCREEN FUNCTION
MOV BH,07 ;NORMAL ATTRIBUTE
MOV CX,0000 ;SCROLL FROM ROW=00,COL=00
MOV DX,184FH ;TO ROW=18H,4FH
INT 10H ;INVOKE INTERRUPT TO CLEAR SCREEN
RET

```

برنامه ۲-۴ (ادامه در صفحه بعد)

```

;THIS SUBROUTINE SETS THE CURSOR TO THE BEGINNING OF THE 3RD LINE
CURSOR PROC
 MOV AH,02 ;SET CURSOR FUNCTION
 MOV BH,00 ;PAGE 0
 MOV DL,01 ;COLUMN 1
 MOV DH,03 ;ROW 3
 INT 10H ;INVOKE INTERRUPT TO SET CURSOR
 RET
CURSOR ENDP

;THIS SUBROUTINE DISPLAYS A STRING ON THE SCREEN
DISPLAY PROC
 MOV AH,09 ;DISPLAY STRING FUNCTION
 MOV DX,OFFSET DATAAREA ;DX POINTS TO BUFFER
 INT 21H ;INVOKE INTERRUPT TO DISPLAY STRING
 RET
DISPLAY ENDP

;THIS SUBROUTINE PUTS DATA FROM THE KEYBOARD INTO A BUFFER
GETDATA PROC
 MOV AH,0AH ;INPUT STRING FUNCTION
 MOV DX,OFFSET BUFFER ;DX POINTS TO BUFFER
 INT 21H ;INVOKE INTERRUPT TO INPUT STRING
 RET
GETDATA ENDP

;THIS SUBROUTINE CONVERTS ANY SMALL LETTER TO ITS CAPITAL
CONVERT PROC
 MOV BX,OFFSET BUFFER
 MOV CL,[BX]+1 ;GET THE CHAR COUNT
 SUB CH,CH ;CX = TOTAL CHARACTER COUNT
 MOV DI,CX ;INDEXING INTO BUFFER
 MOV BYTE PTR[BX+DI]+2,20H ;REPLACE CR WITH SPACE
 MOV SI,OFFSET DATAAREA+2 ;STRING ADDRESS
AGAIN: MOV AL,[BX]+2 ;GET THE KEYED-IN DATA
 CMP AL,61H ;CHECK FOR 'a'
 JB NEXT ;IF BELOW, GO TO NEXT
 CMP AL,7AH ;CHECK FOR 'z'
 JA NEXT ;IF ABOVE GO TO NEXT
 AND AL,11011111B ;CONVERT TO CAPITAL
 MOV [SI],AL ;PLACE IN DATA AREA
 INC SI ;INCREMENT POINTERS
 INC BX
 LOOP AGAIN ;LOOP IF COUNTER NOT ZERO
 RET
CONVERT ENDP
CDSEG ENDS
END MAIN

```

برنامه ۲-۴ (ادامه صفحه قبل)

### استفاده از بازگشت و تعویض خط

در برنامه ۲-۴، عبارت EQU برای برابری CR (بازگشت) با مقدار اسکی اش، 0DH، و LF (تعویض خط) با مقدار اسکی اش، 0AH بکار رفته است. این موجب مفهوم تر شدن برنامه می گردد. چون قرار است نتایج در خط بعدی نمایش داده شوند، رشته با CR و LF پایان یافته است. در غیاب CR، رشته در هر جایی که مکان نما واقع باشد ظاهر می شود. هنگام وجود CR ولی غیاب LF، رشته پس از بازگشت به اول همان خط بر روی بعضی کاراکترهای آن خط نوشته می شود.

برنامه ۳-۴ اجازه می دهد تا کاربر نامی را تایپ کند. نام می تواند حداکثر هشت حرف داشته باشد. پس از تایپ نام، برنامه طول نام را بدست آورده و آن را روی صفحه نمایش نشان می دهد. برنامه ۴-۴ بسیاری از توابع توصیف شده در این فصل را نشان می دهد.

```

TITLE PROG4-3 READS IN LAST NAME AND DISPLAYS LENGTH
PAGE 60,132
STSEG SEGMENT
DB 64 DUP (?)
STSEG ENDS

DTSEG SEGMENT
MESSAGE1 DB "What is your last name?','$'
 ORG 20H
BUFFER1 DB 9,?,9 DUP (0)
 ORG 30H
MESSAGE2 DB CR,LF,"The number of letters in your name is:','$'
DTSEG ENDS
ROW EQU 08
COLUMN EQU 05
CR EQU 0DH ;EQUATE CR WITH ASCII CODE FOR CARRIAGE RETURN
LF EQU 0AH ;EQUATE LF WITH ASCII CODE FOR LINE FEED

CDSEG SEGMENT
MAIN PROC FAR
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
MOV AX,DTSEG
MOV DS,AX
CALL CLEAR
CALL CURSOR
MOV AH,09 ;DISPLAY THE PROMPT
MOV DX,OFFSET MESSAGE1
INT 21H
MOV AH,0AH ;GET LAST NAME FROM KEYBOARD
MOV DX,OFFSET BUFFER1
INT 21H
MOV BX,OFFSET BUFFER1 ;FIND OUT NUMBER OF LETTERS IN NAME
MOV CL,[BX+1] ;GET NUMBER OF LETTERS
OR CL,30H ;MAKE IT ASCII
MOV MESSAGE2+40,CL ;PLACE AT END OF STRING
MOV AH,09 ;DISPLAY SECOND MESSAGE
MOV DX,OFFSET MESSAGE2
INT 21H
MOV AH,4CH
INT 21H ;GO BACK TO DOS

MAIN ENDP

CLEAR PROC ;CLEAR THE SCREEN
MOV AX,0600H
MOV BH,07
MOV CX,0000
MOV DX,184FH
INT 10H
RET
CLEAR ENDP

CURSOR PROC ;SET CURSOR POSITION
MOV AH,02
MOV BH,00
MOV DL,COLUMN
MOV DH,ROW
INT 10H
RET
CURSOR ENDP
CDSEG ENDS
END MAIN

```

برنامه ۳-۴

### انتخاب 07: ورود از صفحه کلید بدون انعکاس

انتخاب 07 از INT21H یک کاراکتر را از ورودی دریافت کرده ولی آن را روی صفحه نمایش نشان نمی‌دهد (بدون انعکاس). پس از اجرای وقفه، PC به انتظار یک کاراکتر باقی‌مانده و سپس کاراکتر را در AL می‌گذارد.

```

MOV AH,07 ;keyboard input without echo
INT 21H

```

## استفاده از رهنمون LABEL برای تعریف یک بافر رشته

راه سیستماتیک‌تر برای تعریف ناحیه بافر برای ورودی رشته‌ای، استفاده از رهنمون LABEL است. رهنمون LABEL می‌تواند در قطعه داده برای تخصیص چندین نام به داده بکار رود. پس از استفاده از آن در قطعه داده چنین داریم:

name LABEL attribute

تخصیص می‌تواند BYTE، WORD، DWORD، FWORD، QWORD یا TBYTE باشد. ساده‌تر بگوییم، رهنمون LABEL برای بکارگیری آدرس تفاوت مکان یکسان برای دو نام استفاده می‌شود. مثلاً

```
JOE LABEL BYTE
TOM DB 20 DUP(0)
```

آدرس تفاوت مکان تخصیص یافته به JOE همان آدرس اختصاص یافته به TOM است زیرا رهنمون LABEL هیچگونه فضایی از حافظه را بخود اختصاص نمی‌دهد (پیوست (پ) برای مشاهده مثال‌هایی از کاربرد رهنمون LABEL ملاحظه شود). اکنون چگونگی کاربرد این رهنمون را برای تعریف یک ناحیه بافر برای

```
DATA_BUF LABEL BYTE
MAX_SIZE DB 10
BUF_COUNT DB ?
BUF_AREA DB 10 DUP(20H)
```

رشته ورودی‌ها از صفحه کلید خواهیم دید:

اکنون می‌توانیم داده را بوسیله نام در قطعه کد دستیابی کنیم:

```
MOV AH,0AH ;load string into buffer
MOV DX,OFFSET DATA_BUF
INT 21H
MOV CL,BUF_COUNT ;load the actual length of string
MOV SI,OFFSET BUF_AREA;SI=address of first byte of string
```

این روش برای دنبال کردن، بسیار ساختاری‌تر و ساده‌تر است. با استفاده از این روش، می‌توان بکمک نام به هر پارامتر اشاره کرد. مثلاً بکمک رهنمون LABEL، می‌توان زیر برنامه CONVERT را در برنامه 4-2 بطریق زیر نوشت.

```
BUFFER LABEL BYTE
BUFSIZE DB 22
BUFCOUNT DB ?
REALDATA DB 22 DUP(' ')
```

در قطعه داده BUFFER بطریق زیر تعریف می‌شود

```
CONVERT PROC ; CONVERT رویه کد، بجای رویه
MOV CL,BUFCOUNT ;load the counter
SUB CH,CH ;CX=counter
MOV DI,CX ;index into data field
MOV BX,OFFSET REALDATA;actual data address in buffer
MOV BYTE PTR[BX+DI],20H ;replace the CR with space
MOV SI,OFFSET DATAAREA ;SI=address of converted data
AGAIN: MOV AL,[BX] ;move the char into AL
CMP AL,61H ;check if is below 'a'
JB NEXT ;if yes then go to next
CMP AL,7AH ;check for above 'z'
JA NEXT ;if yes then go to next
AND AL,11011111B ;if not then mask it to capital
NEXT: MOV [SI],AL ;move the character
INC SI ;increment the pointer
INC BX ;increment the pointer
LOOP AGAIN ;repeat if CX not zero yet
RET ;return to main procedure
CONVERT ENDP
```

برنامه‌ای برای انجام عملیات زیر بنویسید:

(۱) صفحه نمایش پاک شود

(۲) مکان‌نما را که در سطر ۵ ستون ۱ صفحه نمایش مستقر نماید.

(۳) پیام "There is a message for you from Mr Jones. To read enter Y" را نمایش دهید. اگر کاربر "Y" یا "y" را وارد کرد پیام در روی صفحه نمایش ظاهر خواهد شد. اگر کاربر کلید دیگری را وارد کرد آنگاه باید پیام "no more messages for you" در روی خط بعدی ظاهر گردد.

TITLE PROGRAM 4-4

PAGE 60,132

STSEG SEGMENT  
DB 64 DUP (?)  
STSEG ENDS

DTSEG SEGMENT  
PROMPT1 DB 'There is a message for you from Mr. Jones.'  
DB 'To read it enter Y','\$'  
MESSAGE DB CR,LF,'Hi! I must leave town tomorrow,'  
DB 'therefore I will not be able to see you','\$'  
PROMPT2 DB CR,LF,'No more messages for you','\$'  
DTSEG ENDS  
CR EQU 0DH  
LF EQU 0AH

CDSEG SEGMENT  
MAIN PROC FAR  
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG  
MOV AX,DTSEG  
MOV DS,AX  
CALL CLEAR ;CLEAR THE SCREEN  
CALL CURSOR ;SET CURSOR POSITION  
MOV AH,09 ;DISPLAY THE PROMPT  
MOV DX,OFFSET PROMPT1  
INT 21H  
MOV AH,07 ;GET ONE CHAR, NO ECHO  
INT 21H  
CMP AL,'Y' ;IF 'Y', CONTINUE  
JZ OVER  
CMP AL,'y'  
JZ OVER  
MOV AH,09 ;DISPLAY SECOND PROMPT IF NOT Y  
MOV DX,OFFSET PROMPT2  
INT 21H  
JMP EXIT  
OVER: MOV AH,09 ;DISPLAY THE MESSAGE  
MOV DX,OFFSET MESSAGE  
INT 21H  
EXIT: MOV AH,4CH  
INT 21H ;GO BACK TO DOS  
MAIN ENDP

CLEAR PROC ;CLEARS THE SCREEN  
MOV AX,0600H  
MOV BH,07  
MOV CX,0000  
MOV DX,184FH  
INT 10H  
RET  
CLEAR ENDP

CURSOR PROC ;SET CURSOR POSITION  
MOV AH,02  
MOV BH,00  
MOV DL,05 ;COLUMN 5  
MOV DH,08 ;ROW 8  
INT 10H  
RET  
CURSOR ENDP  
CDSEG ENDS  
END MAIN

این برنامه مشابه برنامه ۴-۴ است که بصورت ساده‌تری بازنویسی شده است.

برنامه اعمال زیر را انجام می‌دهد:

(۱) صفحه نمایش پاک می‌شود.

(۲) مکان‌نما که در سطر ۵ ستون ۱ صفحه نمایش مستقر می‌نماید.

(۳) پیام "There is a message for you from Mr Jones. To read enter Y" را روی صفحه نمایش می‌دهد. اگر کاربر 'Y' یا 'y' را وارد

کند، آنگاه پیام "Hi! I must leave town tomorrow, therefore I will not be able to see you" در روی صفحه نشان داده می‌شود. اگر

کاربر کلید دیگری را بفشارد، پیام no more message for you در خط بعدی ظاهر می‌گردد.

#### TITLE PROGRAM 4-4

;rewritten with simplified segment definition

PAGE 60,132

.MODEL SMALL

.STACK 64

```

;-----
.DATA
PROMPT1 DB 'There is a message for you from Mr. Jones.'
 DB 'To read it enter Y','$'
MESSAGE DB CR,LF,'Hi! I must leave town tomorrow,'
 DB 'therefore I will not be able to see you','$'
PROMPT2 DB CR,LF,'No more messages for you','$'
CR EQU 0DH
LF EQU 0AH
;-----
.CODE
MAIN PROC
 MOV AX,@DATA
 MOV DS,AX
 CALL CLEAR ;CLEAR THE SCREEN
 CALL CURSOR ;SET CURSOR POSITION
 MOV AH,09 ;DISPLAY THE PROMPT
 MOV DX,OFFSET PROMPT1
 INT 21H
 MOV AH,07 ;GET ONE CHAR, NO ECHO
 INT 21H
 CMP AL,'Y' ;IF 'Y', CONTINUE
 JZ OVER
 CMP AL,'y'
 JZ OVER
 MOV AH,09 ;DISPLAY SECOND PROMPT IF NOT Y
 MOV DX,OFFSET PROMPT2
 INT 21H
 JMP EXIT
OVER: MOV AH,09 ;DISPLAY THE MESSAGE
 MOV DX,OFFSET MESSAGE
 INT 21H
EXIT: MOV AH,4CH
 INT 21H ;GO BACK TO DOS
MAIN ENDP
;-----
CLEAR PROC ;CLEARS THE SCREEN
 MOV AX,0600H
 MOV BH,07
 MOV CX,0000
 MOV DX,184FH
 INT 10H
 RET
CLEAR ENDP
;-----
CURSOR PROC ;SET CURSOR POSITION
 MOV AH,02
 MOV BH,00
 MOV DL,05 ;COLUMN 5
 MOV DH,08 ;ROW 8
 INT 10H
 RET
CURSOR ENDP
END MAIN

```

برنامه ۴-۴. با تعریف ساده‌تر قطعه بازنویسی شده است.

۱- تابع ... INT در ROM BIOS واقع است در حالی که ... INT بوسیله DOS تهیه شده است.

۲- تفاوت میان دو برنامه زیر چیست؟

```
MOV AH,09
MOV DX,OFFSET BUFFER
INT 21H
```

```
MOV AH,0AH
MOV DX,OFFSET BUFFER
INT 21H
```

۳- تابع INT21H رشته‌ای از داده را از مکانی که بوسیله ثابت DX مشخص می‌شود نمایش می‌دهد. سیستم چگونه انتهای رشته را شناسایی می‌نماید؟

۴- با استفاده از INT21H مکان‌های خالی را پر کنید. MESSAGE1 DB 'What is your last name?\$',

```
MOV AH,____
MOV DX,____
INT 21H
```

۵- می‌خواهیم پیام زیر را نمایش دهیم. اگر این رشته را با تابع 09 از INT21H خارج کنیم چه می‌شود؟

PROMPT1 DB 'Enter(round to nearest \$) youre annual salary'

۶- از رهنمون EQU برای مساوی کردن نام "BELL" با کد اسکی برای بصد در آوردن زنگ استفاده کنید.

۷- برنامه‌ای برای بصد در آوردن زنگ بنویسید.

۸- از رهنمون‌های تعریف داده استفاده کرده و برنامه‌ای برای تعریف ناحیه بافری که INT21H انتخاب 0AH شماره بیمه را وارد آن می‌کند بنویسید.

### بخش ۳-۴: برنامه نویسی صفحه کلید INT16H

آخرین بخش، کاربرد INT21H تابع AH = 07 را نشان داد که در آن به انتظار دریافت یک کاراکتر از کاربر باقی می‌ماند. اگر قرار باشد برنامه‌ای بطور مداوم برای کار خاصی همراه با تست فشردن کلیدی اجرا شود چه خواهد شد؟ در چنین حالاتی استفاده از INT 16H پیشنهاد می‌شود که وقفه BIOS از آن خصوصاً برای صفحه کلید استفاده می‌نماید.

#### تست فشردن کلید

برای تست یک کلید فشرده از تابع AH = 01 در INT16H استفاده می‌کنیم.

```
MOV AH,01 ;check for key press
INT 16H ;using INT 16H
```

پس از بازگشت اگر کلیدی فشرده شده باشد ZF = 0، و در غیر اینصورت ZF = 1 خواهد بود. توجه داشته باشید که این تابع منتظر کاربر برای فشردن کلید نمی‌شود، بلکه فقط کلید فشرده شده‌ای را در آن هنگام، تست می‌کند. کاربرد این تابع در قالب مثالهایی در این رابطه بهتر درک خواهد شد. برنامه ۴-۵ کاراکتر اسکی زنگ، 07H (بیوست ج) ملاحظه شود) را مرتباً به صفحه نمایش می‌فرستد. برای متوقف کردن صدای زنگ، کاربر باید بدلولخواه کلیدی را بفشارد.

#### کدام کلید فشرده شده است؟

موقعی وجود دارد که برنامه نه تنها مایل به اطلاع از فشردگی کلید است بلکه می‌خواهد بداند که کدام کلید فشرده شده است. برای انجام آن، می‌توان از INT16H تابع AH = 0 بلافاصله بعد از INT16H تابع AH = 01 استفاده کرد.



```
MOV AH,0 ;get key pressed
INT 16H ;using INT 16H
```

پس از بازگشت، AL حاوی کاراکتر اسکی کلید فشرده شده است. کلید پویش شده در AH قرار دارد. دقت داشته باشید که این تابع باید بلافاصله پس از INT16H تابع AH = 01 بکار رود. برنامه ۴-۶ کار آن را به نمایش گذاشته است. پویش صفحه کلید بطور مشروح در جلد دوم این مجموعه مورد بحث قرار گرفته است.

```
TITLE PROGRAM 4-5:KEYBOARD HIT USING INT 16H
THIS PROGRAM SOUNDS THE BELL CONTINUOUSLY UNLESS ANY KEY IS PRESSED

.MODEL SMALL
.STACK
.DATA
MESSAGE DB 'TO STOP THE BELL SOUND PRESS ANY KEYS'
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX
MOV AH,09
MOV DX,OFFSET MESSAGE ;DISPLAY THE MESSAGE
INT 21H
AGAIN: MOV AH,02 ;SENDING TO MONITOR A SINGLE CHAR
 MOV DL,07 ;SEND OUT THE BELL CHAR
 INT 21H
 MOV AH,01 ;CHECK THE KEY PRESS
 INT 16H ;USING INT 16H
 JZ AGAIN ;IF NO KEY PRESS STAY IN THE LOOP
 MOV AH,4CH ;IF ANY KEY PRESSED GO BACK TO DOS
 INT 21H
MAIN ENDP
END
```

برنامه ۴-۵

```
TITLE PROGRAM 4-6: MODIFIED VERSION OF PROGRAM 4-5
THIS PROGRAM SOUNDS THE BELL CONTINUOUSLY UNTIL 'Q' OR 'q' IS PRESSED

.MODEL SMALL
.STACK
.DATA
MESSAGE DB 'TO STOP THE BELL SOUND PRESS Q (or q) KEY$'
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX
MOV AH,09
MOV DX,OFFSET MESSAGE ;DISPLAY THE MESSAGE
INT 21H
AGAIN: MOV AH,02
 MOV DL,07 ;SOUND THE BELL BY SENDING OUT BELL CHAR
 INT 21H
 MOV AH,01 ;CHECK FOR KEY PRESS
 INT 16H ;USING INT 16H
 JZ AGAIN ;IF NO KEY PRESS KEEP SOUNDING THE BELL
 MOV AH,0 ;TO GET THE CHARACTER
 INT 16H ;WE MUST USE INT 16H ONE MORE TIME
 CMP AL,'Q' ;IS IT 'Q'?
 JE EXIT ;IF YES EXIT
 CMP AL,'q' ;IS IT 'q'
 JE EXIT ;IF YES EXIT
 JMP AGAIN ;NO, KEEP SOUNDING THE BELL
EXIT: MOV AH,4CH ;GO BACK TO DOS
 INT 21H
MAIN ENDP
END
```

برنامه ۴-۶

- ۱- کدام تابع INT16H برای تشخیص کلید فشرده بکار می‌رود؟
- ۲- در سؤال فوق چگونه می‌فهمید کلیدی فشرده شده است؟
- ۳- در مثال فوق چگونه مقدار اسکی برای کلید فشرده شده بدست می‌آید؟
- ۴- تفاوت عمده بین INT21H تابع AH = 07 و INT16H تابع AH = 01 چیست؟
- ۵- برنامه ساده‌ای را برای بصدا در آوردن زنگ بجز در حالت فشردن حرف 'X' بنویسید. اگر 'X' فشرده شد، برنامه خاتمه یابد.

## بخش ۴-۴: برنامه‌نویس فایل دیسک

بخش‌های قبلی چگونگی ورود داده از صفحه کلید و خروج آن را به مانیتور نشان دادند. با این وجود بسیاری از برنامه‌ها به ذخیره یا بازیابی اطلاعات در دیسک نیاز دارند. اطلاعات در دیسک‌های نرم (فلاپی) و یا سخت (هارد) در فایل‌ها ذخیره می‌شوند. برنامه‌های زبان اسمبلی بوسیله شناسه فایل به یک فایل دسترسی پیدا می‌کنند. این شناسه عدد منحصر بفردی است که آن فایل را، مادامی که برنامه از آن استفاده می‌کند شناسایی می‌نماید. لطفاً برای مروری بر معانی جملاتی مانند فهرست قطاع، دسته و FAT (جدول تخصیص فایل) پیوست (ب) را ملاحظه کنید. در این بخش ابتدا مفهوم شناسه فایل را توضیح داده و سپس بکمک INT21H پردازش فایل را ملاحظه خواهیم کرد.

### شناسه فایل و کد خطا

در DOS ویرایش 1.0، برای دستیابی به یک فایل، برنامه نویس‌ها باید آنچه را که FCB (بلوک کنترل فایل) نامیده می‌شد بکار می‌بردند. با این وجود با شروع DOS ویرایش 2، شناسه فایل معرفی شد. چون در کاربردهای اخیر شناسه فایل راهی استاندارد و پیشنهاد شده در دستیابی فایل‌هاست، ما فقط این روش را بررسی خواهیم کرد. وقتی که فایلی ایجاد می‌شود، DOS یک شماره 16 بیتی که شناسه فایل نامیده می‌شود به آن اختصاص می‌دهد. از آن پس، هرگونه دستیابی به آن به کمک شناسه فایل خواهد بود.

برای ایجاد و دستیابی به یک فایل باید دو مرحله زیر انجام شود.

۱- برای دیسک گردان، مسیر و نام فایل یک رشته ASCII را بکار ببرید. این رشته باید با 00 پایان یابد که به آن اسکی صفر می‌گوئیم.

۲- از INT21H انتخاب 3CH برای باز کردن یا ایجاد فایل استفاده کنید. اگر ایجاد فایل موفقیت‌آمیز باشد DOS شناسه فایل را برای آن فایل تهیه کرده و پرچم نقلی را به 0 می‌نشانند. این شناسه فایل باید ذخیره شود و در تمام دستیابی‌های بعدی، مورد استفاده قرار گیرد. اگر باز کردن فایل موفقیت‌آمیز نباشد، DOS یک کد خطا تولید کرده و پرچم نقلی را 1 می‌کند. در زیر چگونگی باز کردن فایل، خواندن و نوشتن در یک فایل و بستن فایل بکمک INT21H تشریح شده است. بقیه سرویس‌هایی که بوسیله INT21H تهیه می‌شوند در پیوست (ت) آمده است. جدول ۱- (ت) لیستی از کدهای خطا را نشان می‌دهد.

جدول ۲-۴. بایت خصیصه فایل

| مفهوم            | خصیصه |
|------------------|-------|
| فایل معمولی      | 00H   |
| فایل فقط خواندنی | 01H   |
| فایل پنهان       | 02H   |
| فایل سیستم DOS   | 04H   |
| پرچم Volume      | 08H   |
| زیر فهرست        | 10H   |
| فایل آرشیو       | 20H   |

### INT21H، AH = 3CH (باز کردن یک فایل)

قبل از این تابع، یعنی AH = 3CH، DX به رشته ASCII که حاوی نام فایل است اشاره می‌کند، و نیز CX حاوی بایت خصیصه می‌باشد. توجه کنید که اگر انتخابی را با نام فایل موجود بکار ببرید بر روی فایل اطلاعات قدیمی خواهید نوشت و در واقع آن را پاک خواهید کرد. اگر تابع موفق به ایجاد فایل گردد، پرچم نقلی صفر شده، شناسه

فایل در AX قرار گرفته و وارده فهرست برای فایل ایجاد می‌شود. همه مراجعات بعدی به فایل از طریق شناسه فایل انجام خواهد شد. اگر هرگونه خطایی رخ دهد، CF در وضعیت 1 قرار گرفته و کد خطا در AX خواهد بود. برای ملاحظه لیست کدهای خطا به پیوست ۱- (ت) مراجعه نمایید.

```
MOV AH,3CH ;create file option
MOV CX,0 ;normal file
MOV DX,OFFSET FNAME_A ;DX points to ASCIIZ string
INT 21H ;invoke interrupt
JC ERROR ;go display error message
MOV HANDLE_A,AX ;save file handle
```

#### AH = 40H, INT21H (نوشتن در فایل)

قبل از اجرای این تابع، باید AH = 40H و CX برابر با تعداد بایت‌های نوشتنی باشد و نیز BX حاوی شناسه فایل بوده و DX به شروع آدرس تفاوت مکان داده نوشتنی اشاره کند. اگر عملیات موفقیت‌آمیز باشد CX = 0 و AX برابر با بایت‌های نوشته شده خواهد بود. و در غیراینصورت CF = 1 و AX حاوی کد خطاست.

```
OUTBUF DB 'Hello'

MOV AH, 40H ;write to file option
MOV CX,5 ;write 5 bytes
MOV BX,HANDLE_A ;BX contains handle
MOV DX,OFFSET OUTBUF ;DX points to data to write
INT 21H ;invoke interrupt
```

#### AH = 3EH, INT21H (بستن یک فایل)

قبل از اتمام برنامه، تمام فایل‌ها باید بسته شوند. این موجب می‌شود تا همه رکوردهایی که شما برای فایل نوشته‌اید و هنوز در بافر حافظه قرار دارند به فایل اضافه گردند. بستن فایل موجب بهنگام سازی وارده‌های فهرست و FAT نیز می‌شود. قبل از فراخوانی این تابع، BX حاوی شناسه فایل و AH = 3EH است.

```
MOV AH,3EH ;close file option
MOV BX,HANDLE_A ;set up handle
INT 21H ;invoke interrupt
```

#### AH = 3FH, INT21H (خواندن یک فایل)

قبل از فراخوانی این فایل AH = 3FH بوده، و BX حاوی شناسه فایل، CX برابر با تعداد بایت‌های خواندنی، و DX به بافر ورودی (آدرس تفاوت مکان) اشاره می‌کند. در صورت موفقیت عملیات، CF = 0 و AX کد خطا را نگه خواهد داشت.

```
MOV AH,3FH ;read file option
MOV CX,80 ;read 80 bytes
MOV BX,HANDLE_A ;BX holds the handle
MOV DX,OFFSET INBUF ;DX points to input buffer
INT 21H ;invoke interrupt
```

#### AH = 3DH, INT21H (بازکردن یک فایل)

قبل از فراخوانی این تابع، AL برابر با مد (لیست زیر ملاحظه شود) و DX هم به رشته ASCIIZ اشاره می‌کند.

در صورت موفقیت،  $CF = 0$  و  $AX$  برابر با شناسه فایل است. در صورت وجود خطا  $CF = 1$  و  $AX$  کد خطا را نگه می‌دارد.

```
MOV AH,3DH ;open file option
MOV AL,0 ;open for read
MOV DX,OFFSET FNAME_A ;DX points to ASCIIZ string
INT 21H ;invoke interrupt
```

| AL mode: 76543210 (bits) | Result                             |
|--------------------------|------------------------------------|
| 000                      | open for read                      |
| 001                      | open for write                     |
| 010                      | open for read/write                |
| 0                        | reserved                           |
| 000                      | give others compatible access      |
| 001                      | read/write access denied to others |
| 010                      | write access denied to others      |
| 011                      | read access denied to others       |
| 100                      | give full access to others         |
| 0                        | file inherited by child process    |
| 1                        | file private to current process    |

### سؤالات مروری

- ۱- صحیح یا غلط. FCB و شناسه فایل بصورت قابل تعویض می‌توانند با یک تابع DOS 21H بکار روند.
- ۲- اگر قبلاً فایلی وجود داشته باشد و شما سعی کنید با تابع 3CH آن را ایجاد کنید چه رخ می‌دهد؟
- ۳- اگر برنامه شما قبل از بستن فایل‌های خروجی پایان یابد چه می‌شود؟
- ۴- در رشته ASCIIZ زیر چه چیز غلط است
- ۵- در شناسه فایل زیر چه چیز غلط است

### بخش ۴-۵: برنامه نویسی وقفه با C

گرچه C یک زبان سطح بالاست ولی دارای قابلیت دستکاری بیتی است. به این دلیل برخی از برنامه نویسان به C زبان "اسمبلی سطح بالا" می‌گویند. برای برنامه نویسان  $C++$  که اطلاع کاملی از برنامه نویسی بزرگ اسمبلی 80x86 ندارند ولی مایل به نوشتن برنامه‌هایی بکمک وقفه‌های DOS INT21H و BIOS هستند، توابع کمکی int86 و intdos از کامپایلر وجود دارند. تابع int86 برای فراخوانی وقفه‌های PC بکار می‌روند در حالی که تابع intdos فقط توابع DOS INT21H را فرا می‌خواند.

#### برنامه نویسی وقفه‌های BIOS با $C++$

برای بکارگیری تابع int86 ابتدا ثبات‌ها با مقادیر مورد نظر تنظیم می‌شوند، سپس int86 فراخوانی می‌شود. پس از بازگشت از int86، ثبات‌های 80x86 قابل دستیابی خواهند بود. از این لحاظ int86 شبیه دستور "INT#" در زبان اسمبلی 80x86 است. برای دستیابی به ثبات‌های 80x86 از ساختار UNION REGS که قبلاً بوسیله کامپایلر C معرفی شد استفاده می‌شود. این ساختار دارای قالب زیر بوده و regin و regout نام متغیرهاست.

ساختار union REGS امکان دستیابی ثبات‌های 80x86 را در قالب 8 و 16 بیتی میسر می‌سازد. ثبات‌های 16 بیتی با x و ثبات‌های 8 بیتی با h (بمعنی نیم کلمه) ارجاع داده می‌شوند. این مطلب در جدول ۳-۴ آمده است.

| TITLE<br>PAGE | FILE I/O<br>60,132<br>.MODEL SMALL<br>.CODE                                                                                                                                                                                                                          |                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|
|               | MAIN procedure                                                                                                                                                                                                                                                       |                                  |
|               | این برنامه صفحه نمایش را پاک می کند، صفحه کلید را به داخل بافر می خواند، بافر را در مانیتور می نویسد، فایل را می بندد و سپس فایل را باز می کند.<br>فایل را برای بافر می خواند، بافر را در مانیتور می نویسد، فایل را می بندد، و هنگامی که کاربر آماده شد خارج می شود. |                                  |
| MAIN          | PROC FAR                                                                                                                                                                                                                                                             |                                  |
|               | MOV AX,@DATA                                                                                                                                                                                                                                                         |                                  |
|               | MOV DS,AX                                                                                                                                                                                                                                                            |                                  |
|               | CALL CLR_SCR                                                                                                                                                                                                                                                         | ;clear the screen                |
|               | MOV AH,02                                                                                                                                                                                                                                                            | ;set cursor position             |
|               | MOV BH,00                                                                                                                                                                                                                                                            |                                  |
|               | MOV DL,10                                                                                                                                                                                                                                                            | ;column position                 |
|               | MOV DH,10                                                                                                                                                                                                                                                            | ;row position                    |
|               | INT 10H                                                                                                                                                                                                                                                              |                                  |
|               | CALL CREATE_F                                                                                                                                                                                                                                                        | ;create an output file           |
|               | MOV AH,0AH                                                                                                                                                                                                                                                           | ;read keyboard input             |
|               | MOV DX,OFFSET BUFFER_1                                                                                                                                                                                                                                               |                                  |
|               | INT 21H                                                                                                                                                                                                                                                              |                                  |
|               | CALL WRITE_F                                                                                                                                                                                                                                                         | ;write buffer to file            |
|               | CALL CLOSE_F                                                                                                                                                                                                                                                         | ;close file                      |
|               | MOV AH,02                                                                                                                                                                                                                                                            | ;set cursor position             |
|               | MOV BH,00                                                                                                                                                                                                                                                            |                                  |
|               | MOV DL,10                                                                                                                                                                                                                                                            | ;column position                 |
|               | MOV DH,20                                                                                                                                                                                                                                                            | ;row position                    |
|               | INT 10H                                                                                                                                                                                                                                                              |                                  |
|               | CALL OPEN_F                                                                                                                                                                                                                                                          | ;open file for read              |
|               | CALL READ_F                                                                                                                                                                                                                                                          | ;read file to buffer             |
|               | MOV AH,09                                                                                                                                                                                                                                                            | ;copy buffer to screen           |
|               | MOV DX,OFFSET BUFFER_2                                                                                                                                                                                                                                               |                                  |
|               | INT 21H                                                                                                                                                                                                                                                              |                                  |
|               | CALL CLOSE_F                                                                                                                                                                                                                                                         | ;close file                      |
|               | MOV AH,02                                                                                                                                                                                                                                                            | ;set cursor position             |
|               | MOV BH,00                                                                                                                                                                                                                                                            |                                  |
|               | MOV DL,10                                                                                                                                                                                                                                                            | ;column position                 |
|               | MOV DH,24                                                                                                                                                                                                                                                            | ;row position                    |
|               | INT 10H                                                                                                                                                                                                                                                              |                                  |
|               | MOV DX,OFFSET EXIT_MSG                                                                                                                                                                                                                                               | ;print "ANY KEY TO EXIT" message |
|               | CALL ERR_MSG                                                                                                                                                                                                                                                         |                                  |
|               | MOV AH,07                                                                                                                                                                                                                                                            | ;exit when user hits any key     |
|               | INT 21H                                                                                                                                                                                                                                                              |                                  |
|               | MOV AH,4CH                                                                                                                                                                                                                                                           | ;set up to return                |
|               | INT 21H                                                                                                                                                                                                                                                              | ;return to DOS                   |
| MAIN          | ENDP                                                                                                                                                                                                                                                                 |                                  |
|               | این رویه فایل خروجی را ایجاد می کند                                                                                                                                                                                                                                  |                                  |
| CREATE_F      | PROC NEAR                                                                                                                                                                                                                                                            |                                  |
|               | MOV AH,3CH                                                                                                                                                                                                                                                           | ;create file function            |
|               | MOV CX,0                                                                                                                                                                                                                                                             | ;normal file                     |
|               | MOV DX,OFFSET FILE_1                                                                                                                                                                                                                                                 | ;DX points to ASCIIZ             |
|               | INT 21H                                                                                                                                                                                                                                                              | ;invoke interrupt                |
|               | JC C_ERR                                                                                                                                                                                                                                                             |                                  |
|               | MOV HANDLE_F1,AX                                                                                                                                                                                                                                                     | ;save handle if OK               |
|               | JMP C_EXIT                                                                                                                                                                                                                                                           |                                  |
| C_ERR:        | MOV DX,OFFSET ERR_CREATE                                                                                                                                                                                                                                             | ;display error message           |
|               | CALL ERR_MSG                                                                                                                                                                                                                                                         |                                  |
| C_EXIT:       | RET                                                                                                                                                                                                                                                                  |                                  |
| CREATE_F      | ENDP                                                                                                                                                                                                                                                                 |                                  |
|               | این رویه فایل را برای بافر می خواند                                                                                                                                                                                                                                  |                                  |
| READ_F        | PROC NEAR                                                                                                                                                                                                                                                            |                                  |
|               | MOV AH,3FH                                                                                                                                                                                                                                                           | ;read from file                  |
|               | MOV BX,HANDLE_F1                                                                                                                                                                                                                                                     | ;use handle                      |
|               | MOV CX,25                                                                                                                                                                                                                                                            | ;number of bytes to read         |
|               | MOV DX,OFFSET BUFFER_2                                                                                                                                                                                                                                               | ;DX points to buffer             |
|               | INT 21H                                                                                                                                                                                                                                                              | ;invoke interrupt                |
|               | JNC R_EXIT                                                                                                                                                                                                                                                           |                                  |
|               | MOV DX,OFFSET ERR_READ                                                                                                                                                                                                                                               | ;display error message           |
|               | CALL ERR_MSG                                                                                                                                                                                                                                                         |                                  |
| R_EXIT:       | RET                                                                                                                                                                                                                                                                  |                                  |
| READ_F        | ENDP                                                                                                                                                                                                                                                                 |                                  |

برنامه ۷-۴. کاربرد شناسه فایل INT21H

```

WRITE_F PROC NEAR
MOV AH,40H ;write to file
MOV BX,HANDLE_F1
MOV CX,25 ;number of bytes to write
MOV DX,OFFSET BUFFER_1+2 ;DX points to buffer
INT 21H ;invoke interrupt
JNC W_EXIT
MOV DX,OFFSET ERR_WRITE ;display error message
CALL ERR_MSG
W_EXIT:
WRITE_F RET
ENDP

CLOSE_F PROC NEAR
MOV AH,3EH ;close file function
MOV BX,HANDLE_F1 ;use handle
INT 21H ;invoke interrupt
JNC CL_EXIT
MOV DX,OFFSET ERR_CLOSE ;display error message
CALL ERR_MSG
CL_EXIT:
CLOSE_F RET
ENDP

OPEN_F PROC NEAR
MOV AH,3DH ;open file function
MOV AL,0 ;read only
MOV DX,OFFSET FILE_1 ;DX points to ASCIIZ
INT 21H ;invoke interrupt
JC O_ERR
MOV HANDLE_F1,AX ;save handle if OK
JMP O_EXIT
O_ERR: MOV DX,OFFSET ERR_OPEN ;display error message
CALL ERR_MSG
O_EXIT:
OPEN_F RET
ENDP

ERR_MSG PROC NEAR
MOV AH,09H ;DX points to msg before call
INT 21H ;output to monitor
RET ;invoke interrupt
ERR_MSG ENDP

CLR_SCR PROC NEAR
MOV AX,0600H ;scroll screen
MOV BH,07
MOV CX,0000 ;scroll the
MOV DX,184FH ;entire screen
INT 10H
RET
CLR_SCR ENDP

HANDLE_F1 DW 0 ;file1 handle
FILE_1 DB 'C:\FILE1.ASC',0 ;file1 ASCIIZ string
BUFFER_1 DB 25,?,25 DUP (' ')
BUFFER_2 DB 25 DUP (' '), '$'
ERR_CREATE DB 0DH,0AH,'** Error creating file **$'
ERR_OPEN DB 0DH,0AH,'** Error opening file **$'
ERR_READ DB 0DH,0AH,'** Error reading file **$'
ERR_WRITE DB 0DH,0AH,'** Error writing to file **$'
ERR_CLOSE DB 0DH,0AH,'** Error closing file **$'
EXIT_MSG DB 0DH,0AH,'PRESS ANY KEY TO EXIT$'

;
.STACK 32
END MAIN

```

ادامه برنامه ۷-۴

جدول ۳-۴. عناصر REGS union و معادل اسمبلی آنها

| 16 بیت        |             | 8 بیت      |             |
|---------------|-------------|------------|-------------|
| زبان C        | زبان اسمبلی | زبان C     | زبان اسمبلی |
| regin.x.ax    | AX          | regin.h.al | AL          |
|               |             | regin.h.ah | AH          |
| regin.x.bx    | BX          | regin.h.bl | BL          |
|               |             | regin.h.bh | BH          |
| regin.x.cx    | CX          | regin.h.cl | CL          |
|               |             | regin.h.ch | CH          |
| regin.x.dx    | DX          | regin.h.dl | DL          |
|               |             | regin.h.dh | DH          |
| regin.x.si    | SI          |            |             |
| regin.x.di    | DI          |            |             |
| regin.x.cflag | CY          |            |             |

برنامه زیر بار کردن ثبات‌ها و اجرای وقفه را در C و زبان اسمبلی مقایسه می‌کند.

|                                   |                           |    |
|-----------------------------------|---------------------------|----|
| /* C language                     | Assembly language         | */ |
| union REGS regin,regout;          |                           |    |
| regin.h.ah=0x25;                  | /* mov ah,25h ;AH=25H     | */ |
| regin.x.dx=0x4567;                | /* mov dx,4567h ;DX=4567H | */ |
| regin.x.si=0x1290;                | /* mov si,1290h ;SI=1290H | */ |
| int86(interrupt#,&regin,&regout); | /* int #                  | */ |

در برنامه فوق، # interrupt مقداری بین 00 تا 255 (یا 0x00 تا 0xFF) را به hex داراست و از نشانه C برای اعداد مبنای 16 استفاده شده است)، و regin & و regout آدرس/متغیرهای REGS هستند. پس از بازگشت از تابع int86، می‌توان محتوای ثبات‌ها را همچون برنامه‌های زبان اسمبلی دستیابی کرد. این مطلب در زیر نشان داده شده است.

```
mydata=regout.h.ah; /* mov mydata,ah ;assign AH to mydata */
myvalu=regout.x.bx; /* mov myvalu,bx ;assign BX to myvalu */
```

مثال‌های ۴-۷ و ۴-۸ چگونگی بکارگیری int86 را در برنامه‌نویسی C نشان می‌دهند. مثال ۴-۹ چگونگی دستیابی به ثبات‌ها را پس از بازگشت از int86 بیان می‌دارد. مثال ۴-۱۰ از AH = 30H INT21H برای نمایش ویرایش DOS استفاده می‌کند.

مثال ۴-۷. تابع int86 را برای پاک کردن صفحه نمایش بکار برید. معادل دستورالعمل INT10 را نشان دهید.

```
/* example 4-7A using 16-bit registers */
#include <dos.h> /* int86 is part of this library */
main()
{
 union REGS regin,regout;
 regin.x.ax=0x0600; /* MOV AX,0600H */
 regin.x.bx=0x07; /* MOV BX,07H */
 regin.x.cx=0; /* MOV CX,0 */
 regin.x.dx=0x184F; /* MOV DX,184FH */
 int86(0x10,&regin,&regout); /* INT 10H */
}
```

حل:

می‌توانیم ثبات‌های 8 و 16 بیتی را مطابق زیر در هم بیامیزیم.

```
/* example 4-7B using 8-bit registers */
#include <dos.h> /* int86 is part of this library */
main()
{
 union REGS regin,regout;
 regin.h.ah=6; /* MOV AH,6 */
 regin.h.al=0; /* MOV AL,0 */
 regin.h.bl=07; /* MOV BL,7 */
 regin.h.bh=0; /* MOV BH,0 */
 regin.x.cx=0; /* MOV CX,0 */
 regin.h.dl=0x4F; /* MOV DL=4FH */
 regin.h.dh=0x18; /* MOV DH=18H */
 int86(0x10,&regin,&regout); /* INT 10H */
}
```

مثال ۸-۴. تابع int86 را همراه با INT12H برای یافتن ساین حافظه متعارف نصب شده در PC مفروض بکار ببرید.  
حل:

INT12H ساین حافظه متعارف را در ثبات AX تهیه می‌کند

```
#include <stdio.h>
#include <dos.h>
main()
{
 unsigned int convmem;
 union REGS regin,regout;
 int86(0x12,&regin,&regout);
 convmem=regout.x.ax;
 printf("This PC has %dKB of Conventional memory\n",convmem);
}
```

مثال ۹-۴. تابع int86 را برای اجرای اعمال زیر بکار ببرید

(الف) موقعیت جاری مکان نما را ذخیره کنید، (ب) مکان نما را در سطر 12، ستون 8 مستقر کنید و (پ) پیام "Hello" را با تابع print f نمایش دهید.

حل:

```
#include <stdio.h>
#include <dos.h>
main()
{
 unsigned char oldrow;
 unsigned char oldcol;
 union REGS regin,regout;
 regin.h.ah=3; /* MOV AH,3 ;option 3 INT 10H */
 regin.h.bh=0; /* MOV BH,0 ;page 0 */
 int86(0x10,&regin,&regout); /* INT 10H ;video INT */
 oldrow=regout.h.dh; /* MOV oldrow,DH ;save row */
 oldcol=regout.h.dl; /* MOV oldcol,DL ;save col */
 printf("Cursor was at row=%d,column=%d\n",oldrow,oldcol);
 regin.h.ah=2; /* MOV AH,2 ;option 2 of int 10H */
 regin.h.bh=0; /* MOV BH,0 ;Page zero */
 regin.h.dl=8; /* MOV DL,8 ;col location */
 regin.h.dh=12; /* MOV DH,12 ;rol location */
 int86(0x10,&regin,&regout); /* INT 10H */
 printf("Hello\n");
}
```



## برنامه نویسی تابع INT21H با C++

گرچه می‌توانیم تابع int86 را برای فراخوانی INT21H DOS بکار ببریم، ولی تابع مفید دیگری بنام intdos نیز برای فراخوانی تابع DOS وجود دارد. قالب intdos برابر زیر است و مثال ۱۱-۴ هم کاربرد آن را نشان می‌دهد.

```
intdos(&regin,regout); n /*to be used for INT 21H only*/
```

مثال ۱۰-۴. با استفاده از تابع int86 با INT21H انتخاب 30، برنامه‌ای برای نمایش ویرایش DOS بنویسید.  
حل:

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
main()
{
 union REGS regin,regout;
 unsigned char minor, major;
 clrscr();
 regin.h.ah=0x30; //get DOS version using AH=30H of INT 21H
 int86(0x21,&regin,&regout);
 minor=regout.h.ah;
 major=regout.h.al;
 printf("The DOS version on this PC is %d.%d\n",major,minor);
}
```

مثال ۱۱-۴. از INT21H انتخاب 2AH برای نمایش تاریخ بفرم سال - ماه - روز روی صفحه نمایش استفاده کنید  
(الف) توابع indos را بکار برید. (ب) تابع int86 را استفاده نمایید.  
حل:

پس از بازگشت از INT21H تابع 2AH، DL حاوی روز، DH حاوی ماه، و CX حاوی سال می‌باشد  
(الف) این برنامه indos را استفاده می‌کند.

```
#include <stdio.h>
#include <dos.h>
main()
{
 unsigned int year;
 unsigned char month;
 unsigned char day;
 union REGS regin,regout;
 regin.h.ah=0x2A;
 intdos(&regin,&regout);
 day=regout.h.dl;
 month=regout.h.dh;
 year=regout.x.cx;
 printf("Today's date is %d-%d-%d\n",month,day,year);
}
```

(ب) در این برنامه می‌توانیم عبارت indos را با عبارت زیر جایگزین کنیم.

```
int86(0x21,&regin,&regout)
```

## دستیابی به ثبات‌های قطعه

int86، indos دستیابی به ثبات‌های AX، BX، CX، DX، SI و DI را فراهم می‌کنند ولی نمی‌توانند دستیابی به ثبات‌های CS، DS، SS و ES را میسر سازند. در بعضی از سرویس‌های وقفه نیاز به دستیابی به ثبات‌های قطعه هم وجود دارد. در این موارد باید از int86 بجای intdosx و intdos بجای indos استفاده شود. در هنگام استفاده از int86x و intdosx، باید آرگومان SREGS نیز بکار رود. توابع int86x و intdosx دارای

قالب زیر هستند و در آنها &regseg به ساختار زیر اشاره می نماید:

```
int86x(interrupt #,&regin,&regout,&regseg);
intdosx(&regin,&regout,&regseg);
struct SREGS regseg; // struct SREGS given below
```

توابع int86x و intdosx فقط دستیابی به ES و DS را فراهم می کنند ولی ثبات های CS و SS دستیابی نمی شوند. محتوای CS و SS قابل تغییر نیستند زیرا هر تغییری در آنها موجبات بهم ریختگی برنامه را فراهم خواهد ساخت. خوشبختانه توابع BIOS و DOS که ثبات های قطعه را بکار می برند نیاز به تغییر CS و SS ندارند. مثال ۱۲-۴ چگونگی بدست آوردن جدول بردار وقفه را نشان می دهد.

**مثال ۱۲-۴.** با استفاده از INT21H انتخاب 35H، IP، CS را در جدول بردار وقفه برای INT10H بدست آورید.  
**حل:**

از پیوست (ت)، INT21H، AH = 35 و AL برابر با شماره وقفه را داریم. پس از بازگشت، ES حاوی مقدار قطعه کد (CS) و BX دارای مقدار اشاره گردستورالعمل (IP) از جدول بردار است.

```
#include <stdio.h>
#include <dos.h>
main()
{
 unsigned int ipvalu;
 unsigned int csvalu;
 union REGS regin,regout;
 struct SREGS regseg;
 regin.h.ah=0x35; /* MOV AH,35H */
 regin.h.al=0x10; /* MOV AL,10H */
 int86x(0x21,&regin,&regout,&regseg);
 /* or we can use intdosx(&regin,&regout,&regseg) */
 ipvalu=regout.x.bx; /* MOV ipvalu,BX */
 csvalu=regseg.es; /* MOV csvalu,ES */
 printf("The CS:IP of INT 10H is %X:%X\n",csvalu,ipvalu);
}
```

### دستیابی به پرچم نقلی در توابع int86 و intdos

ضمن بازگشت از بسیاری از وقفه ها نیاز به واری پرچم نقلی وجود دارد. توابع int86، intdos، و int86x و intdosx اجازه می دهند تا فقط پرچم نقلی مورد آزمایش قرار گیرند ولی هیچیک از پرچم های دیگر از طریق این توابع در دسترس نمی باشند. برای دستیابی به پرچم نقلی چنین می نویسیم:

```
if(regout.x.cflag)
```

ساختار ثبات های کلمه ای، ثبات های بیتی و ثبات های قطعه در زیر نشان داده شده است.

```
union REGS {
 struct WORDREGS {
 unsigned int ax;
 unsigned int bx;
 unsigned int cx;
 unsigned int dx;
 unsigned int si;
 unsigned int di;
 unsigned int cflag;
 } x;
 struct BYTEREGS {
 unsigned char al,ah;
 unsigned char bl,bh;
 unsigned char cl,ch;
 unsigned char dl,dh;
 } h;
} *inregs;
union REGS *outregs;
struct SREGS {
 unsigned int es;
 unsigned int cs;
 unsigned int ss;
 unsigned int ds;
} *seregs;
```

```
// using in-line assembly to check the zero flag, this program displays a statement
// a certain number of times, using the cx register for a counter
// thanks to Mark Lessley for his input on this example
```

```
#include <stdio.h>
#include <conio.h>
```

```
main()
{
 unsigned char row=10;
 unsigned char col=10; //Byte size data
 unsigned int counter=5;
 clrscr();
 AGAIN: asm MOV CX,counter //cx=counter
 asm MOV AH,2 // AH=02 of INT 10H to set cursor
 asm MOV BH,0 //page 0
 asm MOV DH,row // load the row value
 asm MOV DL,col // load the column value
 asm INT 10H // call INT 10H to set cursor
 asm PUSH CX // save the counter

 printf("This is a test!");
 row++;
 col++;

 asm POP CX //restore the counter
 asm DEC CX; // decrement the counter
 asm JNZ AGAIN // go back if ZF not high

 getch();
}
```

توجه : در حالی که عبارات C باید با ";" پایان یابد ولی برای عباراتی که دارای پیشوند asm هستند این امر اختیاری است

#### برنامه ۸-۴

#### ترکیب C با اسمبلی و تست ZF

اکثریت قریب باتفاق وقفه‌ها در PC از پرچم نقلی برای نشان دادن شرایط خاص مانند خطاها استفاده می‌کنند. مثلاً توابع 3CH = AH، 3DH = AH در وقفه INT21H (پیوست (ت)) که هر دو با فایل‌ها سرو کار دارند برای شناسایی شرایط معینی به پرچم نقلی نیاز دارند. همانطور که قبلاً بحث شد، union REGS فقط CF (پرچم نقلی) را در اختیار می‌گذارد. اگر نیاز به تست پرچم صفر برای برقراری شرایط خاصی موردنظر باشد؛ تنها راه، ترکیب C و اسمبلی است. به این ترکیب، اسمبلی درون برنامه‌ای گویند و در برنامه ۸-۴ آورده شده است. هردستور العمل معتبر در x86 می‌تواند در برنامه C باپیشوند "asm" بکار رود. در برنامه ۸-۴ یک عبارت چندین بار تکرار شده است. در عوض استفاده از "for" برای حلقه C، از ثبات CX بعنوان شمارنده استفاده شده و "JNZ" نیز پرچم صفر را تست می‌نماید.

#### تابع kbhit از C در برابر ورودی صفحه کلید INT16H

تابع kbhit معادل C از INT16H در زبان اسمبلی است. برنامه ۹-۴ ویرایش C برنامه ۵-۴ است.

```
/*keep sounding the bell unless any key is pressed */
```

```
#include <stdio.h>
#include <conio.h>
```

```
main()
{
 clrscr();
 printf("To Stop the Bell Sound Press Any Key\n");
 while(!kbhit()) //continue as long as there is no keyboard hit
 printf("%c",0x07); //send the Bell ASCII character to monitor
}
```

#### برنامه ۹-۴

برنامه ۱۰-۴ در عین حال که کلید مخصوصی را واریسی می‌کند با استفاده از تابع getch() در C فشار کلید را نیز واریسی می‌کند.

```
/*Keep sounding the bell unless 'Q' or 'q' is pressed */
#include <stdio.h>
#include <conio.h>

main()
{
 unsigned char data;
 clrscr();
 printf("To Stop the Bell Sound Press Q or q Key\n");
 do
 {
 while(!kbhit()) //keep sounding Bell unless a key is pressed
 printf("%c",0x07);
 data=getch(); //get the key press
 }
 while(!(data=='q' || data=='Q')); //continue as long as it is not Q or q
}
```

برنامه ۱۰-۴

### سؤالات مروری

- ۱- صحیح یا غلط. تابع int86 برای هر شماره وقفه‌ای قابل استفاده است.
- ۲- صحیح یا غلط. تابع intdos برای هر شماره وقفه‌ای قابل استفاده است.
- ۳- int86 ... آرگومان دارد، در صورتی که intdos دارای ... آرگومان است
- ۴- صحیح یا غلط. عملوند regin.h.al ثابت 16 بیتی را دستیابی می‌کند.
- ۵- آیا کد زیر صحیح است؟
- ۶- برای دستیابی به ثابت‌های قطعه از .... (intdos , int86 , int86x) استفاده می‌کنیم.
- ۷- تابع int86x دارای ... آرگومان است که عبارتند از...
- ۸- صحیح یا غلط. در توابع int86x و intdosx، فقط ثابت‌های ES و DS قابل دسترسی‌اند.

### خلاصه

تابع INT10H قابلیت دستکاری متن و گرافیک را روی صفحه نمایش فراهم می‌نماید. این توابع در ROM BIOS قرار دارند زیرا سرعت در بکارگیری مکرر این روال‌ها اهمیت دارد. توابع فراخوانده شده شامل پاک کردن یا حرکت طوماری صفحه نمایش، تغییر مد تصویر و نوشتن متن یا گرافیک در آن است. در مد متن، برنامه نویس با یک ماتریس 80x25 کاراکتر کار می‌کند. در مد گرافیک، ماتریسی از پیکسل‌ها وجود دارد که تعداد آن‌ها با مد تصویر تغییر می‌نماید.

تابع INT21H بوسیله DOS تهیه شده تا عملیات مفید بسیاری را انجام دهد. توابعی که در این فصل معرفی شدند شامل فراخوانی برای ورود کاراکتر و یا خروج آن‌ها به صفحه نمایش بودند. تابع INT16H دستیابی به صفحه کلید را از طریق برنامه‌های BIOS میسر می‌سازد. توابع INT21H نیز برای پردازش فایل از جمله بازکردن یا بستن یک فایل، نوشتن داده و یا خواندن داده از آن بکار می‌رود.

برنامه‌نویسی وقفه در زبان C نیز قابل اجرا می‌باشد که این از طریق ساختار REGS و توابع int86 و intdos در C صورت می‌گیرد. بعلاوه دستورالعمل‌های زبان اسمبلی می‌توانند در برنامه‌نویسی C، با استفاده از پیشوند asm بکار روند. در اینصورت به آن اسمبلی درون برنامه‌ای می‌گویند.

۱- برنامه‌ای بنویسید که (الف) صفحه نمایش را پاک کند، (ب) مکان نما را در سطر 5 ستون 12 مستقر نماید.

۲- برنامه زیر چه کاری انجام می‌دهد؟

```
MOV AH,02
MOV BH,00
MOV DL,20
MOV DH,10
INT 10H
```

۳- برنامه زیر برای استقرار مکان نما در سطر 14 ستون 20 نوشته شده است. خطا را در آن اصلاح کرده و آن را برای تحقیق صحت عملکرد خود اجرا کنید.

```
MOV AH,02
MOV BH,00
MOV DH,14H
MOV DL,20H
INT 10H
```

۴- برنامه‌ای بنویسید که مکان نما را در سطر 12 ستون 15 مستقر کند، و سپس با استفاده از برنامه زیر موقعیت فعلی مکان نما را در ثبات DX مشخص کنید. DH حاوی سطر و DL حاوی ستون است. سطر و ستون فوق به دهنده‌ی است یا شانزده شانزده؟ در مورد جواب خود تحقیق کنید.

```
MOV AH,03
MOV BH,00
INT 10H
```

۵- در هنگام پاک کردن صفحه نمایش، آیا رشته کدهای قبل از INT10H اهمیت دارند؟ در تنظیم محل مکان نما چگونه؟ موضوع را با جابجایی و اجرای دستورات تحقیق کنید.

۶- می‌خواهید صفحه نمایش را با برنامه زیر پاک کنید، ولی مقداری خطا در آن وجود دارد. خطا را رفع و برنامه را برای تأیید اجرا نمایید.

```
MOV AX,0600H
MOV BH,07
MOV CX,0000
MOV DX,184F
INT 10H
```

۷- برنامه‌ای بنویسید که:

(الف) صفحه نمایش را پاک کند،

(ب) مکان نما را در سطر 8 ستون 14 مستقر کند، و (پ) رشته "IBM Personal Computer" را نمایش دهد.

۸- برنامه زیر را اجرا کرده و حافظه را برای تأیید محتویات مکان‌های 0220H تا 022FH تخلیه نمایید. بشرطی که عبارت "IBM PC with 8088 CPU" وارد شود.

```
ORG 220H
BUFFER DB 15,16 DUP (0FFH)
```

```
MOV AH,0AH
MOV DX,OFFSET BUFFER
INT 21H
```

۹- برنامه‌ای بنویسید که: (الف) صفحه نمایش را پاک کند، (ب) مکان نما را در سطر 15 ستون 20 قرار دهد، (پ) عبارت "what is your name?" را نمایش دهد، پاسخی را از صفحه کلید دریافت و آن را در سطر 17 ستون 20 نمایش دهد.

۱۰- برنامه‌ای بنویسید که مد و ضوح متوسط را برقرار کند، خطی عمودی را در وسط صفحه نمایش ترسیم کند، سپس یک خط افقی در صفحه نمایش بکشد.

۱۱- برنامه‌ای بنویسید که یک شماره رمز بفرم 123-45-6789 را از ورودی دریافت و پس از حذف خطوط تیره آن را

به شکل 123456789 در آورده و به ناحیه دیگری منتقل نمایید. از تعاریف داده زیر استفاده کنید.

|           |       |            |
|-----------|-------|------------|
| SS_AREA   | LABEL | BYTE       |
| SS_SIZE   | DB    | 12         |
| SS_ACTUAL | DB    | ?          |
| SS_DASHED | DB    | 12 DUP (?) |
| SS_NUM    | DB    | 9 DUP (?)  |

۱۲- برنامه‌ای بنویسید تا دو عدد هفت رقمی را در پاسخ به پیام "Enter the first number" و "Enter the second number" وارد کند. سپس آنها را با هم جمع نماید (با استفاده از AAA که در فصل ۳ معرفی شد) و حاصل جمع را با پیام "The total sum is" نمایش دهد.

۱۳- ASCIIZ چیست؟

۱۴- اگر هر بار چندین فایل باز باشد DOS چگونه آنها را از هم تفکیک می‌کند؟

۱۵- نوع فایل مربوط به هر یک از خصیصه‌های زیر را مشخص کنید.

(الف) 00 (ب) 02 (پ) 04

۱۶- وقتی تابع AH=3CH از INT21H برای باز کردن فایل بکار رود، DOS باز شدن موفق یک فایل باز شده قبلی را چگونه مشخص می‌نماید.

۱۷- صحیح یا غلط. در استفاده از تابع INT21H در DOS، برای دستیابی به یک فایل باز شده قبلی باید شناسه فایل را بدانیم.

۱۸- صحیح یا غلط. هنگام استفاده از تابع INT21H برای نوشتن در یک فایل، باید ناحیه داده‌ای در قطعه داده تعریف شود.

۱۹- در چه مقطعی مطمئن می‌شویم که داده در فایل نوشته شده و فهرست زیر بهنگام شده است؟

(الف) هنگام باز کردن فایل (ب) هنگام نوشتن در آن، (پ) هنگام بستن فایل.

۲۰- با استفاده از زبان اسمبلی و INT21H، فایلی ایجاد کنید، آن را "MYFILE" بنامید و عبارت زیر را در آن بنویسید.

"In the world of microprocessors and personal computers, the mother of all battles looms ahead when the 80x86 clone makers and IBM/Motorola Power PC RISC battle Intel".

پس از بستن فایل، فرمان TYPE را برای تخلیه آن روی صفحه تصویر بکار ببرید. ساینز فایل را با فرمان DIR تحقیق نمایید.

۲۱- چگونگی بکاربردن union REGS را برای برقراری AX = 9878، BH = 90H و CL = F4H نشان دهید.

۲۲- برای برقراری مکان نما بکمک int86 تابعی به C بنویسید. سپس بکمک آن مکان نما را در سطر 16 ستون 20 قرار داده و پیام "HELLO" را نمایش دهید.

۲۳- برنامه‌ای به C با اهداف زیر بنویسید.

(الف) صفحه نمایش را با int86 پاک کنید

(ب) مکان نما را در محدوده وسط صفحه قرار دهید. int86 را بکار ببرید.

(پ) مرتباً تاریخ و زمان را با قالب زیر نمایش دهید. intdos را بکار ببرید.

زمان: hr:min:sec

تاریخ: mon/day/yr

(ت) پیامی به شکل "Q" خروج را سؤال کند. توابع C را بکار ببرید.

(ث) وقتی کاربری Q را تایپ می‌نماید، باید ضمن نمایش زمان و تاریخ به DOS باز گردد. توابع C را بکار ببرید.

۲۴- برنامه نویسی union REGS را بفرم زیر بیان کرده است. آیا کار خواهد کرد؟

Union REGS inregs, outregs;

۲۵- دو نمونه برنامه بزبان C بنویسید که با افزایش یک واحد بشمارد و آن را با تأخیرهای یک ثانیه‌ای بین شمارش‌ها

نمایش دهد. اگر کلیدی فشرده شود شمارش متوقف شده و به DOS باز گردد. شمارش از صفر آغاز می شود.  
 (الف) از تابع kbhit استفاده کنید (ب) INT16H تابع 01 را بکار ببرید.  
 ۲۶- بخش (ب) برنامه فوق را با مقدار اولیه دلخواهی بجای صفر تکرار کنید.

## ✓ پاسخ به سؤالات مروری

### بخش ۱-۴: برنامه نویسی INT10H از BIOS

۱- اجرای i/o صفحه تصویر  
 ۲- 80 , 25; 00, 00 و 24 , 79  
 ۳-  
 MOV AH,06 ;SELECT CLEAR SCREEN FUNCTION  
 MOV AL,00 ;AH=0 TO SCROLL ENTIRE PAGE  
 MOV BH,07 ;BH=07 FOR NORMAL ATTRIBUTE  
 MOV CH,00 ;START AT ROW 00  
 MOV CL,00 ;START AT COLUMN 00  
 MOV DH,24 ;END AT ROW 24  
 MOV DL,79 ;END AT ROW 79  
 INT 10H ;INVOKE THE INTERRUPT

۴- مشخص می کند که مکان نما در سطر 5 ستون 34 است  
 ۵- اطلاعاتی درباره شدت پیش زمینه و پس زمینه منجمله چشمک زنی و یا متمایز  
 ۶- کاراکتر 10100111 -v  
 ۸- اولین باری که INT10H فرا خوانده شود، مکان نما را در 0000 می نشانند، در نوبت دوم کاراکتر "\*" را 80 مرتبه با  
 خصیصه های سفید روی سیاه چشمک زن نمایش می دهد.

### بخش ۲-۴ وقفه 21H در DOS

۱- 21H , 10H  
 ۲- سمت چپ ترین کد یک رشته را از صفحه کلید به بافر وارد می کند؛ کد سمت راست رشته را از بافر به مانیتور می فرستد.  
 ۳- در انتهای رشته علامت '\$' وجود دارد  
 ۴- 0AH , OFFSET MESSAGE1  
 ۵- وقتی که به یک '\$' در یک رشته برخورد شود، کامپیوتر نمایش رشته را متوقف می کند.  
 ۶- BELL EQU 07H  
 ۷- با استفاده از EQU در پاسخ 6، قطعه کد حاوی برنامه مقابل است.  
 ۸-  
 MOV AH,02  
 MOV DL,BELL  
 INT 21H  
 SS\_AREA LABEL BYTE  
 SS\_SIZE DB 12  
 SS\_ACTUAL DB ?  
 SS\_NUM DB 12 DUP (?)

### بخش ۳-۴: برنامه نویسی صفحه کلید INT16H

۱- AH = 01 تابع INT16H  
 ۲- پس از بازگشت از INT16H تابع AH = 01، اگر ZF = 1 باشد کلیدی فشرده نشده است، اگر ZF = 0 باشد کلیدی فشرده شده است.  
 ۳- اگر ZF = 0 باشد، از INT16H تابع AH = 0 استفاده می کنیم تا کاراکتر اسکی برای کلید فشرده بدست آید  
 ۴- INT21H برای فشردن کلید بوسیله کاربر به انتظار می ماند، INT16H صفحه کلید را پویش کرده و به برنامه اجازه اجرای دستورات را می دهد ضمن اینکه در انتظار فشردن کلید هم خواهد بود.  
 ۵-

AGAIN: MOV AH,02 ;USE FUNCTION AH=02 OF INT 21H  
 MOV DL,07 ;SOUND THE BELL BY SENDING OUT BELL CHAR  
 INT 21H  
 MOV AH,01 ;CHECK FOR KEY PRESS  
 INT 16H ;USING INT 16H  
 JZ AGAIN ;IF NO KEY PRESS KEEP SOUNDING THE BELL  
 MOV AH,0 ;TO GET THE CHARACTER  
 INT 16H ;WE MUST USE INT 16H ONE MORE TIME  
 CMP AL,'X' ;IS IT 'X'?  
 JE EXIT ;IF YES EXIT  
 JMP AGAIN ;NO. KEEP SOUNDING THE BELL  
 EXIT: MOV AH,4CH ;GO BACK TO DOS  
 INT 21H

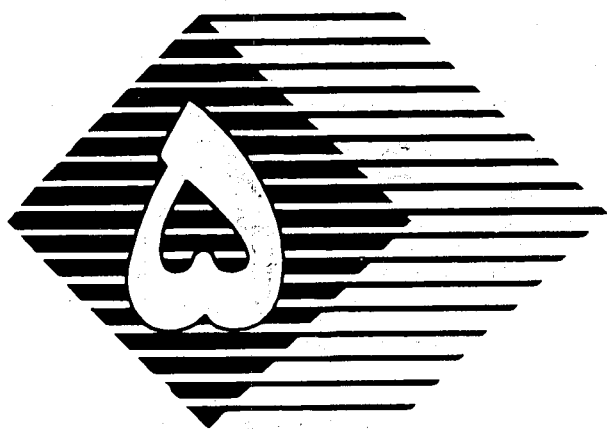
#### بخش ۴-۴ برنامه نویسی فایل دیسک

- ۱- غلط
- ۲- برنامه قدیمی در فایل کاملاً پاک می شود.
- ۳- در بستن فایل هر اطلاعاتی از بافر به فایل منتقل می گردد، ولی اگر قبل از بستن به کار پایان بدهید اطلاعات ممکن است در بافر از بین برود.
- ۴- یک رشته ASCIIZ نیست زیرا در ASCII Zero (null) پایان نمی یابد.
- ۵- یک شناسه فایل باید 16 بیتی باشد (از رهنمون DW استفاده کنید).

#### بخش ۴-۵ برنامه نویسی وقفه با C

- ۱- صحیح
- ۲- غلط، فقط برای INT21H
- ۳- 3,2
- ۴- غلط
- ۵- بله، می توانیم هر اسمی را بکار ببریم. دیگر نام های معمول عبارتند از inregs، outregs و r1، r2.
- ۶- int86x
- ۷- چهار: INT#، &regin، &regout، regseg
- ۸- صحیح





## ماکروها و ماوس

### اهداف

در پایان این فصل قادر خواهید بود:

- « مزیت استفاده از ماکروها را لیست کنید.
- « ماکروها را با برنامه نویسی رهنمون‌های ماکرو MACRO و ENDM تعریف نمایید.
- « دستورات زبان اسمبلی را برای فراخوانی ماکروها بکار ببرید.
- « چگونگی گسترش ماکروها را با اسمبلر بیان کنید.
- « از رهنمون‌های XALL، SALL و LALL برای کنترل گسترش ماکرو در فایل لیست استفاده کنید.
- « رهنمون LOCAL را برای تعریف متغیرهای محلی در ماکرو بکار ببرید.
- « رهنمون‌های INCLUDE را برای لحاظ ماکروها از فایل دیگری به یک برنامه استفاده نمایید.
- « INT33H را برای کنترل تابع ماوس در مدهای متن و گرافیک استفاده کنید.
- « برای استقرار ماوس و یا بدست آوردن موقعیت آن برنامه‌ای بزبان اسمبلی بنویسید.
- « توابع INT33H را برای اخذ اطلاعات مربوط به فشردن و آزاد سازی ماوس بکار ببرید.
- « با تفکیک ناحیه تعریف شده و تعیین مرز، مکان‌نمای ماوس را محدود کنید.

در این فصل مفهوم ماکرو و کاربرد آن را در برنامه‌نویسی اسمبلی می‌شکافیم. قالب و کاربرد ماکروها تعریف شده و مثال‌های بسیاری از کاربرد آنها ارائه شده است. بعلاوه، این فصل استفاده از INT33H را برای کنترل عملیات ماوس در برنامه‌نویسی زبان اسمبلی نشان می‌دهد.

## بخش ۱-۵: ماکرو چیست و چگونه بکار می‌رود

در بعضی از برنامه‌های زبان اسمبلی مواردی وجود دارد که در آن گروهی از دستورالعمل‌ها وظیفه خاصی را مکرراً تکرار می‌نمایند. مثلاً در فصل ۴ دیدیم INT21H تابع 09H رشته داده‌ها را مرتباً نمایش می‌دهد و یا با تابع AH داده‌های کلید زده شده را مکرراً در یک برنامه وارد می‌کند. بنابراین معقول نیست هر بار که لازم شد آنها را بنویسیم. در نتیجه برای کاهش زمان نوشتن این کدها و حذف احتمال خطا، مفهوم ماکرو ارائه شده است. ماکرو به برنامه‌نویس اجازه می‌دهد تا وظیفه مزبور (مجموعه کدها برای انجام کاری خاص) را فقط یکبار بنویسد و هر زمان و هر کجا که لازم شد آن را فراخواند و اجرا نماید.

### تعریف MACRO

تعریف هر ماکرو سه بخش دارد که در زیر آمده است:

```
name MACRO dummy1,dummy2,....,dummyN
.....
.....
ENDM
```

رهنمون MACRO شروع تعریف ماکرو و رهنمون ENDM انتهای آن را مشخص می‌نماید. آنچه در فاصله بین رهنمون‌های MACRO و ENDM قرار می‌گیرد بدنه ماکرو نامیده می‌شود. نام باید منحصر به فرد بوده و از مقررات متعارف نام‌گذاری در زبان اسمبلی پیروی کند. عناصر ساختگی، نام‌ها، پارامترها و یا حتی ثبات‌هایی هستند که در بدنه ماکرو ذکر می‌شوند. پس از نوشتن ماکرو، می‌توان آن را فراخواند و برای پارامترهای ساختگی مقادیر مناسبی تعیین کرد. نمایش رشته‌ای از داده بکمک تابع 09 بطور گسترده‌ای بکار می‌رود. در زیر ماکرو برای این سرویس آورده شده است.

```
STRING MACRO DATA1
MOV AH,09
MOV DX,OFFSET DATA1
INT 21H
ENDM
```

آنچه در بالا آمد تعریف ماکرو است. توجه داشته باشید که DATA1 به عنوان آرگومان ساختگی در بدنه ماکرو ذکر شده است. در مثال زیر فرض کنید که پیامی قبلاً در قطعه داده طبق آنچه در زیر دیده می‌شود قرار داشته باشد. در قطعه کد، ماکرو قابل فراخوانی بوسیله نام و داده واقعی کاربر فراخوانی می‌شود.

```
MESSAGE1 DB 'What is your name?','$'
...
STRING MESSAGE1
```

دستورالعمل "STRING MESSAGE" ماکرو را فراخوانی می‌کند. اسمبلر ماکرو را با تهیه برنامه زیر در

فایل LST توسعه می‌دهد.

```
1 MOV AH,09
1 MOV DX,OFFSET MESSAGE1
1 INT 21H
```

(1) به این معنی است که کد از ماکرو است. در انواع قبلی MASM، (+) مفهوم مشابهی داشت.

### توضیحات در ماکرو

اکنون سؤال این است که: آیا ماکرو می تواند توضیح داشته باشد؟ پاسخ مثبت است، ولی معمولاً توضیحات فشرده بیان می شوند و سبب می گردند تا اسمبلر فقط خطوط مولد کد را نشان دهد. اساساً دو نوع توضیح در ماکرو وجود دارد: لیست شدنی و لیست نشدنی. اگر توضیحات مذکور مثل برنامه نویسی زبان اسمبلی بدنبال نقطه ویرگول (;) آورده شود در فایل "lst" نشان داده می شوند، ولی اگر توضیحات دو عدد نقطه ویرگول (;;) گذاشته شود، هنگام اسمبل کردن در فایل "lst" ظاهر نمی شود. سه رهنمون هم طراحی شده اند تا برنامه های ماکرو راحت تر خوانده شوند، به این معنی که آنها فقط "lst" را متأثر کنند و بر روی فایل های "obj" و "exe" تأثیری ندارند. این رهنمون ها در زیر توضیح داده شده اند.

**LALL** . (List ALL). این رهنمون همه دستور العمل ها و توضیحاتی که بدنبال یک نقطه ویرگول آمده است را در فایل "lst" لیست می کند. توضیحات که پس از دو نقطه ویرگول آمده باشد بهیچ وجه قابل لیست شدن در فایل "lst" نیست.

**SALL** . (Suppress ALL). این رهنمون برای کوتاه تر کردن فایل لیست بکار رفته هنگام خواندن هم ساده تر است. این رهنمون بدنه ماکرو و توضیحات را فشرده می نماید. این عمل خصوصاً هنگامی مفید است که ماکرو در یک برنامه چندین بار فراخوانی شود و دیدن لیست آن در هر نوبت لزومی نداشته باشد. باید تأکید کرد که استفاده از SALL، هیچ کدی را از فایل obj حذف نمی نماید. با این وجود تنها در فایل "lst"، لیست را تحت تأثیر قرار می دهد.

**XALL** . (Executable ALL). این رهنمون که رهنمون لیست بندی پیش فرض است تنها در آن بخش از ماکرو بکار می رود که کدها را تولید می نماید.

مثال ۱-۵. برنامه ای به ماکرو برای استقرار محل مکان نما، نمایش رشته ای و پاک کردن صفحه نمایش بنویسید.  
حل:

```
CURSOR MACRO ROW,COLUMN
;THIS MACRO SETS THE CURSOR LOCATION TO ROW,COLUMN
;;USING BIOS INT 10H FUNCTION 02
 MOV AH,02 ;SET CURSOR FUNCTION
 MOV BH,00 ;PAGE 00
 MOV DH,ROW ;ROW POSITION
 MOV DL,COLUMN ;COLUMN POSITION
 INT 10H ;INVOKE THE INTERRUPT
ENDM

DISPLAY MACRO STRING
;THIS MACRO DISPLAYS A STRING OF DATA
;;DX = ADDRESS OF STRING. USES FUNCTION 09 INT 21H.
 MOV AH,09 ;DISPLAY STRING FUNCTION
 MOV DX,OFFSET STRING ;DX = OFFSET ADDRESS OF DATA
 INT 21H ;INVOKE THE INTERRUPT
ENDM

CLEARSCR MACRO
;THIS MACRO CLEARS THE SCREEN
;;USING OPTION 06 OF INT 10H
 MOV AX,0600H ;SCROLL SCREEN FUNCTION
 MOV BH,07 ;NORMAL ATTRIBUTE
 MOV CX,0 ;FROM ROW=00,COLUMN=00
 MOV DX,184FH ;TO ROW=18H,COLUMN=4FH
 INT 10H ;INVOKE THE INTERRUPT
ENDM
```

بخاطر داشته باشید که توضیحاتی که در جلو آن ها ";;" آمده است در فایل "lst"، آنطور که در برنامه ۱-۵ دیده می شود، لیست نمی گردد.

مثال ۱-۵ تعریف ماکرو را نشان می دهد.

با استفاده از تعریف ماکرو در مثال ۵-۱، برنامه‌ای بنویسید که صفحه نمایش را پاک کند و سپس در هر یک از مکان‌های صفحه نمایش پیام‌های مربوطه را نمایش دهد.

در سطر ۲ ستون ۴ "My name" در سطر ۱۲ ستون ۴۴ "what is"

در سطر ۷ ستون ۲۴ "is Joe" در سطر ۱۹ ستون ۶۴ "Your name"

TITLE PROG5-1  
PAGE 60,132

CLEARSCR MACRO

THIS MACRO CLEARS THE SCREEN

USING OPTION 06 OF INT 10H

```
MOV AX,0600H ;SCROLL SCREEN FUNCTION
MOV BH,07 ;NORMAL ATTRIBUTE
MOV CX,0 ;FROM ROW=00,COLUMN=00
MOV DX,184FH ;TO ROW=18H,COLUMN=4FH
INT 10H ;INVOKE THE INTERRUPT
ENDM
```

DISPLAY MACRO STRING

THIS MACRO DISPLAYS A STRING OF DATA

DX = ADDRESS OF STRING. USES FUNCTION 09 INT 21H.

```
MOV AH,09 ;DISPLAY STRING FUNCTION
MOV DX,OFFSET STRING ;DX = OFFSET ADDRESS OF DATA
INT 21H ;INVOKE THE INTERRUPT
ENDM
```

CURSOR MACRO ROW,COLUMN

THIS MACRO SETS THE CURSOR LOCATION TO ROW,COLUMN

USING BIOS INT 10H FUNCTION 02

```
MOV AH,02 ;SET CURSOR FUNCTION
MOV BH,00 ;PAGE 00
MOV DH,ROW ;ROW POSITION
MOV DL,COLUMN ;COLUMN POSITION
INT 10H ;INVOKE THE INTERRUPT
ENDM
```

STSEG SEGMENT

DB 64 DUP (?)

STSEG ENDS

DTSEG SEGMENT

```
MESSAGE1 DB 'My name ','$'
MESSAGE2 DB 'is Joe ','$'
MESSAGE3 DB 'What is ','$'
MESSAGE4 DB 'your name? ','$'
DTSEG ENDS
```

CDSEG SEGMENT

MAIN PROC FAR  
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG

```
MOV AX,DTSEG
MOV DS,AX
```

```
.LALL ;LIST ALL
CLEARSCR ;INVOKE CLEAR SCREEN MACRO
CURSOR 2,4 ;SET CURSOR TO ROW2,COL 2
DISPLAY MESSAGE1 ;INVOKE DISPLAY MACRO
.XALL ;LIST ALL EXECUTABLE
CURSOR 7,24 ;SET CURSOR TO ROW 7,COL 24
DISPLAY MESSAGE2 ;INVOKE DISPLAY MACRO
.SALL ;SUPPRESS ALL
CURSOR 12,44 ;SET CURSOR TO ROW 12,COL 44
DISPLAY MESSAGE3 ;INVOKE DISPLAY MACRO
CURSOR 19,64 ;SET CURSOR TO ROW 19,COL 64
DISPLAY MESSAGE4 ;INVOKE DISPLAY MACRO
```

```
MOV AH,4CH
INT 21H ;GO BACK TO DOS
```

MAIN ENDP  
CDSEG ENDS

END MAIN

## تحلیل برنامه ۱-۵

فایل های ".asm" و ".lst" را مقایسه کنید تا کاربرد LALL، XALL و SALL را ملاحظه نمایید. رهنمون LALL برای هر ماکرو بکار رفته و آنگاه XALL برای دو ماکرو بکار برده شده است. از آن پس همه آنها فشرده شده اند.

Microsoft (R) Macro Assembler Version 5.10  
PROG5-1

1/13/92 00:17:15  
Page 1-1

TITLE PROG5-1  
PAGE 60,132

```

CLEARSCR MACRO
;THIS MACRO CLEARS THE SCREEN
;;USING OPTION 06 OF INT 10H
 MOV AX,0600H ;SCROLL SCREEN FUNCTION
 MOV BH,07 ;NORMAL ATTRIBUTE
 MOV CX,0 ;FROM ROW=00,COLUMN=00
 MOV DX,184FH ;TO ROW=18H,COLUMN=4FH
 INT 10H ;INVOKE THE INTERRUPT
ENDM

```

```

DISPLAY MACRO STRING
;THIS MACRO DISPLAYS A STRING OF DATA
;;DX = ADDRESS OF STRING. USES FUNCTION 09 INT 21H.
 MOV AH,09 ;DISPLAY STRING FUNCTION
 MOV DX,OFFSET STRING ;DX = OFFSET ADDRESS OF DATA
 INT 21H ;INVOKE THE INTERRUPT
ENDM

```

```

CURSOR MACRO ROW,COLUMN
;THIS MACRO SETS THE CURSOR LOCATION TO ROW,COLUMN
;;USING BIOS INT 10H FUNCTION 02
 MOV AH,02 ;SET CURSOR FUNCTION
 MOV BH,00 ;PAGE 00
 MOV DH,ROW ;ROW POSITION
 MOV DL,COLUMN ;COLUMN POSITION
 INT 10H ;INVOKE THE INTERRUPT
ENDM

```

```

0000 STSEG SEGMENT
0000 0040 [DB 64 DUP(?)
 ??
]

```

0040 STSEG ENDS

```

0000 DTSEG SEGMENT
0000 4D 79 20 6E 61 6D MESSAGE1 DB 'My name ','$'
 65 20 24
0008 69 73 20 4A 6F 65 MESSAGE2 DB 'is Joe ','$'
000E 57 68 61 74 20 69 MESSAGE3 DB 'What is ','$'
 73 20 24
0016 79 6F 75 72 20 6E MESSAGE4 DB 'your name? ','$'
 61 6D 65 3F 24
0020 DTSEG ENDS

```

```

0000 CDSEG SEGMENT
0000 MAIN PROC FAR
 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
0000 B8 ---- R MOV AX,DTSEG
0003 8E D8 MOV DS,AX
 LALL ;LIST ALL
 CLEARSCR ;INVOKE CLEAR SCREEN MACRO

```

فایل لیست برای برنامه ۱-۵

```

1;THIS MACRO CLEARS THE SCREEN
1;
0005 B8 0600 1 MOV AX,0600H ;SCROLL SCREEN FUNCTION
0008 B7 07 1 MOV BH,07 ;NORMAL ATTRIBUTE
000A B9 0000 1 MOV CX,0 ;FROM ROW=00,COLUMN=00
000D BA 184F 1 MOV DX,184FH ;TO ROW=18H,COLUMN=4FH
0010 CD 10 1 INT 10H ;INVOKE THE INTERRUPT
CURSOR 2,4;CURSOR MACRO WILL SET CURSOR TO 2, 2
1;THIS MACRO SETS THE CURSOR LOCATION
1;
0012 B4 02 1 MOV AH,02 ;SET CURSOR FUNCTION
0014 B7 00 1 MOV BH,00 ;PAGE 00
0016 B6 02 1 MOV DH,2 ;ROW POSITION
0018 B2 04 1 MOV DL,4 ;COLUMN POSITION
001A CD 10 1 INT 10H ;INVOKE THE INTERRUPT
DISPLAY MESSAGE1 ;INVOKE DISPLAY MACRO
1;THIS MACRO DISPLAYS A STRING OF DATA
1;
001C B4 09 1 MOV AH,09 ;DISPLAY STRING FUNCTION
001E BA 0000 R1 MOV DX,OFFSET MESSAGE1 ;DX = OFFSET ADDRESS OF DATA
0021 CD 21 1 INT 21H ;INVOKE THE INTERRUPT
XALL ;LIST ALL EXECUTABLE
CURSOR 7,24 ;SET CURSOR TO ROW=7,COL= 24
0023 B4 02 1 MOV AH,02 ;SET CURSOR FUNCTION
0025 B7 00 1 MOV BH,00 ;PAGE 00
0027 B6 07 1 MOV DH,7 ;ROW POSITION
0029 B2 18 1 MOV DL,24 ;COLUMN POSITION
002B CD 10 1 INT 10H ;INVOKE THE INTERRUPT
DISPLAY MESSAGE2 ;INVOKE DISPLAY MACRO
002D B4 09 1 MOV AH,09 ;DISPLAY STRING FUNCTION
002F BA 0008 R1 MOV DX,OFFSET MESSAGE2 ;DX = OFFSET ADDRESS OF DATA
0032 CD 21 1 INT 21H ;INVOKE THE INTERRUPT
SALL ;SUPPRESS ALL
CURSOR 12,44 ;SET CURSOR TO ROW=12,COL=44
DISPLAY MESSAGE3 ;INVOKE DISPLAY MACRO
CURSOR 19,64 ;SET CURSOR TO ROW=19,COL=64
DISPLAY MESSAGE4 ;INVOKE DISPLAY MACRO
0056 B4 4C MOV AH,4CH
0058 CD 21 INT 21H ;GO BACK TO DOS
005A MAIN ENDP
005A CDSEG ENDS
END MAIN

```

ادامه فایل لیست برای برنامه ۱-۵

### رهنمون LOCAL و کاربرد آن در ماکرو

تاکنون در بحث ماکرو، مثال‌هایی انتخاب شدند که فاقد برچسب یا نام در بدنه ماکرو بودند. اگر یک ماکرو بیش از یکبار در یک برنامه گسترش یابد و برچسبی در میدان برچسب بدنه ماکرو وجود داشته باشد، به آنها برچسب‌های LOCAL گویند. در غیراینصورت، به هنگام مواجه با برچسب در دو یا سه محل دیگر، خطایی بوسیله اسمبلر اعلام خواهد شد. قوانین زیر باید در بدنه ماکرو مشاهده شود.

- ۱- همه برچسب‌ها در میدان برچسب باید بصورت LOCAL، اعلام شوند.
- ۲- رهنمون LOCAL باید بلافاصله پس از رهنمون MACRO قرار گیرد. به بیان دیگر باید آن را حتی قبل از توضیحات و بدنه ماکرو قرار داد. در غیر اینصورت اسمبلر اعلام خطا می‌کند.
- ۳- رهنمون LOCAL می‌تواند برای اعلام همزمان همه نام‌ها و برچسب‌ها مطابق زیر بکار رود.

LOCAL name1,name2,name3

یا یکی یکی قابل نوشتن هستند:

```

- LOCAL name1
 LOCAL name2
 LOCAL name3

```

برای تشریح این نکات به مثال ۲-۵ توجه کنید.

مثال ۵-۲. یک ماکرو بنویسید تا دو کلمه را با تکرار جمع، درهم ضرب کند. سپس نتیجه را ذخیره نمایید.

حل:

ماکرو زیر می‌تواند به هر تعداد در یک برنامه تکرار شود زیرا برچسب "back" بصورت LOCAL اعلام شده است.

```
MULTIPLY MACRO VALUE1, VALUE2, RESULT
 LOCAL BACK
 ; THIS MACRO COMPUTES RESULT = VALUE1 X VALUE2
 ; BY REPEATED ADDITION
 ; VALUE1 AND VALUE2 ARE WORD OPERANDS; RESULT IS A DOUBLEWORD
 MOV BX,VALUE1 ;BX=MULTIPLIER
 MOV CX,VALUE2 ;CX=MULTIPLICAND
 SUB AX,AX ;CLEAR AX
 MOV DX,AX ;CLEAR DX
BACK: ADD AX,BX ;ADD BX TO AX
 ADC DX,00 ;ADD CARRIES IF THERE IS ONE
 LOOP BACK ;CONTINUE UNTIL CX=0
 MOV RESULT,AX ;SAVE THE LOW WORD
 MOV RESULT+2,DX ;SAVE THE HIGH WORD
ENDM
```

با استفاده از تعریف ماکرو در مثال ۵-۲، برنامه‌ای برای ضرب اعداد زیر بنویسید.

300 × 400 (۳)      2500 × 500 (۲)      2000 × 500 (۱)

TITLE PROG5-2  
PAGE 60,132

```
MULTIPLY MACRO VALUE1, VALUE2, RESULT
 LOCAL BACK
 ; THIS MACRO COMPUTES RESULT = VALUE1 X VALUE2
 ; BY REPEATED ADDITION
 ; VALUE1 AND VALUE2 ARE WORD OPERANDS; RESULT IS A DOUBLEWORD
 MOV BX,VALUE1 ;BX=MULTIPLIER
 MOV CX,VALUE2 ;CX=MULTIPLICAND
 SUB AX,AX ;CLEAR AX
 MOV DX,AX ;CLEAR DX
BACK: ADD AX,BX ;ADD BX TO AX
 ADC DX,00 ;ADD CARRIES IF THERE IS ONE
 LOOP BACK ;CONTINUE UNTIL CX=0
 MOV RESULT,AX ;SAVE THE LOW WORD
 MOV RESULT+2,DX ;SAVE THE HIGH WORD
ENDM
```

```
STSEG SEGMENT
 DB 64 DUP (?)
STSEG ENDS
```

```
DTSEG SEGMENT
RESULT1 DW 2 DUP (0)
RESULT2 DW 2 DUP (0)
RESULT3 DW 2 DUP (0)
DTSEG ENDS
```

```
CDSEG SEGMENT
MAIN PROC FAR
 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
 MOV AX,DTSEG
 MOV DS,AX
 MULTIPLY 2000,500,RESULT1
 MULTIPLY 2500,500,RESULT2
 MULTIPLY 300,400,RESULT3
 MOV AH,4CH
 INT 21H ;GO BACK TO DOS
MAIN ENDP
CDSEG ENDS
 END
```

برنامه ۵-۲

توجه کنید که برچسب "BACK" در مثال ۲-۵ بصورت LOCAL و درست پس از رهنمون MACRO آمده است. تعریف این برچسب در هر جای دیگر موجب خطا می شود. کاربرد رهنمون LOCAL اجازه می دهد تا اسمبلر هنگام مواجهه هر بار آنها را جداگانه تعریف کند. فایل لیست زیر نشان می دهد که هر بار ماکرو برای اولین بار گسترش یابد، فایل لیست "??0000" دارد. در نوبت دوم این مقدار "??0001" و در بار سوم "??0002" در محل برچسب "BACK" است و این بدان معنی است که "BACK" محلی (Local) است. برای روشن تر شدن این مفهوم، مثال ۲-۵ را بدون رهنمون LOCAL تکرار کنید تا ببینید اسمبلر چگونه خطا تولید می کند.

Microsoft (R) Macro Assembler Version 5.10  
PROG5-2

1/13/92 00:33:14

Page 1-1

```

TITLE PROG5-2
PAGE 60,132

MULTIPLY MACRO VALUE1, VALUE2, RESULT
 LOCAL BACK
;THIS MACRO COMPUTES RESULT = VALUE1 X VALUE2
;BY REPEATED ADDITION
;VALUE1 AND VALUE2 ARE WORD OPERANDS; RESULT IS A DOUBLE
 MOV BX,VALUE1 ;BX=MULTIPLIER
 MOV CX,VALUE2 ;CX=MULTIPLICAND
 SUB AX,AX ;CLEAR AX
 MOV DX,AX ;CLEAR DX
BACK: ADD AX,BX ;ADD BX TO AX
 ADC DX,00 ;ADD CARRIES IF THERE IS ONE
 LOOP BACK ;CONTINUE UNTIL CX=0
 MOV RESULT,AX ;SAVE THE LOW WORD
 MOV RESULT+2,DX ;SAVE THE HIGH WORD
 ENDM

0000 STSEG SEGMENT
0000 0040 [DB 64 DUP(?)
 ??
]

0040 STSEG ENDS

0000 DTSEG SEGMENT
0000 0002 [RESULT1 DW 2 DUP (0)
 0000
]

0004 0002 [RESULT2 DW 2 DUP (0)
 0000
]

0008 0002 [RESULT3 DW 2 DUP (0)
 0000
]

000C DTSEG ENDS

0000 CDSEG SEGMENT
0000 MAIN PROC FAR
 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
 MOV AX,DTSEG
 MOV DS,AX
 MULTIPLY 2000,500,RESULT1
0005 BB 07D0 1 MOV BX,2000 ;BX=MULTIPLIER
0008 B9 01F4 1 MOV CX,500 ;CX=MULTIPLICAND
000B 2B C0 1 SUB AX,AX ;CLEAR AX
000D 8B D0 1 MOV DX,AX ;CLEAR DX
000F 03 C3 1 ??0000: ADD AX,BX ;ADD BX TO AX
0011 83 D2 00 1 ADC DX,00 ;ADD CARRIES IF THERE IS ONE
0014 E2 F9 1 LOOP ??0000 ;CONTINUE UNTIL CX=0
0016 A3 0000 R 1 MOV RESULT1,AX ;SAVE THE LOW WORD
0019 89 16 0002 R 1 MOV RESULT1+2,DX ;SAVE THE HIGH WORD

```

فایل لیست برای برنامه ۲-۵



```

MULTIPLY 2500,500,RESULT2
001D BB 09C4 1 MOV BX,2500 ;BX=MULTIPLIER
0020 B9 01F4 1 MOV CX,500 ;CX=MULTIPLICAND
0023 2B C0 1 SUB AX,AX ;CLEAR AX
0025 8B D0 1 MOV DX,AX ;CLEAR DX
0027 03 C3 1??0001: ADD AX,BX ;ADD BX TO AX
0029 83 D2 00 1 ADC DX,00 ;ADD CARRIES IF THERE IS ONE
002C E2 F9 1 LOOP ??0001 ;CONTINUE UNTIL CX=0
002E A3 0004 R 1 MOV RESULT2,AX ;SAVE THE LOW WORD
0031 89 16 0006 R 1 MOV RESULT2+2,DX ;SAVE THE HIGH WORD
MULTIPLY 300,400,RESULT3
0035 BB 012C 1 MOV BX,300 ;BX=MULTIPLIER
0038 B9 0190 1 MOV CX,400 ;CX=MULTIPLICAND
003B 2B C0 1 SUB AX,AX ;CLEAR AX
003D 8B D0 1 MOV DX,AX ;CLEAR DX
003F 03 C3 1??0002: ADD AX,BX ;ADD BX TO AX
0041 83 D2 00 1 ADC DX,00 ;ADD CARRIES IF THERE IS ONE
0044 E2 F9 1 LOOP ??0002 ;CONTINUE UNTIL CX=0
0046 A3 0008 R 1 MOV RESULT3,AX ;SAVE THE LOW WORD
0049 89 16 000A R 1 MOV RESULT3+2,DX ;SAVE THE HIGH WORD
004D B4 4C MOV AH,4CH
004F CD 21 INT 21H ;GO BACK TO DOS
0051 MAIN ENDP
0051 CDSEG ENDS
 END MAIN

```

فایل لیست برای برنامه ۲-۵ (ادامه صفحه قبل)

## رهنمون INCLUDE

فرض کنید که چندین ماکرو وجود داشته و در هر برنامه هم بکار رفته باشند. آیا لازم است هر بار نوشته شوند؟ اگر مفهوم رهنمون INCLUDE را بدانیم جواب منفی خواهد بود. رهنمون INCLUDE به برنامه نویس اجازه می دهد تا ماکروها را نوشته و آنها را در فایل ذخیره کرده و بعداً به هر فایلی وارد کند. مثلاً، فرض کنید که ماکرو زیر نوشته شده و تحت نام "MYMACRO 1 MAC" ذخیره شود.

```

CLEARSCR MACRO ;the clear screen macro:
 MOV AX,0600H
 MOV BH,07
 MOV CX,0000
 MOV DX,184FH
 INT 10H
 ENDM

DISPLAY MACRO STRING ;the string macro
 MOV AH,09
 MOV DX,OFFSET STRING
 INT 21H
 ENDM

REGSAVE MACRO ;this macro saves all the registers
 PUSH AX
 PUSH BX
 PUSH CX
 PUSH DX
 PUSH DI
 PUSH SI
 PUSH BP
 PUSHF
 ENDM

```

```

REGRESTO MACRO ;this macro restores all the registers
 POPF
 POP BP
 POP SI
 POP DI
 POP DX
 POP CX
 POP BX
 POP AX
 ENDM

```

با این فرض که این ماکروها تحت نام فایل "MYMACRO1.MAC" در دیسک ذخیره شوند، رهنمون INCLUDE برای آوردن آن به هر فایل ".asm" می تواند مورد استفاده قرار گیرد. آنگاه برنامه می تواند هر یک از آنها را به هر تعداد که لازم باشد فرا بخواند. وقتی فایلی شامل همه ماکروها باشد ماکروها در آغاز فایل ".lst" لیست شده و پس از گسترش، بخشی از برنامه خواهند بود.

برنامه ۵-۳ شامل ماکروها برای پاک کردن، استقرار مکان نما و نمایش رشته هاست. همه این ماکروها تحت نام فایل "MY MACRO2.MAC" ذخیره شده اند. نوع "asm" و ".lst" برنامه که از ماکروهای پاک کردن صفحه نمایش و نمایش رشته برای نشان دادن "This is a test of macro concepts" بکار رفته اند در صفحات بعد آورده شده است. توجه کنید که در فایل لیست برنامه ۵-۳، حرف "C" در جلو خطوط بیانگر این است که آن فایل کپی فایل دیگر است که در فایل فعلی منظور شده است.

```

TITLE PROG5-3
PAGE 60,132
INCLUDE MYMACRO2.MAC

STSEG SEGMENT
 DB 64 DUP (?)
STSEG ENDS

DTSEG SEGMENT
MESSAGE1 DB 'This is a test of macro concepts','$'
DTSEG ENDS

CDSEG SEGMENT
MAIN PROC FAR
 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
 MOV AX,DTSEG
 MOV DS,AX
 CLEARSCR ;INVOKE CLEAR SCREEN MACRO
 DISPLAY MESSAGE1 ;INVOKE DISPLAY MACRO
 MOV AH,4CH
 INT 21H ;GO BACK TO DOS
MAIN ENDP
CDSEG ENDS
 END MAIN

```

برنامه ۵-۳

### سؤالات مروری

- ۱- مزایای برنامه نویسی ماکرو را بحث کنید.
- ۲- سه بخش یک ماکرو را لیست نمایید.
- ۳- تعریف ماکرو، فراخوانی ماکرو و گسترش آن را توضیح دهید.
- ۴- صحیح یا غلط. یک برچسب در یک ماکرو بطور خودکار بوسیله اسمبلر به عنوان محلی تلقی می شود.
- ۵- صحیح یا غلط. در فایل لیست برای برنامه ۵-۳، "C" در ابتدای یک خط به معنی این است که خط یک توضیح می باشد.

```

 TITLE PROG5-3
 PAGE 60,132
 INCLUDE MYMACRO2.MAC
 MYMACRO2 (MAC) FOR PROGRAM5-3
C:-----
C CURSOR MACRO ROW,COLUMN
C:THIS MACRO SETS THE CURSOR LOCATION AT ROW,COLUMN
C:;USING BIOS INT 10H FUNCTION 02
C MOV AH,02 ;SET CURSOR FUNCTION
C MOV BH,00 ;PAGE 00
C MOV DH,ROW ;ROW POSITION
C MOV DL,COLUMN ;COLUMN POSITION
C INT 10H ;INVOKE THE INTERRUPT
C ENDM
C:-----
C DISPLAY MACRO STRING
C:THIS MACRO DISPLAYS A STRING OF DATA
C:;DX = ADDRESS OF STRING. USES FUNCTION 09 INT 21H.
C MOV AH,09 ;DISPLAY STRING FUNCTION
C MOV DX,OFFSET STRING ;DX = OFFSET ADDRESS OF DATA
C INT 21H ;INVOKE THE INTERRUPT
C ENDM
C:-----
C CLEARSCR MACRO
C:THIS MACRO CLEARS THE SCREEN
C:;USING OPTION 06 OF INT 10H
C MOV AX,0600H ;SCROLL SCREEN FUNCTION
C MOV BH,07 ;NORMAL ATTRIBUTE
C MOV CX,0 ;FROM ROW=00,COLUMN=00
C MOV DX,184FH ;TO ROW=18H,COLUMN=4FH
C INT 10H ;INVOKE THE INTERRUPT
C ENDM
C:-----
0000 STSEG SEGMENT
0000 0040 [DB 64 DUP (?)
 ??
]

0040 STSEG ENDS
0000 DTSEG SEGMENT
0000 54 68 69 73 20 69 MESSAGE1 DB 'This is a test of macro concepts','$'
 73 20 61 20 74 65
 73 74 20 6F 66 20
 6D 61 63 72 6F 20
 63 6F 6E 63 65 70
 74 73 24
0021 DTSEG ENDS
0000 CDSEG SEGMENT
0000 MAIN PROC FAR
 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
0000 B8 ---- R MOV AX,DTSEG
0003 8E D8 MOV DS,AX
 CLEARSCR ;INVOKE CLEAR SCREEN MACRO
0005 B8 0600 1 MOV AX,0600H ;SCROLL SCREEN FUNCTION

```

فایل لیست برای برنامه ۵-۳ (ادامه در صفحه بعد)

```
0008 B7 07 1 MOV BH,07 ;NORMAL ATTRIBUTE
000A B9 0000 1 MOV CX,0 ;FROM ROW=00,COLUMN=00
000D BA 184F 1 MOV DX,184FH ;TO ROW=18H,COLUMN=4FH
0010 CD 10 1 INT 10H ;INVOKE THE INTERRUPT
 DISPLAY MESSAGE1
 ;INVOKE DISPLAY MACRO
0012 B4 09 1 MOV AH,09 ;DISPLAY STRING FUNCTION
0014 BA 0000 R 1 MOV DX,OFFSET MESSAGE1 ;DX =OFFSET ADDRESS OF DATA
0017 CD 21 1 INT 21H ;INVOKE THE INTERRUPT
0019 B4 4C 1 MOV AH,4CH
001B CD 21 1 INT 21H ;GO BACK TO DOS
001D MAIN ENDP
 CDSEG ENDS
001D END MAIN
```

فایل لیست برای برنامه ۳-۵ (ادامه صفحه قبل)

## بخش ۲-۵: برنامه نویسی ماوس با وقفه 33H

پس از صفحه کلید، ماوس یکی از پر مصرف ترین وسایل ورودی است. این بخش چگونگی استفاده از INT33H را توضیح می دهد.

### INT33H

IBM PC و DOS اولیه ماوس را پشتیبانی نمی کردند. به این دلیل، وقفه INT33H بخشی از BIOS یا DOS نیست. این برخلاف INT21H و INT10H است که بترتیب وقفه های DOS و BIOS هستند. INT33H بخشی از نرم افزار راه انداز است که به هنگام بوت PC نصب می شود.

### تشخیص وجود ماوس

در حالی که PC های جدید همراه با یک ماوس و راه انداز آن که قبلاً بوسیله سازنده PC در آن نصب شده ارائه می شوند، بسیاری از PC های نسل قدیمی تر فاقد ماوس می باشند. بنابراین اولین وظیفه INT33H تشخیص وجود ماوس و تعداد دکمه های آن است. این کار به عهده تابع  $AX = 0$  از INT33H است. پس از بازگشت از INT33H، اگر  $AX = 0$  باشد آنگاه ماوس پشتیبانی نمی شود. ولی اگر  $AX = FFFFH$  باشد ماوس پشتیبانی شده است. تعداد دکمه ها در BX است. گرچه بسیاری از ماوس ها دو دکمه چپ و راست دارند، ولی برخی دارای دکمه وسطی هم هستند. به برنامه زیر توجه کنید.

```
MOV AX,0 ;mouse initialization option
INT 33H
CMP AX,0 ;check AX contents upon return from INT 33H
JE EXIT ;exit if AX=0 since no mouse available
MOV M_BUTTON,BX ;mouse is there, save the number of buttons
```

EXIT:

به نکات زیر درباره فراخوانی INT33H توجه کنید.

۱- در INT21H و INT10H، ثبات AH برای انتخاب توابع بکار می رود. این مطلب در INT33H صحیح نیست. در INT33H، ثبات AL برای انتخاب انواع توابع بکار می رود و  $AH=0$  انتخاب می شود. این، دلیل وجود دستورالعمل "MOV AX,0" می باشد.

۲- از نوشتن "H" به معنی مبنای شانزده در INT33H فراموش نکنید. در غیاب "H" کامپایلر دهندهی را فرض خواهد کرد و DOS INT21H را اجرا خواهد کرد زیرا 33 دهندهی برابر با 21H است.

## برخی از اصطلاحات ماوس

قبل از بحث اضافی در INT33H، بعضی از اصطلاحات ماوس معرفی می‌شوند. اشاره گر ماوس (یا مکان‌نما) اشاره‌گری بر روی صفحه نمایش است که محل اشاره شده بوسیله ماوس در زمان معینی را مشخص می‌کند. در مد گرافیک اشاره گر ماوس یک فلش است و در مد متن یک بلوک خاموش و روشن شونده است. در هر یک از دو مد، اشاره گر ماوس با حرکت ماوس جابجا می‌شود. در حالی که جابجایی ماوس به اینچ (یا سانتیمتر) اندازه‌گیری می‌شود، جابجایی مکان‌نما یا اشاره گر ماوس به میکی است. میکی حساسیت ماوس را مشخص می‌کند. مثلاً ماوسی که بتواند در قبال هر اینچ جابجایی، 200 واحد مکان را جابجا کند دارای حساسیت 200 میکی است. در این حال یک میکی نمایشگر 1/200 اینچ در صفحه نمایش است. بعضی از ماوس‌ها بر خلاف ماوس‌های 200 میکی، دارای 400 میکی حساسیت هستند. در این حال برای هر اینچ جابجایی، مکان‌نما در صفحه نمایش 400 میکی جابجا می‌شود.

## نمایش و پنهان کردن مکان‌نمای ماوس

تابع AX=01 از INT33H برای نمایش مکان‌نمای ماوس بکار می‌رود.

```
MOV AX,01
INT 33H
```

پس از اجرای کد فوق، اشاره گر ماوس نمایان می‌شود. اگر مد تصویر، گرافیک است، فلش ماوس قابل رؤیت می‌گردد. اگر مد تصویر، متن باشد یک بلوک مستطیلی که مکان‌نمای ماوس را نشان می‌دهد آشکار خواهد شد. در مد متن رنگ بلوک مکان‌نمای ماوس مخالف رنگ زمینه است تا قابل رویت باشد. بهتر است پس از قابل رؤیت شدن ماوس، آن را پنهان کنیم. این عمل با اجرای تابع AX=02 از INT33H میسر است. این عمل در مثال ۳-۵ نشان داده شده است. مثال ۳-۵ را در DE3UG انجام دهید و بخاطر داشته باشید که H را حذف و از INT3 استفاده کنید. سپس با AH=03 از INT10H آن را اجرا نمایید تا مکان‌نمای ماوس را در مد متن ملاحظه نمایید.

مثال ۳-۵. (الف) از انتخاب 0F در INT10 برای بدست آوردن مد تصویر استفاده کرده آن را در BL ذخیره نمایید. (ب) مد تصویر گرافیک را با استفاده از انتخاب AH=10H از INT10H برگزینید. (پ) ماوس را با AX=0 از INT33H مقداردهی اولیه کنید. (ت) ماوس را قابل رؤیت نمایید. (ث) برای فشردن کلید با AH=01 از INT21H منتظر بمانید. (ج) اگر کلیدی فشرده شود مد تصویر اولیه را برقرار نمایید.

حل:

```
MOV AH,0FH ;get the current video mode
INT 10H
MOV BL,AL ;and save it
MOV AH,0 ;set the video mode
MOV AL,10H ;to VGA graphics
INT 10H
MOV AX,0 ;initialize the mouse
INT 33H
MOV AX,01 ;make the mouse cursor visible
INT 33H
MOV AH,01 ;wait for key press
INT 21H
MOV AX,2 ;when any key is pressed
INT 33H ;make mouse invisible
MOV AH,0
MOV AL,BL ;and restore original video mode
INT 10H
```

| تعداد کاراکتر در صفحه تصویر | وضوح یا دقت ماوس | نوع      | وضوح تصویر | مد تصویر |
|-----------------------------|------------------|----------|------------|----------|
| 80x25                       | 640x200          | Text     | 640x200    | AL=03    |
| 80x25                       | 640x200          | Graphics | 640x200    | AL=0EH   |
| 80x44                       | 640x350          | Graphics | 640x350    | AL=0FH   |
| 80x44                       | 640x350          | Graphics | 640x350    | AL=10H   |
| 80x60                       | 640x480          | Graphics | 640x480    | AL=11H   |
| 80x60                       | 640x480          | Graphics | 640x480    | AL=12H   |

جدول ۱-۵. دقت یا وضوح ماوس و ویدئو در بعضی از مدهای تصویر (ویدئو)

### وضوح تصویر در مقابل دقت ماوس در مد متن

همانطور که در فصل ۴ بحث شد، صفحه تصویر در مد متن به  $640 \times 200$  پیکسل تقسیم شده است. باین معنی که در مد متن با  $80 \times 25$  هر کاراکتر پیکسل‌های  $8 \times 8$  ( $80 \times 8 = 640, 25 \times 8 = 200$ ) بکار می‌برد. در مد متن ( $AH=03$  از  $INT=10H$ ) ماوس بطور خودکار همان وضوح (دقت  $640 \times 200$ ) را برای محورهای افقی و عمودی خواهد داشت. بنابراین در مد متن وقتی که برنامه‌ای محل مکان‌نما را بدست می‌آورد، مقادیر بر حسب پیکسل بوده و باید تقسیم بر ۸ شوند تا محل مکان‌نما بر حسب محل کاراکتر از ۰ تا ۷۹ (افقی) و از ۰ تا ۲۴ (عمودی) در روی صفحه نمایش معین شود.

### وضوح تصویر در برابر دقت ماوس در مد گرافیک

در گرافیک وضوح نه تنها  $640 \times 200$  بلکه  $640 \times 350$  و  $640 \times 480$  نیز می‌باشد. وقتی وضوح تصویر به این مدها تغییر یافت ماوس نیز دقت گرافیک را برمی‌گزیند. جدول ۱-۵ ملاحظه شود.

### بدست آوردن مکان جاری مکان‌نمای ماوس

انتخاب  $AX=03$  از  $INT33H$  مکان جاری مکان‌نمای ماوس را بدست می‌آورد. پس از بازگشت، محورهای X و Y در ثبات‌های CX (افقی) و DX (عمودی) می‌باشند. BX حاوی کلیدهای وضعیت است: D0 کلید وضعیت به چپ، D1 برابر کلید وضعیت به راست، D2 برای کلید وضعیت وسط می‌باشد. وضعیت برای سمت پائین ۱ و برای سمت بالا ۰ است. توجه کنید که محل مکان‌نمای حاصل از این تابع بر حسب پیکسل داده می‌شود. مثلاً برای یک صفحه  $640 \times 200$  در مد متن یا گرافیک محل بازگشت در محدوده ۰-۶۳۹ (افقی) و ۰-۱۹۹ (عمودی) است. با این وجود محل مکان‌نمای ماوس اغلب بر حسب مکان کاراکتر مانند  $80 \times 25$  مورد نظر است و نه بر حسب پیکسل. برای بدست آوردن محل کاراکتر ماوس، هر دو مقدار افقی و عمودی CX و DX را بر ۸ تقسیم کنید. برنامه‌های ۴-۵ و ۵-۵ ملاحظه شود.

#### TITLE PROGRAM 5.4 DISPLYING MOUSE POSITIN

PAGE 60,132

CURSOR MACRO ROW,COLUMN

MOV AH,02H

MOV BH,00

MOV DH,ROW

MOV DL,COLUMN

INT 10H

ENDM

DISPLAY MACRO STRING

MOV AH,09H

MOV DX,OFFSET STRING ;load string address

INT 21H

ENDM

این برنامه اعمال زیر را انجام می‌دهد: (الف) مد جاری تصویر را بدست می‌آورد و آن را ذخیره می‌کند، (ب) مد را در یک مد تصویر جدید قرار می‌دهد. (پ) مکان اشاره‌گر ماوس را بدست می‌آورد، و آن را به مکان کاراکتر تبدیل می‌کند و سپس آن را مرتباً نشان می‌دهد مگر اینکه کلیدی فشرده شود. (ت) پس از فشردن هر کلیدی، مد اولیه تصویر را بازگردانده و به DOS باز می‌گردد.

```

 .MODEL SMALL
 .STACK
 DATA
MESSAGE_1 DB 'PRESS ANY KEY TO GET OUT','$'
MESSAGE_2 DB 'THE MOUSE CURSOR IS LOCATED AT ','$'
POS_HO DB '?,?' AND '$'
POS_VE DB '?,?' '$'
OLDVIDEO DB ? ;current video mode
NEWVIDEO DB 0EH ;new video mode
 .CODE
MAIN PROC
 MOV AX,@DATA
 MOV DS,AX
 MOV AH,0FH ;get current video mode
 INT 10H
 MOV OLDVIDEO,AL ;save it
 MOV AX,0600H ;clear screen
 MOV BH,07
 MOV CX,0
 MOV DX,184FH
 INT 10H
 MOV AH,00H ;set new video mode
 MOV AL,NEWVIDEO
 INT 10H
 MOV AX,0 ; initialize mouse
 INT 33H
 MOV AX,01 ;show mouse cursor
 INT 33H
 CURSOR 20,20
 DISPLAY MESSAGE_1
AGAIN: MOV AX,03H ;get mouse location
 INT 33H
 MOV AX,CX ;get the hor. pixel position
 CALL CONVERT ;convert to displayable data
 MOV POS_HO,AL ;save the LSD
 MOV POS_HO+1,AH ;save the MSD
 MOV AX,DX ;get the vert. pixel position
 CALL CONVERT ;convert
 MOV POS_VE,AL ;save
 MOV POS_VE+1,AH
 CURSOR 5,20
 DISPLAY MESSAGE_2 ;
 DISPLAY POS_HO
 DISPLAY POS_VE
 MOV AH,01 ;check for key press
 INT 16H
 JZ AGAIN ;if no key press, keep monitoring mouse position
 MOV AH,02 ;hide mouse
 INT 33H
 MOV AH,0 ;restore original video mode
 MOV AL,OLDVIDEO ;load original video mode
 INT 10H
 MOV AH,4CH ;go back to DOS
 INT 21H
MAIN ENDP

;divide pixels position by 8 and convert to ASCII to make it displayable
;ax=pixels position (it is in hex)
;on return ax= two ASCII digits
CONVERT PROC
 SHR AX,1 ;divide
 SHR AX,1 ;by 8
 SHR AX,1 ;to get screen position by character
 MOV BL,10
 SUB AH,AH
 DIV BL ;divide by ten to convert from hex. to decimal
 OR AX,3030H ;make it ASCII
 RET
 ;return with AX=two ASCII digits
CONVERT ENDP
END MAIN

```

# TITLE PROG 5-5:MOUSE BOX PROGRAM

اعمال زیر را انجام می‌دهد: (الف) مد تصویر جاری را بدست آورده، آن را ذخیره می‌کند، (ب) مد تصویر را با مد جدید جایگزین کرده و صفحه تصویر را پاک می‌کند. (پ) یک جعبه رنگی رسم می‌کند و مکان ماوس را بدست می‌آورد. (ت) بسته کلیک خوردن ماوس در داخل یا خارج جعبه، پیام مختلفی می‌فرستد فشردن هر کلید موجب بازگشت به DOS می‌گردد.

```
CURSOR MACRO ROW,COLUMN
MOV AH,02H
MOV BH,00
MOV DH,ROW
MOV DL,COLUMN
INT 10H
ENDM

DISPLAY MACRO STRING
MOV AH,09H
MOV DX,OFFSET STRING ;load string address
INT 21H
ENDM

FILL MACRO ROW_START,COL_START,ROW_END,COL_END,COLOR
LOCAL START,AGAIN
MOV DX,ROW_START
MOV CX,COL_START
AGAIN: MOV AH,0CH
MOV AL,COLOR
INT 10H
INC CX
CMP CX,COL_END
JNE AGAIN
INC DX
CMP DX,ROW_END
JNE START
ENDM

.MODEL SMALL
.STACK
.DATA
MESSAGE_1 DB 'AN EXAMPLE OF HOW TO USE INTERRUPT 33H FOR MOUSE.', '$'
MESSAGE_2 DB 'IT WORKS!', '$'
MESSAGE_3 DB 'CLICK IN THE BOX TO SEE WHAT HAPPENS!', '$'
MESSAGE_4 DB 'No, NO, NO I SAID IN THE BOX!', '$'
MESSAGE_5 DB 'NOW PRESS ANY KEY TO GET BACK TO DOS. '$'
OLDVIDEO DB ?
NEWVIDEO DB 12H

.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX
MOV AH,0FH ;get the current video mode
INT 10H
MOV OLDVIDEO,AL ;save it
MOV AX,0600H ;clear screen
MOV BH,07
MOV CX,0
MOV DX,184FH
INT 10H
MOV AH,00H ;set new video mode
MOV AL,NEWVIDEO
INT 10H
CURSOR 0,0
FILL 150,250,250,350,4 ;draw red box
CURSOR 1,1
DISPLAY MESSAGE_1;
CURSOR 5,22
DISPLAY MESSAGE_3;
MOV AX,0000H ;initialize mouse
INT 33H
MOV AX,01H
INT 3 3H ;show mouse cursor
```



```

BACK: MOV AX,03H ;check for mouse button press
 INT 33H ;now CX =COL and DX=ROW location
 CMP BX,0001H ;check to see if left button is pressed
 JNE BACK ;if not keep checking
 CMP CX,250 ;see if on right side of box
 JB NOT_INSIDE ;if less it must be outside box
 CMP CX,350 ;see if on left side of box
 JA NOT_INSIDE ;if not then it is outside the box
 CMP DX,150 ;check for the top of the box
 JB NOT_INSIDE ;if not then outside the box
 CMP DX,250 ;see if bottom of the box
 JA NOT_INSIDE
 CURSOR 18,18 ;then it must be inside box
 DISPLAY MESSAGE_2 ;indicate mouse is inside the box
 JMP EXIT ;go prepare to exit to DOS

NOT_INSIDE:
 CURSOR 20,18 ;indicate mouse is not inside box
 DISPLAY MESSAGE_4

EXIT: MOV AH,02H ;hide mouse before exiting to DOS
 INT 33H
 CURSOR 22,18
 DISPLAY MESSAGE_5
 MOV AH,07 ;wait for a key press
 INT 21H
 MOV AH,0 ;restore original video mode
 MOV AL,OLDVIDEO
 INT 10H
 MOV AH,4CH ;exit
 INT 21H ;to DOS

MAIN ENDP
 END

```

#### ادامه برنامه ۵-۵

#### استقرار محل اشاره گر ماوس (AX=04)

این تابع موجب می شود تا برنامه اشاره گر ماوس را در هر مکان جدیدی در صفحه نمایش مستقر سازد. قبل از فراخوانی این تابع مختصات افقی محل جدید باید در CX (محور x) و عمودی در DX (محور y) قرار گیرد. این مقادیر برای وضوح 640x200 باید بین 0-639 و 0-199 باشد. مختصات (0,0) سمت چپ بالا در صفحه نمایش است. مثلاً برای استقرار مکان نما در محل 9x5 (در صفحه 80x25)، هر دو مقدار را در 8 ضرب کنید تا بر حسب پیکسل بدست آید. بنابراین مختصات کاراکتری 9x5 برابر با 72x40 در مختصات پیکسل خواهد شد.

#### بدست آوردن اطلاعات حاصل از فشردن کلید ماوس

این تابع برای بدست آوردن اطلاعات درباره فشردن دکمه خاص از آخرین فراخوانی این تابع است. این تابع بدین ترتیب تنظیم می شود.

AX=05

BX برای دکمه چپ 0، برای دکمه راست 1، برای دکمه وسط 2 می باشد.

پس از بازگشت:

|     |                                                  |
|-----|--------------------------------------------------|
| AX= | دکمه وضعیت                                       |
| DO= | اگر 1 یعنی پائین، اگر 0 یعنی بالا برای دکمه چپ   |
| D1= | اگر 1 یعنی پائین، اگر 0 یعنی بالا برای دکمه راست |
| D2= | اگر 1 یعنی پائین، اگر 0 یعنی بالا برای دکمه وسط  |
| BX= | تعداد فشردگی دکمه                                |
| CX= | مختصات X در آخرین فشردن دکمه (افقی)              |
| DX= | مختصات Y در آخرین فشردن دکمه (عمودی)             |

در تابع AX=05، پس از بازگشت از INT33H، ثبات AX وضعیت دکمه (بالا یا پائین) را داراست، در حالی که ثبات BX تعداد دفعات فشردن یک کلید خاص را از آخرین فراخوانی به بعد دارد. برنامه ۵-۶ روال استفاده از این تابع را نشان می دهد.

## برنامه ردیابی و نمایش دفعات فشردن کلید

برنامه ۵-۶ تابع AX=05 را برای ردیابی دفعات فشردن دکمه چپ و نمایش تعداد آن بکار می‌برد. این تابع به کاربر پیام می‌دهد که دکمه چپ را چندین بار فشرده است. هنگامی که کاربر مایل به دیدن تعداد دفعات شد باید یک کلید را بفشارد.

### TITLE PROGRAM 5-6: DISPLAY MOUSE PRESS COUNT

این برنامه منتظر فشردن کلید و شمردن آن می‌ماند و هر کلیدی که فشرده شد آن دفعات را نشان می‌دهد.

PAGE 60,132

```
CURSOR MACRO ROW,COLUMN
MOV AH,02H
MOV BH,00
MOV DH,ROW
MOV DL,COLUMN
INT 10H
ENDM
```

```
DISPLAY MACRO STRING
MOV AH,09H
MOV DX,OFFSET STRING ;LOAD STRING ADDRESS
INT 21H
ENDM
```

```
.MODEL SMALL
.STACK
.DATA
```

```
MESSAGE_1 DB 'PRESS LEFT BUTTON A NUMBER OF TIMES:LESS THAN 99.','$'
MESSAGE_2 DB 'TO FIND OUT HOW MANY TIMES, PRESS ANY KEY','$'
MESSAGE_3 DB 'YOU PRESSED IT ','$'
P_COUNT DB '?',' TIMES $'
MESSAGE_4 DB 'NOW PRESS ANY KEY TO GO BACK TO DOS','$'
OLDVIDEO DB ? ;current video mode
NEWVIDEO DB 12H ;new video mode
```

```
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX
MOV AH,0FH ;get current video mode
INT 10H
MOV OLDVIDEO,AL ;save it
MOV AX,0600H ;clear screen
MOV BH,07
MOV CX,0
MOV DX,184FH
INT 10H
MOV AH,00H ;set new video mode
MOV AL,NEWVIDEO
INT 10H
MOV AX,0 ;initialize mouse
INT 33H
MOV AX,01 ;show mouse cursor
INT 33H
CURSOR 2,1
DISPLAY MESSAGE_1
CURSOR 4,1
DISPLAY MESSAGE_2
MOV AH,07 ;wait for key press
INT 21H
MOV AX,05H ;get mouse press count
MOV BX,0 ;check press count for left button
INT 33H
```

برنامه ۵-۶ (ادامه در صفحه بعد)

```

MOV AX,BX ;BX=button press count
MOV BL,10
DIV BL
OR AX,3030H ;convert it to ASCII
MOV P_COUNT,AL ;save the number
MOV P_COUNT+1,AH
CURSOR 10,2
DISPLAY MESSAGE_3
DISPLAY P_COUNT
CURSOR 20,2
DISPLAY MESSAGE_4
MOV AH,07 ;wait for a key press to get out
INT 21H
MOV AH,02 ;hide mouse
INT 33H
MOV AH,0 ;restore original video mode
MOV AL,OLDVIDEO ;load original vide mode
INT 10H
MOV AH,4CH ;go back to DOS
INT 21H
MAIN
ENDP
END MAIN

```

برنامه ۶-۵ (ادامه صفحه قبل)

### بدست آوردن اطلاعات دکمه رها شده در ماوس

این تابع مشابه  $AX=05$  است. با این تفاوت که اطلاعات را در مورد دکمه رها شده فراهم می نماید. فراخوانی با تنظیم های زیر صورت می گیرد:

$AX=06$

برای دکمه چپ  $BX=0$  ، برای دکمه راست  $BX=1$  ، برای دکمه وسط  $BX=2$

پس از بازگشت :

$AX=$  وضعیت دکمه

$DO=$  برای دکمه چپ اگر 1 یعنی پائین، اگر 0 یعنی بالا

$D1=$  برای دکمه راست اگر 1 یعنی پائین، اگر 0 یعنی بالا

$D2=$  برای دکمه وسط اگر 1 یعنی پائین، اگر 0 یعنی بالا

$BX=$  دفعات رهاسازی دکمه

$CX=$  مختصات x در آخرین رهاسازی بر حسب پیکسل

$DX=$  مختصات y در آخرین رهاسازی بر حسب پیکسل

### تنظیم محدوده افقی برای اشاره گر ماوس ( $AX=07$ )

وقتی که ماوس با  $AX=0$  مقداردهی اولیه می شود، مکان نمای ماوس قابل جابجایی به هر نقطه در صفحه نمایش است. این تابع و تابع بعدی مرزهای جابجایی را در راستای x و y معین می نمایند. در این تابع

$AX=07$

$CX=$  حداقل مختصات x بر حسب پیکسل (0-639)

$DX=$  حداکثر مختصات x بر حسب پیکسل (0-639)

توجه داشته باشید که CX باید کمتر از DX باشد در غیر اینصورت آنها تعویض خواهند شد.

### تنظیم محدوده عمودی برای اشاره گر ماوس ( $AX=08$ )

این تابع و تابع قبلی امکان تنظیم مختصات x و y را که در آن جابجایی ماوس تعریف می شود، فراهم می سازند.

AX=08

CX=

DX=

حداقل مختصات y بر حسب پیکسل (0-199)

حداکثر مختصات y بر حسب پیکسل (0-199)

توجه کنید که CX باید کوچکتر از DX باشد در غیر اینصورت آنها تعویض خواهند شد.

### تنظیم ناحیه خارج از محدوده (exclusion) برای اشاره گر ماوس (AX=10H)

این تابع برای تنظیم ناحیه خارج از محدوده ای برای مکان‌نمای ماوس بکار می‌رود. اگر مکان‌نمای ماوس به ناحیه غیرمجاز وارد شود ناپدید می‌گردد. در این تابع،

AX=10H

CX= مختصات x بالایی

DX= مختصات y بالایی

SI= مختصات x پائینی

DI= مختصات y پائینی

### بدست آوردن اطلاعات راه‌انداز ماوس (ویرایش) (AX=24H)

این تابع اجازه می‌دهد تا برنامه ویرایش راه‌انداز ماوس معین شود. همچنین نوع ماوس، مانند ماوس گذرگاه داخلی یا ماوس گذرگاه خارجی از طریق پورت سریال را مشخص می‌کند:

AX=24H

پس از بازگشت داریم:

BH= بخش کوچکتر ویرایش ، BL= بخش بزرگتر ویرایش

مثلاً در "7.20" عدد 7 بخش بزرگتر و 20 بخش کوچکتر است

CH=

=1

=2

=3

PS/2 =4

5=HP

CL=0

نوع ماوس که

ماوس گذرگاهی

ماوس سریال

ماوس درگاهی

ماوس

ماوس

اگر ماوس PS/2 باشد اگر ماوس سریال باشد عدد بین 2 و 7 بوده و

نشان دهنده این حقیقت است که شماره IRQ (مثل IRQ2، IRQ3

و غیره) بکار رفته است.

برنامه ۵-۷ مثالی از برنامه‌ای بزیان C است که ماوس را بکار برده است. وقفه INT 33H با استفاده از تابع int86 فراخوانی می‌شود تا وجود ماوس ملاحظه شود. اگر ماوس موجود باشد تعداد دکمه‌های ماوس چاپ خواهند شد.

```
// This Program uses INT 33H to see if mouse driver is installed and if so it displays
// the number of buttons on the mouse
#include <stdio.h>
#include <dos.h>
#include <conio.h>
main()
{
 union REGS ri,ro;
 clrscr();
 ri.x.ax=0x0; //check to see if there is mouse
 int86(0x33,&ri,&ro);
 if(ro.x.ax>0)
 printf("\n\nYou have %d button(s) on the mouse on this PC",ro.x.bx);
 else
 printf("This PC has no mouse\n");
}
```

- ۱- کدام تابع INT33H برای تشخیص وجود ماوس در PC بکار می‌رود؟ در کدام ثبات انتظار دریافت اطلاعات می‌رود؟ در کدام ثبات تعداد دکمه‌های ماوس را پیدا می‌کنیم؟
- ۲- کد زیر یک پیام برای فراخوانی تابع 2 از INT33H است. آیا صحیح است؟ اگر خیر، آن را اصلاح کنید.  
`MOVAH,02`  
`INT 33H`
- ۳- چرا مد تصویر اولیه را قبل از تغییر آن برای ماوس باید ذخیره کرد؟
- ۴- در INT33H تابع `AX=03` چگونه فشردگی کلید چپ تشخیص داده می‌شود؟
- ۵- صحیح یا غلط. مختصات ماوس همانند مد تصویر است.

## خلاصه

ماکروها بوسیله برنامه‌نویسان جهت صرفه‌جویی در کدنویسی و رفع عیب مورد استفاده قرار گرفتند. هر وقت مجموعه‌ای از دستورالعمل‌ها مکرراً اجرا شوند، استفاده از ماکرو ایده‌آل است. می‌توان مقادیری را که بوسیله دستورالعمل‌ها بکار می‌روند در میان ماکروها ردوبدل کرد. برنامه‌نویسان می‌توانند چندین ماکرو پر مصرف را در یک فایل قرار دهند و به موقع آن‌ها را وارد یک یا چند برنامه نمایند. در سال‌های اخیر ماوس بعنوان یک وسیله ورودی استاندارد در IBM PC معرفی شده است. IBM PC BIOS اولیه و DOS برای ماوس تهیه نشده‌اند. INT33H بخشی از نرم افزار راه انداز ماوس است که هنگام بوت شدن PC نصب می‌شود. تابع INT33H برای ورودی ماوس و کنترل مکان نما مورد استفاده قرار می‌گیرد.

## مسائل

- ۱- هر ماکرو باید با یک رهنمون ... آغاز و با رهنمون ... پایان یابد.
- ۲- نام، بدنه و آرگومان ساختگی را در ماکرو زیر بدست آورید:

```
WORK_HOUR MACRO OVRTME_HR
 MOV AL,40 ;WEEKLY HRS
 ADD AL,OVRTME_HR ;TOTAL HRS WORKED
ENDM
```

- ۳- تفاوت میان رهنمون‌های SALL، LALL و XALL را توضیح دهید.
- ۴- مقدار کل موجود در ثبات‌های DX و AX پس از فراخوانی ماکرو زیر چیست؟

```
WAGES MACRO SALARY,OVERTIME,BONUSES
;TOTAL WAGES=SALARY + OVERTIME + BONUSES
SUB AX,AX ;CLEAR
MOV DX,AX ;AX AND DX
ADD AX,SALARY
ADD AX,OVERTIME
ADC DX,0 ;TAKE CARE OF CARRY
ADD AX,BONUSES
ADC DX,0
ENDM
```

ماکرو بصورت زیر فراخوانی می شود:

WAGES 60000,25000,3000

- ۵- در مسئله ۴، در بدنه ماکرو، ساختگی ها از چپ به راست لیست بکار رفته اند. آیا با هر ترتیبی قابل استفاده اند؟ با افزودن OVERTIME بدنه را بازنویسی کنید (پارامترهای ساختگی را دست نزنید).
- ۶- در مسئله ۴ توضیحات را اگر ماکرو بصورت زیر گسترش یابد بیان کنید.

.LALL  
WAGES X,Y,Z

- ۷- ماکروها می توانند ثبات ها را به عنوان مکان های ساختگی بکار ببرند. فایل "lst" را نشان دهید و بگوئید ماکرو در مسئله ۴ اگر به شکل زیر فراخوانی شود چه کاری انجام می دهد.

WAGES BX,CX,SI

- ۸- مکان ها خالی را برای ماکروهای زیر پر کنید تا آرایه ای از بایت ها با هم جمع شوند. بعضی از مکان های خالی نیازی به پر شدن ندارند.

```
SUMMING MACRO COUNT,VALUES
LOCAL
;;this macro adds an array of byte size elements.
;;ax will hold the total sum
MOV CX,.... ;size of array
MOV SI,OFFSET ;load offset address of array
SUB AX,AX ;clear ax
AGAIN: ADD AL,[SI]
 ADC AH,0 ;add bytes and takes care of carries
 INC SI ;point to next byte
 LOOP AGAIN ;continue until finished
 ENDM
```

- ۹- ماکرو فوق را برای داده زیر فراخوانی و اجرا نمایید.  
در قطعه داده داریم:

```
DATA1 DB 89,75,98,91,65
SUM1 DW ?
DATA2 DB 86,69,99,14,93,99,54,39
SUM2 DW ?
DATA3 DB 10 DUP (99)
SUM3 DW ?
```

(تذکر: مسئله ۱۰ ملاحظه شود)

- ۱۰- رهنمون های لیست کردن را در مسئله و مطابق زیر وارد کرده و فایل "lst" را تحلیل کنید. از قطعه کد داریم:

```
.LALL
SUMMING 5,DATA1 ;adding and saving data1
.XALL
SUMMING ;adding and saving data2
.....
.SALL
..... ;adding and saving data3
.....
```

- ۱۱- مسئله ۸ را به منظور داشتن سومین آرگومان ساختگی بازنویسی کنید. آنگاه مسئله ۹ را تکرار نمایید.
- ۱۲- برنامه ۲-۵ را با رهنمون DD برای RESULT1، RESULT2 و RESULT3 دوباره بنویسید.
- ۱۳- با استفاده از وقفه INT33H، یک برنامه ب زبان اسمبلی برای تست وجود ماوس در یک PC بنویسید. اگر راه انداز ماوس نصب باشد باید تعداد دکمه هایی را که پشتیبانی می نماید بیان کند. اگر راه انداز ماوس نصب نشده، باید آن را بیان نماید.
- ۱۴- مد تصویر برنامه ۴-۵ را به مقادیر زیر تغییر دهید و سپس صحت جدول ۱-۵ را تحقیق کنید.

(الف) 10H

(ب) 12H

- ۱۵- سایز و رنگ جعبه را در برنامه ۵-۵ به مختصات و رنگ انتخابی خود تغییر دهید و سپس آن را اجرا و تست کنید. جعبه را باندازه کافی انتخاب کنید تا نام و نام خانوادگی شما در آن جا می‌گیرد.
- ۱۶- برنامه ۵-۶ را برای نمایش تعداد دفعات فشردن دکمه سمت راست اصلاح کنید.
- ۱۷- مسئله فوق را برای مشاهده تعداد دفعات رها سازی دکمه سمت راست اصلاح نمایید.
- ۱۸- با استفاده از INT33H برای نمایش ویرایش راه انداز و تعداد دکمه‌های پشتیبانی شده بوسیله ماوس در یک PC مفروض برنامه‌ای بزبان اسمبلی بنویسید.
- ۱۹- نوع زبان C برنامه ۵-۴ را بنویسید. برنامه شما اعمال زیر را انجام می‌دهد  
(الف) صفحه را پاک می‌نماید  
(ب) مرتباً مکان اشاره گر ماوس را بدست آورده و در مختصات کاراکتر صفحه نمایش آن را نمایش می‌دهد.  
(ت) با فشردن هر یک از کلیدها، مد تصویر اولیه را بازگردانده و به DOS بر می‌گردد.
- ۲۰- مسئله فوق را تکرار نمایید. این بار محل اشاره گر ماوس را برحسب پیکسل نشان دهید. آن را برای مد تصویر  $AL = 10H$  و  $AL = 12H$  تست نمایید.
- ۲۱- برنامه‌ای بزبان C برای نمایش اینکه کدام دکمه ماوس در یک لحظه فشرده شده بنویسید. با فشردن هرکلید برنامه به DOS بازگردد.
- ۲۲- برای برنامه ۵-۶ برنامه‌ای بزبان C بنویسید تا اعمال زیر را انجام دهد.  
(الف) مد تصویر را ذخیره کنید و سپس آن را به یکی از مدهای گرافیکی تغییر دهید.  
(ب) از کاربر بخواهید تا چندین بار دکمه سمت چپ را بفشارد.  
(پ) وقتی کلیدی فشرده می‌شود عددی را نمایش دهید  
(ت) با فشردن هر کلید، مد تصویر اولیه را بازگردانید و به DOS بروید.
- ۲۳- برنامه فوق را برای دکمه سمت راست تکرار کنید.
- ۲۴- با استفاده از  $AX = 0$  و  $AX = 24H$  از INT33، یک برنامه بزبان C برای نمایش موارد زیر بنویسید.  
(الف) تعداد دکمه‌های ماوس  
(ب) نوع ماوس (ماوس گذرگاهی، ماوس سریال، ماوس PS/2 و غیره)  
(پ) ویرایش راه انداز ماوس در فرمی مشابه "7.5"  
(ت) شماره IRQ، اگر بوسیله ماوس بکار رفته است.

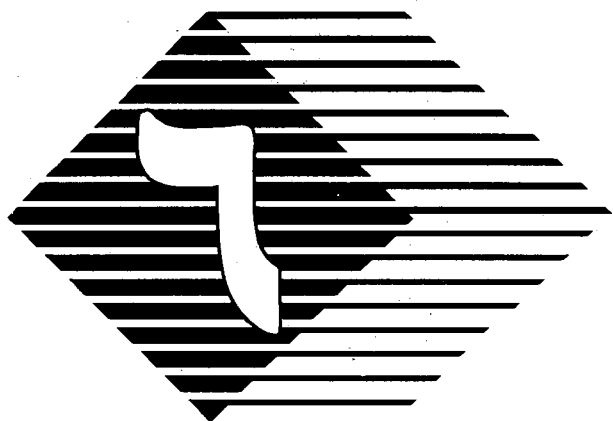
## ✓ پاسخ به سؤالات مروری

### بخش ۱-۵: ماکرو چیست و چگونه بکار می‌رود.

- ۱- برنامه نویسی ماکرو می‌تواند موجب صرفه جویی در زمان برنامه نویسی گردد، باین ترتیب که دستورات بکار رفته تکراری قابل فراخوانی با یک خط برنامه می‌باشند.
- ۲- سه بخش از یک ماکرو عبارتند از رهنمون MACRO، بدنه و رهنمون ENDM.
- ۳- تعریف ماکرو لیستی از عبارات است که ماکرو آن‌ها را اجرا خواهد کرد. این تعریف با رهنمون MACRO آغاز و با رهنمون ENDM پایان می‌یابد. فراخوانی ماکرو هنگامی صورت می‌گیرد که از درون یک برنامه زبان اسمبلی فراخوانده شود. گسترش ماکرو هنگامی است که اسمبلی خط فرا خوانده شده ماکرو را با کدهای زبان اسمبلی در بدنه ماکرو جایگزین نماید.
- ۴- غلط. یک برچسب که قرار است برای یک ماکرو محلی باشد با رهنمون LOCAL محلی خواهد شد.
- ۵- غلط. حرف C در ابتدای خط باین معنی است که این خط از فایل دیگری به وسیله رهنمون INCLUDE وارد شده است.

### بخش ۲-۵: برنامه نویسی ماوس با وقفه 33H

- ۱-  $AX = 0$ ؛ ثبات AX، مقدار دکمه‌ها در BX
- ۲- غلط است. ثبات  $AL = 02$  و  $AH = 0$ ، INT33H
- ۳- هنگامی که آن را انجام ندهیم، و به DOS بازگردیم، اگر برنامه ماوس تصویر رابه گرافیک تغییر داده باشد مکان‌نما گم می‌شود
- ۴- ما محتوای ثبات BX را برای 01 جستجو می‌کنیم.
- ۵- صحیح



## اعداد علامت دار، رشته ها و جداول

### اهداف



- در پایان این فصل قادر خواهید بود:
- « اعداد 8 و 16 بیتی را آن طور که در کامپیوتر بکار می روند ارائه دهید
  - « یک عدد را به متمم 2 تبدیل کنید
  - « با دستورالعمل های حسابی علامت دار، جمع، تفریق و ضرب و تقسیم یعنی ADD، SUB، IMUL و IDIV برنامه نویسی نمایید.
  - « نشان دهید که چگونه دستورات حسابی پرچم علامت را تغییر می دهند.
  - « تفاوت میان یک نقلی و سرریز را در حساب علامت دار توضیح دهید.
  - « با علامت گسترش یافته در داده از بروز خطاهای سرریز جلوگیری کنید
  - « با دستورالعمل های جابجایی علامت دار SAM و SAL برنامه بنویسید.
  - « با اعداد علامت دار، دستور منطقی CMP را استفاده کنید و درک مقایسه اثر روی ثبات پرچم را نشان دهید.
  - « با دستورالعمل JUMP بعد از CMP اعداد علامت دار، برنامه نویسی کنید.
  - « عملکرد ثبات های SI و DI را در عملیات رشته ای توضیح دهید.
  - « عملکرد پرچم جهت را در دستورالعمل های رشته ای توصیف نمایید.
  - « برای کنترل پرچم جهت با دستورات CLD و STD برنامه نویسی کنید.
  - « عملکرد پیشوند REP را توصیف کنید.
  - « با دستورات رشته ای برنامه های زیر را بنویسید:
  - « MOVS و MOVSW برای انتقال داده
  - « LODS، STOS برای ذخیره و بار کردن محتوای AX
  - « CMPS برای مقایسه دو رشته داده
  - « SCAS برای پویش یک رشته داده جهت تطبیق آن با AX
  - « XLAT برای پردازش جدول



در اولین بخش از این فصل ما بر مفهوم اعداد علامت‌دار در مهندسی نرم افزار تکیه می‌کنیم. عملیات مربوط به اعداد علامت‌دار همراه با مثال‌هایی مورد بررسی قرار خواهند گرفت. در بخش دوم عملیات زشته‌ای و پردازش جدولی را بحث خواهیم کرد.

## بخش ۱-۶: عملیات محاسباتی اعداد علامت‌دار

همه اقلام داده‌ای که تاکنون بکار رفته‌اند اعداد بدون علامت بودند. به این معنی که همه 8 یا 16 بیت از عملوند بکار رفته، اندازه را نشان می‌دادند. بسیاری از کاربردها نیاز به اعداد علامت‌دار دارند. در این بخش مفهوم اعداد علامت‌دار همراه با دستورالعمل‌های مربوطه مورد بحث قرار می‌گیرند.

### مفهوم اعداد علامت‌دار در کامپیوترها

در زندگی روزمره اعداد بکار رفته می‌توانند مثبت یا منفی باشند. مثلاً، دمای 5 درجه زیر صفر می‌تواند با 5- و 20 درجه بالای صفر با 20+ نشان داده شود. کامپیوترها باید قادر به نگهداری چنین اعدادی باشند. برای تحقق این مطلب متخصصین کامپیوتر پیشنهاد زیر را برای اعداد مثبت و منفی ارائه نمودند: با ارزش‌ترین بیت (MSB) برای علامت (+ یا -) کنار گذاشته می‌شود و از بقیه بیت‌ها برای مقدار استفاده می‌گردد. علامت 0 برای اعداد مثبت و 1 برای اعداد منفی در نظر گرفته می‌شوند. نمایش بایت و کلمه علامت‌دار در زیر بحث شده‌اند.

### عملوندهای بایتی علامت‌دار

| D7     | D6 | D5 | D4 | D3 | D2 | D1 | D0    |
|--------|----|----|----|----|----|----|-------|
| اندازه |    |    |    |    |    |    | علامت |

در عملوندهای بایتی علامت‌دار، D7 برای علامت و D0 تا D6 برای اندازه اعداد کنار گذاشته شده‌اند. اگر  $D7 = 0$  باشد عملوند مثبت و اگر  $D7 = 1$  باشد منفی است.

### اعداد مثبت

محدوده اعداد مثبتی که قابل نمایش با قالب فوق هستند از 0 تا +127 است.

|      |      |      |
|------|------|------|
| 0    | 0000 | 0000 |
| +1   | 0000 | 0001 |
| +5   | 0000 | 0101 |
| ..   | .... | .... |
| ..   | .... | .... |
| +127 | 0111 | 1111 |

اگر عدد مثبت بزرگتر از +127 باشد عملونددی با سائز کلمه باید بکار برود. عملوندهای کلمه‌ای بعداً بحث خواهند شد.

### اعداد منفی

برای اعداد منفی  $D7=1$  است، ولی مقدار به متمم 2 نشان داده می‌شود. اگر چه اسمبلر تبدیل را انجام می‌دهد ولی دانستن چگونگی تبدیل اهمیت خاص خود را دارد. برای تبدیل نمایش اعداد منفی مراحل زیر را دنبال کنید.

۱- مقدار عدد را در فرم 8 بیت دودویی بنویسید (بدون علامت)

۲- هر بیت را معکوس کنید

۳- حاصل را با 1 جمع کنید

مثال‌های ۱-۶، ۲-۶ و ۳-۶ این مراحل را نشان می‌دهند.

مثال ۱-۶. کامپیوتر، ۵- را چگونه نشان می‌دهد

حل:

۱- ۵ در هشت بیت دودویی 0000 0101  
۲- جمع با 1 (FBH) = به شانزده شانزدهی 1111 1011  
این عدد نمایش متمم 2 عدد 5- می‌باشد

مثال ۲-۶. 34H- در درون کامپیوتر چگونه نمایش داده می‌شود

حل:

۱- 0011 0100 ۲- 1100 1011 ۳- 1100 1100 که برابر با CCH است

مثال ۳-۶. 128<sub>10</sub>- را نشان دهید.

حل:

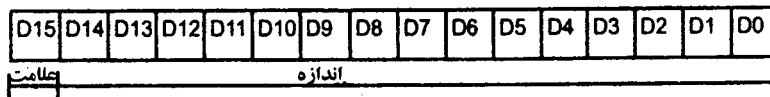
۱- 1000 0000 ۲- 0111 1111 ۳- 1000 0000 توجه کنید که این عدد صفر منفی نیست.

در مثال‌های فوق دیده می‌شود که محدوده اعداد منفی بایستی از 1- تا 128- است.

در زیر محدوده اعداد علامت‌دار بایستی لیست شده‌اند.

| دهدهی | دودویی    | شانزده شانزدهی |
|-------|-----------|----------------|
| -128  | 1000 0000 | 80             |
| -127  | 1000 0001 | 81             |
| -126  | 1000 0010 | 82             |
| ...   | .....     | ...            |
| -2    | 1111 1110 | FE             |
| -1    | 1111 1111 | FF             |
| 0     | 0000 0000 | 00             |
| +1    | 0000 0001 | 01             |
| +2    | 0000 0010 | 02             |
| ...   | .....     | ...            |
| +127  | 0111 1111 | 7F             |

اعداد علامت‌دار با سایز کلمه



در کامپیوترهای 80x86 یک کلمه 16 بیت طول دارد. با کنار گذاشتن MSB (D15) برای علامت، جمعاً 15 بیت (D14-D0) باقی خواهد گذاشت تا برای اندازه آن بکار رود. به این تعداد بیت، محدوده 32768- تا 32767+ در دسترس خواهد بود. اگر عددی بزرگتر از این باشد باید به عنوان عملوند چند بایتی متصور شود و جزء به جزء پردازش گردد، (فصل ۳). در زیر عملوندهای کلمه‌ای علامت‌دار دیده می‌شوند. برای تبدیل یک منفی به شکل عملوندها کلمه سه مرحله بکار رفته در بحث عملوندهای منفی بایستی بکار می‌رود.

| دهدهی   | دودویی              | شانزده شانزدهی |
|---------|---------------------|----------------|
| -32 768 | 1000 0000 0000 0000 | 8000           |
| -32 767 | 1000 0000 0000 0001 | 8001           |
| -32 766 | 1000 0000 0000 0010 | 8002           |
| .....   | .....               | .....          |
| .....   | .....               | .....          |
| -2      | 1111 1111 1111 1110 | FFFE           |
| -1      | 1111 1111 1111 1111 | FFFF           |
| 0       | 0000 0000 0000 0000 | 0000           |
| +1      | 0000 0000 0000 0001 | 0001           |
| +2      | 0000 0000 0000 0010 | 0002           |
| ...     | .....               | .....          |
| ...     | .....               | .....          |
| +32 766 | 0111 1111 1111 1110 | 7FFE           |
| +32 767 | 0111 1111 1111 1111 | 7FFF           |

## مسئله سرریز در عملیات اعداد علامت دار

وقتی اعداد علامت دار بکار می‌روند، مسئله جدی مورد بحث، مسئله سرریز است. CPU وجود مشکل را با بالا بردن پرچم سرریز OF اعلام می‌دارد ولی توجه به آن به عهده برنامه نویسی است. CPU فقط 0 و 1 را می‌شناسد و قرارداد اعداد مثبت و منفی انسان را چشم پوشی می‌نماید. و اما سرریز چیست؟ اگر نتیجه عملی بر روی اعداد علامت دار خیلی بزرگتر از ابعاد ثبات باشد، سرریز رخ داده است. برنامه نویسی باید از آن آگاه شود. به مثال ۴-۶ توجه نمایید.

مثال ۴-۶. به قطعات کد و داده توجه کنید:

```
DATA1 DB +96
DATA2 DB +70

MOV AL, DATA1 ;AL=0110 0000 (AL=60H)
MOV BL, DATA2 ;BL=0100 0110 (BL=46H)
ADD AL, BL ;AL=1010 0110 (AL=A6H= -90 invalid!)
```

```
+ 96 0110 0000
+ 70 0100 0110
+166 1010 0110
```

طبق محاسبات CPU این مقدار (OF = 1, SF = 1, CF = 0)

-90 است که غلط می‌باشد.

در مثال فوق 96+ و 70+ با هم جمع شده و حاصل -90 شده است. چرا؟ دلیل این است که نتیجه بزرگتر از قابلیت AL است. همچون دیگر ثبات‌ها، AL فقط می‌تواند تا 127+ را در خود جای دهد. طراحان CPU پرچم سرریز را به این خاطر طراحی کرده‌اند تا به برنامه نویسی خطای موجود در عملیات اعداد علامت دار اطلاع داده شود.

## چه وقت پرچم سرریز در عملیات 8 بیت نشانده می‌شود

در عملیات اعداد علامت دار 8 بیتی، OF در یکی از دو حالت زیر 1 می‌گردد.

۱- از D6 به D7 رقم نقلی وجود داشته باشد ولی از D7 رقم نقلی خارج نشود (CF = 0)

۲- از D7 رقم نقلی خارج شود (CF = 1) ولی از D6 به D7 رقم نقلی وجود نداشته باشد.

به بیان دیگر پرچم سرریز هنگامی به 1 نشانده می‌شود که از D6 به D7 و یا فقط از D7 رقم نقلی وجود داشته باشد ولی نه هر دو آنها بطور همزمان. این، چنین معنی می‌دهد که اگر بطور همزمان از D6 به D7 و یا از D7 رقم نقلی وجود داشته باشد OF = 0 خواهد بود.

در مثال ۴-۶ چون رقم نقلی فقط از D6 به D7 وجود داشته ولی به بیرون از D7 وجود ندارد، پس OF = 1 است. مثال‌های ۵-۶، ۶-۶ و ۶-۷ توضیحات بیشتری را در مورد پرچم سرریز در محاسبات علامت دار ارائه کرده‌اند.

مثال ۵-۶. نتایج حاصل را ملاحظه کنید.

```
MOV DX, 542FH ; 21,551
MOV BX, 12E0H ; + 4,832
ADD DX, BX ; =26,383

543F 0101 0100 0010 1111
+ 12E0 0001 0010 1110 0000
670F 0110 0111 0000 1111 = 26,383 (جواب صحیح) OF = 0, CF = 0, SF = 0
```

طبق محاسبات CPU، نتیجه 126+ است که غلط می‌باشد. خطا بوسیله OF = 1 اعلام می‌گردد.

مثال ۶-۶. نتایج حاصل را ملاحظه نمایید

|     |       |                                         |
|-----|-------|-----------------------------------------|
| MOV | AL,-2 | ;AL=1111 1110 (AL=FEH)                  |
| MOV | CL,-5 | ;CL=1111 1011 (CL=FBH)                  |
| ADD | CL,AL | ;CL=1111 1001 (CL=F9H= -7 ( که صحیح است |

|      |           |                                           |
|------|-----------|-------------------------------------------|
| -2   | 1111 1110 |                                           |
| + -5 | 1111 1011 |                                           |
| -7   | 1111 1001 | OF = 0. CF = 0. SF = 1 (منفی) . می باشد . |

نتیجه صحیح است زیرا OF = 0

مثال ۶-۷. نتایج زیر را ملاحظه کنید

|     |        |                                  |
|-----|--------|----------------------------------|
| MOV | DH,+7  | ;DH=0000 0111 (DH=07H)           |
| MOV | BH,+18 | ;BH=0001 0010 (BH=12H)           |
| ADD | BH,DH  | ;BH=0001 1001 (BH=19H=+25, صحیح) |

|       |           |                                  |
|-------|-----------|----------------------------------|
| +7    | 0000 0111 |                                  |
| + +18 | 0001 0010 |                                  |
| +25   | 0001 1001 | OF = 0, CF = 0 و SF = 0 ( مثبت ) |

### پرچم سرریز در عملیات 16 بیتی

در عملیات 16 بیتی، OF در هر یک از دو حالت زیر در 1 نشانده می شود:

- ۱- از D14 به D15 رقم نقلی وجود داشته باشد ولی از D15 به خارج رقم نقلی نباشد (CF = 0)
  - ۲- از D15 به خارج رقم نقلی وجود دارد (CF = 1) ولی از D14 به D15 رقم نقلی وجود ندارد.
- مجدداً اگر رقم نقلی از D14 به D15 و از D15 به خارج وجود داشته باشد پرچم سرریز پایین خواهد بود. اگر از D14 به D15 یا از D15 به بیرون نقلی داشته باشیم ولی هر دو همزمان نباشند OF = 1 است. مثال های ۶-۸ و ۶-۹ ملاحظه شوند.

مثال ۶-۸. نتایج زیر را ملاحظه نمایید

|     |          |                              |
|-----|----------|------------------------------|
| MOV | AX,6E2FH | ; 28,207                     |
| MOV | CX,13D4H | ; + 5,076                    |
| ADD | AX,CX    | ; = 33,283 جواب مورد نظر است |

|        |                     |                                           |
|--------|---------------------|-------------------------------------------|
| 6E2F   | 0110 1110 0010 1111 |                                           |
| + 13D4 | 0001 0011 1101 0100 |                                           |
| 8203   | 1000 0010 0000 0011 | = -32,253 غیر صحیح OF = 1, CF = 0, SF = 1 |

مثال ۶-۹. نتایج حاصل را ملاحظه فرمایید.

|     |          |           |
|-----|----------|-----------|
| MOV | DX,542FH | ; 21,551  |
| MOV | BX,12E0H | ; + 4,832 |
| ADD | DX,BX    | ; =26,383 |

|        |                     |                                               |
|--------|---------------------|-----------------------------------------------|
| 543F   | 0101 0100 0010 1111 |                                               |
| + 12E0 | 0001 0010 1110 0000 |                                               |
| 670F   | 0110 0111 0000 1111 | = 26,383 ( پاسخ صحیح ) OF = 0, CF = 0, SF = 0 |

### پیشگیری از نتایج غلط در عملیات اعداد علامت دار

برای پرهیز از مشکلات ناشی از عملیات اعداد علامت دار می توان علامت عملوند را گسترش داد. در گسترش علامت، بیت علامت D7 از بایت مرتبه پایین تر از یک ثبات، در بیت های بالاتر ثبات و یا بیت علامت ثبات 16 بیتی را در ثبات دیگر کپی می کند. CBW (تبدیل بایت علامت دار به کلمه علامت دار) و CWD (تبدیل کلمه علامت دار به کلمه مضاعف علامت دار) برای اجرای گسترش علامت بکار می روند.

CBW بیت D7 (پرچم علامت) را در تمام بیت های AH کپی می نماید.

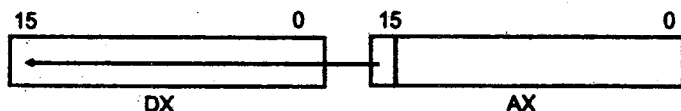
این عمل در زیر نشان داده شده است. توجه کنید که فرض می شود عملوند در AL باشد و نیز محتوای AH نیز از بین می رود.

```
MOV AL,+96 ;AL=0110 0000
CBW ;now AH=0000 0000 and AL=0110 0000
```

```
MOV AL,-2 ;AL=1111 1110
CBW ;AH=1111 1111 و AL=1111 1110
```

یا

**CWD** ، علامت AX را گسترش می دهد. این دستور D15 از AX را در تمام بیت های ثبات DX کپی می نماید. این دستور برای عملوندهایی که اندازه کلمه ای را دارا هستند بکار می رود. در زیر این عمل نشان داده شده است.



به مثال زیر توجه کنید:

```
MOV AX,+260 ;AX=0000 0001 0000 0100 یا AX=0104H
CWD ;DX=0000H and AX=0104H
```

مثالی دیگر:

```
MOV AX,-32766 ;AX=1000 0000 0000 0010B یا AX=8002H
CWD ;DX=FFFF و AX=8002
```

همانطور که در مثال های فوق دیده می شود، **CWD** محتوای AX را تغییر نمی دهد. علامت AX در ثبات DX کپی می شود. چگونه این دستورات یک خطای سرریز صحیح را نشان می دهند؟ برای پاسخ به این سؤال، مثال ۴-۶ در قالب مثال ۱۰-۶ بازنویسی شده و برای اصلاح مشکل سرریز بکار رفته است.

در مثال ۱۰-۶، اگر پرچم سرریز بالا نرود ( $OF = 0$ )، نتایج حاصل از عدد علامت دار صحیح بوده و **JNO** (پرش اگر سرریز وجود ندارد) به **OVER** پرش خواهد کرد. با این وجود اگر  $OF = 1$  باشد، که به معنی نتیجه غلط است، هر عملوند باید گسترش علامت یافته و سپس جمع شود. این عمل نقش کدهای بعد از **JNO** در برنامه است. برنامه مثال ۱۰-۶ برای هر دو عدد علامت دار بایتی کار خواهد کرد.

مثال ۱۰-۶: مثال ۴-۶ را برای اداره شکل سرریز بازنویسی کنید.

حل:

```
DATA1. DB +96
DATA2. DB +70
RESULT DW ?

.....
SUB AH,AH ;AH=0
MOV AL,DATA1 ;GET OPERAND 1
MOV BL,DATA2 ;GET OPERAND 2
ADD AL,BL ;ADD THEM
JNO OVER ;IF OF=0 THEN GO TO OVER
MOV AL,DATA2 ;OTHERWISE GET OPERAND 2 TO
CBW ;SIGN EXTEND IT
MOV BX,AX ;SAVE IT IN BX
MOV AL,DATA1 ;GET BACK OPERAND 1 TO
CBW ;SIGN EXTEND IT
ADD AX,BX ;ADD THEM AND
OVER: MOV RESULT,AX ;SAVE IT
```

در زیر تحلیل مقادیر مثال ۱۰-۶ آورده شده است. هر یک از آنها گسترش علامت داده شده و سپس جمع

شده اند.

| S | AH       | AL        |      |                   |
|---|----------|-----------|------|-------------------|
| 0 | 000 0000 | 0110 0000 | +96  | پس از گسترش علامت |
| 0 | 000 0000 | 0100 0110 | +70  | پس از گسترش علامت |
| 0 | 000 0000 | 1010 0110 | +166 |                   |

در اینجا بعنوان یک قانون چنین می گوئیم: اگر احتمال وقوع سرریز وجود دارد، همه اعداد علامت دار بایتی

باید گسترش علامت شده و به کلمه تبدیل شوند، و نیز همه عملوندهای علامت‌دار کلمه‌ای قبل از پردازش باید گسترش علامت یابند. بزودی این مطلب در برنامه ۱-۶ نشان داده خواهد شد. قبل از بحث در اینمورد، لازم است ضرب و تقسیم علامت‌دار مورد بحث قرار گیرند.

### IDIV (تقسیم عدد علامت‌دار)

راهنمای اینتل چنین بیان می‌دارد که IDIV بمعنی "تقسیم صحیح" است. این دستور برای تقسیم عدد علامت‌دار بکار می‌رود. در واقع همه دستورات محاسباتی 8088/86 بدون توجه به علامت‌دار بودن یا نبودن آنها، برای اعداد صحیح می‌باشند. برای اجرای اعمال روی اعداد حقیقی، پردازنده کمکی 8087 بکار می‌رود. بخاطر داشته باشید که اعداد حقیقی مانند "3.56" دارای بخش اعشاری هستند. تقسیم اعداد علامت‌دار بسیار شبیه اعداد بدون علامت است که در فصل ۳ هم مورد بحث قرار گرفت. جدول ۱-۶ تقسیم عدد علامت‌دار را خلاصه کرده است. این جدول بسیار شبیه جدول ۲-۳ است که در آن تقسیم عدد بدون علامت خلاصه شده است.

جدول ۱-۶. خلاصه تقسیم علامت‌دار

| باقیمانده | خارج قسمت       | مخرج          | صورت            | تقسیم           |
|-----------|-----------------|---------------|-----------------|-----------------|
| AH        | AL              | ثبات یا حافظه | CBW بایت = AL   | بایت / بایت     |
| DX        | AX              | ثبات یا حافظه | CWD کلمه = AX   | کلمه / کلمه     |
| AH        | AL <sup>1</sup> | ثبات یا حافظه | AX = کلمه       | بایت / کلمه     |
| DX        | AX <sup>2</sup> | ثبات یا حافظه | DXAH = جفت کلمه | کلمه / جفت کلمه |

! توجه:

- ۱- وقفه خطای تقسیم اگر  $-127 > AL > +127$
- ۲- وقفه خطای تقسیم اگر  $-32767 > AL > +32767$

|           |                                      |                                |
|-----------|--------------------------------------|--------------------------------|
| TITLE     | PROG 6-1                             | FIND THE AVERAGE TEMPERATURE   |
| PAGE      | 60,132                               |                                |
| STSEG     | SEGMENT                              |                                |
|           | DB 64 DUP (?)                        |                                |
| STSEG     | ENDS                                 |                                |
| DTSEG     | SEGMENT                              |                                |
| SIGN_DAT  | DB +13,-10,+19,+14,-18,-9,+12,-9,+16 |                                |
|           | ORG 0010H                            |                                |
| AVERAGE   | DW ?                                 |                                |
| REMAINDER | DW ?                                 |                                |
| DTSEG     | ENDS                                 |                                |
| CDSEG     | SEGMENT                              |                                |
| MAIN:     | PROC FAR                             |                                |
|           | ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG    |                                |
|           | MOV AX,DTSEG                         |                                |
|           | MOV DS,AX                            |                                |
|           | MOV CX,9                             | :LOAD COUNTER                  |
|           | SUB BX,BX                            | :CLEAR BX, USED AS ACCUMULATOR |
|           | MOV SI,OFFSET SIGN_DAT               | :SET UP POINTER                |
| BACK:     | MOV AL,[SI]                          | :MOVE BYTE INTO AL             |
|           | CBW                                  | :SIGN EXTEND INTO AX           |
|           | ADD BX,AX                            | :ADD TO BX                     |
|           | INC SI                               | :INCREMENT POINTER             |
|           | LOOP BACK                            | :LOOP IF NOT FINISHED          |
|           | MOV AL,9                             | :MOVE COUNT TO AL              |
|           | CBW                                  | :SIGN EXTEND INTO AX           |
|           | MOV CX,AX                            | :SAVE DENOMINATOR IN CX        |
|           | MOV AX,BX                            | :MOVE SUM TO AX                |
|           | CWD                                  | :SIGN EXTEND THE SUM           |
|           | IDIV CX                              | :FIND THE AVERAGE              |
|           | MOV AVERAGE,AX                       | :STORE THE AVERAGE (QUOTIENT)  |
|           | MOV REMAINDER,DX                     | :STORE THE REMAINDER           |
|           | MOV AH,4CH                           |                                |
|           | INT 21H                              | :GO BACK TO DOS                |
| MAIN      | ENDP                                 |                                |
| CDSEG     | ENDS                                 |                                |
|           | END                                  | MAIN                           |

جدول ۶-۲. خلاصه ضرب علامت‌دار

| نتیجه    | عملوند 2      | عملوند 1      | ضرب         |
|----------|---------------|---------------|-------------|
| $AX^1$   | ثبات یا حافظه | AL            | بایت × بایت |
| $DXAX^2$ | ثبات یا حافظه | AX            | کلمه × کلمه |
| $DXAX^3$ | ثبات یا حافظه | AL = CBW بایت | بایت × کلمه |

توجه:

- ۱- اگر  $CF = 1$  و  $OF = 1$ ، بخشی از نتیجه را خواهد داشت، ولی اگر نتیجه به قدر کافی بزرگ نباشد تا AH بکار رود، بیت علامت در بیت‌های بکار نرفته کپی شده و CPU پرچم‌های  $CF = 0$  و  $OF = 0$  را برای آن ایجاد می‌کند.
- ۲- اگر DX بخشی از نتیجه را دارا باشد  $CF = 1$  و  $OF = 1$  بوده است. ولی اگر نتیجه به قدر کافی بزرگ نباشد و بنابراین DX بکار نرود، بیت علامت در بیت‌های بکار نرفته کپی شده و CPU پرچم‌های  $CF = 0$  و  $OF = 0$  را ایجاد خواهد کرد.
- می‌توان از J شرطی برای یافتن یکی از شرایط فوق استفاده کرد. بقیه پرچم‌ها تعریف نشده‌اند.

### IMUL (ضرب عدد علامت‌دار)

ضرب عدد علامت‌دار مشابه ضرب بدون علامتی است که در فصل ۳ مورد بحث قرار گرفت. تنها اختلاف این است که عملوندها در عملیات اعداد علامت‌دار ممکن است مثبت یا منفی باشند. بنابراین نتیجه باید علامت را هم نشان دهد. جدول ۶-۲ ضرب عدد علامت‌دار را خلاصه کرده است. این جدول مشابه جدول ۳-۱ است.

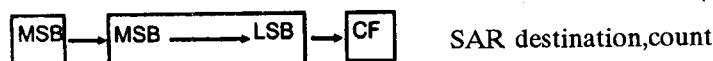
کاربردی از محاسبات عدد علامت‌دار در برنامه ۱-۶ داده شده است. این برنامه متوسط دمای حرارت سلسیوس را برای درجات 13، +، 10، -، 19، +، 14، -، 18، -، 9، +، 12، -، 19، - و 16، + حساب می‌کند. برنامه بطریقی نوشته شده است تا هر گونه سرریزی کنترل شود. در برنامه ۱-۶، هر بایت داده گسترش علامت شده و سپس به BX اضافه شده است تا حاصل جمع که یک کلمه علامت‌دار است، محاسبه شود. آنگاه حاصل جمع و تعداد بایت‌ها گسترش علامت داده شده و حاصل جمع بر تعداد (که در این حالت 9 است) تقسیم شده تا متوسط دما محاسبه شود.

در صفحه بعد فایل "lst" از برنامه فوق داده شده است. توجه کنید که قالب عدد علامت‌دار بوسیله اسمبلر تهیه می‌شود.

### نقل مکان حسابی

همانطور که در فصل ۳ بحث شد، دو نوع نقل مکان (شیفت) وجود دارد: منطقی و حسابی. نقل مکان منطقی که برای اعداد بدون علامت بکار می‌رود قبلاً مورد بحث واقع شدند. نقل مکان حسابی برای اعداد علامت‌دار مورد استفاده قرار می‌گیرد. این دستور در اصل مثل نقل مکان منطقی است با این تفاوت که بیت علامت در بیت‌های جابجا شده کپی می‌شود. SAR (نقل مکان حسابی به راست) و SAL (نقل مکان حسابی به چپ) دو دستوری هستند که برای نقل مکان حسابی در نظر گرفته شده‌اند.

### SAR (نقل مکان حسابی به راست)



با نقل مکان بیت‌های عملوند مقصد به راست و به داخل CF، بیت‌های خالی بابت علامت پر می‌شود. می‌توان از دستورالعمل SAR برای تقسیم عدد علامت‌دار بر 2 استفاده کرد.

مثال ۱۱-۶ کاربرد دستورالعمل SAR را نشان می‌دهد.

```
MOV AL, -10 ;AL=-10=F6H=1111 0110
SAR AL, 1 ;AL is shifted right arithmetic once
 ;AL=1111 1011=FDH=-5
```

```

TITLE PROG6-1 FIND THE AVERAGE TEMPERATURE
PAGE 60,132

0000 STSEG SEGMENT
0000 0040 [DB 64 DUP (?)
 ??
]

0040 STSEG ENDS

0000 DTSEG SEGMENT
0000 0D F6 13 0E EE F7 SIGN_DAT DB +13,-10,+19,+14,-18,-9,+12,-9,+16
 0C F7 10

0010 ORG 0010H
0010 0000 AVERAGE DW ?
0012 0000 REMAINDER DW ?
0014 DTSEG ENDS

0000 CDSEG SEGMENT
0000 MAIN PROC FAR
 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG

0000 B8 -- R MOV AX,DTSEG
0003 8E D8 MOV DS,AX
0005 B9 0009 MOV CX,9 ;LOAD COUNTER
0008 2B DB SUB BX,BX ;CLEAR BX, USED AS ACCUMULATOR
000A BE 0000 R MOV SI,OFFSET SIGN_DAT ;SET UP POINTER
000D 8A 04 BACK: MOV AL,[SI] ;MOVE BYTE INTO AL
000F 98 CBW ;SIGN EXTEND INTO AX
0010 03 D8 ADD BX,AX ;ADD TO BX
0012 46 INC SI ;INCREMENT POINTER
0013 E2 F8 LOOP BACK ;LOOP IF NOT FINISHED
0015 B0 09 MOV AL,9 ;MOVE COUNT TO AL
0017 98 CBW ;SIGN EXTEND INTO AX
0018 8B C8 MOV CX,AX ;SAVE DENOMINATOR IN CX
001A 8B C3 MOV AX,BX ;MOVE SUM TO AX
001C 99 CWD ;SIGN EXTEND THE SUM
001D F7 F9 IDIV CX ;FIND THE AVERAGE
001F A3 0010 R MOV AVERAGE,AX ;STORE THE AVERAGE (QUOTIENT)
0022 89 16 0012 R MOV REMAINDER,DX ;STORE THE REMAINDER
0026 B4 4C MOV AH,4CH
0028 CD 21 INT 21H ;GO BACK TO DOS
002A MAIN ENDP
002A CDSEG ENDS
 END MAIN

```

فایل لیست برای برنامه ۱-۶

### SAL (نقل مکان حسابی به چپ) و SHL (نقل مکان به چپ)

این دو دستور دقیقاً مثل هم عمل می‌کنند. در واقع آن‌ها یک دستور ولی با دو علامت اختصاری هستند از دید اعداد علامت‌دار، نیازی به SAL نیست. برای بحث در مورد SHL (SAL) به فصل ۳ مراجعه کنید.

مثال ۱۱-۶. با استفاده از DEBUG، نتایج حاصله را ارزیابی کنید:

```

MOV AX,-9
MOV BL,2
IDIV BL ;divide -9 by 2 results in FCH
MOV AX,-9
SAR AX,1 ;divide -9 by 2 with arithmetic shift results in FBH

```

حل:

ردیابی DEBUG نشان می‌دهد که تقسیم -9 بر 2، FCH (-4) را نتیجه می‌دهد در حالی که 9 - SAR برابر با FBH (-5) خواهد بود. این بدان علت است که SAR اعداد منفی را به پایین گرد می‌کند ولی IDIV آن‌ها را به بالا گرد می‌نماید.



CMP dest,source

گرچه دستورالعمل CMP (مقایسه) برای اعداد علامت دار و بدون علامت یکی است، ولی دستور J شرطی برای تصمیم گیری روی اعداد علامت دار و بدون علامت متفاوت است. در حالی که در مقایسه اعداد بی علامت، CF و ZF برای شرایط بزرگتر، مساوی یا کوچکتر (فصل ۳ ملاحظه شود) چک می شوند، در اعداد علامت دار، بیت های OF، ZF و SF چک می گردند.

|             |                   |
|-------------|-------------------|
| منبع > مقصد | OF = SF یا ZF = 0 |
| منبع = مقصد | ZF = 1            |
| منبع < مقصد | OF = SF متمم      |

نمادهای اختصاری بکار رفته برای شناسایی شرایط فوق عبارتند از:

|     |                       |                                  |
|-----|-----------------------|----------------------------------|
| JG  | Jump Greater          | jump if OF=SF or ZF=0            |
| JGE | Jump Greater or Equal | jump if OF=SF                    |
| JL  | Jump Less             | jump if OF=inverse of SF         |
| JLE | Jump Less or Equal    | jump if OF=inverse of SF or ZF=1 |
| JE  | Jump if Equal         | jump if ZF = 1                   |

مثال ۶-۱۲ می تواند به درک چگونگی تغییر پرچم های شرایط با دستور مقایسه کمک کند. برنامه ۲-۶ مثالی از کاربرد مقایسه عدد علامت دار است. این برنامه داده های برنامه ۱-۸ را بکار می برد تا پایین ترین دما را پیدا کند.

مفهوم حساب عدد علامت دار بسیار اهمیت داشته و بطور وسیعی مورد استفاده است و حتی ریزپردازنده های RISC که از مجموعه دستورات خط سیری بهره می برند این دستورات را نمی توانند حذف کنند

مثال ۶-۱۲: ردیابی DEBUG را برای دستورات مقایسه اعداد علامت دار زیر نشان دهید.

|     |       |
|-----|-------|
| MOV | AL,-5 |
| CMP | AL,-9 |
| CMP | AL,-2 |
| CMP | AL,-5 |
| CMP | AL,+7 |

```
C>debug
-a 100
103D:0100 mov al,-5
103D:0102 cmp al,-9
103D:0104 cmp al,-2
103D:0106 cmp al,-5
103D:0108 cmp al,7
103D:010A int 3
103D:010B
-t=100,5
```

حل:

```
AX=00FB BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0102 NV UP DI PL NZ NA PO NC
103D:0102 3CF7 CMP AL,F7
```

```
AX=00FB BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0104 NV UP DI PL NZ NA PO NC
103D:0104 3CFE CMP AL,FE
```

```
AX=00FB BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0106 NV UP DI NG NZ AC PO CY
103D:0106 3CFB CMP AL,FB
```

```
AX=00FB BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0108 NV UP DI PL ZR NA PE NC
103D:0108 3C07 CMP AL,07
```

```
AX=00FB BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=010A NV UP DI NG NZ NA PO NC
103D:010A CC INT 3
```

|          |                                      |   |                                     |
|----------|--------------------------------------|---|-------------------------------------|
| TITLE    | PROG6-2                              | ; | FIND THE LOWEST TEMPERATURE         |
| PAGE     | 60,132                               |   |                                     |
| STSEG    | SEGMENT                              |   |                                     |
|          | DB 64 DUP (?)                        |   |                                     |
| STSEG    | ENDS                                 |   |                                     |
| DTSEG    | SEGMENT                              |   |                                     |
| SIGN_DAT | DB +13,-10,+19,+14,-18,-9,+12,-9,+16 |   |                                     |
|          | ORG 0010H                            |   |                                     |
| LOWEST   | DB ?                                 |   |                                     |
| DTSEG    | ENDS                                 |   |                                     |
| CDSEG    | SEGMENT                              |   |                                     |
| MAIN     | PROC FAR                             |   |                                     |
|          | ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG    |   |                                     |
|          | MOV AX,DTSEG                         |   |                                     |
|          | MOV DS,AX                            |   |                                     |
|          | MOV CX,8                             |   | ;LOAD COUNTER (NUMBER ITEMS - 1)    |
|          | MOV SI,OFFSET SIGN_DAT               |   | ;SET UP POINTER                     |
|          | MOV AL,[SI]                          |   | ;AL HOLDS LOWEST VALUE FOUND SO FAR |
| BACK:    | INC SI                               |   | ;INCREMENT POINTER                  |
|          | CMP AL,[SI]                          |   | ;COMPARE NEXT BYTE TO LOWEST        |
|          | JLE SEARCH                           |   | ;IF AL IS LOWEST, CONTINUE SEARCH   |
|          | MOV AL,[SI]                          |   | ;OTHERWISE SAVE NEW LOWEST          |
| SEARCH:  | LOOP BACK                            |   | ;LOOP IF NOT FINISHED               |
|          | MOV LOWEST,AL                        |   | ;SAVE LOWEST TEMPERATURE            |
|          | MOV AH,4CH                           |   |                                     |
|          | INT 21H                              |   | ;GO BACK TO DOS                     |
| MAIN     | ENDP                                 |   |                                     |
| CDSEG    | ENDS                                 |   |                                     |
|          | END MAIN                             |   |                                     |

برنامه ۶-۲

### سؤالات مروری

- ۱- در یک عملوند ۸ بیتی، بیت .... برای علامت بکار رفته است، در صورتی که در یک عملوند ۱۶ بیتی، بیت ... برای آن استفاده شده است.
- ۲- ۱۶ در BCD را به فرم متمم ۲ در آورید
- ۳- محدوده عملوندهای علامت‌دار بایتی از ... تا ... می‌باشد. محدوده کلمه علامت‌دار از ... تا ... است.
- ۴- تفاوت میان یک نقلی و سرریز چیست.
- ۵- هدف از دستور CBW و CWD چیست. اثر CBW روی  $AL = F6H$  را نشان دهید. اثر CWD را روی  $AX = 124CH$  نشان دهید.
- ۶- دستور ضرب علامت‌دار ... است. دستورالعمل تقسیم علامت‌دار ... است.
- ۷- تفاوت میان دستور SHR (در فصل ۳ بحث شد) و SAR چیست؟
- ۸- برای هر یک از دستورات زیر، شرط پرچم لازم برای هر پرش را مشخص کنید  
(الف) JLE (ب) JG

### بخش ۶-۲: عملیات رشته‌ای و جدولی

در خانواده پردازنده‌های ۸۰۸۶ گروهی از دستورات وجود دارند که به آنها دستورات رشته‌ای می‌گویند. این دستورات قادرند اعمالی را روی رشته‌ای از عملوندها واقع در مکان‌های متوالی حافظه انجام دهند. مثلاً در حالی که دستور CMP می‌تواند فقط دو بایت (یا دو کلمه) را مقایسه کند، دستور CMPS (مقایسه رشته‌ای) قادر است دو آرایه داده واقع در مکان‌های حافظه اشاره شده با ثبات SI و DI را مقایسه کند. این دستورات بسیار قدرتمند بوده و در بسیاری از کاربردها قابل استفاده‌اند و بزودی بررسی خواهند شد.

جدول ۳-۶. خلاصه عملیات رشته‌ای

| دستور             | نماد رمزی | مقصد  | مبدأ  | پیشوند     |
|-------------------|-----------|-------|-------|------------|
| انتقال رشته بایت  | MOVSB     | ES:DI | DS:SI | REP        |
| انتقال رشته کلمه  | MOVSW     | ES:DI | DS:SI | REP        |
| ذخیره رشته بایت   | STOSB     | ES:DI | AL    | REP        |
| ذخیره رشته کلمه   | STOSW     | ES:DI | AX    | REP        |
| بارکردن رشته بایت | LODSB     | AL    | DS:SI | none       |
| بارکردن رشته کلمه | LODSW     | AX    | DS:SI | none       |
| مقایسه رشته بایت  | CMPSB     | ES:DI | DS:SI | REPE/REPNE |
| مقایسه رشته کلمه  | CMPSW     | ES:DI | DS:SI | REPE/REPNE |
| پویش رشته بایت    | SCASB     | ES:DI | AL    | REPE/REPNE |
| پویش رشته کلمه    | SCASW     | ES:DI | AX    | REPE/REPNE |

استفاده از SI و DI، DS و ES در دستورات رشته‌ای

برای عملیات رشته‌ای طراحان CPU باید ثبات‌های معینی را برای توابع مشخصی کنار بگذارند. این ثبات‌ها باید مرتباً عملوندهای مبدأ و مقصد را فراهم سازند. این دقیقاً همان کاری است که 80x86 انجام داده است. در ریز پردازنده‌های 8088/86، ثبات‌های SI و DI همیشه بترتیب به عملوندهای مبدأ و مقصد اشاره دارند. حال سؤال این است که، کدام قطعه‌ها برای تولید آدرس فیزیکی 20 بیتی با هم ترکیب می‌شوند؟ برای تولید آدرس فیزیکی، 8088/86 همیشه از SI بعنوان تفاوت مکان ثبات DS (قطعه داده) و DI بعنوان تفاوت مکان ES (قطعه اضافی) استفاده می‌کند. این مد پیش فرض می‌باشد. باید متذکر شد که ثبات ES باید برای عملیات رشته‌ای مقدار دهی اولیه شود.

عملوندهای بایتی و کلمه‌ای در دستورات رشته‌ای

در هر یک از دستورات رشته‌ای، عملوند می‌تواند یک بایت یا یک کلمه باشد، عملوندها در دستور بوسیله حروف B (بایت) و W (کلمه) مشخص می‌شوند. جدول ۳-۶ خلاصه‌ای از همه دستورات رشته‌ای فراهم کرده است. هر یک از آنها در مثال‌های متن مورد بحث قرار خواهند گرفت.

پرچم جهت، DF

در پردازش عملوندهای واقع در مکان‌های متوالی حافظه نیاز به افزایش یا کاهش اشاره گر است. این عمل در عملیات رشته‌ای با پرچم جهت ممکن می‌شود. از 16 بیت ثبات پرچم (D0-D15)، بیت 11 (D10) برای جهت (DF) در نظر گرفته شده است. این به عهده دستور رشته‌ای است تا اشاره گرهای SI و DI را افزایش یا کاهش دهد، ولی انتخاب افزایش یا کاهش به عهده برنامه‌نویس می‌باشد که طی آن پرچم جهت را در سطح بالا یا پایین قرار دهد. دستورالعمل CLD (پاک کردن پرچم جهت) و STD (برقراری پرچم جهت) خصوصاً برای این کار طراحی شده‌اند.

CLD موجب پاک شدن (قراردادن صفر) DF شده و بمعنی افزایش خودکار اشاره گر دستور رشته است. این افزایش خودکارگاهی خود افزایش خوانده می‌شود.

STD (برقرار کردن پرچم جهت) عملکرد مخالفی را نسبت به دستور CLD انجام می‌دهد. این دستور DF را در 1 می‌نشانند و بمعنی این است که اشاره گرهای SI و DI مربوط به دستور رشته‌ای باید بطور خودکار کاهش یابند.

## پیشوند REP

پیشوند REP (تکرار) اجازه می‌دهد تا یک دستور رشته‌ای عملیات را تکرار کند. اکنون سؤال این است که چند بار تکرار می‌شود؟ REP فرض می‌کند که CX تعداد دفعات تکرار دستورالعمل را نگه می‌دارد. به بیان دیگر، پیشوند REP به CPU اجرای دستور رشته‌ای را فرمان داده و سپس ثبات CX را بطور خودکار کاهش می‌دهد. عملیات تا صفر شدن CX ادامه می‌یابد. برای درک بعضی از مفاهیم مورد بحث، به مثال ۶-۱۳ توجه کنید.

**مثال ۶-۱۳** با استفاده از دستورات رشته‌ای، برنامه‌ای بنویسید که بلوکی از ۲۰ بایت داده را انتقال دهد.

**حل:**

در قطعه داده:

```
DATA1 DB ABCDEFGHIJKLMNOPQRST
 ORG 30H
DATA2 DB 20 DUP (?)
```

در قطعه کد:

```
ASSUME CS:CODESEG,DS:DATSEG,SS:STASG,ES:DATSEG
MOV AX,DATSEG
MOV DS,AX ;INITIALIZE THE DATA SEGMENT
MOV ES,AX ;INITIALIZE THE EXTRA SEGMENT
CLD
MOV SI,OFFSET DATA1 ;LOAD THE SOURCE POINTER
MOV DI,OFFSET DATA2 ;LOAD THE DESTINATION POINTER
MOV CX,20 ;LOAD THE COUNTER
REP MOVSB ;REPEAT UNTIL CX BECOMES ZERO
```

در مثال ۶-۱۳، پس از انتقال هر بایت بوسیله دستورالعمل MOVSB، هر دو ثبات SI و DI بطور خودکار فقط یکبار افزایش می‌یابند (به CLD توجه کنید) پیشوند REP موجب می‌شود تا شمارنده CX کاهش یافته و MOVSB تا صفر شدن CX تکرار می‌گردد. توجه کنید که در عبارت ASSUME هر دو ثبات DS و ES تعریف شده‌اند. بجای ES می‌توان قطعه متفاوتی بکار برد. در این حال E باید در برنامه‌نویسی و عبارت ASSUME مقداردهی اولیه شود.

راه‌حل دیگری برای مثال ۶-۱۳ تنها دو خط از برنامه را تغییر می‌دهد:

```
MOV CX,10
REP MOVSW
```

در این حال MOVSW یک کلمه (۲ بایت) را هر بار انتقال داده و هر یک از ثبات‌های SI و DI دوبار افزایش می‌یابند. REP عملیات را تا صفر شدن CX ادامه می‌دهد. توجه کنید که CX حاوی ۱۰ است زیرا ۱۰ کلمه برابر با ۲۰ بایت است.

## دستورالعمل‌های STOS و LODS

دستورالعمل STOSB بایت موجود در ثبات AL را در حافظه‌ای که بوسیله ES:DI اشاره می‌شود ذخیره می‌نماید و یکبار DI را (اگر  $DF = 0$  باشد) اضافه می‌کند. اگر  $DF = 1$  بود در اینصورت DI کم می‌گردد. دستور STOSW محتوای AX را در مکان‌های حافظه ES:DI و  $ES:DI + 1$  ذخیره می‌نماید (AL در ES:DI و AH در  $ES:DI + 1$ ) سپس DI را دوباره افزایش می‌دهد (اگر  $DF = 0$  باشد). در صورتی که  $DF = 1$  باشد، DI دوباره کم می‌شود.

دستورالعمل LODSB محتوای حافظه اشاره شده با SI:DS را در AL بار کرده و اگر  $DF = 0$  یا  $DF = 1$  باشد SI را یکواحد افزایش (یا کاهش) می‌دهد. LODSW محتوای حافظه‌های اشاره شده با SI:DS را در AL و  $SI + 1$  در AH بار می‌نماید. SI، اگر  $DF = 0$  باشد، دوباره افزایش می‌یابد. در غیراینصورت دوبار کاهش خواهد یافت. LODS هیچوقت همراه با پیشوند REP بکار نمی‌رود.

## تست حافظه با STOSB و LODSE

مثال ۱۴-۶ دستورات رشته‌ای STOSB و LODSB را برای تست یک ناحیه از حافظه RAM بکار می‌برد.

مثال ۱۴-۶. برنامه‌ای بنویسید که:

(۱) STOSB را برای ذخیره بایت AAH در ۱۰۰ مکان حافظه بکار ببرید.

(۲) LODSB را برای تست محتوای هر مکان بکار برده و ببینید آیا AAH در آنجاست. اگر تست موفق نباشد سیستم باید پیام "bad memory" را نمایش دهد.

حل:

```

;PUT PATTERN AAAAH IN TO 50 WORD LOCATIONS
MOV AX,DTSEG ;INITIALIZE
MOV DS,AX ;DS REG
MOV ES,AX ;AND ES REG
CLD ;CLEAR DF FOR INCREMENT
MOV CX,50 ;LOAD THE COUNTER (50 WORDS)
MOV DI,OFFSET MEM_AREA ;LOAD THE POINTER FOR DESTINATION
MOV AX,0AAAAH ;LOAD THE PATTERN
REP STOSW ;REPEAT UNTIL CX=0
;BRING IN THE PATTERN AND TEST IT ONE BY ONE
MOV SI,OFFSET MEM_AREA ;LOAD THE POINTER FOR SOURCE
MOV CX,100 ;LOAD THE COUNT (COUNT 100 BYTES)
AGAIN: LODSB ;LOAD INTO AL FROM DS:SI
 XOR AL,AH ;IS PATTERN THE SAME?
 JNZ OVER ;IF NOT THE SAME THEN EXIT
 LOOP AGAIN ;CONTINUE UNTIL CX=0
 JMP EXIT ;EXIT PROGRAM
OVER: MOV AH,09 ;{ DISPLAY
 MOV DX,OFFSET MESSAGE ;{ THE MESSAGE
 INT 21H ;{ ROUTINE
EXIT: ...

```

فرض کنید که ES و DS به رهنمون ASSUME تخصیص یافته باشند. در زیر قطعه کد آورده شده است. در برنامه مثال ۱۴-۶، ابتدا AAH در ۱۰۰ مکان بکمک عملوند کلمه‌ای AAAAH و شماره ۵۰ نوشته می‌شود. در بخش تست، LODSB محتوای مکان‌های حافظه را یک به یک به درون AL می‌آورد و هر بار با AAH ، OR انحصاری می‌شود (ثبات AH مقدار AAH را دارد). اگر آنها یکی باشند  $ZF = 1$  گشته و پردازش ادامه می‌یابد. در غیر اینصورت الگوی نوشته شده در آنجا نبوده و برنامه خارج می‌گردد. این مطلب تا حدودی شبیه روالی است که در IBM PC BIOS بکار رفته است با این تفاوت که BIOS بیشتر درگیر شده و الگوهای متفاوتی از داده را برای تست بر می‌دارد و قادر است هر بخش از RAM یعنی RAM اصلی یا RAM تصویر را تست نماید.

## پیشوندهای REPZ و REPNZ

این پیشوندها قابل استفاده با دستورات CMPS و SCAS برای اهداف تست می‌باشند. در زیر هر یک از این پیشوندها توضیح داده می‌شوند.

**REPZ** (تکرار اگر صفر)، که شبیه REPE (تکرار اگر مساوی) است رشته عملیات مادامی که عملوندهای مبدأ و مقصد برابر ( $ZF = 1$ ) باشند و یا تا هنگامی که CX صفر شود ادامه می‌دهد.

**REPNZ** (تکرار اگر غیر صفر) که مثل REPNE (تکرار اگر غیر مساوی) می‌باشد و رشته عملیات اگر عملوندهای مبدأ یکی نیستند ( $ZF=0$ ) یا تا CX صفر شود ادامه می‌یابد این دو پیشوند پس از توضیحات دستورات CMPS و SCANS بکار برده خواهند شد.

**CMPS** (مقایسه رشته) اجازه می‌دهد تا مقایسه‌ای بین دو آرایه داده که بوسیله ثبات‌های SI و DI اشاره می‌شوند صورت گیرد. می‌توان برابری یا نابرابری داده را با استفاده از پیشوندهای REPE یا REPNE مشخص کرد. مقایسه با بکارگیری CMPSB یا CMPSW بایت به بایت یا کلمه به کلمه انجام شود.

مثلاً اگر "Euorop" و "Europe" برای برابری مقایسه شوند، مقایسه با REPE CMPS انجام شده و تا زمانی که دو آرایه یکی باشند ادامه می‌یابد.  
از قطعه داده:

DATA1 DB 'Europe'  
DATA2 DB 'Euorop'

از قطعه کد:

|      |                 |                                 |
|------|-----------------|---------------------------------|
| CLD  |                 | ;DF=0 for increment             |
| MOV  | SI,OFFSET DATA1 | ;SI=DATA1 offset                |
| MOV  | DI,OFFSET DATA2 | ;DI=DATA2 offset                |
| MOV  | CX,06           | ;load the counter               |
| REPE | CMPSB           | ;repeat until not equal or CX=0 |

در مورد فوق، دو آرایه حرف به حرف مقایسه می‌شوند. ابتدا اولین کاراکترهایی که بوسیله SI, DI اشاره می‌شوند مقایسه خواهند شد. در این حال آنها یکسانند ('E') بنابراین پرچم Z در وضعیت 1 قرار خواهد گرفت و هر دو ثبات SI و DI افزایش می‌یابند. چون  $ZF = 1$  است، پیشوند REPE مقایسه را تکرار خواهد کرد. پردازش تا سومین حرف ادامه می‌یابد. سومین حروف "o" و "r" یکسان نیستند، بنابراین ZF به صفر باز نشاندن می‌شود و مقایسه متوقف خواهد شد. می‌توان از ZF برای تصمیم‌گیری استفاده کرد. مثال ۱۵-۶ را ملاحظه کنید.

مثال ۱۵-۶. فرض کنید می‌خواهیم "Europe" را در فرهنگ الکترونیک هجی کنیم و کاربری "Euorope" را تایپ می‌کند، برنامه‌ای بنویسید که این دو را مقایسه کرده و پیام‌های زیر را بسته به نتیجه نمایش دهد.  
۱- اگر مساوی باشند پیام "The spelling is correct" نمایش داده شود  
۲- اگر مساوی نباشد پیام "Wrong spelling" نمایش داده شود.

حل:

DAT\_DICT DB 'Europe'  
DAT\_TYPED DB 'Euorope'  
MESSAGE1 DB 'The spelling is correct','\$'  
MESSAGE2 DB 'Wrong spelling','\$'

از قطعه کد:

|          |                     |                                        |
|----------|---------------------|----------------------------------------|
| CLD      |                     | ;DF=0 FOR INCREMENT                    |
| MOV      | SI,OFFSET DAT_DICT  | ;SI=DATA1 OFFSET                       |
| MOV      | DI,OFFSET DAT_TYPED | ;DI=DATA2 OFFSET                       |
| MOV      | CX,06               | ;LOAD THE COUNTER                      |
| REPE     | CMPSB               | ;REPEAT AS LONG AS EQUAL OR UNTIL CX=0 |
| JE       | OVER                | ;IF ZF=1 THEN DISPLAY MESSAGE1         |
| MOV      | DX,OFFSET MESSAGE2  | ;IF ZF=0 THEN DISPLAY MESSAGE2         |
| JMP      | DISPLAY             |                                        |
| OVER:    | MOV                 | DX,OFFSET MESSAGE1                     |
| DISPLAY: | MOV                 | AH,09                                  |
|          | INT                 | 21H                                    |

می‌توان برنامه مثال ۱۵-۶ را با کمی دستکاری و افزودن دستورات JMP کاراتر نمود ولی بخاطر سادگی به فرم فوق ارائه شد.

از CMPS می‌توان برای نابرابری آرایه‌ها نیز استفاده کرد که در این صورت "REPNE CMPS" بکار می‌رود. مثلاً وقتی شماره‌های شناسایی زیر را مقایسه کنیم، مقایسه تا آخرین رقم ادامه می‌یابد زیرا هیچ دو رقمی در دو مکان مساوی یکسان نیستند.

231-24-7656

564-77-1338

## SCAS (پویش رشته)

دستور رشته SCASB هر بایت از آرایه اشاره شده با ES:DI را با محتوای ثبات AL مقایسه و بسته به

پیشوند REPE یا REPNE تصمیمی برای برابری یا نابرابری گرفته می‌شود. مثلاً در آرایه "Mr.Jones" می‌توان با بار کردن حرف "J" در ثبات AL حرف "J" را پویش کرد، و سپس با استفاده از REPNE SCASB بدنبال حرف مزبور گشت.  
در قطعه داده:

DATA1 DB 'Mr. Gones'

و در قطعه کد:

```
CLD ;DF=0 FOR INCREMENT
MOV DI,OFFSET DATA1 ;DI=ARRAY OFFSET
MOV CX,09 ;LENGTH OF ARRAY
MOV AL,'G' ;SCANNING FOR THE LETTER 'G'
REPNE SCASB ;REPEAT THE SCANNING IF NOT EQUAL OR
 ;UNTIL THE CX IS ZERO
```

در مثال فوق، حرف G با "M" مقایسه شده است. چون آنها برابر نیستند، DI افزایش و CX کاهش می‌یابد، و عمل پویش تا رسیدن حرف "G" یا صفر شدن CX ادامه می‌یابد. در این مثال، چون "G" یافت می‌شود، ZF = 1 شده و بیان می‌کند که "G" در آرایه وجود دارد.

### جایگزینی کاراکتر پویش یافته

SCASB را می‌توان برای جستجوی هر کاراکتر در یک آرایه بکار برد، و اگر کاراکتر پیدا شد با یک کاراکتر مورد نظر جایگزینی می‌شود. مثال ۱۶-۶ ملاحظه گردد.

**مثال ۱۶-۶.** برنامه‌ای بنویسید که نام "MR.GONES" را یافته و حرف "G" را با حرف "J" جایگزین کند. سپس نام اصلاح شده را نمایش دهد.

**حل:**

در برنامه زیر تعلق ES و DS به رهنمون ASSUME فرض شده است  
در قطعه داده،

DATA1 DB 'Mr. Gones','\$'

در قطعه کد

```
MOV AX,DTSEG
MOV DS,AX
MOV ES,AX
CLD
MOV DI,OFFSET DATA1 ;DF=0 FOR INCREMENT
MOV CX,09 ;ES:DI=ARRAY OFFSET
MOV AL,'G' ;LENGTH OF ARRAY
REPNE SCASB ;SCANNING FOR THE LETTER 'G'
JNE OVER ;REPEAT THE SCANNING IF NOT EQUAL OR
 ;UNTIL CX IS ZERO. JUMP IF Z=0
DEC DI ;DECREMENT TO POINT AT 'G'
MOV BYTE PTR [DI],'J' ;REPLACE 'G' WITH 'J'
OVER: MOV AH,09 ;DISPLAY
MOV DX,OFFSET DATA1 ;THE
INT 21H ;CORRECTED NAME
```

در عملیات رشته‌ای، پس از هر اجرا اشاره‌گر افزایش می‌یابد (اگر DF = 0 باشد). بنابراین در مثال فوق، DI باید کم شود تا بدینوسیله اشاره‌گر به کاراکتر پویش شده اشاره نموده و سپس آن را جایگزین نماید.

### دستور XLAT و جداول نظاره

اغلب در کاربردهای کامپیوتر نیاز به جداولی است که برخی از اطلاعات مهم را نگه می‌دارند. برای دستیابی به اجزاء جدول، ریزپردازنده‌های 8088/86 دستور XLAT (ترجمه) را فراهم نموده‌اند. برای درک دستور XLAT

ابتدا باید جداول فهمیده شوند. جدول معمولاً به نام جدول نظاره خوانده می‌شود. فرض کنید جدولی برای مقادیر  $x^2$  لازم باشد که در آن  $x$  بین 0 و 9 تغییر می‌نماید. ابتدا جدول تولید، و در حافظه ذخیره می‌شود.

SQUR\_TABLE DB 0,1,4,9,16,25,36,49,64,81

اکنون می‌توان با استفاده از XLAT به مربع اعداد 0 تا 9 دسترسی پیدا کرد. برای انجام آن، BX باید آدرس تفاوت مکان جدول نظاره را داشته باشد و عددی که مربعش مورد نظر است نیز در ثبات AL قرار گیرد. آنگاه پس از اجرای XLAT، ثبات AL مربع عدد را خواهد داشت. برنامه زیر چگونگی بدست آوردن مربع 5 را نشان می‌دهد.

```
MOV BX,OFFSET SQUR_TABLE ;load the offset address of table
MOV AL,05 ;AL=05 will retrieve 6th element
XLAT ;pull out of table the element
 ;and put in AL
```

پس از اجرای این برنامه ثبات AL دارای 25 (19H) که مربع 5 است خواهد بود. باید متذکر شد برای این که XLAT عمل بکند واردهای جدول نظاره باید بترتیب بوده و رابطه یک به یکی با خود عنصر داشته باشند. این بدان علت است که طریق کار کردن XLAT چنین است. در واقع XLAT دستورالعملی است که معادل برنامه زیر می‌باشد.

```
SUB AH,AH ;AH=0
MOV SI,AX ;SI=000X
MOV AL,[BX+SI] ;GET THE SIth ENTRY FROM BEGINNING
 ;OF THE TABLE POINTED AT BY BX
```

به بیان دیگر، اگر دستورالعمل XLAT نمی‌بود، برنامه فوق می‌باید همان کار را می‌کرد و این همان کاری است که RISC انجام داده است تا این عمل را انجام دهد. سؤال این است که چرا بخواهیم با استفاده از XLAT مربع عددی را از جدول نظاره بدست آوریم در حالیکه می‌توان از دستور MUL استفاده کرد. پاسخ این است که MUL بیشتر طول می‌کشد.

### تبدیل کد بکممک XLAT

در بسیاری از سیستم‌های مبتنی بر ریزپردازنده، صفحه کلید از نوع اسکی نیست. می‌توان از دستور XLAT برای تبدیل کلید چنین صفحه کلیدهایی به اسکی استفاده کرد. فرض کنید که کلیدها از 0-F باشند. در زیر برنامه تبدیل ارقام مبنای شانزده به معادل اسکی آورده شده است.

قطعه داده:

```
ASC_TABL DB '0','1','2','3','4','5','6','7','8'
 DB '9','A','B','C','D','E','F'
HEX_VALU DB ?
ASC_VALU DB ?
```

قطعه کد:

```
MOV BX,OFFSET ASC_TABL ;BX= TABLE OFFSET
MOV AL,HEX_VALU ;AL=THE HEX DATA
XLAT ;GET THE ASCII EQUIVALENT
MOV ASC_VALU,AL ;MOVE IT TO MEMORY
```



- ۱- در عملیات رشته‌ای، ثبات ... برای اشاره به عملوند مبدأ و ثبات ... برای اشاره به عملوند مقصد بکار می‌روند.
- ۲- SI بعنوان تفاوت مکان در قطعه ..... و DI بعنوان تفاوت مکان در قطعه ..... بکار می‌روند.
- ۳- پرچم .....، بیت ... از ثبات پرچم به CPU می‌گوید که اشاره‌گرها را در عملیات رشته افزایش و یا کاهش دهد.
- ۴- هدف از دستورات CLD و STD چیست؟
- ۵- اگر یک دستورالعمل رشته‌ای چندین بار بدلیل پیشوند REP تکرار شود، CPU چگونه متوقف کردن آن را می‌فهمد.
- ۶- در قطعه برنامه زیر، چه شرایطی موجب عدم موفقیت REP NZ خواهد شد؟

```
MOV SI, OFFSET DATA1
MOV DI, OFFSET DATA2
MOV CX, LENGTH
REP NZ CMPSB
```

## خلاصه

نمایش اعداد علامت‌دار 8088/86 با بکارگیری MSB (با ارزشترین بیت) بعنوان بیت علامت حاصل شد. در عملوند بایستی، بیت علامت، D7 است و در عملوند کلمه‌ای این بیت D15 می‌باشد. بیت علامت صفر بمعنی عدد مثبت و بیت علامت 1 بمعنی عدد منفی است. اعداد منفی بفرم متمم 2 نشان داده می‌شوند. دستورات جمع علامت‌دار و تفریق دستورات یکسانی را با جمع و تفریق بدون علامت بکار می‌برند: این دستورات ADD و SUB هستند. با این وجود ضرب و تقسیم علامت‌دار از دستور IMUL و IDIV بجای MUL و DIV استفاده می‌کنند. در محاسبات اعداد علامت‌دار، برنامه‌نویس باید سر ریز را تست کند. سر ریز هنگامی رخ می‌دهد که یک رقم نقلی به MSB وارد شود و یا یک رقم نقلی بخارج اتفاق بیفتد بدون آنکه رقم نقلی وارد MSB گردد. مسئله سرریز می‌تواند با استفاده از دستورات علامت گسترش یافته CBW و CWD حل شود.

دستورات نقل مکان حسابی مشابه دستورات نقل مکان منطقی کار می‌کنند بجز اینکه دستورات نقل مکان حسابی باید بیت علامت را هم به حساب آورند. بنابراین آن‌ها بیت علامت را در بیت‌های جابجا شده کپی می‌کنند. دستور مقایسه (CMP) برای دستورات علامت‌دار و بدون علامت بطور یکسان کار می‌کند ولی دستورات پرش شرطی متفاوتی پس از CMP بکار گرفته می‌شوند.

80x86 دارای دستورات بسیاری برای کار بر روی داده‌های رشته‌ای است. این دستورات شامل STOS و LODS برای ذخیره و بار کردن داده و دستور پویش SCAS است. عملیات رشته‌ای از ثبات‌های DI و SI بعنوان اشاره‌گر به داده در قطعات داده و قطعه اضافی است. این دستورات می‌توانند با استفاده از انواع پیشوندهای REP تکرار شوند. اینکه با هر تکرار اشاره‌گرهای DI و SI افزایش یا کاهش خواهند یافت به تنظیم پرچم جهت بستگی دارد. و نیز دستورالعملی برای پردازش جدول بنام دستورالعمل XLAT وجود دارد.

## مسائل

- ۱- نشان دهید که کامپیوتر 80x86 چگونه اعداد زیر را نمایش می‌دهد و بکمک DEBUG هر یک را تحقیق کنید.
 

|           |          |          |            |
|-----------|----------|----------|------------|
| (الف) 23- | (ب) 12+  | (پ) 28H- | (ت) 6FH+   |
| (ث) 128-  | (ج) 127+ | (چ) 365+ | (ح) 32767- |
- ۲- برای هر حالت پرچم سرریز را معین کرده و نتیجه را بکمک DEBUG تحقیق کنید.
 

|                     |                       |                   |
|---------------------|-----------------------|-------------------|
| (الف) (-12) + (+15) | (ب) (-127) + (-123)   | (پ) (+34) + (+25) |
| (ت) (+127) + (-127) | (ث) (-1000) + (+1000) |                   |

- ۳- موارد زیر را گسترش علامت دهید و برنامه ساده‌ای در DEBUG برای تحقیق صحت آنها بنویسید.  
 (الف) 122- (ب) 999H- (پ) 17H+ (ت) 127+ (ث) 129-  
 ۴- برنامه ۲-۶ را برای یافتن بالاترین دما اصلاح کنید صحت برنامه خود را تحقیق کنید.  
 ۵- کدام دستور العمل‌ها برای نشان دادن و باز نشان دادن جهت پرچم بکار می‌روند؟ هدف از پرچم جهت چیست.  
 ۶- دستور العمل REP با کدامیک از دستورات زیر بکار می‌رود؟  
 (الف) MOVSB (ب) MOVSW (پ) CMPSB (ت) LODSB (ث) STOSW (ج) SCASW  
 ۷- در مسئله ۶، عملوندهای مبدأ و مقصد را برای هر دستور مشخص نمایید.  
 ۸- برنامه‌ای نوشته و آن را برای انتقال 200 کلمه داده تست کنید.  
 ۹- دستورات LODSX و STOSX را برای پوشش 3 رقم اسکی از 50 رقم موجود بکار ببرید و سپس نتیجه را به مکان متفاوتی در حافظه منتقل نمایید. آنگاه از اسکی به BCD فشرده نشده تبدیل نموده و حاصل را در مکان دیگری ذخیره کنید. مثلاً

| مبدأ          | مقصد                   |
|---------------|------------------------|
| ASCII for '5' | 0011 0101    0000 0101 |

- ۱۰- کدام پیشوند برای حالت نابرابری با دستورات CMPS و SCAS بکار می‌روند.  
 ۱۱- برنامه‌ای بنویسید که "IbM" را پوشش کرده و حرف کوچک "b" را با حرف بزرگ "B" جایگزین کند.  
 ۱۲- با توجه به زمان بندی در پیوست ۲- ب تعداد پالس‌های ساعت دستور العمل XLAT را بشمارید و آن را با معادله مقایسه نمایید تا مشخص شود کدام یک کاراترند.  
 ۱۳- برنامه‌ای را با استفاده از یک جدول نظاره و XLAT بنویسید تا مقدار y را در معادله  $y = x^2 + 2x + 5$  بازاء مقادیر x از 0 تا 9 بازیابی کند.

## ✓ پاسخ به سؤالات مروری

- بخش ۱-۶: عملیات حسابی اعداد علامت‌دار  
 ۱- d15, d7  
 ۲-  $16H = 0001\ 0110_2$  و در فرم متمم 2 برابر است با  $1110\ 1010_2$   
 ۳- 128- تا 127+ ؛ 32768- تا 32767+ (دهدهی)  
 ۴- سرریز، یک رقم نقلی به بیت علامت است؛ یک نقلی، نقلی خروجی از ثبات است.  
 ۵- دستور CBW یک بایت را گسترش علامت داده آن را به کلمه تبدیل می‌کند؛ دستور CWD یک کلمه را گسترش علامت داده آن را به جفت کلمه تبدیل می‌نماید.  
 F6H گسترش علامت یافته به  $AX = FFF6$  بدل می‌شود  
 124C در DXAX گسترش علامت یافته و به  $DX = 0000$  و  $AX = 124CH$  تبدیل می‌شود.  
 ۶- IDIV, IMUL  
 ۷- SHR هر بیت را یک مکان به راست جابجا کرده و MSB را با صفر پر می‌نماید  
 SAR هر بیت را یک مکان به راست جابجا کرده و MSB را با بیت علامت پر می‌کند در هر کدام؛ LSB به پرچم نقلی جابجا می‌شود  
 ۸- (الف) JLE بشرطی پرش خواهد کرد که OF معکوس SF یا اگر  $ZF = 1$  باشد.  
 (ب) JG بشرطی پرش خواهد کرد که OF برابر SF یا اگر  $ZF = 0$  باشد.

## بخش ۲-۶: عملیات رشته‌ای و جدولی

- ۱- DI, SI  
 ۲- داده، اضافی  
 ۳- پرچم جهت، بیت 11 یا D10  
 ۴- DF, CLD را صفر می‌کند؛ STD, DF را 1 می‌کند  
 ۵- وقتی  $CX = 0$   
 ۶- اگر  $CX = 0$  و یا نقطه‌ای که در آن DATA1 و DATA2 مساوی نباشند.



## مدول‌ها: برنامه‌نویسی مدولی و C

### اهداف

- در پایان این فصل قادر خواهید بود:
- « مزایای برنامه‌نویسی مدولی را بحث کنید.
  - « برنامه‌های بزرگ را به مدول‌ها بشکنید و مدول‌ها و برنامه‌های فراخوانی را برنامه‌نویسی نمایید.
  - « نام‌های تعریف شده در دیگر مدول‌ها را با رهنمون EXTRN مشخص کنید.
  - « نام‌های بکار رفته در دیگر مدول‌ها را با رهنمون PUBLIC مشخص نمایید.
  - « زیر برنامه‌ها را به هم پیوند بزنید تا یک برنامه اجرایی حاصل گردد.
  - « رهنمون‌های قطعه کد، برای پیوند قطعات داده، کد یا پشته از مدول‌های دیگر را در یک قطعه بکار ببرید.
  - « برنامه‌هایی با استفاده از تعاریف قطعه کامل بنویسید.
  - « روش‌های مختلف تحویل پارامترها را به مدول فهرست کرده و مزیت و عیب هر یک را بحث نمایید.
  - « برنامه‌ای جهت تحویل پارامترها از طریق ثبات‌ها، حافظه یا پشته بنویسید.
  - « برنامه‌ای بزبان اسمبلی به کمک روش درون برنامه‌ای در C بنویسید.
  - « مدول‌هایی بزبان اسمبلی برای برنامه‌های C بنویسید.
  - « از فراخوانی متعارف C برای تحویل پارامتر استفاده نمایید.
  - « مدول‌های زبان اسمبلی را با برنامه‌های C پیوند بزنید.

در این فصل مفهوم مدول‌ها همراه با قوانین نوشتن و پیوند آنها ارائه شده است. برخی از مدول‌های مفید همراه با روش‌های تحویل پارامترها در میان مدول‌ها داده شده است. در بخش پایانی ترکیب برنامه‌های C و اسمبلی را نشان خواهیم داد.

## بخش ۷-۱: نوشتن و پیوند مدول‌ها

### چرا مدول‌ها؟

در نوشتن بسته‌های نرم‌افزاری رسم بر این است که طرح به مدول‌های کوچکتر شکسته شده و نوشتن این مدول‌ها در میان چندین برنامه‌نویس تقسیم شود. این کار نه تنها موجب مدیریت بهتر طرح می‌شود بلکه مزایای دیگری نیز دارد، از جمله:

- ۱- هر مدول قابل نوشتن، عیب‌یابی و تست جداگانه است.
  - ۲- خرابی یک مدول موجب توقف کل پروژه نمی‌گردد.
  - ۳- استقرار و جدایی وظیفه‌های هر برنامه ساده‌تر و کمتر وقت‌گیر است.
  - ۴- می‌توان مدول‌ها را به زبان‌های سطح بالایی همچون C، پاسکال یا BASIC پیوند زد.
  - ۵- طراحی موازی، زمان تکمیل پروژه را بطور مؤثری کاهش می‌دهد.
- در این بخش چگونگی نوشتن و پیوند مدول‌ها را برای ایجاد یک برنامه اجرایی توضیح می‌دهیم.

### نوشتن مدول‌ها

در فصل‌های قبل رویه نوشته شده اصلی چند زیر روال دیگر را فراخوانی می‌کرد. در مثال‌های مربوط به آنها، اگر زیر روالی صحیح کار نمی‌کرد تمام برنامه باید بازنویسی و مجدداً اسمبل می‌شد. راه کاراتر طراحی نرم‌افزار این است که با هر زیر روال، به عنوان یک برنامه جدا همراه با نام جداگانه برخورد شود. آنگاه هر یک قابل اسمبل و تست شدن هستند. پس از تست هر یک و اطمینان از عملکرد صحیح آن، می‌توان همه آنها را یکپارچه کرده و یک برنامه بوجود آورد. برای اینکه این مدول‌ها بهم پیوند بخورند از رهنمون‌های خاصی در زبان اسمبلی باید استفاده کرد. از جمله این رهنمون‌ها می‌توان از EXTRN (بیرونی)، و PUBLIC نام برد. هر کدام در زیر توضیح داده شده است.

### رهنمون EXTRN

وظیفه رهنمون EXTRN این است تا به اسمبلی و پیوندزن بفهماند که نام‌ها و متغیرهایی که در مدول جاری تعریف نشده‌اند، در جایی بیرون از آن تعریف شده‌اند. در غیاب رهنمون EXTRN، اسمبلی خطا اعلام خواهد کرد زیرا نمی‌تواند جایی که نام‌ها تعریف شده‌اند را بیابد. رهنمون EXTRN، دارای قالب زیر است:

```
EXTRN name1:type ;each name can be in a separate EXTRN
EXTRN name2:type
EXTRN name1:type,name2:type ;or many can be listed in the same EXTRN
```

نام رویه‌های بیرون می‌تواند NEAR، FAR یا PROC باشند. این نام‌ها برای مدول‌های کوچک NEAR و برای مدول‌های بزرگتر FAR خواهد بود. در زیر نمونه‌هایی از نام داده‌ها و تعداد بایت‌های آنها در پرانتز داده شده‌اند: (1) BYTE، (2) WORD، (4) DWORD، (6) FWORD، (8) QWORD یا (10) TBYTE

### رهنمون PUBLIC

نام‌ها یا پارامترهایی که تحت EXTRN تعریف شده‌اند (بیانگر خارج از مدول فعلی) باید تحت PUBLIC

در مدولی که تعریف شده‌اند معرفی شوند. تعریف نامی بصورت PUBLIC به اسمبلی و پیوندزن اجازه تطبیق با همتای EXTRN خود می‌دهد. در زیر قالب رهنمون PUBLIC آمده است.

**PUBLIC name1** ;each name can be in a separate directive  
**PUBLIC name2**

**PUBLIC name1, name2** ;or many can be listed in the same PUBLIC

مثال ۷-۱ به درک این مفاهیم کمک می‌کند. این مثال نشان می‌دهد که برای هر تعریف EXTRN یک رهنمون PUBLIC در مدول دیگر وجود دارد. در این مثال رهنمون‌های EXTRN و PUBLIC به نام یک رویه FAR مربوط شده‌اند.

### رهنمون END در مدول‌ها

در مثال ۷-۱، به نقاط ورود و خروج توجه نمایید. نقطه ورود MAIN و خروج نیز END MAIN است. مدول‌هایی که بوسیله مدول اصلی فراخوانی می‌شوند دارای رهنمون END ولی فاقد نام، پس از آن هستند. دقت کنید که هر یک از SUBPROG1 و SUBPROG2 دارای رهنمون END ولی بدون پرچسب پس از آن می‌باشند.

**مثال ۷-۱.** فرض کنید برنامه‌ای وجود دارد که متشکل از یک روال اصلی همراه با دو زیر روال با نام‌های SUBPROG1 و SUBPROG2 می‌باشند. زیر روال‌ها از روال اصلی فراخوانی می‌شوند. در زیر کاربرد EXTRN و PUBLIC نشان داده شده است.  
**حل:**

```

;-----
;one file will contain the main module:
EXTRN SUBPROG1:FAR
EXTRN SUBPROG2:FAR
CODSG7_1 SEGMENT
MAIN PROC FAR
ASSUME CS:CODSG7_1,DS:.....,SS:.....,ES:....
...
CALL SUBPROG1
CALL SUBPROG2
...
MOV AH,4CH
INT 21H
MAIN ENDP
CODSG7_1 ENDS
END MAIN

```

```

;-----
;----- and in a separate file: -----
PUBLIC SUBPROG1
CODSEG1 SEGMENT
SUBPROG1 PROC FAR
ASSUME CS:CODSEG1
...
RET
SUBPROG1 ENDP
CODSEG1 ENDS
END

```

```

;-----
;----- and in another file: -----
PUBLIC SUBPROG2
CODSEG2 SEGMENT
SUBPROG2 PROC FAR
ASSUME CS:CODSEG2
...
RET
SUBPROG2 ENDP
CODSEG2 ENDS
END

```

## پیوند مدول ها و تبدیل آن به یک واحد قابل اجرا

فرض کنید که هر یک از مدول ها در مثال ۷-۱ جداگانه اسمبل شده و تحت نام های EXAMPLE1,OBJ, PROC1,OBJ و PROC2,OBJ ذخیره شده باشد. در زیر چگونگی پیوند آنها در MASM برای ایجاد یک برنامه قابل اجرا دیده می شود.

**C> LINK EXAMPLE1.OBJ + PROC1.OBJ + PROC2.OBJ**

برنامه ۷-۱ نیز چگونگی اعمال رهنمون های EXTRN و PUBLIC را به متغیرهای داده نشان می دهد. در برنامه ۷-۱ مدول اصلی حاوی یک قطعه کد و یک قطعه پشته است. ولی زیر روال ها فاقد آنهاند. هر مدول می تواند قطعه کد و داده خود را داشته باشد. در حالی مدول ها می توانند قطعات داده خود را در صورت لزوم داشته باشند ولی عموماً فقط یک پشته و آنها در برنامه اصلی تعریف می شود. تاکنون در همه برنامه ها بخاطر سادگی این کار صورت نگرفته است و این دلیل تولید پیام "no stack segment" بوسیله برنامه LINK است. بعداً در همین فصل، نشان می دهیم که چگونه می توان پیام خطا را حذف کرد و چندین قطعه از مدول ها را برای تولید یک قطعه یکنواخت برای هر قطعه برنامه، داده یا پشته تولید نمود.

از ساختار مثال ۷-۱ برای:

۱- جمع دو کلمه و

۲- ضرب دو کلمه استفاده کنید.

هر یک از آنها باید با مدول جداگانه اجرا شود. داده در مدول اصلی تعریف شده است و مدول های جمع و ضرب فاقد داده خاص خود هستند.

TITLE PROG7-1 DEMONSTRATES MODULAR PROGRAMMING  
PAGE 60,132

EXTRN SUBPROG1:FAR  
EXTRN SUBPROG2:FAR  
PUBLIC VALUE1, VALUE2, SUM, PRODUCT

STSEG SEGMENT  
DB 64 DUP (?)  
STSEG ENDS

DTSEG SEGMENT  
VALUE1 DW 2050  
VALUE2 DW 500  
SUM DW 2 DUP (?)  
PRODUCT DW 2 DUP (?)  
DTSEG ENDS

CODSG\_A SEGMENT  
MAIN PROC FAR  
ASSUME CS:CODSG\_A,DS:DTSEG,SS:STSEG  
MOV AX,DTSEG  
MOV DS,AX  
CALL SUBPROG1 ;CALL SUBPROG TO ADD VALUE1 + VALUE2  
CALL SUBPROG2 ;CALL SUBPROG TO MUL VALUE1 \* VALUE2  
MOV AH,4CH  
INT 21H ;GO BACK TO DOS  
MAIN ENDP  
CODSG\_A ENDS  
END MAIN

برنامه ۷-۱. مدول اصلی

```

;THIS PROGRAM FINDS THE SUM OF TWO EXTERNALLY DEFINED WORDS
;AND STORES THE SUM IN A LOCATION DEFINED BY THE CALLING MODULE
TITLE SUBPROG1 PROGRAM TO ADD TWO WORDS
PAGE 60,132
 EXTRN VALUE1:WORD
 EXTRN VALUE2:WORD
 EXTRN SUM:WORD
 PUBLIC SUBPROG1

CODSG B SEGMENT
SUBPROG1 PROC FAR
 ASSUME CS:CODSG_B
 SUB BX,BX ;INITIALIZE CARRY COUNT
 MOV AX,VALUE1
 MOV DX,VALUE2
 ADD AX,DX ;ADD VALUE1 + VALUE2
 ADC BX,00 ;ACCUMULATE CARRY
 MOV SUM,AX ;STORE SUM
 MOV SUM+2,BX ;STORE CARRY
 RET
SUBPROG1 ENDP
CODSG_B ENDS
 END

```

برنامه ۱-۷. مدول ۲

```

;THIS PROGRAM FINDS THE PRODUCT OF TWO EXTERNALLY DEFINED WORDS
;AND STORES THE PRODUCT IN A LOCATION DEFINED BY THE CALLING MODULE
TITLE SUBPROG2 PROGRAM TO MULTIPLY TWO WORDS
PAGE 60,132
 EXTRN VALUE1:WORD
 EXTRN VALUE2:WORD
 EXTRN PRODUCT:WORD
 PUBLIC SUBPROG2

CODSG C SEGMENT
SUBPROG2 PROC FAR
 ASSUME CS:CODSG_C
 MOV AX,VALUE1
 MOV CX,VALUE2
 MUL CX ;MULTIPLY VALUE1 * VALUE2
 MOV PRODUCT,AX ;STORE PRODUCT
 MOV PRODUCT+2,DX ;STORE PRODUCT HIGH WORD
 RET
SUBPROG2 ENDP
CODSG_C ENDS
 END

```

برنامه ۱-۷. مدول ۳

### تحلیل برنامه ۱-۷

توجه کنید که در برنامه اصلی دو زیر روال با رهنمون EXTRN معرفی شده‌اند و بدان معنی است که رویه‌ها در فایل دیگری تعریف خواهند شد. در این حال زیر روال‌ها بصورت FAR تعریف شده‌اند. در فایل‌ها هر زیر روال بصورت PUBLIC تعریف می‌گردد تا از دیگر برنامه‌ها قابل فراخوانی باشد. در مدول اصلی، نام‌های VALUE1، VALUE2، SUM و PRODUCT بصورت PUBLIC آمده‌اند تا برنامه‌های دیگر بتوانند به این اقلام داده دسترسی داشته باشند. در زیر روال‌ها (زیر برنامه‌ها) این اقلام داده بصورت EXTRN داده شده‌اند. این سه برنامه بطریق زیر بهم پیوند می‌خورند:

C>LINK PROG7-1.OBJ + SUBPROG1.OBJ + SUBPROG2.OBJ

برنامه پیوندزن ارجاع بیرونی را با تطبیق نام‌های PUBLIC و EXTRN، مطابقت می‌کند. برنامه پیوندزن در فایل‌هایی که در دستور LINK آمده است بدنبال زیر روال‌های بیرونی جستجو می‌نماید. توجه داشته باشید که نام فایل‌ها به نام‌های رویه‌ها مرتبطند. "MAIN" در فایل "PROG7-1.OBJ" واقع است.

## رهنمون SEGMENT

در فصول قبل، وقتی که قطعه‌ای تعریف می‌شد، هیچگونه خصیصه‌ای پس از آن ذکر نمی‌گردید. در واقع آن را بسادگی چنین می‌نوشتیم:

```
name SEGMENT
```

اینگونه تعریف قطعات قابل قبول بود زیرا از هر نوع قطعه کد، داده و پشته فقط یک نمونه موجود بود. با این وجود، اگر مدول‌های پیوند شونده متعدد باشند تعریف قطعه باید اصلاح شود. تعریف قطعه کامل که بگرات در برنامه‌نویسی مدولی بکار گرفته می‌شود بطرق زیر است.

```
name SEGMENT alignment combine type class name
```

پیوست (پ) (SEGMENT ملاحظه شود) توضیح کاملی از میدان‌های رهنمون SEGMENT را ارائه نموده است. در زیر مختصری برای هر میدان آورده شده است.

میدان تراز (alignment) باین معنی است که آیا قطعه باید از مرز بایت، کلمه، پاراگراف و یا صفحه آغاز شود. مثلاً اگر در میدان تراز، از WORD استفاده شود قطعه از کلمه موجود بعدی آغاز می‌شود. وقتی که مرز WORD استفاده می‌شود، اگر قطعه قبلی در تفاوت مکان 0048H پایان یافته باشد، قطعه بعدی از 004A آغاز خواهد شد. انتخاب پیش‌فرض PARA در میدان تراز بدان معنی است که هر قطعه از مرز یک پاراگراف آغاز می‌گردد. یک پاراگراف در DOS از 16 بایت شروع می‌شود. هنگام استفاده از PARA، اگر قطعه قبلی در 0048H پایان یافته باشد قطعه بعدی از مرز پاراگراف بعدی که 0050H است شروع خواهد شد. مرزهای پاراگراف با 0 پایان یافته و قابل تقسیم بر 16 (10H) می‌باشند. میدان نوع ترکیب (Combine type) برای پیوندن مشخص می‌کند که آیا قطعاتی از یک‌نوع بهم پیوند می‌خورند یا خیر. نمونه‌ای از این نوع ترکیب STACK و یا PUBLIC است. مثالی در زیر نشان می‌دهد که استفاده از این میدان در تعریف قطعه پشته موجب می‌شود، قطعه پشته برنامه با پشته سیستم ترکیب شده و پیام "warning: no stack segment" که بوسیله پیوندن تولید شده حذف گردد. اگر نوع ترکیب PUBLIC باشد، پیوندن آن قطعه را با دیگر قطعات همانند در سایر مدول‌ها پیوند خواهد زد. از این میدان می‌توان برای ترکیب قطعات کد که نام‌های مختلف دارند و تبدیل آن به یک نام استفاده کرد.

میدان نام طبقه‌بندی (class name) دارای چهار انتخاب 'CODE'، 'STACK'، 'DATA' و 'EXTRA' می‌باشند. این میدان باید در داخل نقل قول تکی قرار گیرد. این علامت برای ترکیب قطعات همانند در مدول‌های مختلف بکار می‌رود.

### تعریف قطعه پشته کامل

تعریف قطعه پشته کامل زیر در مدول پیام "Warning: no stack segment" را که بوسیله پیوندن تولید شده حذف می‌نماید:

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

### تعاریف قطعه کد و داده

در زیر تعریف قطعه داده‌ای نشان داده شده و اگر مدول دیگری قطعه داده‌ای تعریف نکرده باشد می‌توان از آن استفاده کرد.

```
name SEGMENT PARA 'DATA'
```

اگر مدول دیگری قطعه داده‌ای تعریف کرده باشد باید بین PARA و DATA از PUBLIC استفاده شود. در زیر قطعه داده و کد برای ترکیب قطعات از مدول‌های مختلف آورده شده است.

```
name SEGMENT PARA PUBLIC 'CODE'
name SEGMENT PARA PUBLIC 'DATA'
```



یک نکته اضافی دیگر که باید روشن شود این است که برای ترکیب انواع قطعات از مدول‌های مختلف به یک قطعه، نام قطعات باید یکی باشد. این مطلب در مثال ۷-۲ نشان داده شده است.

برنامه ۷-۲ تعریف قطعه کامل را برای تعریف مجدد همه قطعات برنامه ۱ نشان می‌دهد. تحلیلی از چگونگی ترکیب در طرح پیوند بدنبال برنامه آورده شده است. قطعات کد در این مثال PUBLIC نشده‌اند.

```

TITLE PROG7-2 PROG7-1 REWRITTEN WITH FULL SEGMENT DEFINITION
PAGE
EXTRN SUBPROG1:FAR
EXTRN SUBPROG2:FAR
PUBLIC VALUE1, VALUE2, SUM, PRODUCT

STSEG SEGMENT PARA STACK 'STACK'
DB 100 DUP(?)
STSEG ENDS

DTSEG SEGMENT PARA 'DATA'
VALUE1 DW 2050
VALUE2 DW 500
SUM DW 2 DUP (?)
PRODUCT DW 2 DUP (?)
DTSEG ENDS

CODSG_A SEGMENT PARA 'CODE'
MAIN PROC FAR
 ASSUME CS:CODSG_A,DS:DTSEG,SS:STSEG
 MOV AX,DTSEG
 MOV DS,AX
 CALL SUBPROG1 ;CALL SUBPROG TO ADD VALUE1 + VALUE2
 CALL SUBPROG2 ;CALL SUBPROG TO MUL VALUE1 * VALUE2
 MOV AH,4CH
 INT 21H ;GO BACK TO DOS
MAIN ENDP
CODSG_A ENDS
END MAIN

```

برنامه ۷-۲. مدول اصلی

```

;THIS PROGRAM FINDS THE SUM OF TWO EXTERNALLY DEFINED WORDS
;AND STORES THE SUM IN A LOCATION DEFINED BY THE CALLING MODULE
TITLE SUBPROG1 PROGRAM TO ADD TWO WORDS
PAGE
EXTRN VALUE1:WORD
EXTRN VALUE2:WORD
EXTRN SUM:WORD
PUBLIC SUBPROG1

CODSG_B SEGMENT PARA 'CODE'
SUBPROG1 PROC FAR
 ASSUME CS:CODSG_B
 SUB BX,BX ;INITIALIZE CARRY COUNT
 MOV AX,VALUE1
 MOV DX,VALUE2
 ADD AX,DX ;ADD VALUE1 + VALUE2
 ADC BX,00 ;ACCUMULATE CARRY
 MOV SUM,AX ;STORE SUM
 MOV SUM+2,BX ;STORE CARRY
 RET
SUBPROG1 ENDP
CODSG_B ENDS
END

```

برنامه ۷-۲. مدول ۲

THIS PROGRAM FINDS THE PRODUCT OF TWO EXTERNALLY DEFINED WORDS  
AND STORES THE PRODUCT IN A LOCATION DEFINED BY THE CALLING MODULE

```
TITLE SUBPROG2 PROGRAM TO MULTIPLY TWO WORDS
PAGE 60,132
EXTRN VALUE1:WORD
EXTRN VALUE2:WORD
EXTRN PRODUCT:WORD
PUBLIC SUBPROG2
CODSG_C SEGMENT PARA 'CODE'
SUBPROG2 PROC FAR
ASSUME CS:CODSG_C
MOV AX,VALUE1
MOV CX,VALUE2
MUL CX ;MUL VALUE1 * VALUE2
MOV PRODUCT,AX ;STORE PRODUCT
MOV PRODUCT+2,DX ;STORE PRODUCT HIGH WORD
RET
SUBPROG2 ENDP
CODSG_C ENDS
END
```

| Start  | Stop   | Length | Name    | Class |
|--------|--------|--------|---------|-------|
| 00000H | 00063H | 00064H | STSEG   | STACK |
| 00070H | 0007BH | 0000CH | DTSEG   | DATA  |
| 00080H | 00092H | 00013H | CODSG_A | CODE  |
| 000A0H | 000B5H | 00016H | CODSG_B | CODE  |
| 000C0H | 000D0H | 00011H | CODSG_C | CODE  |

برنامه ۷-۲. مدول ۳ و نقشه پیوند

### تحلیل نقشه پیوند برنامه ۷-۲

نقشه پیوند، شروع و ختم هر قطعه را نشان می‌دهد. توجه کنید که هر قطعه از مرز 16 بایت آغاز می‌شود: یعنی 00070H، 00080H و غیره. قطعه کد برای مدول اصلی دارای نام "CODSG-A" بوده و از 00080H شروع و در 00092H ختم می‌شود و بدین ترتیب جمعاً 13H بایت را اشغال می‌نماید. این قطعه را "CODE" می‌نامیم. قطعه کد بعدی تحت نام "CODSG\_B" خوانده می‌شود. این قطعه از مرز 16 بایت 000A0H آغاز می‌شود زیرا بصورت PARA تعریف شده است. باین معنی که از 00092H تا 0009FH بکار برده نشده است. به همین طریق سومین مدول از 000C0H شروع می‌شود. دقت داشته باشید که قطعات کد از یکدیگر جدا هستند. می‌توان همه آنها را با استفاده از انتخاب PUBLIC درهم ادغام نمود. این مطلب در مثال ۷-۲ نشان داده شده است. برای ادغام قطعات کد، هر قطعه کد باید نام یکسانی داشته و بصورت PUBLIC تعریف شود.

مثال ۷-۲. طرح پیوند را برای برنامه ۷-۲ به منظور ترکیب قطعات کد نشان دهید. از مرز PARA و رهنمون CDSEG SEGMENT

CDSEG SEGMENT PARA PUBLIC 'CODE' استفاده کنید. حل:

| Start  | Stop   | Length | Name  |
|--------|--------|--------|-------|
| 00000H | 00063H | 00064H | STSEG |
| 00070H | 0007BH | 0000CH | DTSEG |
| 00080H | 000D0H | 00051H | CDSEG |

در زیر رهنمون‌های SEGMENT به کمک مرزهای کلمه دیده می‌شوند.

```
SEGMENT WORD STACK 'STACK'
SEGMENT WORD 'DATA'
SEGMENT WORD PUBLIC 'CODE'
```

در زیر طرح پیوند وقتی برنامه از مرز WORD استفاده کند دیده می‌شود.

| Start  | Stop   | Length | Name  | Class |
|--------|--------|--------|-------|-------|
| 00000H | 00063H | 00064H | STSEG | STACK |
| 00064H | 0006FH | 0000CH | DTSEG | DATA  |
| 00070H | 000AAH | 0003BH | CDSEG | CODE  |

## برنامه‌نویسی مدولی و استفاده از تعریف قطعه جدید

در این بخش مثالی از برنامه‌نویسی مدولی را برای کسانی که از تعریف قطعه جدید استفاده می‌کنند ارائه می‌دهیم. البته می‌توان این بخش را بدون اینکه تداوم مطلب از دست برود چشم‌پوشی کرد. در مثال ۷-۳، مثال ۷-۱ برای تعریف قطعات با استفاده از تعریف قطعه جدید بازنویسی شده است.

مثال ۷-۳. لایه‌های برنامه مدولی برای رهنمون‌های قطعه جدید.

در فایل اصلی:

```
.MODEL SMALL
.STACK 64
.DATA
....
....
.CODE
EXTRN SUBPROG1:NEAR
EXTRN SUBPROG2:NEAR
MAIN: MOV AX,@DATA ;this is the program entry point
 MOV DS,AX
 CALL SUBPROG1
 CALL SUBPROG2
 MOV AH,4CH
 INT 21H
 END MAIN ;this is the program exit point
```

.....و در فایل دیگر.....

```
.MODEL SMALL
.CODE
PUBLIC SUBPROG2
SUBPROG2 PROC
...
RET
SUBPROG2 ENDP
END
```

.....و در فایل دیگر.....

```
.MODEL SMALL
.CODE
PUBLIC SUBPROG1
SUBPROG1 PROC
...
RET
SUBPROG1 ENDP
END
```

توجه کنید که در مدول اصلی مثال ۷-۳ نام MAIN دارای دو نقطه "." است و در برابر اولین دستور قابل اجرا آورده شده است. این نقطه ورود برنامه است. نقطه خروج برنامه با همان برچسب مشخص می‌شود و در رهنمون END ذکر می‌گردد. هیچ برنامه‌ای نمی‌تواند بیش از یک نقطه ورود و یک نقطه خروج داشته باشد. در اینجا نام MAIN اختیار شده است ولی هر نام دیگری هم می‌تواند انتخاب شود. توجه کنید که رهنمون‌های END در دیگر مدول‌ها دارای برچسب پس از کلمه "END" نیستند. برنامه ۷-۳ مثل برنامه ۷-۱، برای تعریف یک قطعه جدید بازنویسی شده است. دو برنامه را با هم مقایسه کنید تا سادگی تعریف قطعه جدید را ملاحظه نمایید.

وقتی که فرم ساده تعریف قطعه را مانند آنچه در مثال ۷-۳ ملاحظه شد بکار ببریم رویه‌ها NEAR را برای مدل‌های فشرده یا کوچک پیش‌فرض خواهند کرد و برای مدل‌های بزرگ و متوسط از FAR به عنوان پیش‌فرض استفاده می‌شود.

```

;THIS PROGRAM FINDS THE PRODUCT OF TWO EXTERNALLY DEFINED WORDS
;AND STORES THE PRODUCT IN A LOCATION DEFINED BY THE CALLING MODULE
TITLE SUBPROG2 PROGRAM TO MULTIPLY TWO WORDS
PAGE 60,132
.MODEL SMALL
EXTRN VALUE1:WORD
EXTRN VALUE2:WORD
EXTRN PRODUCT:WORD
.CODE
PUBLIC SUBPROG2
SUBPROG2 PROC NEAR
MOV AX,VALUE1
MOV CX,VALUE2
MUL CX ;MULTIPLY VALUE1 * VALUE2
MOV PRODUCT,AX ;STORE PRODUCT
MOV PRODUCT+2,DX ;STORE PRODUCT HIGH WORD
RET
SUBPROG2 ENDP
END

```

برنامه ۳-۷. جدول اصلی

```

;THIS PROGRAM FINDS THE SUM OF TWO EXTERNALLY DEFINED WORDS
;AND STORES THE SUM IN A LOCATION DEFINED BY THE CALLING MODULE
TITLE SUBPROG1 PROGRAM TO ADD TWO WORDS
PAGE 60,132
.MODEL SMALL
EXTRN VALUE1:WORD
EXTRN VALUE2:WORD
EXTRN SUM:WORD
.CODE
PUBLIC SUBPROG1
SUBPROG1 PROC NEAR
SUB BX,BX ;INITIALIZE CARRY COUNT
MOV AX,VALUE1
MOV DX,VALUE2
ADD AX,DX ;ADD VALUE1 + VALUE2
ADC BX,00 ;ACCUMULATE CARRY
MOV SUM,AX ;STORE SUM
MOV SUM+2,BX ;STORE CARRY
RET
SUBPROG1 ENDP
END

```

برنامه ۳-۷. جدول ۲

```

TITLE PROG7-3 MODULAR PROGRAMMING WITH NEW SEGMENT DEFINITION
PAGE 60,132
.MODEL SMALL
.STACK 64
.DATA
PUBLIC VALUE1, VALUE2, SUM, PRODUCT
VALUE1 DW 2050
VALUE2 DW 500
SUM DW 2 DUP (?)
PRODUCT DW 2 DUP (?)

.CODE
EXTRN SUBPROG1:NEAR
EXTRN SUBPROG2:NEAR
MAIN: MOV AX,@DATA
 MOV DS,AX
 CALL SUBPROG1 ;CALL SUBPROG TO ADD VALUE1 + VALUE2
 CALL SUBPROG2 ;CALL SUBPROG TO MUL VALUE2 * VALUE2
 MOV AH,4CH
 INT 21H ;GO BACK TO DOS
 END MAIN

```

برنامه ۳-۷. جدول ۳

- ۱- سه مزیت برنامه نویسی مدولی را فهرست نمایید.
- ۲- رهنمون ... در یک مدول باین علت بکار می رود تا متغیر نام گذاری شده قابل استفاده بوسیله مدول های دیگر باشد.
- ۳- رهنمون ... در یک مدول هنگامی بکار می رود که متغیر نام گذاری شده در مدول دیگر تعریف شده است.
- ۴- یک سیستم چگونه نقاط ورود و خروج برنامه ای را که از چند مدول تشکیل شده شناسایی می کند.
- ۵- پاراگراف چیست؟
- ۶- رهنمونی بنویسید که قطعه پشته ای را تعریف کند و با پشته سیستم ترکیب شود.
- ۷- اگر یک قلم داده با سایز کلمه به نام TOTAL در مدولی بکار رفته باشد ولی آن قلم داده در مدول دیگری تعریف شده باشد، رهنمون مناسب TOTAL را در مدول بنویسید.
- ۸- اگر PARA برای تراز قطعه کدی که در 56H ختم می شود بکار رفته باشد، قطعه کد بعدی از کجا شروع می شود.
- ۹- رهنمون های قطعه کد را برای یک برنامه فراخوان و یک مدول بنویسید بطوری که بتوانند در یک قطعه کد ترکیب شوند.

## بخش ۲-۷: برخی از مدول های بسیار مفید

در این بخش ما طراحی دو مدول مفید را نشان خواهیم داد. ابتدا، برنامه هایی برای تبدیل اعداد مبنای شانزده به مبنای ده و بالعکس طراحی می شوند. آنگاه آنها را بصورت مدول بازنویسی می کنیم که قابل فراخوانی از هر برنامه ای باشند. نهایتاً برنامه فراخوان نوشته می شود.

### تبدیل دودویی (مبنای شانزده) به اسکی (دهدی)

می دانیم که نتیجه عملیات محاسباتی دودویی است. برای نمایش نتایج به دهدی، ابتدا آن را به دهدی و سپس هر رقم را با 30H برچسب می زنیم تا بتوان آن را نمایش داد و یا چاپ کرد. اولین قدم تبدیل عدد دودویی به دهدی است. به مثال زیر توجه کنید که در آن 34DH به دهدی تبدیل شده است.

روش دیگری برای تبدیل یک عدد مبنای شانزده تقسیم متوالی آن بر 10 (0AH) است و در آن باقیمانده های کوچکتر از 10 جدا می شوند. مراحل زیر اجرا می شوند.

$$34DH / A = 84 \text{ باقیمانده کاست}$$

$$84H / A = 8 \text{ باقیمانده 4 است}$$

$$(A < 8) \text{ بنابراین عملیات پایان یافته است}$$

برنامه ۴-۷ روند تبدیل را برای عددی با ابعاد کلمه (16 بیت) نشان می دهد که در آن از تقسیم متوالی استفاده شده است. چون یک عدد با مشخصات فوق بین 0 تا FFFFH قرار دارد، نتیجه حاصل به دهدی تا 65535 خواهد بود. بنابراین یک رشته 5 تایی برای نگهداری نتیجه کافی خواهد بود. عدد دودویی مورد تبدیل در BINNUM می باشد. در برنامه ۴-۷ توجه داشته باشید که با استقرار رقم دهدی (باقیمانده) در DL، به آن 30H اضافه می شود تا به اسکی بدل شود. آنگاه در ناحیه ای از حافظه به نام ASCNUM قرار می گیرد. ارقام اسکی طوری در حافظه قرار می گیرند که پایین ترین رقم در بالاترین حافظه قرار گیرد. این روالی است که در ذخیره سازی اسکی در DOS مقرر شده است.

|           |                                   |                                  |
|-----------|-----------------------------------|----------------------------------|
| TITLE     | PROG7-4                           | CONVERT BINARY TO ASCII          |
| PAGE      | 60,132                            |                                  |
| STSEG     | SEGMENT PARA STACK 'STACK'        |                                  |
|           | DB 64 DUP (?)                     |                                  |
| STSEG     | ENDS                              |                                  |
| DTSEG     | SEGMENT                           |                                  |
| BINNUM    | DW 246DH                          |                                  |
|           | ORG 10H                           |                                  |
| ASCNUM    | DB 5 DUP ('0')                    |                                  |
| DTSEG     | ENDS                              |                                  |
| CDSEG     | SEGMENT 'CODE'                    |                                  |
| B2ASC_CON | PROC FAR                          |                                  |
|           | ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG |                                  |
|           | MOV AX,DTSEG                      |                                  |
|           | MOV DS,AX                         |                                  |
|           | MOV BX,10                         | ;BX=10 THE DIVISOR               |
|           | MOV SI,OFFSET ASCNUM              | ;SI = BEGINNING OF ASCII STRING  |
|           | ADD SI,5                          | ;ADD LENGTH OF STRING            |
|           | DEC SI                            | ;SI POINTS TO LAST ASCII DIGIT   |
|           | MOV AX,BINNUM                     | ;LOAD BINARY (HEX) NUMBER        |
| BACK:     | SUB DX,DX                         | ;DX MUST BE 0 IN WORD DIVISION   |
|           | DIV BX                            | ;DIVIDE HEX NUMBER BY 10 (BX=10) |
|           | OR DL,30H                         | ;TAG '3' TO MAKE IT ASCII        |
|           | MOV [SI],DL                       | ;MOVE THE ASCII DIGIT            |
|           | DEC SI                            | ;DECREMENT POINTER               |
|           | CMP AX,0                          | ;CONTINUE LOOPING WHILE AX > 0   |
|           | JA BACK                           |                                  |
|           | MOV AH,4CH                        |                                  |
|           | INT 21H                           | ;GO BACK TO DOS                  |
| B2ASC_CON | ENDP                              |                                  |
| CDSEG     | ENDS                              |                                  |
|           | END B2ASC_CON                     |                                  |

برنامه ۴-۷

### تبدیل اسکی (دهدهی) به دودویی (مبنای 16)

وقتی کاربری ارقام 0 تا 9 را کلید زنی می‌کند، صفحه کلید ارقام را بصورت اسکی برای کامپیوتر فراهم می‌نماید. مثلاً وقتی که کلید مربوط به عدد 9 فشرده شود، صفحه کلید در واقع اسکی آن، یعنی 00111001 (39H) را برای سیستم تهیه می‌نماید. در فصل ۳ نشان دادیم که در بعضی حالات مانند جمع اعداد به صورت اسکی پردازش شده و نیاز به تبدیل آنها به دهدهی نیست. با این وجود در اکثر موارد، لازم است عدد به مبنای شانزده تبدیل شود تا بوسیله CPU پردازش گردد. به مثال تبدیل عدد دهدهی 482 به شانزده شانزدهی توجه نمایید. در زیر مراحل تبدیل این عدد ارائه شده است.

$$\begin{aligned}
 482 / 16^2 &= 482 / 256 = 1 \\
 482 - (1 \times 256) &= 226 \quad 226 / 16^1 = 226 / 16 = 14 = E \\
 226 - (14 \times 16) &= 2 \\
 482 &= 1E2 \text{ دهدهی}
 \end{aligned}$$

با این وجود کامپیوتر ممکن است روش متفاوتی را بکار برد زیرا در حساب دودویی کار می‌کند و نه دهدهی. ابتدا باید رقم اسکی اضافه شده به آن، یعنی 30H حذف گردد. سپس هر رقم بوسیله یک وزن (توانی از 10) مانند 1، 10، 100 یا 1000 ضرب می‌شود و سپس با یکدیگر جمع می‌شوند تا نتیجه نهایی شانزده شانزدهی حاصل گردد. تبدیل 482 به مبنای شانزده مراحل زیر را شامل می‌شود. ابتدا کاربر '482' را از طریق صفحه کلید اسکی PC تایپ می‌کند، بنابراین 343832 که اسکی 482 است بدست می‌آید. آنگاه مراحل زیر اجرا می‌گردد.

$$\begin{aligned}
 2 \times 1 &= &= & 2 \\
 8 \times 10 &= 80 = &= & 50H \\
 4 \times 100 &= 400 = &= & 190H \\
 &&& \text{شانزده شانزدهی } 1E2
 \end{aligned}$$

```

TITLE PROG7-5 CONVERT ASCII TO BINARY
PAGE 60,132
STSEG SEGMENT PARA STACK 'STACK'
 DB 64 DUP (?)
STSEG ENDS

DTSEG SEGMENT
TEN DW 10
ASCNUM DB '09325'
STRLEN DB 5
 ORG 10H
BINNUM DW 0
DTSEG ENDS

CDSEG SEGMENT
ASC2B_CON PROC FAR
 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
 MOV AX,DTSEG
 MOV DS,AX
 SUB DI,DI ;CLEAR DI FOR THE BINARY(HEX) RESULT
 MOV SI,OFFSET ASCNUM ;SI = BEGINNING OF ASCII STRING
 MOV BL,STRLEN ;BL = LENGTH OF ASCII STRING
 SUB BH,BH ;BH=0 USE BX IN BASED INDEX MODE
 DEC BX ;BX IS OFFSET TO LAST DIGIT
 MOV CX,1 ;CX = WEIGHT FACTOR
AGAIN: MOV AL,[SI+BX] ;GET THE ASCII DIGIT
 AND AL,0FH ;STRIP OFF '3'
 SUB AH,AH ;CLEAR AH FOR WORD MULTIPLICATION
 MUL CX ;MULTIPLY BY THE WEIGHT
 ADD DI,AX ;ADD IT TO BINARY (HEX)RESULT
 MOV AX,CX ;MULTIPLY THE WEIGHT FACTOR
 MUL TEN ;BY TEN
 MOV CX,AX ;FOR NEXT ITERATION
 DEC BX ;DECREMENT DIGIT POINTER
 JNS AGAIN ;JUMP IF COUNTER >= 0
 MOV BINNUM,DI ;SAVE THE BINARY(HEX)RESULT
 MOV AH,4CH
 INT 21H ;GO BACK TO DOS
ASC2B_CON ENDP
CDSEG ENDS
 END ASC2B_CON

```

#### برنامه ۵-۷

برنامه ۵-۷ اعداد اسکی را به دودویی تبدیل می‌کند. این برنامه حداکثر عدد دهمی را 65535 فرض می‌نماید. بنابراین حداکثر نتیجه نیز FFFFH خواهد بود که یک کلمه 16 بیتی است. ابتدا با کم ارزش ترین رقم شروع کرده و عدد 3 را از آن حذف می‌کند و سپس آن را در فاکتور وزن ضرب می‌نماید. ثبات CX وزن را نگه می‌دارد که در این حالت 1 است. برای رقم بعدی CX برابر 10 (0AH) خواهد شد. برای رقم بعد، محتوای این ثبات، 100 (64H) می‌گردد و به همین ترتیب. برنامه، کم ارزشترین رقم اسکی را در بالاترین مکان حافظه داده فرض می‌نماید. به بیان دیگر بنا به قرارداد ذخیره‌سازی اعداد اسکی، ارقام با ارزشتر در پایین ترین آدرس حافظه و کم ارزشترین رقم در بالاترین آدرس حافظه قرار می‌گیرند. مثلاً اگر "749" را در تفاوت مکان 200 قرار دهیم، داریم: 200=(37)، 201=(34) و 202=(39). تابع 0A از DOS 21H نیز اعداد اسکی را به همین ترتیب می‌چیند.

برنامه‌های ۷-۴ و ۷-۵ با داده‌های نمونه نوشته شده و تست گردیده است و اکنون می‌تواند از فرم برنامه به مدول‌هایی که بوسیله هر برنامه قابل فراخوانی است تبدیل گردد.

#### مدول دودویی به اسکی

برنامه ۷-۶ فرم مدوله شده برنامه ۷-۴ است. رویه بصورت PUBLIC معرفی شده است و بنابراین قابل فراخوانی بوسیله برنامه‌های دیگر می‌باشد. همه مقادیر بصورت بیرونی تلقی شده و داده بوسیله فراخوانی برنامه تهیه می‌شود. بنابراین این مدول نیاز به داشتن قطعه داده خود ندارد.

به نکات زیر در مورد مدول توجه کنید:

۱- چون این مدول بوسیله دیگر مدول‌ها فراخوانی می‌شود نقطه ورود و خروج داده نشده است. بنابراین رهنمون EN فاقد برچسب B2ASC - CON است.

۲- مدول به برنامه فراخوان برمی گردد و همچون برنامه ۵-۷ به DOS باز نمی گردد.

۳- این مدول به قطعات داده و پشته مخصوص به خود نیازی ندارد.

```

TITLE PROG7-6 BINARY TO DECIMAL CONVERSION MODULE
PAGE 60,132
;this module converts a binary (hex) number up to FFFFH to decimal
; then makes it displayable (ASCII)
;CALLING PROGRAM SETS
; AX = BINARY VALUE TO BE CONVERTED TO ASCII
; SI = OFFSET ADDRESS WHERE ASCII VALUE TO BE STORED
 PUBLIC B2ASC_CON
CDSEG SEGMENT PARA PUBLIC 'CODE'
B2ASC_CON PROC FAR
 ASSUME CS:CDSEG
 PUSHF ;STORE REGS CHANGED BY THIS MODULE
 PUSH BX
 PUSH DX
 MOV BX,10 ;BX=10 THE DIVISOR
 ADD SI,4 ;SI POINTS TO LAST ASCII DIGIT
B2A_LOOP: SUB DX,DX ;DX MUST BE 0 IN WORD DIVISION
 DIV BX ;DIVIDE HEX NUMBER BY 10 (BX=10)
 OR DL,30H ;TAG '3' TO REMAINDER TO MAKE IT ASCII
 MOV [SI],DL ;MOVE THE ASCII DIGIT
 DEC SI ;DECREMENT POINTER
 CMP AX,0 ;CONTINUE LOOPING WHILE AX > 0
 JA B2A_LOOP
 POP DX ;RESTORE REGISTERS
 POP BX
 POPF
 RET
B2ASC_CON ENDP
CDSEG ENDS
 FND

```

برنامه ۶-۷

```

TITLE PROG7-7 ASCII TO BINARY CONVERSION MODULE
PAGE 60,132
;this module converts any ASCII number between 0 to 65535 to binary
;CALLING PROGRAM SETS
; SI = OFFSET OF ASCII STRING
; BX = STRING LENGTH - 1 (USED AS INDEX INTO ASCII NUMBER)
;THIS MODULE SETS
; AX = BINARY NUMBER
 EXTRN TEN:WORD
 PUBLIC ASC2B_CON
CDSEG SEGMENT PARA PUBLIC 'CODE'
ASC2B_CON PROC FAR
 ASSUME CS:CDSEG
 PUSHF ;STORE REGS CHANGED IN THIS MODULE
 PUSH DI
 PUSH CX
 SUB DI,DI ;CLEAR DI FOR THE BINARY(HEX) RESULT
 MOV CX,1 ;CX = WEIGHT FACTOR
A2B_LOOP: MOV AL,[SI+BX] ;GET THE ASCII DIGIT
 AND AL,0FH ;STRIP OFF '3'
 SUB AH,AH ;CLEAR AH FOR WORD MULTIPLICATION
 MUL CX ;MULTIPLY BY THE WEIGHT
 ADD DI,AX ;ADD IT TO BINARY (HEX) RESULT
 MOV AX,CX ;MULTIPLY THE WEIGHT FACTOR
 MUL TEN ; BY TEN
 MOV CX,AX ; FOR NEXT ITERATION
 DEC BX ;DECREMENT DIGIT POINTER
 JNS A2B_LOOP ;JUMP IF OFFSET >= 0
 MOV AX,DI ;STORE BINARY NUMBER IN AX
 POP CX ;RESTORE FLAGS
 POP DI
 POPF
 RET
ASC2B_CON ENDP
CDSEG ENDS
 END

```

برنامه ۷-۷



|          |                                   |                                            |
|----------|-----------------------------------|--------------------------------------------|
| TITLE    | PROG7-8                           | CALLING PROGRAM TO CONVERT ASCII TO BINARY |
| PAGE     | 60,132                            |                                            |
| STSEG    | PUBLIC TEN                        |                                            |
| STSEG    | SEGMENT PARA STACK 'STACK'        |                                            |
| STSEG    | DB 64 DUP (?)                     |                                            |
| STSEG    | ENDS                              |                                            |
| DTSEG    | SEGMENT                           |                                            |
| ASC_AREA | LABEL                             | BYTE                                       |
| MAX_LEN  | DB                                | 6                                          |
| ACT_LEN  | DB                                | ?                                          |
| ASC_NUM  | DB                                | 6 DUP (?)                                  |
|          | ORG                               | 10H                                        |
| BINNUM   | DW                                | 0                                          |
| PROMPT1  | DB                                | 'PLEASE ENTER A 5 DIGIT NUMBER','\$'       |
| TEN      | DW                                | 10                                         |
| DTSEG    | ENDS                              |                                            |
| CDSEG    | SEGMENT                           |                                            |
| MAIN     | EXTRN ASC2B_CON:FAR               |                                            |
| MAIN     | PROC FAR                          |                                            |
|          | ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG |                                            |
|          | MOV AX,DTSEG                      |                                            |
|          | MOV DS,AX                         |                                            |
|          | :DISPLAY THE PROMPT               |                                            |
|          | MOV AH,09                         |                                            |
|          | MOV DX,OFFSET PROMPT1             |                                            |
|          | INT 21H                           |                                            |
|          | :INPUT STRING                     |                                            |
|          | MOV AH,0AH                        |                                            |
|          | MOV DX,OFFSET ASC_AREA            |                                            |
|          | INT 21H                           |                                            |
|          | MOV SI,OFFSET ASC_NUM             |                                            |
|          | MOV BH,00                         |                                            |
|          | MOV BL,ACT_LEN                    |                                            |
|          | DEC BX                            |                                            |
|          | CALL ASC2B_CON                    |                                            |
|          | MOV BINNUM,AX                     | ;SAVE THE BINARY (HEX) RESULT              |
|          | MOV AH,4CH                        |                                            |
|          | INT 21H                           | ;GO BACK TO DOS                            |
| MAIN     | ENDP                              |                                            |
| CDSEG    | ENDS                              |                                            |
|          | END                               | MAIN                                       |

#### برنامه ۷-۸

#### مدول اسکی به دودویی

برنامه ۷-۷ فرم مدوله شده برنامه ۷-۵ است به نکات زیر درباره مدول توجه کنید:

۱- TEN در برنامه فراخوان تعریف شده است.

۲- این مدول باید به فراخوان بازگردد و نه به DOS.

#### مدول فراخوان

برنامه ۷-۸، مدول یک برنامه فراخوان است که اسکی را به دودویی تبدیل می‌کند. این برنامه قطعه داده را ایجاد می‌کند، ورودی‌های داده اسکی از صفحه کلید را دریافت و آنها را در حافظه قرار می‌دهد و سپس روال را برای تبدیل عدد به دودویی فرا می‌خواند. نهایتاً، نتیجه شانزده شانزدهی در حافظه ذخیره می‌گردد.

#### سؤالات مروری

- ۱- تحلیل قدم به قدمی از برنامه ۷-۴ با داده F624H را نشان دهید. رشته دستورات و مقادیر داده را نشان دهید.
- ۲- تحلیل قدم به قدمی از برنامه ۷-۵ را با داده '1546' نمایش دهید. رشته دستورات و مقادیر داده را نیز نشان دهید.

## بخش ۳-۷: تبادل پارامترها در میان مدول‌ها

گاه مواردی پیش می‌آید که در آن پارامترهایی باید میان مدول‌های متفاوت زبان اسمبلی و پایین زبان اسمبلی و برنامه‌های زبان BASIC، پاسکال یا C رد و بدل شود. پارامتر می‌تواند مقداری ثابت، متغیرها، آرایه‌ای از داده یا حتی اشاره‌گرهایی به حافظه باشد. پارامترها می‌توانند از طریق ثبات‌ها، حافظه و یا پشته از مدولی به مدولی دیگر منتقل شوند. در این بخش موضوع رد و بدل شدن پارامترها را بین مدول‌های زبان اسمبلی مورد بررسی قرار می‌دهیم.

### تبدیل پارامترها از طریق ثبات‌ها

هنگامی که نیاز به تبادل پارامترها میان انواع مدول‌ها وجود دارد می‌توان از ثبات‌های CPU استفاده کرد. مثلاً اگر روال اصلی زیر روالی را فرا می‌خواند، مقادیر در ثبات‌های روال اصلی قرار می‌گیرند و سپس زیرروال برای پردازش داده فرا خوانده می‌شود. در این حالات برنامه‌نویس باید ثبات‌های بکار رفته برای ورود داده و ثبات‌های نگهداری نتایج پس از اجرای زیر روال را مستند نماید. در فصل ۴ مفهوم با INT21H و INT10H بیان شد. برنامه ۸-۷ این روش را شرح می‌دهد. در این برنامه، ثبات‌های BX و SI برای اشاره به اقلام مشخصی از داده قبل از فراخوانی مدول بکار رفته‌اند و مدول فرا خوانده شده نیز نتایجش را قبل از بازگشت به برنامه فرخوان در ثبات AX قرار داده است.

### تبادل پارامترها از طریق حافظه

گرچه رد و بدل پارامترها از طریق ثبات‌ها بطور گسترده‌ای در میان توابع وقفه DOS و BIOS بکار می‌رود، محدودیت تعداد ثبات‌ها در CPU خود محدودیت بزرگی برای این روش تبادل پارامتر است. بنابراین مدیریت ثبات را بسیار مشکل می‌نماید. روش دیگر، تبادل پارامترها از طریق حافظه است که در آن ناحیه‌ای از RAM تعریف می‌شود و پارامترها به این مکان‌های RAM منتقل می‌شوند. DOS و BIOS IBM کرراً از این روش استفاده می‌کنند. مشکل تبادل پارامتر با یک انتقال پارامتر به ناحیه ثابتی از حافظه این است که باید توافق نامه جهانی برای ناحیه آدرس حافظه وجود داشته باشد تا مطمئن شویم مدول‌ها می‌توانند با سخت افزار و نرم افزار کمپانی‌های مختلف کار کنند. این نوع استانداردسازی بسختی صورت می‌گیرد. یکی از دلایلی که DOS و BIOS از ناحیه‌ای از حافظه برای تبادل پارامترها استفاده می‌کنند این است که IBM و مایکروسافت در مورد آدرس‌های حافظه با هم همکاری کرده‌اند. انتخاب دیگر و در واقع متداول‌ترین روش تبادل پارامترها، از طریق پشته صورت می‌گیرد. این روش در زیر مورد بررسی قرار گرفته است. تبادل پارامترها از طریق پشته، آنها را از ثبات و حافظه مستقل می‌نماید.

### تبادل پارامترها از طریق پشته

پشته بخش بسیار حساس هر برنامه‌ای است و بازی با آن می‌تواند یک ریسک باشد. وقتی مدولی فراخوانده می‌شود، این پشته است که آدرس بازگشت برنامه را پس از اجرا نگه می‌دارد. بنابراین اگر محتوای پشته تغییر یابد برنامه فرو خواهد ریخت. باین دلیل کار با پشته و رد و بدل پارامترها از طریق آنها باید بخوبی قبل از آغاز دانسته شود.

در زیر، برنامه ۹-۷ این روش رد و بدل پارامترها را نشان داده و براساس نیازهای زیر نوشته شده است. مدول اصلی سه عملوند کلمه‌ای را از قطعه داده دریافت کرده و سپس آنها را در پشته ذخیره می‌کند و آنگاه زیر روال را فراخوانی می‌کند. زیر روال، عملوندها را از پشته دریافت و آنها را با هم جمع نموده و نتیجه را در یک ثبات نگه می‌دارد. آنگاه کنترل را به مدول اصلی می‌سپارد. برنامه اصلی نتیجه جمع را ذخیره می‌کند، بدنال برنامه، شرحی از تحلیل محتوای پشته آمده و نشان می‌دهد که چگونه پارامترها در پشته بوسیله روال اصلی ذخیره شده و بوسیله روال فراخوانی از پشته بازیابی می‌شوند.

|        |                                          |
|--------|------------------------------------------|
| TITLE  | PROG7-9 PASSING PARAMETERS VIA THE STACK |
| PAGE   | 60,132                                   |
| EXTRN  | SUBPROG6:FAR                             |
| STSEG  | SEGMENT PARA STACK 'Stack'               |
|        | DB 64 DUP (?)                            |
| STSEG  | ENDS                                     |
| DTSEG  | SEGMENT PARA 'Data'                      |
| VALUE1 | DW 3F62H                                 |
| VALUE2 | DW 1979H                                 |
| VALUE3 | DW 25F1H                                 |
| RESULT | DW 2 DUP (?)                             |
| DTSEG  | ENDS                                     |
| CDSEG  | SEGMENT PARA PUBLIC 'Code'               |
| MAIN   | PROC FAR                                 |
|        | ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG        |
|        | MOV AX,DTSEG                             |
|        | MOV DS,AX                                |
|        | PUSH VALUE3 ;SAVE VALUE3 ON STACK        |
|        | PUSH VALUE2 ;SAVE VALUE2 ON STACK        |
|        | PUSH VALUE1 ;SAVE VALUE1 ON STACK        |
|        | CALL SUBPROG6 ;CALL THE ADD ROUTINE      |
|        | MOV RESULT,AX ;STORE                     |
|        | MOV RESULT+2,BX ; THE RESULT             |
|        | MOV AH,4CH                               |
|        | INT 21H                                  |
| MAIN   | ENDP                                     |
| CDSEG  | ENDS                                     |
|        | END MAIN                                 |

برنامه ۹-۷. مدول اصلی.

|          |                                                              |
|----------|--------------------------------------------------------------|
|          | in a separate file:                                          |
| TITLE    | SUBPROG6 MODULE TO ADD THREE WORDS BROUGHT IN FROM THE STACK |
| PAGE     | 60,132                                                       |
|          | PUBLIC SUBPROG6                                              |
| CDSEG    | SEGMENT PARA PUBLIC 'Code'                                   |
| SUBPROG6 | PROC FAR                                                     |
|          | ASSUME CS:CDSEG                                              |
|          | SUB BX,BX ;CLEAR BX FOR CARRIES                              |
|          | PUSH BP ;SAVE BP                                             |
|          | MOV BP,SP ;SET BP FOR INDEXING                               |
|          | MOV AX,[BP]+6 ;MOV VALUE1 TO AX                              |
|          | MOV CX,[BP]+8 ;MOV VALUE2 TO CX                              |
|          | MOV DX,[BP]+10 ;MOV VALUE3 TO DX                             |
|          | ADD AX,CX ;ADD VALUE2 TO VALUE1                              |
|          | ADC BX,00 ;KEEP THE CARRY IN BX                              |
|          | ADD AX,DX ;ADD VALUE3                                        |
|          | ADC BX,00 ;KEEP THE CARRY IN BX                              |
|          | POP BP ;RESTORE BP BEFORE RETURNING                          |
|          | RET 6 ;RETURN AND ADD 6 TO SP TO BYPASS DATA                 |
| SUBPROG6 | ENDP                                                         |
| CDSEG    | ENDS                                                         |
|          | END                                                          |

برنامه ۹-۷. مدول ۲

### تحلیل محتوای پشته برای برنامه ۹-۷

برای آشکار کردن مفهوم تبادل پارامتر از طریق پشته تحلیل قدم به قدم زیر از اشاره گر پشته و محتوای آن آورده شده است. فرض کنید که اشاره گر پشته دارای مقدار  $SP = 17FEH$  قبل از اجرای دستور "PUSH VALUE3" در برنامه اصلی باشد.

۱-  $VALUE3 = 25F1H$  در پشته درج شده و  $SP = 17FC$  می گردد (از منتهی به کوچکتر فراموش نشود که در آن بایت مرتبه پایین در آدرس پایین و بایت بالا در آدرس بالا قرار می گیرد).

۲-  $VALUE2 = 1979H$  در پشته درج می گردد و سپس  $SP = 17FA$  می شود.

|      |    |        |
|------|----|--------|
| 17F0 |    |        |
| 17F1 |    |        |
| 17F2 | BP |        |
| 17F3 |    |        |
| 17F4 | IP |        |
| 17F5 |    |        |
| 17F6 | CS |        |
| 17F7 |    |        |
| 17F8 | 62 | VALUE1 |
| 17F9 | 3F |        |
| 17FA | 79 | VALUE2 |
| 17FB | 19 |        |
| 17FC | F1 | VALUE3 |
| 17FD | 25 |        |
| 17FE |    |        |

نمودار محتوای پشته برنامه ۹-۷

۳-  $VALUE\ 1 = 3F62H$  در پشته درج و سپس  $SP = 17F8$  می‌شود.

۴-  $CALL\ SUBPROG6$  یک فراخوانی دور است؛ بنابراین هر دو ثبات  $CS$  و  $IP$  به پشته ارسال می‌شوند و لذا  $SP = 17F4$  خواهد شد. اگر فراخوانی نزدیک باشد فقط  $IP$  در پشته درج می‌شود.

۵- در مدول زیر برنامه ثبات،  $BP$  با درج در پشته ذخیره می‌شود و بنابراین  $SP = 17F2$  خواهد شد. در زیر برنامه،  $BP$  برای دستیابی مقادیر در پشته بکار می‌رود. ابتدا  $SP$  در  $BP$  کپی می‌شود زیرا تنها  $BP$  در مد اندیسی با ثبات پشته ( $SS$ ) قابل استفاده است. به بیان دیگر  $[MOV\ AX, [SP+4]]$  مقدار  $AX, [SP+4]$  موجب خطا می‌گردد.  $MOV\ AX, [BP+6]$  مقدار  $VALUE1$  را در  $AX$  بار می‌کند.  $17F2+6=17F8$   $[BP+6]$  است. دقیقاً همان جایی که  $VALUE1$  در آن ذخیره شده است. بطور مشابه  $BP+8=17F2+8=17FA$  مکان  $VALUE2$  می‌باشد، و  $BP+10=17F2+10=17FC$  مکان  $VALUE3$  می‌باشد.

۶- در پایان، پارامترها بوسیله مدول جاری بداخل  $CPU$  آورده شده و پردازش می‌شوند (در این حالت جمع می‌گردند) و مدول با برداشت از پشته،  $BP$  اولیه را باز می‌گرداند، و آنگاه  $SP = 17F4$  خواهد شد.

۷-  $RET6$ : این دستورالعمل جدید است.  $RET$  هایی که قبلاً ملاحظه شدند فاقد عددی در انتها بودند. دستور  $"RETn"$  باین معنی است که ابتدا  $IP$  :  $CS$  از بالا پشته برداشته می‌شود (اگر  $CALL$  از نوع نزدیک بود فقط  $IP$  برداشته می‌شد) و سپس مقدار  $n$  به  $SP$  اضافه می‌گردد. همانطور که از نمودار برنامه ۹-۷ ملاحظه می‌شود پس از برداشت  $CS$  و  $IP$  از پشته اشاره گر پشته چهار بار اضافه می‌شود و بنابراین  $SP = 17F8$  خواهد شد. با افزایش ۶ واحد برای چشم پوشی از ۶ مکانی که پارامترها در آن ذخیره شده‌اند اشاره گر  $SP = 17FEH$  خواهد شد، که همان مقدار اولیه آن است. سؤال این است که اگر برنامه به جای  $"RET6"$  از  $RET$  استفاده کند چه پیش خواهد آمد؟ مشکل این خواهد بود که هر بار زیر برنامه اجرا شود، پشته ۶ مکان را از دست خواهد داد. اگر در مثال فوق این کار را انجام دهیم وقتی روال مجدداً فراخوانی شود پشته از  $17F8$  بجای  $17FE$  شروع خواهد کرد. اگر این عمل از دست دادن بخشی از ناحیه پشته تداوم یابد نهایتاً وضعی پیش خواهد آمد که برنامه بدون پشته کار کرده و فرو خواهد ریخت.

### سوالات مروری

- ۱- یک مزیت و یک عیب از هر روش ردوبدل پارامتر را بیان کنید.
- (الف) از طریق ثبات (ب) از طریق پشته (پ) از طریق حافظه
- ۲- فرض کنید می‌خواهیم به بعضی از پارامترهای پشته دسترسی پیدا کنیم. کدامیک از موارد زیر راه صحیح دستیابی به پشته است.

(ب)  $MOV\ AX, [SP]+20$   
(ت)  $MOV\ AX, [SP+SI]$

(الف)  $MOV\ AX, [BP] + 20$   
(پ)  $MOV\ AX, [BP+DI]$

### بخش ۴-۷: ترکیب زبان اسمبلی و برنامه‌های C

گرچه زبان اسمبلی سریعترین زبان موجود برای یک  $CPU$  است، ولی قابل اجرا بر روی  $CPU$  های مختلف نیست. مثلاً برنامه‌های اسمبلی  $80x86$  اینتل قابل اجرا بر روی کامپیوترهای سری ۶۸۰۰۰ نیست زیرا کد، نماد رمزی، نام ثبات‌ها و سائز آنها کاملاً با هم متفاوتند. بنابراین به یک زبان قابل حمل نیاز است.

## چرا C؟

گر چه رؤیای زبان جهانی در میان مردم جهان هنوز رسمیت نیافته است، ولی زبان C در میان انواع CPUها در حال جهانی شدن است. امروز بخش عمده‌ای از برنامه‌های نوشته شده برای کامپیوترها از PCها تا سوپر کامپیوترهایی مانند CRAY بزبان C می‌باشند. جهانی بودن زبان برنامه نویسی C به حدی است که بر روی هر معماری CPU با مختصر اصلاح یا حتی بدون اصلاح قابل اجرا هستند. تنها کافی است آن را برای CPU مورد نظر مجدداً کامپایل کنیم. این واقعیت که C زبان قابل حملی است موجب انتخاب آن بوسیله برنامه نویس‌ها شده است. با این وجود C دارای سرعت زبان اسمبلی نیست. ترکیبی از C و زبان اسمبلی مزیت قابل حمل بودن C و سرعت اسمبلی را خواهد داشت. امروزه در پروژه‌های نرم‌افزاری، 70 الی 80% از C و بقیه از زبان اسمبلی استفاده می‌کنند.

دو راه برای ترکیب C و اسمبلی وجود دارد. یکی اینکه دستورات اسمبلی را در برنامه‌های C وارد کنیم که به آن اسمبلی درون برنامه‌ای می‌گویند. روش دوم این است که از C یک رویه زبان اسمبلی بیرونی فراخوانی شود. در این بخش ابتدا چگونگی برنامه نویسی درون برنامه‌ای را مورد بحث قرار می‌دهیم و سپس یک رویه اسمبلی را از برنامه زبان C فراخوانی خواهیم کرد. خوانندگانی که با C آشنا نیستند می‌توانند بدون اینکه تداوم مطالعه را از دست بدهند از آن چشم‌پوشند. این بخش هر دو نوع زبان Micro soft's Quick C و Borland's Turbo C++ را در بر می‌گیرد.

## وارد کردن کد اسمبلی 80x86 در برنامه‌های C

در این بخش ما درون برنامه‌ای را با Borland's Turbo C بحث خواهیم کرد. درون برنامه‌ای Microsoft C خیلی شبیه به Borland است ولی ویرایش قدیمی آن را پشتیبانی نمی‌کند. برای دیگر کامپایلرهای C، مستندات C را مطالعه فرمائید. برنامه زیر نشان می‌دهد که چگونه محل مکان‌نما را به سطر 10، ستون 20 در برنامه C تغییر دهیم. دستورات اسمبلی با پیشوند "asm" که کلمه رزروی است ذکر می‌شوند. میکرو سافت از پیشوند "\_asm" استفاده می‌کند. در زیر دو نوع قالب بورلند برای اسمبلی درون برنامه‌ای آورده شده است.

*/\* version 1: using keyword asm before each line of in-line code \*/*

```
main ()
{
asm mov ah,2; /* each line should end with semicolon or <CR> */
asm mov bh,0;
asm mov dl,20; /* comments must be C style, not ";" assembly style */
asm mov dh,10;
asm int 10h;
}
```

*/\* version 2: using the keyword asm before a block of in-line code \*/*

```
main ()
{
asm {
mov ah,2
mov bh,0
mov dl,20
mov dh,10
int 10h
 }
}
```

همانطور که در فوق دیده می شود، هر خط از کد درون برنامه ای بوسیله کلمه کلید "asm" پیشوند شده است، یا در نوع دوم یک بلوک از کد بوسیله "asm" پیشوند شده است. هر خط با نقطه ویرگول یا خط جدید پایان می یابد و هر توضیحی هم باید در فرم صحیح خود ذکر گردد.

مثال ۴-۷ دو برنامه نمایش رشته داده ای را نشان می دهد. حل (الف) فقط از زبان C استفاده می کند. حل (پ) از بورلند C و کد اسمبلی درون برنامه ای استفاده می نماید. توجه کنید که در ترکیب C و کد اسمبلی، رهنمون های اسمبلی مانند OFFSET در "MOV DX,OFFSET MESSAGE" قابل تشخیص بوسیله C نیستند.

مثال ۵-۷ نیز زبان اسمبلی درون برنامه ای را نشان می دهد. کد درون برنامه مکان نما را در سطر ۱۰ ستون ۲۰ نشانده و سپس رشته ای از داده را با ترکیب بورلند C و اسمبلی بنمایش می گذارد.

#### مثال ۴-۷.

حل (الف): برنامه زبان C

```
#include <stdio.h>
main ()
{
 printf("The planet Earth. \n");
}
```

حل (ب): توربو C با اسمبلی درون برنامه ای

```
char const *MESSAGE = "The planet Earth.\n$";
main ()
{
 asm mov ah,9
 asm mov dx, MESSAGE
 asm int 21h
}
```

#### مثال ۵-۷. برنامه بورلند C و اسمبلی درون برنامه ای

```
main ()
{
 int const row = 10;
 int const column = 20;
 char const *MESSAGE = "The planet Earth. \n$";

 asm {
 mov ah,2
 mov bh,0
 mov dl,column
 mov dh,row
 int 10h /* set cursor position */
 mov ah,09
 mov dx,MESSAGE
 int 21h /* display message */
 }
}
```

#### برنامه های C فراخوان رویه های اسمبلی

گرچه اسمبلی درون برنامه ای سریع و ساده است، در کاربردهای واقعی رسم بر این است که زیر روال هایی بزبان اسمبلی نوشته شود و سپس در دسترس C قرار گیرند تا مثل فراخوانی توابع C، فراخوانی شوند. آنچه در C باصطلاح تابع خوانده می شود، در زبان اسمبلی به آن رویه می گویند. قبل از آغاز نوشتن روال های اسمبلی برای C، باید ابتدا چگونگی ردوبدل شدن پارامترها از C به زبان اسمبلی مشخص گردد. همه زبان های سطح بالا، مانند C، BASIC، FORTRAN و Pascal پارامترها را از طریق پشته به زیر روال ها رد می کنند. بعضی از آنها مثل C و پاسکال خود پارامتر را تحویل می دهند و برخی دیگر مثل BASIC و FORTRAN آدرس آن را تحویل می نمایند. در BASIC، تنها آدرس تفاوت مکان رد می شود، در حالی که FORTRAN آدرس های قطعه و تفاوت مکان را تحویل می نماید. در میان زبان های برنامه نویسی سطح بالا، حتی ترتیب تحویل پارامترها تفاوت می کند. اصلاح استاندارد فراخوانی به روشی اطلاق می شود که در یک زبان مفروض پارامترها به زیر روالی که فرا می خواند منتقل شود. در زیر استاندارد فراخوانی توضیح داده می شود.

```
extern cursor (int, int);
main ()
{
 cursor (15,12);
 printf("This program sets the cursor");
}
```

```
----- in cursor.asm: -----
.MODEL SMALL
.CODE
PUBLIC _CURSOR
;this procedure is written to be called by a C program
_CURSOR PROC
 PUSH BP ;save the BP (it is being altered)
 MOV BP,SP ;use BP as indexing into stack
 MOV DH,[BP+4] ;get the x (row) value from stack
 MOV DL,[BP+6] ;get the y (column) value from stack
 MOV AH,02 ;set registers for INT call
 MOV BH,00
 INT 10H
 POP BP ;restore BP
 RET
_CURSOR ENDP
```

### استاندارد فراخوانی C

|      |    |                            |
|------|----|----------------------------|
| 1982 |    |                            |
| 1983 |    |                            |
| 1984 | BP | SP=BP                      |
| 1985 |    |                            |
| 1986 | IP | BP +2 holds return address |
| 1987 |    |                            |
| 1988 | 0F |                            |
| 1989 | 00 | BP+4 holds x value 15=FH   |
| 198A | 0C |                            |
| 198B | 00 | BP+6 holds y value 12=CH   |
| 198C |    |                            |

نمودار محتوای پشته مثال ۶-۷

C چگونه پارامترها را به توابع رد می‌کند؟ دانستن این نکته بسیار مهم است زیرا عدم موفقیت موجب دسترسی به داده‌های غلط از طریق اسمبلی خواهد شد. در زیر استاندارد فراخوانی برای ترکیب زبان اسمبلی MASM و C آورده شده است. فراخوانی رویه‌های زبان اسمبلی بوسیله C باید طبق قوانین زیر باشد.

۱- مقدار پارامترها بصورت معکوس برخورد، به پشته تحویل می‌شوند. مثلاً، در تابع

Prg (x,y,z) ابتدا z، سپس y و غیره انتقال می‌یابند.

۲- پس از انتقال معکوس پارامترها، C آدرس (IP، CS) را ذخیره می‌کند. اگر C در مدل حافظه کوچک یا فشرده کامپایل شود (و یا رویه از نوع NEAR باشد) فقط IP ذخیره می‌شود. اگر C برای LARGE، MEDIUM یا HUGE کامپایل گردد (یا رویه FAR باشد)، هر دو مقدار CS و IP به پشته ارسال می‌شوند (CS ابتدا و سپس IP). ۳- BP باید در پشته ذخیره شود و سپس پارامترها باید بوسیله ثابت BP و مقدار جابجایی دستیابی گردند، زیرا BP تفاوت مکان ثبات قطعه پشته (SS) است.

۴- آخرین دستور باید RET و بدون هر عددی پس از آن باشد، زیرا بازگرداندن پشته به محل اولیه‌اش، در هنگام باز پس گرفتن کنترل، بعهد C است.

۵- هر نام مشترک با C باید با یک خط تیره پیشوند شود و تنها هشت کاراکتر اول نام بوسیله C شناسایی می‌شود.

۶- مقدار پارامترها را تحویل می‌دهد بجز آرایه‌ها که با ارجاع تحویل می‌گردند.

۷- اگر C در مدل LARGE، MEDIUM یا HUGE کامپایل شود، انتخاب FAR را برای رویه زبان اسمبلی اختیار کنید. اگر با مدل SMALL کامپایل شود از انتخاب NEAR برای رویه زبان اسمبلی استفاده نمایید.

برای درک مفاهیم فوق، فرض کنید که تابعی به زبان C با نام cursor (x,y) وجود داشته باشد، که در آن x و y بترتیب مقادیر سطر و ستون هستند. مثال ۶-۷ چگونگی تحویل x و y را به پشته و سپس به برنامه اسمبلی نشان می‌دهد. مراحل قدم به قدم محتوای ثبات نیز نشان داده شده است.

وقتی C مکان نما (cursor) را فرامی خواند، ابتدا y، سپس x، آنگاه آدرس IP را ذخیره کرده و بالاخره کنترل را به برنامه اسمبلی تحویل می دهد. اولین دستور رویه اسمبلی باید ثبات BP را با "PUSH BP" ذخیره کند. آخرین دستور باید دستور RET باشد.

توجه کنید که دو دستور اول رویه باید همیشه درج BP و انتقال SP به BP باشد. بطور مشابه دو دستور آخر باید بازیافت BP (POP BP) و RET باشد. بدنه برنامه اسمبلی در بین آنها خواهد بود. این طریقه دستیابی به آرگومان ها در پشته، استاندارد است و ذخیره کردن هر ثبات دیگری بر محاسبه جابجایی بی اثر خواهد بود بشرطی که تعداد PUSH و POP در آن برابر باشند.

مثال ۷-۷ روال مکان نما را نشان می دهد که برای ذخیره همه ثبات های تغییر یافته بوسیله روال بازنویسی شده است. بوسیله روال تغییر یافته است.

تحلیل محتوای پشته در نمودار نشان داده شده است.

مثال ۷-۷:

```

.MODEL SMALL
.CODE
PUBLIC _CURSOR
;this procedure is written to be called by C language
_CURSOR PROC
 PUSH BP ;save the BP since contents are altered
 MOV BP,SP
 PUSH AX ;push regs altered by this module
 PUSH DX
 PUSH BX
 MOV DH,[BP+4] ;get x the row value from stack
 MOV DL,[BP+6] ;get y the column value from stack
 MOV AH,02 ;set up for INT call
 MOV BH,00
 INT 10H
 POP DX ;restore registers
 POP BX
 POP AX
 POP BP
_CURSOR END
END

```

|      |    |                            |
|------|----|----------------------------|
| 197E | BL | BX                         |
| 197F | BH |                            |
| 1980 | DL | DX                         |
| 1981 | DH |                            |
| 1982 | AL | AX                         |
| 1983 | AH |                            |
| 1984 | BP | SP=BP                      |
| 1985 | IP | BP +2 holds return address |
| 1986 | 0F | BP+4 holds x value 15=FH   |
| 1987 | 00 |                            |
| 1988 | 0C | BP+6 holds y value 12=CH   |
| 1989 | 00 |                            |

نمودار محتوای پشته مثال ۷-۷

چگونه پارامترها به C بازگردانده می شوند.

در بخش قبل چگونگی تحویل آرگومان ها از C به پشته و از آنجا به رویه اسمبلی را توصیف کردیم. اگر قرار باشد که C آرگومانی را از رویه اسمبلی دریافت کند انتظار خواهد داشت تا پارامتر بازگشته را بسته به سبب پارامتر در ثبات معینی دریافت نماید. جدول ۷-۱

در جدول ۷-۱ ستون، ثباتی را که باید برای مقدار برگشت بکار رود مشخص می کند. اگر مقدار برگشتی اشاره گر (آدرس) باشد، AX مقدار IP را برای NEAR و

AX : DX مقدار IP : CS را برای FAR نگه

می دارد. این مطلب در مثال ۷-۸ که در آن مجموع x و y و z از طریق DX و AX به C بازگردانده شده دیده می شود. DX حاوی کلمه بالاتر و AX حاوی کلمه پایین تر است. در توضیح مربوط به چهارچوب پشته توجه کنید که چون رویه FAR است هر دو مقدار CS و IP در پشته ذخیره می شوند. بنابراین برای دستیابی به



| نوع داده C  | اندازه | نات   |
|-------------|--------|-------|
| char, short | 1 بایت | AL    |
| int         | 2 بایت | AX    |
| long        | 4 بایت | DX:AX |

جدول ۱-۷. مقادیر بازگشته از رویه‌های اسمبلی

آرگومان‌های C، لازم است تا جابجایی BP + 6، BP + 8 و BP + 10 بکار روند. به عنوان یک قاعده اگر رویه اسمبلی NEAR باشد، آخرین آرگومان انتقالی توسط C، با جابجایی BP+4 دستیابی می‌شود و اگر FAR باشد با جابجایی BP + 6 دستیابی خواهد شد. به منظور پرهیز از هرگونه نگرانی در مورد اسمبلرهای جدید ساده‌تری که برای کاربر نوشته شده‌اند، در بخش زیر توضیحات لازم داده شده است.

```
extern unsigned long sum (int, int, int);
{
main()
printf("The sum is equal to %u", sum (500,6500,200));
}
The following is sum.asm
```

مثال ۸-۷: مقادیر int بوسیله یک تابع C به یک رویه اسمبلی تحویل شده است. کد اسمبلی آنها را با هم جمع کرده و حاصل را با جمع کل به C باز می‌گرداند تا نتیجه نمایش داده شود.

```
.MODEL MEDIUM
.CODE
PUBLIC _SUM
;this far procedure gets three words form stack and adds
;them together. At the end DX:AX has the total sum
_SUM PROC FAR
PUSH BP ;save BP
MOV BP,SP ;use it as SP
SUB AX,AX ;clear AX
MOV DX,AX ;and DX
ADD AX,[BP+6] ;add the first
ADC DX,0 ;add the carry
ADD AX,[BP+8] ;add the second
ADC DX,0 ;add the carry
ADD AX,[BP+10] ;add the third
ADC DX,0 ;add the carry
POP BP ;restore BP
RET ;go back to C
_SUM END
```

### اسمبلرهای جدید و پیوند با C

|      |    |                             |
|------|----|-----------------------------|
| 17F0 |    |                             |
| 17F1 |    |                             |
| 17F2 | BP | SP=BP                       |
| 17F3 |    |                             |
| 17F4 | IP |                             |
| 17F5 |    |                             |
| 17F6 | CS |                             |
| 17F7 |    |                             |
| 17F8 |    | x value pointed at by BP+6  |
| 17F9 |    |                             |
| 17FA |    | y value pointed at by BP+8  |
| 17FB |    |                             |
| 17FC |    | z value pointed at by BP+10 |
| 17FD |    |                             |

نمودار محتوای پشته مثال ۸-۷

در سال‌های اخیر برخی از اسمبلرها بسیار ساده با C پیوند داده شده‌اند. استفاده از MASM 5.1 یا TASM1.0 و بالاتر، به نگرانی‌های جابجایی، و یا نام‌های مشترک C و اسمبلی همراه با خط تیره (-)، یا در مورد ذخیره سازی BP پایان می‌دهد. مثال برنامه ۸-۷ در زیر دوباره نوشته شده است. به حرف C در رهنمون "MUDEL SMALL C" توجه کنید. این موجب می‌شود تا اسمبلر [BP + n] را برای همه پارامترها محاسبه کنید. این دو برنامه را مقایسه کنید تا راحتی اسمبلرهای جدید را دریابید. مثال ۸-۷ در قالب جدید چنین است.

```

.MODEL MEDIUM, C
.CODE
PUBLIC SUM
;this far procedure gets three words from stack and adds
;them together. At the end DX:AX has the total sum
SUM PROC FAR DATA1:WORD, DATA2:WORD, DATA3:WORD
SUB AX,AX ;CLEAR AX
MOV DX,AX ;CLEAR DX
ADD AX,DATA1
ADC DX,0
ADD AX,DATA2
ADC DX,0
ADD AX,DATA3
ADC DX,0
RET
SUM ENDP
END

```

### انتقال آدرس‌های آرایه‌ای از C به پشته

زبان C متغیرها را براساس مقدار و یا بصورت آرایه‌ای، به کمک یک اشاره‌گر منتقل می‌نماید. به بیان دیگر، آدرس تفاوت مکان آرایه اگر مدل حافظه SMALL یا MEDIUM باشد در پشته درج می‌شود در غیر این صورت هر دو آدرس قطعه و تفاوت مکان از آرایه در پشته درج خواهند شد. مثال ۷-۹ این نکته را تشریح می‌نماید. این برنامه از آرایه زبان C برای تعریف حقوق روزانه ۵ روز در هفته با استفاده از یک تعریف داده (255 تا 0) استفاده می‌کند. سپس از برنامه اسمبلی برای جمع آنها و بازگشت جمع کل به C برای نمایش استفاده کرده است.

مثال ۷-۹.

```

int wages[5] = {154, 169, 98, 129, 245};
extern unsigned short weekpay(int wages[]);
main()
{
printf("Weekly pay = %u", weekpay(wages));
}

```

weekpay.asm is as follows:

```

.MODEL MEDIUM
.CODE
PUBLIC _WEEKPAY
;this procedure add five bytes together.
;At the end AX has the total sum.
_WEEKPAY PROC FAR
PUSH BP ;save BP
MOV BP,SP
PUSH SI ;save SI
SUB AX,AX
MOV CX,5
MOV SI,[BP+6]
AGAIN: ADD AL,[SI] ;add a day's wages
ADC AH,0
INC SI ;increment pointer to next wage
INC SI
LOOP AGAIN
POP SI
POP BP
RET
_WEEKPAY ENDP
END

```

### پیوند روال‌های زبان اسمبلی با C

مراحل زیر چگونگی پیوند Micro soft Quick C را با روال‌های زبان اسمبلی MASM نشان می‌دهد.

- ۱- اطمینان حاصل کنید که رویه زبان اسمبلی به عنوان PUBLIC مشخص شده باشد. نام رویه باید با علامت زیر خط، در آغاز شود. مثلاً اگر رویه در C "sum" نام گذاری شود باید آن را در اسمبلی "sum\_" نام نهاد. رویه زبان اسمبلی را برای مدل کوچک از نوع NEAR و برای مدل متوسط از نوع FAR اختیار کنید.
- ۲- در برنامه C، رویه را به عنوان بیرونی (external) معرفی نمایید.
- ۳- برنامه زبان اسمبلی را با MASM برپا کنید تا فایل مقصود تولید شود. مثلاً Module 1.obj.
- ۴- برنامه C را برای تولید فایل مقصود (object) کامپایل نمایید: مثلاً Prog1.obj.
- توجه کنید که وقتی فرمان کامپایلر QCL را بکار می‌برید، سیستم تولید یک مدل کوچک را پیش فرض می‌کند. در محیط‌های مجتمع Quick C، پیش فرض مدل متوسط خواهد بود.
- ۵- آنها را برای تولید یک فایل قابل اجرا بهم پیوند بزنید.

C> link prog10obj + module.obj

در بورلند C++، بطریق زیر می‌توانند پیوند بخورند:

C> bcc prog1.cmodule1.asm

فرمان "bcc" برنامه C را کامپایل خواهد کرد. برای اسمبل کردن برنامه زبان اسمبلی از اسمبلر TASM استفاده نمایید و سپس آنها را بهم پیوند بزنید. توجه داشته باشید که حالت بورلند C حساس است. اگر رویه شما در برنامه زبان اسمبلی "SUM\_" و در زبان C "SUM" نامیده می‌شود، پیوندزن قادر نیست آنها را بهم پیوند بزند. نام را در برنامه زبان اسمبلی به حروف کوچک تبدیل کنید. بهنگام استفاده از بورلند C++، پیشنهاد می‌شود از توسعه نام فایل "cpp" استفاده نکنید زیرا این موجب می‌شود تا نام تابع تکه پاره شود و بنابراین پیوندزن قادر نخواهد بود تابع را پیدا کند. اگر در بکارگیری توسعه "CPP" اجبار دارید باید برنامه C را با انتخاب /S کامپایل کنید تا لیستی از زبان اسمبلی بدست آید و ببینید چگونه نام تابع تکه پاره شده و لذا آن نام را در برنامه اسمبلی خود بکار برید. مثلاً اگر نام تابع "sum" بصورت @Sum q \$ iii لیست شده باشد در اینصورت تمام ارجاعات به "sum" در زبان اسمبلی باید به @Sum q \$ iii اصلاح شود تا امکان پیوند برنامه به برنامه C++ فراهم گردد.

## سؤالات مروری

- ۱- یک برنامه C می‌تواند یک برنامه زبان اسمبلی را فرا بخواند و یا کد ... را بکار برده و کدهای زبان اسمبلی را در برنامه C تزریق کند.
- ۲- استاندارد تحویل پارامترها به توابع را در C توصیف نمایید.

## خلاصه

برنامه‌نویسی مدولی شامل شکستن یک پروژه به زیر برنامه‌های مستقل است. هر زیر برنامه مجموعه وظایف مشخصی را برعهده دارد. تجربیات برنامه‌نویسی صحیح، مستندسازی متغیرهای ورودی و خروجی را اجتناب ناپذیر می‌سازد. متغیرهای درون زیر برنامه‌ها با رهنمون‌های EXTRN و PUBLIC تعریف می‌شوند. این رهنمون موجب می‌شوند تا کامپیوتر بتواند متغیرها را مستقر سازد. روش‌های متنوعی برای تحویل پارامترها بکار رفته است که از آن جمله روش با ثبات، با حافظه و یا با پشته را می‌توان نام برد.

زبان برنامه‌نویسی C محبوبیت بسیاری را کسب کرده است زیرا می‌توان براحتی کد را از یک ماشین به ماشین دیگر منتقل کرد. با این وجود گاهی پیشنهاد می‌شود که برنامه‌های زبان اسمبلی به آن الصاق گردد تا بدین ترتیب سرعت افزایش یابد. روال‌های زبان اسمبلی می‌توانند بوسیله برنامه‌های C فراخوانی شوند و یا کدهای زبان اسمبلی می‌توانند مستقیماً در برنامه C با تکنیک درون برنامه‌ای نوشته شوند.

۱- در برنامه زیر محل های خالی را پر کنید. برنامه اصلی داده ها را تعریف کرده و مدول دیگر را برای جمع 5 بایت از داده فرا می خواند، و سپس نتیجه را ذخیره می نماید. توجه: بعضی از مکان های خالی ممکن است نیاز به پر شدن نداشته باشند.

```

.MODEL SMALL
.STACK 100H
.DATA
PUBLIC
DATA1 DB 25,12,34,56,98
RESULT DW ?
.CODE
EXTRN :FAR
HERE: MOV AX,@DATA
 MOV DS,AX
 CALL SUM
 MOV AH,4CH
 INT 21H
 END

```

در فایل دیگر مدولی برای جمع 5 بایت وجود دارد

```

.MODEL SMALL
 DATA1:BYTE
 RESULT:WORD
COUNT EQU 5
.CODE
 SUM
SUM PROC
MOV BX,OFFSET DATA1
SUB AX,AX
MOV CX,COUNT
 : ADD AL,BYTE PTR [BX]
 ADC AH,0
 INC BX
 LOOP AGAIN
 MOV RESULT,AX
 RET
 ENDP
 END

```

- ۲- اگر یک برجسب یا پارامتر در یک مدول تعریف نشود، باید به عنوان ... شناخته شود.
- ۳- اگر یک برجسب یا پارامتر بوسیله مدول های دیگر بکار رود، باید به عنوان ... در مدول جاری شناخته شود.
- ۴- انتخاب های رهنمون EXTRN را وقتی که به یک رویه ارجاع می دهد فهرست کنید.
- ۵- انتخاب های رهنمون EXTRN را وقتی که به یک داده ارجاع می دهد فهرست کنید.
- ۶- انتخاب های رهنمون PUBLIC را وقتی که به یک رویه ارجاع می دهد لیست نمایید.
- ۷- انتخاب های رهنمون PUBLIC را وقتی که به یک داده ارجاع می دهد لیست نمایید.
- ۸- برنامه ۱-۴ را با تبدیل هر زیر روال INT به یک مدول تبدیل نمایید. هر مدول باید NEAR باشد. برنامه را اسمبل و تست نمایید.

۹- فرض کنید چهار مدول وجود دارد که نام قطعه آنها CODSG-1 CODSG-2 CODSG-3 و CODSG-4 می باشد. نقطه ورود برنامه طبق نقشه پیوند برای GODSG-1 برابر 00040H است.

| شروع   | توقف  | طول    | نام     |
|--------|-------|--------|---------|
| 00040H | ----- | 0000BH | CODSG_1 |
| -----  | ----- | 00014H | CODSG_2 |
| -----  | ----- | 00025H | CODSG_3 |
| -----  | ----- | 00041H | CODSG_4 |

(الف) محل های خالی را با فرض اینکه قطعات از نوع PARA فرض شده باشند پر کنید.

(ب) محل های خالی را با فرض اینکه قطعات از نوع WORD باشند پر کنید.

۱۰- فرض کنید هر چهار قطعه در مسئله ۹ بصورت زیر تعریف شوند:

CDSEG SEGMENT PARAPUBLIC 'CODE'

نقشه پیوند را برای قطعات کد بنویسید.

۱۱- نقشه پیوند تعریف قطعه کامل را با تعریف قطعه مرسوم مقایسه کنید. آیا رشته قطعات نقشی در نقشه پیوند

دارد؟ اگر جواب مثبت است توضیح دهید که چرا.

۱۲- برنامه ای بنویسید که دو عدد بی علامت را (هر کدام کمتر از ۹۹۹) از صفحه کلید بپذیرد، آنها را به مبنای شانزده

تبدیل کند، متوسط بگیرد و نتیجه را روی مانیتور نمایش دهد. از مدول های تبدیل شانزده شانه دهی به دهی

و اسکی به شانزده شانه دهی در متن درس استفاده کنید.

۱۳- برنامه ای با اجزاء زیر مشابه با برنامه ۷-۲ بنویسید

(الف) در برنامه اصلی، دو مقدار تعریف شده اند: ۱۲۲۸ و ۵۲۴۰۰.

(ب) برنامه اصلی دو مدول جدا را فرا می خواند، مقادیر را بوسیله پشته انتقال می دهد.

(پ) در مدول اول دو عدد درهم ضرب می شوند و نتیجه به مدول اصلی بازگردانده می شود.

(ت) مدول دوم تقسیم دو عدد (۵۲۴۰۰/۱۲۲۸) را انجام داده و خارج قسمت و باقیمانده را برای ذخیره شدن به

مدول اصلی باز می گرداند.

(ث) پشته و محتوای آن را برای هر مدول بررسی کنید بشرطی که قبل از اولین دستور CALL مقدار

SP=FFFFH باشد.

۱۴- برنامه ۷-۳ را برای درج آدرس های تفاوت مکان داده اصلاح کنید. نمودار پشته را بکشید.

۱۵- یک برنامه اسمبلی درون برنامه ای برای استقرار مکان نما در سطر ۱۴ ستون ۲۷ نوشته و سپس پیام

"This is a Test" را نمایش دهید.

۱۶- مثال ۷-۸ را برای جمع حقوق دوازده ماهه اصلاح کنید. جمع کل حقوق سالانه بیشتر از \$65535 نیست.

## ✓ پاسخ به سؤالات مروری

بخش ۷-۱: نوشتن و پیوند مدول ها

۱- (۱) هر مدول می تواند جداگانه ایجاد شود و این موجب ایجاد موازی آنها شده و زمان طراحی کوتاه می گردد،

(۲) عیوب راحت تر مکان یابی می شوند، (۳) این مدول ها قابل پیوند با زبانهای سطح بالا مانند C هستند.

۲- PUBLIC -۳ EXTRN

۴- در مدولی که دارای نقطه ورود و خروج است، پس از عبارت END برچسب وجود دارد.

۵- یک پاراگراف مشکل از ۱۶ بایت بوده و آدرس آن به 0H ختم می شود.

۶- name SEGMENT PARA STACK 'STACK'

۷- EXTRN TOTAL:WORD

۸- 60H

۹- name SEGMENT PARA PUBLIC 'CODE'

بخش ۷-۲: برخی از مدول های بسیار مفید

۱-

|                        |               |                |
|------------------------|---------------|----------------|
| 1st iteration: AX=F624 | F624/A = 189D | remainder DL=2 |
| 2nd iteration: AX=189D | 189D/A=0276   | remainder DL=1 |
| 3rd iteration: AX=0276 | 0276/A=003F   | remainder DL=0 |
| 4th iteration: AX=003F | 003F/A=6      | remainder DL=3 |
| 5th iteration: AX=0006 | 0006/A=0      | remainder DL=6 |

AX is now zero, so the conversion is complete: F624H=63012<sub>10</sub>

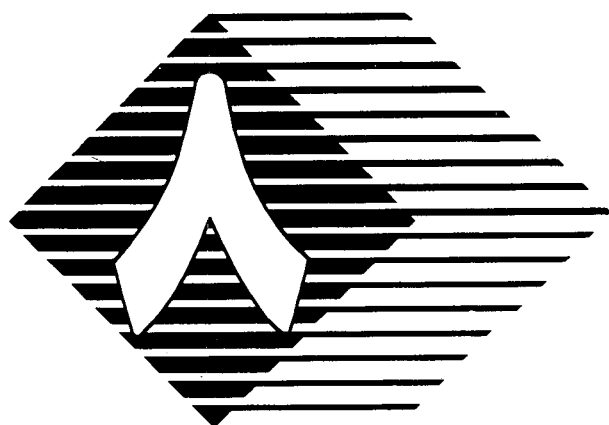
1st iteration: AL=36      06x1=6      DI=6  
 2nd iteration: AL=35      05xA=32      DI=6+32=38  
 3rd iteration: AL=34      04x64=190      DI=38+1C8  
 4th iteration: AL=31      01x03E8=03E8      DI=1C8+03E8=05B0  
 5th iteration: AL=30      0x2710=0      DI=05B0  
 BX has been decremented from 4 to 0, is now -1, so the conversion is complete  
 01456<sub>10</sub>=05B0<sub>H</sub>

### بخش ۳-۷: تبادل پارامترها در میان مدول‌ها

- (الف) بوسیله ثبات: یک مزیت، سرعت پردازش ثبات‌هاست؛ یک عیب این است که تعداد ثبات‌های موجود محدود است بنابراین مقادیر قابل توجهی ردوبدل نمی‌شوند.
- (ب) بوسیله پشته؛ یک مزیت این است که ثبات‌های موجود را استفاده نمی‌کند؛ عیب این است که هر خطایی در پردازش پشته موجب فروریختگی سیستم می‌گردد.
- (پ) بوسیله حافظه؛ یک مزیت، ناحیه و سعی برای ذخیره داده وجود دارد؛ عیب آن این است که برنامه قابل حمل به کامپیوترهای دیگر نیست.
- ۲- (الف) و (پ) صحیح‌اند، (ب) و (ت) صحیح نیستند زیرا SP نمی‌تواند در مد اندیسی به SS بکار رود.

### بخش ۴-۷ ترکیب زبان اسمبلی و برنامه‌های C

- ۱- درون برنامه‌ای
- ۲- بجز در آرایه‌ها که با ارجاع انتقال می‌یابند، پارامترها با مقدار تحویل می‌شوند؛ پارامترها بترتیب معکوس لیست آرگومان منتقل می‌گردند؛ پس از درج آرگومان‌ها در پشته، C درج می‌شود، CS:IP برای رویه‌های FAR و IP برای رویه‌های NEAR درج می‌گردند.



---

# برنامه نویسی 32 بیتی برای ماشین های 386 و 486

---

## اهداف

در پایان این فصل قادر خواهید بود:

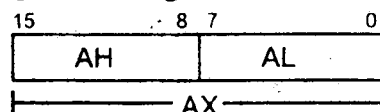
- « تفاوت های مهم بین 80386/486 و 8086/88 را بیان کنید.
- « ثبات های ماشین های 80386/486 را بیان نمایید.
- « اندازه ثبات های موجود در 386/486 را معین کنید.
- « تفاوت میان مد واقعی و حفاظت شده را توصیف نمایید.
- « تفاوت میان کاربرد ثبات میان 8086/286 و 386/486 را توضیح دهید.
- « ملاحظه کنید که چگونه افزایش ثبات موجب افزایش محدوده حافظه در 386/486 می گردد.
- « ببینید چگونه قرار داد ذخیره سازی "منتهی به کوچکتر" ماشین های 80x86 عملوندهایی با اندازه کلمه مضاعف را ذخیره می کند.
- « با استفاده از ثبات توسعه یافته و رهنمون های جدید در 386/486 برنامه نویسی نمایید.
- « عبارات محاسباتی را به کمک ثبات های محاسباتی 386/486 برنامه نویسی نمایید.
- « فاکتور هایی که موجب افزایش کارایی 386/486 نسبت پردازنده های قبلی می شوند را توضیح دهید.

همه برنامه‌هایی که تاکنون مورد بحث قرار گرفتند برای ماشین‌هایی همچون 8088، 8086 و 80286 کامپیوترهای IBM و سازگار با آنها بودند. گرچه این برنامه‌ها روی ماشین‌های مبتنی بر 80386 و 80486 با سرعت بهتری کار می‌کنند، توان واقعی پردازنده‌های 386 و 486 هنگامی ارائه می‌شود که در مد حفاظت شده کار کنند. و اما مد حفاظت شده چیست؟ همانطور که در فصل ۱ ذکر شد، 386 و 486 می‌توانند در دو مد واقعی و حفاظت شده کار کنند. در مد واقعی همه آنها همانند ماشین‌های 8086/286 عمل می‌کنند. با این استثناء که، ثبات‌ها بصورت 32 بیت در دسترسند. این ماشین‌ها هنوز قابلیت آدرس دهی حداکثر 1 مگا بایتی از حافظه را دارند. از آن مهمتر، آنها همه برنامه‌های MS-DOS را بدون هرگونه خطایی اجرا می‌کنند. در مد حفاظت شده می‌توانند تا 4 گیگا بایت از حافظه را دستیابی کنند ولی نیاز به یک سیستم عامل پیچیده‌ای دارند که یکی از وظایفشان تخصیص سطح مزیت 0 تا 3 (0 بالاترین است) به اجرای هر برنامه در CPU است. در این مقطع از زمان فقط سیستم عامل یونیکس از قابلیت حفاظت شده در ریزپردازنده‌های 386 و 486 بهره می‌برد. در مارس ۱۹۹۲، OS/2 IBM ویرایش 0 را برای بهره‌گیری از مد حفاظت شده در ریز پردازنده‌های 386 و بالاتر نوشت. سیستم عامل مذکور ویرایش 32 بیتی سیستم عامل OS/2 بود که برای بهره‌گیری از مزایای مد حفاظت شده در 386 و بالاتر نوشته شد. این در مقابل OS/2 ویرایش 1 است که برای مد حفاظت شده در 80286 و بالاتر نوشته شده بود. ترکیب توان پردازش 32 بیتی و سیستم عامل چند کاره و چند کاربره 386 و 486 را قابل قیاس با مینی کامپیوترهای دهه ۱۹۷۰ می‌نماید با این تفاوت که قیمت آنها کسری از قیمت مینی هاست. جمله چند کاربره به سیستمی اطلاق می‌شود که قادر به پشتیبانی بیش از یک پایانه / صفحه کلید در هر زمان می‌باشد. چند کاره‌ای به سیستم‌هایی گفته می‌شود که قادر به اجرای بیش از یک برنامه در هر زمان است.

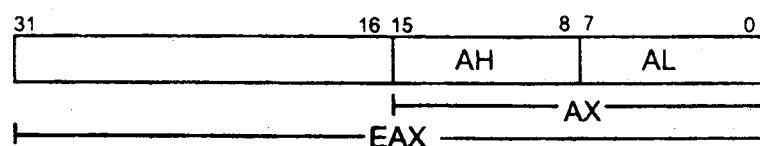
در این بخش ما مشخصات ریزپردازنده‌های 386 و 486 را در مد واقعی و تأثیرشان را در برنامه نویسی بحث خواهیم کرد. آنگاه تعدادی برنامه‌های نمونه که از توانمندی 32 بیتی این ماشین‌ها سود می‌جویند داده شده است. در پایان زمان‌های اجرای چند برنامه که بر روی ماشین‌های 80x86 اجرا می‌شوند مقایسه می‌گردند تا سرعت کامپیوترهای مبتنی بر 386/486 ملاحظه شود. بحث کاملی از مد حفاظت شده و دیگر توانمندی‌های این پردازنده‌ها در این کتاب بحث نشده زیرا این خود یک جلد را شامل می‌شود.

## بخش ۸-۱: ماشین‌های 80386/80486 در مد واقعی

در این بخش ما بر روی برخی از تفاوت‌های عمده بین 8086/286 و 386/486 در مد واقعی تأکید خواهیم کرد. یکی از تفاوت‌های عمده در میان آنها سایز ثبات است. در حالی که در 8086/286 حداکثر اندازه ثبات 16 بیت است، در 386/486 این قابلیت به 32 بیت می‌رسد. به منظور انعکاس این گسترش نام همه ثبات‌ها تعویض شده است. بنابراین AX تبدیل به EAX، BX به EBX و غیره شده‌اند که موضوع مذکور در زیر تشریح و در جدول ۸-۱ نیز آورده شده است. مثلاً، 386/486 دارای ثبات‌های AL، AH، AX و EAX با ترتیب 8، 8، 16 و 32 بیت است. در 86/286 ثبات AX قابل دسترسی بصورت AL، یا AH یا AX است، در حالی که در 386/486، ثبات EAX قابل دسترسی بصورت AL، AH، AX و یا EAX می‌باشد. به بیان دیگر، 16 بیت فوقانی EAX به عنوان یک ثبات جداگانه نیست. قانون مشابهی برای EBX، ECX، EDX نیز صادق است. ثبات‌های DI، SI، BP و SP بترتیب EDI، ESI، EBP و ESP خواهند شد. در واقع 386/486 قادر است DI، SI، ESI، BP، EBP، SP و ESP را دستیابی نماید. همه این ثبات‌ها در مد واقعی قابل دستیابی‌اند.



ثبات 16 بیت AX  
از 8086/80286



ثبات 32 بیت EAX  
از 80386/486



بیتی موجب می شود تا محدوده آدرس از 1M تجاوز کند. این وضعیتی است که برنامه نویس باید از آن در کامپیوترهای 386/486 که با DOS ویرایش 5 و پایین تر کار می کنند پرهیز نماید. مثلاً برای اجرای دستور "MOV AX,[ESI]" برنامه نویس باید همه 16 بیت بالایی ESI را صفر کند. یعنی تفاوت مکان قطعه در مد واقعی 386/486 تحت DOS از 0000 تا FFFFH می باشد. در زیر بعضی از حالات برنامه نویسی مجاز و غیر مجاز برای مد واقعی 386/486 تحت DOS آمده است.

|     |                   |                                                    |
|-----|-------------------|----------------------------------------------------|
| ADD | EAX,[BX]          | :LEGAL                                             |
| ADD | ECX,[DX]          | :ILLEGAL! EDX CAN BE USED<br>AS POINTER BUT NOT DX |
| MOV | AX,WORD PTR [ECX] | :LEGAL                                             |
| ADD | EBX,[EDX]         | :LEGAL                                             |

### دستیابی به ثبات های 32 بیتی با اسمبلی های رایج

در زبان اسمبلی، رهنمون "386" برای دستیابی به ثبات های 32 بیتی کامپیوترهای 386/486 که در مد واقعی تحت DOS کار می کنند بکار می رود تا بتوان دستورالعمل های جدید ریزپردازنده 386 را بکار برد. هر نسل جدیدی از 80x86 تعدادی دستورالعمل جدید دارد که در پردازنده های پایین تر اجرا نمی شوند و در واقع بسمت بالا سازگارند. به بیان دیگر، استفاده از رهنمون "386" در برنامه به این معنی است که برنامه باید روی کامپیوترهای 386 و بالاتر اجرا شود (486 یا 586) ولی نمیتوان آنها را روی کامپیوترهای مبتنی بر 8088/86 و 286 اجرا نمود. بالعکس همه برنامه هایی که در فصل قبل نوشته شدند قابل اجرا بر روی کامپیوترهای 80x86 می باشند. در زیر رهنمون های اضافی اسمبلی همراه با ریزپردازنده پشتیبانی شده توسط اسمبلی میکروسافت (MASM) و اسمبلی توربو بورلند (TASM) آورده شده اند.

| MASM | TASM  | مفهوم                                                                                |
|------|-------|--------------------------------------------------------------------------------------|
| .86  | 8086P | بصورت پیش فرض روی هر کامپیوتر 80x86 کار می کند                                       |
| .286 | P286  | روی هر کامپیوتر 286 و بالاتر کار می کند و نیز از دستورات جدید 286 نیز استفاده می شود |
| .386 | P386  | روی هر کامپیوتر 386 و بالاتر کار می کند، و نیز از دستورات جدید 386 استفاده می نماید  |
| .486 | P486  | روی هر کامپیوتر 486 و بالاتر کار می کند و نیز از دستورات جدید 486 استفاده می نماید.  |

برنامه ۸-۱ چگونگی استفاده از رهنمون "386" و دستورات 32 بیتی را نشان می دهد. از فرم تعریف قطعه ساده شده استفاده شده است. برنامه از ثبات 32 بیتی EAX برای جمع و تفریق مقادیر با سائزهای مختلف استفاده کرده است تا برنامه نویسی 32 بیتی 386/486 تحت DOS را بنمایش بگذارد. اکنون سؤال این است که چگونه این برنامه را اجرا کنیم و محتوای ثبات های 386/486 را ببینیم. متأسفانه ابزار DEBUG که در فصل های قبلی بکار رفت نمی تواند مورد استفاده قرار گیرد زیرا فقط قادر است ثبات های 16 بیتی را نشان دهد. در بسیاری از اسمبلرها، از جمله MASM، TASM، ابزارهای پیشرفته عیب یابی وجود دارد که می توانند برای ملاحظه اجرای برنامه های 386/486 بکار می روند. در ارتباط با MASM، ابزار Code View و برای TASM، Turbo Debugger ابزارهایی هستند که برای ردیابی اجرای 386/486 و نیز برنامه های 8086/88 و 286 بکار می روند. در زیر ردیابی برنامه ۸-۱، به کمک برنامه Code View از میکروسافت نشان داده شده است. برای واریسی ثبات های 32 بیتی در Code View، F را فشارید تا ثبات ها نمایش داده شوند؛ سپس از فهرست انتخاب ها در بالای صفحه نمایش، انتخاب خود را انجام دهید تا منوی پایین تری ظاهر شود. از این منو "386" را برای نمایش ثبات ها در قالب 32 بیتی برگزینید.

با استفاده از ثبات‌های 32 بیتی در 386 برنامه‌ای بنویسید تا مقادیر 100000، 200000 و 40000 را جمع کند. سپس از حاصل جمع کل 80000، 35000 و 250 را کسر کنید. نتیجه را با استفاده از رهنمون کلمه مضاعف DD که برای اعداد 32 بیت بکار می‌رود ذخیره کنید.

```

TITLE ADD AND SUBTRACT USING 32-BIT REGISTERS IN 386 MACHINES
PAGE 60,132
.MODEL SMALL
.386
.STACK 200H
.DATA
RESULT DD ?
.CODE
BEGIN: MOV AX,@DATA
 MOV DS,AX
 SUB EAX,EAX
 ADD EAX,100000 ;EAX = 186A0H
 ADD EAX,200000 ;EAX = 186A0H + 30D40 H = 493E4H
 ADD EAX,40000 ;EAX = 493E4H + 9C40H = 53020H
 SUB EAX,80000 ;EAX = 53020H - 13880H = 3F7A0H
 SUB EAX,35000 ;EAX = 3F7A0H - 88B8H = 36EE8H
 SUB EAX,250 ;EAX = 36338H - FAH = 36DEEH = 224750 decimal
 MOV RESULT,EAX
 MOV AH,4CH
 INT 21H
 END BEGIN

```

#### برنامه ۸-۱

| File View Search Run Watch Options Language Calls Help   F8=Trace F5=Go |              |     |                      |  |  |  |  |              |  |
|-------------------------------------------------------------------------|--------------|-----|----------------------|--|--|--|--|--------------|--|
| 4833:0000                                                               | B83648       | MOV | AX,4836              |  |  |  |  | EAX=00036DEE |  |
| 4833:0003                                                               | 8ED8         | MOV | DS,AX                |  |  |  |  | EBX=00000000 |  |
| 4833:0005                                                               | 662BC0       | SUB | EAX,EAX              |  |  |  |  | ECX=00000000 |  |
| 4833:0008                                                               | 6605A0860100 | ADD | EAX,000186A0         |  |  |  |  | EDX=00000000 |  |
| 4833:000E                                                               | 6605400D0300 | ADD | EAX,00030D40         |  |  |  |  | ESP=00000200 |  |
| 4833:0014                                                               | 6605409C0000 | ADD | EAX,00009C40         |  |  |  |  | EBP=00000000 |  |
| 4833:001A                                                               | 662D80380100 | SUB | EAX,00013880         |  |  |  |  | ESI=00000000 |  |
| 4833:0020                                                               | 662DB8880000 | SUB | EAX,000088B8         |  |  |  |  | EDI=00000000 |  |
| 4833:0026                                                               | 662DFA000000 | SUB | EAX,000000FA         |  |  |  |  | DS=...4836   |  |
| 4833:002C                                                               | 66A30400     | MOV | DWord Ptr [0004],EAX |  |  |  |  | ES=...4823   |  |
| 4833:0030                                                               | B44C         | MOV | AH,4C                |  |  |  |  | FS=...0000   |  |
| 4833:0032                                                               | CD21         | INT | 21                   |  |  |  |  | GS=...0000   |  |
| 4833:0034                                                               | 0000         | ADD | Byte Ptr [BX+SI],AL  |  |  |  |  | SS=...4837   |  |
| 4833:0036                                                               | 0000         | ADD | Byte Ptr [BX+SI],AL  |  |  |  |  | CS=...4833   |  |
| 4833:0038                                                               | 0000         | ADD | Byte Ptr [BX+SI],AL  |  |  |  |  | IP=0000002C  |  |
| 4833:003A                                                               | 0000         | ADD | Byte Ptr [BX+SI],AL  |  |  |  |  |              |  |
| 4833:003C                                                               | 0000         | ADD | Byte Ptr [BX+SI],AL  |  |  |  |  | NV UP        |  |
| 4833:003E                                                               | 0000         | ADD | Byte Ptr [BX+SI],AL  |  |  |  |  | EI NG        |  |
|                                                                         |              |     |                      |  |  |  |  | NZ AC        |  |
|                                                                         |              |     |                      |  |  |  |  | PE NC        |  |
| >t                                                                      |              |     |                      |  |  |  |  |              |  |
| >t                                                                      |              |     |                      |  |  |  |  |              |  |
| >t                                                                      |              |     |                      |  |  |  |  | DS:0004      |  |
| >                                                                       |              |     |                      |  |  |  |  | 00000000     |  |

برنامه ۸-۱. صفحه نمایش Code View از اجرای برنامه

## بازدید مجدد منتهی به کوچتر

در تحلیل اینکه چگونه 386/486 داده 32 بیتی را در حافظه ذخیره می‌کند و یا عملوند 32 بیتی را در ثبات بار می‌کند، قرارداد منتهی به کوچتر را بیاد آورید؛ که در آن بایت پائینی به آدرس پائین‌تر و بایت بالایی به آدرس بالاتر می‌رفت. مثلاً در دستورالعمل مثل "MOV RESULT,EAX" در برنامه ۱-۸ داده بطریق زیر ذخیره می‌گردد.

| تغییر مکان | محتوا   |
|------------|---------|
| RESULT     | d0-d7   |
| RESULT+1   | d8-d15  |
| RESULT+2   | d16-d23 |
| RESULT+3   | d24-d31 |

مثال ۱-۸: فرض کنید SI=1298 و EAX=41992F56H باشند. محتوای مکان‌های حافظه را پس از اجرای دستور "MOV [SI],EAX" نشان دهید.

حل: (در شانزده شانزدهی)

DS:1298 = (56)

DS:1299 = (2F)

DS:129A = (99)

DS:129B = (41)

## سؤالات مروری

- در 80386/486 بیت‌های ثبات EDX می‌تواند مثل DL، بیت‌های ... تا ...، یا DH، بیت‌های ... تا ... یا DX، بیت‌های ... تا ... یا EAX، بیت‌های ... تا ... دستیابی شوند.
- در قطعه 386/486 ساین قطعه ... بایت است.
- مد واقعی چیست؟ مد حفاظت شده چیست؟
- صحیح یا غلط: دستورالعمل "MOV DX,[AX]" در 8086 غیرمجاز است و در 386/486 دستور "MOV DX,[EAX]" مجاز می‌باشد.
- ثبات قطعه پیش فرض وقتی EAX به عنوان اشاره‌گر بکار رود چیست؟
- هدف از رهنمون "386" چیست؟
- اگر DI = 148F، FBX = 6B2415F9 باشند، محتوای حافظه را بعد از اجرای دستور "MOV[DI],EBX" نشان دهید.

## بخش ۲-۸: تعدادی برنامه ساده 386/486

یکی از راه‌های افزایش توان پردازش ریز پردازنده تعویض ساین ثبات است. این موجب می‌شود تا بتوان، به جای شکستن آنها به قطعات کوچتر به منظور جای دادن آنها در ثبات‌های کوچتر، آنها را بصورت اعداد بزرگتر پردازش نمود. در ریزپردازنده‌های جدید ثبات‌های 32 بیتی بصورت استاندارد در آمده است. سوپر کامپیوترهای توانمند، ثبات‌های 64 بیتی را بکار می‌برند. در این بخش ما مروری بر برخی از برنامه‌های قبلی خواهیم داشت تا تاثیر تعویض ثبات‌ها را در برنامه نویسی ببینیم. با مقایسه نوع 32 بیتی این برنامه‌ها با 16 بیتی، می‌توان افزایش کارایی برنامه‌های 32 بیتی را مشاهده کرد. تاثیر سرعت در بخش آخر بحث شده است.

### جمع کلمات 16 بیتی با استفاده از ثبات‌های 32 بیتی

برنامه ۱-۳ (ب) از ثبات‌های 16 بیتی برای جمع چند کلمه داده استفاده کرد. حاصل جمع در یک ثبات ذخیره شد و ثبات دیگری هم برای افزودن رقم نقلی مورد استفاده قرار گرفت. هنگامی که از ثبات‌های 32 بیتی استفاده کنیم دیگر نیازی به این کار نیست. ابتدا ذهن خود را با ملاحظه برنامه ۱-۳ (ب) آماده کنید و سپس برنامه ۲-۸ که نوع 32 بیتی همان برنامه است را بررسی نمایید. این برنامه برای CPU های 386/486 نوشته شده است.

|        |                                                 |                               |                          |
|--------|-------------------------------------------------|-------------------------------|--------------------------|
| TITLE  | REVISION OF PROGRAM 3-1B USING 32-BIT REGISTERS |                               |                          |
| PAGE   | 60,132                                          |                               |                          |
|        | .MODEL SMALL                                    |                               |                          |
|        | .386                                            |                               |                          |
|        | .STACK 200H                                     |                               |                          |
|        | .DATA                                           |                               |                          |
| DATA1  | DD                                              | 27345,28521,29533,30105,32375 |                          |
| SUM    | DD                                              | ?                             |                          |
| COUNT  | EQU                                             | 5                             |                          |
|        | .CODE                                           |                               |                          |
| BEGIN: | MOV                                             | AX,@DATA                      |                          |
|        | MOV                                             | DS,AX                         |                          |
|        | MOV                                             | CX,COUNT                      | ;CX is loop counter      |
|        | MOV                                             | SI,OFFSET DATA1               | ;SI is data pointer      |
|        | SUB                                             | EAX,EAX                       | ;EAX will hold sum       |
| BACK:  | ADD                                             | EAX,DWORD PTR[SI]             | ;add next word to EAX    |
|        | ADD                                             | SI,4                          | ;SI points to next dword |
|        | DEC                                             | CX                            | ;decrement loop counter  |
|        | JNZ                                             | BACK                          | ;continue adding         |
|        | MOV                                             | SUM,EAX                       | ;store sum               |
|        | MOV                                             | AH,4CH                        |                          |
|        | INT                                             | 21H                           |                          |
|        | END                                             | BEGIN                         |                          |

### برنامه ۲-۸

برنامه ۲-۳ از فصل ۳ را برای جمع دو عملوند ۸ بایتی با استفاده از ثبات های ۳۲ بیتی بنویسید.

|        |                                                         |                    |                                    |
|--------|---------------------------------------------------------|--------------------|------------------------------------|
| TITLE  | ADD TWO 8-BYTE NUMBER USING 32-BIT REGISTERS IN THE 386 |                    |                                    |
| PAGE   | 60,132                                                  |                    |                                    |
|        | .MODEL SMALL                                            |                    |                                    |
|        | .386                                                    |                    |                                    |
|        | .STACK 200H                                             |                    |                                    |
|        | .DATA                                                   |                    |                                    |
| DATA1  | DQ                                                      | 548FB9963CE7H      |                                    |
|        | ORG                                                     | 0010H              |                                    |
| DATA2  | DQ                                                      | 3FCD4FA23B8DH      |                                    |
|        | ORG                                                     | 0020H              |                                    |
| DATA3  | DQ ?                                                    |                    |                                    |
|        | .CODE                                                   |                    |                                    |
| BEGIN: | MOV                                                     | AX,@DATA           |                                    |
|        | MOV                                                     | DS,AX              |                                    |
|        | CLC                                                     |                    | ;clear carry before first addition |
|        | MOV                                                     | SI,OFFSET DATA1    | ;SI is pointer for operand1        |
|        | MOV                                                     | DI,OFFSET DATA2    | ;DI is pointer for operand2        |
|        | MOV                                                     | BX,OFFSET DATA3    | ;BX is pointer for the sum         |
|        | MOV                                                     | CX,02              | ;CX is the loop counter            |
| BACK:  | MOV                                                     | EAX,DWORD PTR [SI] | ;move the operand to EAX           |
|        | ADC                                                     | EAX,DWORD PTR [DI] | ;add the operand to EAX            |
|        | MOV                                                     | DWORD PTR [BX],EAX | ;store the sum                     |
|        | INC                                                     | SI                 | ;point to next dword of operand1   |
|        | INC                                                     | SI                 |                                    |
|        | INC                                                     | SI                 |                                    |
|        | INC                                                     | SI                 |                                    |
|        | INC                                                     | DI                 |                                    |
|        | INC                                                     | DI                 | ;point to next dword of operand2   |
|        | INC                                                     | DI                 |                                    |
|        | INC                                                     | DI                 |                                    |
|        | INC                                                     | BX                 |                                    |
|        | INC                                                     | BX                 |                                    |
|        | INC                                                     | BX                 | ;point to next dword of sum        |
|        | INC                                                     | BX                 |                                    |
|        | INC                                                     | BX                 |                                    |
|        | LOOP                                                    | BACK               | ;if not finished, continue adding  |
|        | MOV                                                     | AH,4CH             |                                    |
|        | INT                                                     | 21H                | ;go back to DOS                    |
|        | END                                                     | BEGIN              |                                    |

### برنامه ۳-۸ (الف)

### جمع داده چند کلمه‌ای در ماشین‌های 386/486

در برنامه ۲-۳، دو عدد چند کلمه‌ای با استفاده از ثبات‌های 16 بیت جمع شدند. هر عدد می‌توانست تا 8 بایت عرض داشته باشد. برنامه چهار بار تکرار می‌شد. با استفاده از ثبات‌های 32 بیتی همانطور که در برنامه ۳-۸ (الف) دیده می‌شود، 386/486 نیاز به دو تکرار دارد. این نوع برنامه جمع چند کلمه‌ای حلقه‌ای، خیلی طولانی و ناکارآمد می‌باشد. با ذخیره‌سازی ثبات پرچم که بیت نقلی جمع 32 بیتی را در پشت‌نگه می‌دارد و سپس جمع عدد چهار، در عوض چهار بار افزایش آن، می‌توان کارایی را بالا برد. حلقه در برنامه ۳-۸ (ب) نشان داده شده است.

this revision of Program 3-1a shows how to save the flags  
before updating the pointers

|       |       |                    |                                   |
|-------|-------|--------------------|-----------------------------------|
| BACK: | MOV   | EAX,DWORD PTR [SI] | ;move the operand to EAX          |
|       | ADC   | EAX,DWORD PTR [DI] | ;add the operand to EAX           |
|       | MOV   | DWORD PTR [BX],EAX | ;store the sum                    |
|       | PUSHF |                    | ;save the flags                   |
|       | ADD   | SI,4               | ;point to next dword of operand1  |
|       | ADD   | DI,4               | ;point to next dword of operand2  |
|       | ADD   | BX,4               | ;point to next dword of sum       |
|       | POPF  |                    | ;restore the flags                |
|       | LOOP  | BACK               | ;if not finished, continue adding |

برنامه ۳-۸ (ب)

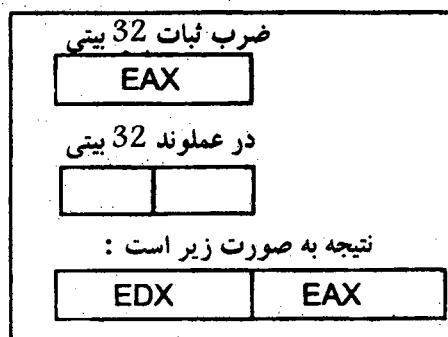
بدلیل جریمه سنگین مربوط به دستورات انشعاب مانند LOOP و J شرطی در 386/486، بهتر است از نوع غیر حلقه‌ای این برنامه، طبق برنامه ۴-۸ استفاده شود.

ابتدا توجه کنید که داده بطریق مشابهی با نوع حلقه‌ای ذخیره می‌گردد. رهنمون داده DQ برای ایجاد امکان ذخیره‌سازی بکار می‌رود. ابتدا کلمه مضاعف (4 بایت) پایین‌تر داده DATA1 به EAX انتقال می‌یابد و بخش پائینی کلمه مضاعف DATA2 با EBX به همراه رقم نقلی حاصل از کلمات پایین رتبه جمع می‌شود. اکنون EAX چهار بایت پایین رتبه نتیجه را نگهداری می‌نماید و EBX هم چهار بایت بالا رتبه نتیجه را حفظ می‌کند. برنامه ۴-۸ از بکارگیری روش حلقه، بسیار کاراتر است. برای ملاحظه علت و نیز به منظور بحث تأثیر دستورات انشعاب و عملکرد برنامه‌های 80386/486، بخش ۳-۸ ملاحظه شود.

|        |                                                                            |                       |                                     |
|--------|----------------------------------------------------------------------------|-----------------------|-------------------------------------|
| TITLE  | ADD TWO 8-BYTE NUMBERS USING 32-BIT REGISTERS IN THE 386 (NO-LOOP VERSION) |                       |                                     |
| PAGE   | 60,132                                                                     |                       |                                     |
|        | .MODEL SMALL                                                               |                       |                                     |
|        | .386                                                                       |                       |                                     |
|        | .STACK 200H                                                                |                       |                                     |
|        | .DATA                                                                      |                       |                                     |
| DATA1  | DQ                                                                         | 548FB9963CE7H         |                                     |
|        | ORG                                                                        | 0010H                 |                                     |
| DATA2  | DQ                                                                         | 3FCD4FA23B8DH         |                                     |
|        | ORG                                                                        | 0020H                 |                                     |
| DATA3  | DQ                                                                         | ?                     |                                     |
|        | .CODE                                                                      |                       |                                     |
| BEGIN: | MOV                                                                        | AX,@DATA              |                                     |
|        | MOV                                                                        | DS,AX                 |                                     |
|        | MOV                                                                        | EAX,DWORD PTR DATA1   | ;move lower dword of DATA1 into EAX |
|        | ADD                                                                        | EAX,DWORD PTR DATA2   | ;add lower dword of DATA2 to EAX    |
|        | MOV                                                                        | EBX,DWORD PTR DATA1+4 | ;move upper dword of DATA1 into EBX |
|        | ADC                                                                        | EBX,DWORD PTR DATA2+4 | ;add upper dword of DATA2 to EBX    |
|        | MOV                                                                        | DWORD PTR DATA3,EAX   | ;store lower dword of result        |
|        | MOV                                                                        | DWORD PTR DATA3+4,EBX | ;store upper dword of result        |
|        | MOV                                                                        | AH,4CH                |                                     |
|        | INT                                                                        | 21H                   |                                     |
|        | END                                                                        | BEGIN                 |                                     |

برنامه ۴-۸

## ضرب عملوند 32 بیتی در عملوند 16 بیتی در 386/486



شکل ۸-۱. ضرب 386/486

به عنوان آخرین مثال از توان ثبات‌های 32 بیتی در ماشین‌های 386/486 در این بخش ما به ضرب عملوند 32 بیتی در 16 بیتی نگاه می‌کنیم. در این رابطه شکل ۸-۱ ملاحظه گردد. مقایسه نوع 386/486 با 86/286 این برنامه، براحتی کارایی برنامه‌نویسی با ثبات‌های 32 بیتی سیستم‌های 386/486 را آشکار می‌سازد. ابتدا به نوع 386/486 ضرب 32 بیتی در 16 بیتی طبق برنامه ۵-۸ نگاه می‌کنیم. توجه کنید که عملوند 16 بیتی در ثبات 32 بیتی قرار گرفته است تا محاسبات 32 بیتی اجرا شود.

با استفاده از 386/486 برنامه ای برای ضرب عملوند 32 بیتی در عملوند 16 بیتی بنویسید.

```
TITLE MULTIPLICATION OF DOUBLE WORD BY WORD USING 386/486
PAGE 60,132
.MODEL SMALL
.386
.STACK 200H
.DATA
DATA1 DD 500000 ;MULTIPLICAND (UP TO 32-BIT SIZE DATA)
DATA2 DD 50000 ;MULTIPLIER (UP TO 16-BIT SIZE)
RESULT DQ ? ;PRODUCT (UP TO 48-BIT SIZE)
.CODE
MAIN: MOV AX,@DATA
 MOV DS,AX
 MOV EAX,DATA1 ;32-BIT OPERAND
 MUL DWORD PTR DATA2 ;TIMES 16-BIT OPERAND
 MOV DWORD PTR RESULT,EAX ;SAVE THE RESULT
 MOV DWORD PTR RESULT+4,EDX
 MOV AH,4CH
 INT 21H
 END MAIN
```

برنامه ۵-۸

به منظور درک توانمندی ثبات‌های 32 بیتی 386/486، عنوان بعدی همان برنامه ۵-۸ را با پردازنده 8086/286 شازنده بیتی نشان می‌دهد.

## ضرب 32 بیت در 16 بیت با ثبات‌های 8086/286

به منظور روشن‌تر کردن بحث زیر، W2 و W1 با سائز کلمه برای نمایش مضروب 32 بیت بکار خواهند رفت. برای ضرب آن در عملوند 16 بیتی W3، الگوریتم زیر باید دنبال شود. فرض کنید که همه مقادیر به مبنای شانزده هستند.

|   |             |                                                                     |
|---|-------------|---------------------------------------------------------------------|
|   | W2 W1       | مضروب                                                               |
| x | W3          | مضروب فیه                                                           |
|   | -----       |                                                                     |
|   | W3xW1       | نتیجه 32 بیتی                                                       |
| + | W3xW2       | نتیجه 32 بیتی باید به چپ به میزان چهار بیت جابجا شود و سپس جمع گردد |
|   | -----       |                                                                     |
|   | X3 X2 X1    | نتیجه 48 بیتی است (X1, X2, X3 سائز کلمه هستند)                      |
|   | W1 DW _____ |                                                                     |
|   | W2 DW _____ | ;W2:W1 IS A 32-BIT MULTIPLICAND                                     |
|   | W3 DW _____ | ;W3 IS A 16-BIT MULTIPLIER                                          |

X1 DW?  
X2 DW? ;X3X2X1 THE 48-BIT PRODUCT RESULT  
X3 DW?

```

...
MOV AX,W1 ;GET THE LOW WORD
MUL W3 ;MULTIPLY
MOV X1,AX ;SAVE THE LOW WORD OF THE PRODUCT
MOV X2,DX ;SAVE THE HIGH WORD OF THE PRODUCT
MOV AX,W2 ;GET THE HIGH WORD
MUL W3 ;MULTIPLY
ADD X2,AX ;ADD THE MIDDLE 16-BIT WORD
ADC DX,0 ;PROPAGATE THE CARRY TO DX
MOV X3,DX ;SAVE THE HIGH WORD RESULT

```

اکنون پس از درک روند فوق، برنامه ۶-۸ قادر خواهد بود تا کد نویسی مربوطه را نشان دهد ابتدا، یک رهنمون DD برای تعریف داده 32 بیتی بجای دوبار نوشتن DW بکار می‌رود. بطور مشابه، چون رهنمونی برای تعریف داده 48 بیتی نداریم، از DQ (تعریف چهار کلمه) استفاده می‌شود که قادر است عملوند 64 بیتی را تعریف کند. بیت‌های استفاده نشده صفر خواهند شد.

با مقایسه این دو برنامه می‌توان دید که چرا ثبات‌های 32 بیتی برای همه نسل‌های جدید ریزپردازنده‌ها استاندارد هستند.

|        |                                                   |                                        |                                    |
|--------|---------------------------------------------------|----------------------------------------|------------------------------------|
| TITLE  | MULTIPLICATION OF DOUBLEWORD BY WORD USING 86/286 |                                        |                                    |
| PAGE   | 60,132                                            |                                        |                                    |
|        | .MODEL SMALL                                      |                                        |                                    |
|        | .STACK 200H                                       |                                        |                                    |
|        | .DATA                                             |                                        |                                    |
| DATA1  | DD 500000                                         | ;MULTIPlicAND (UP TO 32-BIT SIZE DATA) |                                    |
| DATA2  | DW 50000                                          | ;MULTIPLIER (UP TO 16-BIT SIZE)        |                                    |
| RESULT | DQ ?                                              | ;PRODUCT (UP TO 48-BIT SIZE)           |                                    |
|        | .CODE                                             |                                        |                                    |
| MAIN:  | MOV                                               | AX,@DATA                               |                                    |
|        | MOV                                               | DS,AX                                  |                                    |
|        | MOV                                               | AX,WORD PTR DATA1                      | ;GET LOW WORD OF MULTIPlicAND      |
|        | MUL                                               | WORD PTR DATA2                         | ;MULTIPLY THE MULTIPLIER           |
|        | MOV                                               | WORD PTR RESULT,AX                     | ;SAVE LOW WORD OF THE PRODUCT      |
|        | MOV                                               | WORD PTR RESULT + 2,DX                 | ;SAVE MIDDLE WORD OF PRODUCT       |
|        | MOV                                               | AX,WORD PTR DATA1 + 2                  | ;GET THE HIGH WORD OF MULTIPlicAND |
|        | MUL                                               | WORD PTR DATA2                         | ;MULTIPLY THE MULTIPLIER           |
|        | ADD                                               | WORD PTR RESULT + 2 ,AX                | ;ADD THE MIDDLE 16-BIT WORD        |
|        | ADC                                               | DX,0                                   | ;PROPAGATE THE CARRY TO DX         |
|        | MOV                                               | WORD PTR RESULT + 4,DX                 | ;SAVE THE HIGH WORD RESULT         |
|        | MOV                                               | AH,4CH                                 |                                    |
|        | INT                                               | 21H                                    |                                    |
|        | END                                               | MAIN                                   |                                    |

برنامه ۶-۸

### سؤالات مروری

- ۱- تعداد تکرار را برای جمع دو عدد 8 بیتی برای CPU های زیر مقایسه نمایید.  
(الف) برای 8085 CPU (هشت بیت)  
(ب) 8086/88/286  
(پ) 386/486  
(ت) سوپر کامپیوتر Cray
- ۲- برای تعریف عملوندهای 32 و 46 بیتی چه رهنمون‌های داده‌ای بکار می‌روند.
- ۳- برای تفهیم استفاده از دستورات 386 در برنامه به اسمبلر از چه رهنمونی در MASM استفاده می‌شود؟ در ASM T چگونه؟

### بخش ۳-۸: مقایسه عملکرد 80x86

نسل‌های جدید خانواده 80x86 نه فقط ویژگی‌های یک کامپیوتر بزرگ مانند توانمندی حفاظت را دارند بلکه دستورات نسل‌های قدیمی را هم بسیار سریع‌تر اجرا می‌کنند. در بخش قبل دیدیم که یک برنامه‌نویسی 32 بیتی کارا چگونه است. در این بخش ما عملکرد پردازنده‌های خانواده 80x86 را مقایسه خواهیم کرد. برای انجام آن، تعداد سیکل‌های ساعت (تیک) لازم را برای اجرای هر برنامه در CPU های 8086، 286، 386 و 486 مقایسه می‌کنیم. برای درک بقیه مطالب در این بخش، لازم است تا مقدمه بخش ۲- (ب) از پیوست (ب) را مرور کنیم. جدول سیکل‌های ساعت در پیوست (ب) زمان کل اجرا را برای هر پردازنده نشان نمی‌دهد. دلیل این است که محاسبات برحسب میکروثانیه یا میلی ثانیه به طراحی سخت افزار سیستم و عمدتاً روی فاکتورهای فوکانس و طراحی حافظه بستگی دارد. چنین مباحث سخت‌افزاری و تأثیر آن‌ها روی عملکرد کامپیوتر و رای دیدگاه این کتاب است و در جلد دوم این کتاب بحث خواهد شد. مقایسه عملکرد خانواده 80x86 می‌تواند بیکی از دو صورت زیر باشد.

۱- برنامه‌ای را برای 8086 در نظر بگیرید و تعداد پالس‌های ساعت لازم برای اجرای آن را روی پردازنده‌های 8086، 80286، 386 و 486 حساب کنید.

#### اجرای یک برنامه 8086 در خانواده 80x86

اینتل از تکنیک‌های پیشرفته در خط لوله برای افزایش توان پردازش ریزپردازنده‌های 386 و 486 استفاده کرد. مثلاً در بسیاری از دستورات که در 8086 چهار یا پنج پالس ساعت برای اجرا نیاز دارند، در ماشین‌های 386 و 486 با تنها یک یا دو پالس ساعت قابل اجرا هستند. این مطلب در زیر نشان داده شده است.

مسئله ۱-۳ (ب) برنامه محاسبه حاصل جمع پنج کلمه از داده را نشان داد. چون حلقه بخش زمان بر در این برنامه است، در زیر مقایسه‌ای از تعداد کل یک بار تکرار برای همه پردازنده‌های اینتل 80x86 آمده است. تعداد پالس‌های ساعت هر دستور از پیوست (ب) اخذ شده است.

|                   | 8086 | 286 | 386 | 486 |
|-------------------|------|-----|-----|-----|
| BACK: ADD AX,[SI] | 14   | 7   | 6   | 2   |
| ADC BX,0          | 4    | 3   | 2   | 1   |
| INC SI            | 2    | 2   | 2   | 1   |
| INC SI            | 2    | 2   | 2   | 1   |
| DEC CX            | 2    | 2   | 2   | 1   |
| JNZ BACK          | 16/4 | 7/3 | 7/3 | 3/1 |

تعداد پالس‌های ساعت در هر تکرار: 40      23      21      9

برای محاسبه تعداد پالس‌های ساعت برای پنج تکرار تنها کافی است پس از محاسبه صحیح پنجمین تکرار، زمان هر پنج تکرار را با هم جمع کنیم، زیرا تعداد پالس‌ها برای عدم پرش در آخرین تکرار کمتر از حالتی است که پرش وجود دارد. برای تنظیم آن، تفاوت پالس دو حالت را بدست آورید. مثلاً، تعداد کل ساعت برای پنج تکرار در ستون 8086 برابر با  $200 = 40 \times 5$  است. و با اصلاح آخرین تکرار  $(16-4=12)$  را از 200 کسر می‌کنیم که حاصل 188 می‌گردد. با بکارگیری روال مشابهی در 80286 تعداد کل پالس‌های ساعت  $111 = 4 - (23 \times 5)$  را بدست می‌دهد.

داده فوق بزرگ‌تر می‌توان نسل جدید خانواده 80x86 را نشان می‌دهد. برنامه‌ای که ابتدا برای ماشین‌های 8086 نوشته شده است در 386 دو برابر و در 486 چهار برابر سریع‌تر اجرا می‌شود. این ویژگی بعلاوه 32 بیتی بودن گذرگاه داده که خروج و ورود داده را از CPU بعده دارند موجب قویتر شدن 386/486 می‌گردد. اکنون سؤال این است که اگر برنامه با توجه به قدرت پردازش 32 بیتی CPU های 386/486 دوباره نوشته شود سرعت چقدر افزایش می‌یابد. این مطلب در زیر نشان داده شده است. برنامه ۲-۸ نوع 32 بیتی همین برنامه را نشان داد. بخش تکرار برنامه و تعداد پالس‌ها بقرار زیر است.



|                              | <u>386</u> | <u>486</u> |
|------------------------------|------------|------------|
| BACK: ADD EAX,DWORD PTR [SI] | 6          | 2          |
| ADD SI,4                     | 2          | 1          |
| DEC CX                       | 2          | 1          |
| JNZ BACK                     | 7/3        | 3/1        |

تعداد پالس های ساعت در هر تکرار: 17 7

پس از ضرب برای پنج تکرار داریم  $80 = 4 - (5 \times 17)$  که تعداد پالس ساعت برای 386 و  $33 = 2 - (5 \times 7)$  که پالس ساعت برای 486 می باشد. از مقایسه نتایج فوق، می توان نتیجه گرفت که بازنویسی برنامه های 16 بیتی با ثبات های 32 بیتی در 386/486 حدود 25% سریعتر خواهد بود. این مطلب برای بعضی از کاربردها ولی نه برای همه آنها صحیح است. مثلاً حالت جمع عملوندهای چند بایتی را که در زیر آمده است بررسی کنید. سپس عملکرد برنامه ۲-۳ را روی CPU های 80x86 مقایسه نمایید. در زمان های زیر برای برداشت از حافظه (m) دو پالس ساعت فرض می شود.

|                   | <u>8086</u> | <u>286</u> | <u>386</u> | <u>486</u> |
|-------------------|-------------|------------|------------|------------|
| BACK: MOV AX,[SI] | 10          | 5          | 4          | 1          |
| ADC AX,[DI]       | 14          | 7          | 6          | 2          |
| MOV [BX],AX       | 10          | 3          | 2          | 1          |
| INC SI            | 2           | 2          | 2          | 1          |
| INC SI            | 2           | 2          | 2          | 1          |
| INC DI            | 2           | 2          | 2          | 1          |
| INC DI            | 2           | 2          | 2          | 1          |
| INC BX            | 2           | 2          | 2          | 1          |
| INC BX            | 2           | 2          | 2          | 1          |
| LOOP BACK         | 17/5        | (10)/4     | (13)/11    | 7/6        |
| تعداد کل در تکرار | 53          | 37         | 37         | 17         |
| برای چهار تکرار : | 200         | 142        | 146        | 67         |

اکنون این نتایج را با نوع 32 بیتی همان برنامه مقایسه نمایید. نوع اصلاح شده در بخش قبل بحث شد (برنامه ۴-۸) و تعداد پالس های ساعت بقرار زیر است.

|                                           | <u>386</u> | <u>486</u> |
|-------------------------------------------|------------|------------|
| MOV EAX,DWORD PTR DATA1                   | 4          | 1          |
| ADD EAX,DWORD PTR DATA2                   | 6          | 2          |
| MOV EBX,DWORD PTR DATA1+4                 | 4          | 1          |
| ADC EBX,DWORD PTR DATA2+4                 | 6          | 2          |
| MOV DWORD PTR DATA3,EAX                   | 2          | 1          |
| MOV DWORD PTR DATA3+4,EBX                 | 2          | 1          |
| تعداد کل پالس های ساعت برای عملکرد کامل : | 24         | 8          |

در 386، تعداد پالس ساعت از 146 به 24 و در 486 از 67 به 8 کاهش یافته است. با استفاده از سخت افزار یکسان و استفاده از توانمندی 32 بیتی سبب می شود تا سرعت بترتیب 6 بار (146 تقسیم بر 24) و 8.5 بار (67 تقسیم بر 8) برای 386 و 486 افزایش می یابد. بحث فوق بصراحت بیان می کند که در پرازنده های پیشرفته جریمه قابل توجهی وجود دارد. همچنین اگر بخواهیم از منابع قبلی استفاده کرده و برنامه های 16 بیتی را با معماری 32 بیتی بازنویسی کنیم تلاش بیهوده ای نخواهد بود.

- ۱- پالس‌های ساعت را برای اجرای "ADD BX,AX" (ADD reg,reg) در پردازنده‌های 8086، 286، 386 و 486 مقایسه کنید.
- ۲- با استفاده از 8086 به عنوان مبنا، افزایش سرعت را برای هر یک از پردازنده‌های سؤال ۱ مشخص نمایید.
- ۳- در دستور JNZ، جریمه انشعاب را در 8086، 286 و 386 مقایسه کنید. فرض شود  $m = 2$  است.  
(الف) برحسب پالس ساعت (ب) به درصد ( $100 \times$  انشعاب / بدون انشعاب).

## خلاصه

CPU های 386/486 اصلاحات قابل توجهی را نسبت به 8086/286 در چندین زمینه دارا هستند. آنها نه تنها به لحاظ اجرا سریع‌ترند، بلکه توان پردازش بالاتری نیز دارند زیرا 32 بیتی می‌باشند. آنها قادر به دستیابی به فضای حافظه 4 گیگا بیتی نیز می‌باشند. 386/486 چنان طراحی شده‌اند که همه برنامه‌های 8086/286 بدون هر گونه اصلاحی بر روی آنها قابل اجرا باشند. دیگر تغییرات در 386/486 عبارتند از دو ثبات قطعه اضافی و توانمندی استفاده از ثبات‌های همه منظوره به عنوان اشاره‌گر.

## مسائل

- ۱- در برنامه 386/486 محتوای هر ثباتی که در پرانتز نوشته شده را پس از اجرای دستور بدست آورید.

|                   |                    |       |
|-------------------|--------------------|-------|
| MOV EAX,9823F4B6H | (AL,AH,AX and EAX) | (الف) |
| MOV EBX,985C2H    | (BL,BH,BX,EBX)     | (ب)   |
| MOV EDX,2000000   | (DL,DH,DX,EDX)     | (پ)   |
| MOV ESI,120000H   | (SI,ESI)           | (ت)   |

- ۲- مقصد و محتوای هر یک از حالات زیر را نشان دهید

|                    |       |
|--------------------|-------|
| MOV EAX,299FF94H   | (الف) |
| ADD EAX,34FFFFH    | (ب)   |
| MOV EBX,500 000    | (پ)   |
| ADD EBX,700 000    | (ت)   |
| MOV EDX,40 000 000 | (ث)   |
| SUB EDX,1 500 000  |       |
| MOV EAX,39393834H  |       |
| AND EAX,0F0F0F0FH  |       |
| MOV EBX,9FE35DH    |       |
| XOR EBX,0F0F0F0H   |       |

- ۳- با استفاده از قرارداد منتهی به کوچکتر محتوای مقصد را در هر حالت نشان دهید

|              |                                                           |       |
|--------------|-----------------------------------------------------------|-------|
| MOV [SI],EAX | ;ASSUME SI=2000H AND EAX=9823F456H                        | (الف) |
| MOV [BX],ECX | ;ASSUME BX,348CH AND ECX=1F23491H                         | (ب)   |
| MOV EBX,[DI] | ;ASSUME DI=4044H WITH THE<br>;FOLLOWING DATA. ALL IN HEX. | (پ)   |
| DS:4044=(92) |                                                           |       |
| DS:4045=(6D) |                                                           |       |
| DS:4046=(A2) |                                                           |       |
| DS:4047=(4C) |                                                           |       |

- ۴- تعداد پالس‌های ساعت را برای ریزپردازنده 80x86 در هر یک از حالات زیر مقایسه کنید.

(الف) برنامه‌ای برای 8086/286 جهت انتقال 50 کلمه داده، هر بار یک کلمه (16 بیت) بنویسید. از دستورات رشته‌ای استفاده نشود.

(ب) برنامه بخش (الف) را طوری اصلاح کنید که هر بار دو کلمه (32 بیت) را منتقل نماید و تعداد پالس ساعت را برای 386 و 486 محاسبه کنید.

در هر دو حالت (الف) و (ب)، تعداد پالس‌های ساعت باید برای یک تکرار و سپس برای هر تکرارها محاسبه شود.

۵- دستورالعمل DAA که در فصل ۳ بحث شد، فقط روی ثبات AL کار می‌کند. کاربرد نوع پردازنده 80x86

بی‌اهمیت است. برنامه‌ای بنویسید که دو بسته عدد BCD چند بایتی را جمع کند که هر کدام 10 بایت عرض

CPU های 8086، 286، 386 و 486 برای یک تکرار مقایسه

دارند (رهنمون DT

نمایید.

## ✓ پاسخ به سؤالات مروری

بخش ۸-۱: ماشین‌های 80386/80486 در مد واقعی

۱- 0 تا 7، 8 تا 15، 0 تا 15، 0 تا 31

۲- 64k بایت

۳- مد واقعی مشابه عملکرد ماشین 8086 است؛ مد حفاظت شده به برنامه‌ها اولویت می‌بخشد و نیز از مزیت توان 386 بهره می‌گیرد.

۴- صحیح

۵- DS

۶- امکان استفاده از دستورالعمل‌های 386 را اجازه می‌دهد.

۷- DS:1492=6B، DS:1491=24، DS:1490 = 15، DS:148F = F9

بخش ۸-۲: تعدادی برنامه‌های ساده در 386/486

(الف) 8، (ب) 4، (پ) 2، (ت) 1

۲- DQ، DD

۳- 386، P386

بخش ۸-۳: مقایسه عملکرد 80x86

۱- 8086:3 286:2 386:2 486:1

۲- 286 با اندازه 33% نسبت به 8086 اصلاح شده است

386 نسبت به 286 0% اصلاح شده است.

486 نسبت به 386 100% اصلاح شده است.

۳- (الف) 386:6 286:6 8086:8

(ب) 386:300 286:300 8086:400



---

## برنامه نویسی DEBUG

---

DEBUG برنامه‌ای است که در سیستم‌های عامل MS-DOS و PC-DOS گنجانده شد تا به برنامه‌نویس اجازه نظارت بر برنامه را برای رفع عیب بدهد. خصوصاً این برنامه برای بررسی و تغییر محتویات حافظه، ورود و اجرای برنامه‌ها و توقف اجرا در نقاط معین برای واری و حتی تغییر داده نیز مورد استفاده قرار می‌گیرد. این پیوست مقدمه‌ای آموزشی بر برنامه DEBUG است. شما چگونگی ورود و خروج از DEBUG، برنامه‌های ورود، اجرا و عیب‌یابی را خواهید آموخت. چگونگی بررسی و تغییر محتویات ثبات‌ها و حافظه، بعلاوه برخی ویژگی‌های DEBUG را که در نوشتن برنامه‌ها مفیدند می‌آموزید. مثال‌های متعددی از برنامه‌نویسی به زبان اسمبلی در سرتاسر این پیوست داده شده و به همراه آن ارجاع سریع به فرمان‌های خلاصه DEBUG نیز فراهم شده است.

ابتدا چند کلمه‌ای درباره مثال‌های این پیوست گفته می‌شود. آنچه را که شما باید تایپ کنید نیز به خط ایتالیک درشت نوشته خواهد شد؛ یعنی

حروف ایتالیک آنچه را که کاربر تایپ می‌کند نشان خواهد داد  
و پاسخ برنامه DEBUG نیز با حروف ضخیم نشان داده می‌شود

**حروف ضخیم پاسخ کامپیوتر خواهد بود.**

در مثال‌های این پیوست فرض خواهد شد که برنامه DEBUG در دیسک‌گردان A و برنامه شما در دیسک گردان B است اگر سیستم شما به جز این برپا شده است، به هنگام تایپ مشخصات دیسک‌گردان (مانند "B") آن را در خاطر داشته باشید. در اینجا قویاً پیشنهاد می‌شود که مثال‌های DEBUG را تایپ کرده و برای خود تکرار کنید. بهترین راه آموزش، انجام آن است.

## بخش ۱-الف: ورود و خروج از DEBUG

برای ورود به برنامه DEBUG نام آن را در سطح DOS تایپ کنید،

> بازگشت < A>DEBUG

"DEBUG" را ممکن است با حروف بزرگ یا کوچک تایپ نمود. دوباره اجازه دهید تا تصور کنیم که در این مثال برنامه DEBUG در دیسک A فرض شده است. پس از تایپ "DEBUG" و بازگشت نورد، (carriage return)، (کلید enter)، نشانه DEBUG "-" در خط بعد ظاهر می شود. اکنون DEBUG منتظر است تا فرمانی را تایپ کنید.

اکنون که فهمیدید چگونه وارد DEBUG شوید، برای آموختن فرمان های DEBUG آماده اید. اولین فرمان، فرمان خروج است تا به این ترتیب از DEBUG خارج شوید.

فرمان خروج، Q می تواند با حروف بزرگ یا کوچک تایپ شود. این مطلب برای همه فرمان های DEBUG معتبر است. پس از ورود Q و بازگشت، به سطح DOS بازخواهید گشت. این در مثال ۱ - الف نشان داده شده است.

مثال ۱ - الف: ورود و خروج از DEBUG

```
A>DEBUG <return>
-Q <return>
A>
```

## بخش ۲-الف: بررسی و تغییر محتوای ثبات ها

فرمان ثبات به شما اجازه می دهد تا محتوای ثبات های درونی CPU را بررسی و یا تغییر دهید. فرمان R دارای ترکیب زیر است.

R <نام ثبات>

فرمان R همه ثبات ها را نمایش می دهد مگر اینکه میدان خاصی <نام ثبات> وارد شود، که در این حالت فقط ثبات نام برده شده نمایش داده خواهد شد.

مثال ۲-الف: با استفاده از فرمان R تمام ثبات ها را نمایش دهید.

```
A>DEBUG <return>
-R <return>
```

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C44 ES=0C44 SS=0C44 CS=0C44 IP=0100 NV UP DI PL NZ NA PO NC
0C44:0100 0000 ADD [BX+SI],AL DS:0000=CD
```

پس از تایپ R و enter، DEBUG با سه خط اطلاعات به آن پاسخ می دهد. اولین خط، ثبات های همه منظوره و محتوای ثبات های اندیس را نمایش می دهد. خط دوم محتوای ثبات های قطعه، مقدار جاری ثبات دستورالعمل و بیت های ثبات پرچم را نشان می دهد. کدهای دو خط آخر، "NV UP DI ... NC" نشانگر وضعیت هشت بیت ثبات پرچم است. ثبات پرچم و نمایش آن در بخش ۶ - الف بحث شده است. خط سوم برخی اطلاعات مفید را وقتی در DEBUG برنامه نویسی می نمایید در اختیار می گذارد. این خط دستوری را نشان می دهد که با CS:IP به آن اشاره می شود. خط سوم فوق ممکن است با آنچه که در سیستم خود دارید متفاوت باشد. برای اهداف خود به دو خط اول بسنده کنید. توضیحات لازم برای خط سوم تا آخر این پیوست به تعویق می افتد.

وقتی در ابتدا وارد DEBUG می شوید، ثبات های همه منظوره صفر شده و بیت های پرچم پاک می شوند. محتویات ثبات های قطعه به سیستمی که بکار می برید بستگی دارد، ولی همه ثبات های قطعه دارای مقدار یکسانی می باشند، که بوسیله سیستم عامل DOS معین می گردد. مثلاً در مثال ۲ - الف فوق همه ثبات های قطعه حاوی 0C44H هستند. پیشنهاد می شود که محتویات ثبات ها را عوض نکنید زیرا این مقادیر بوسیله سیستم عامل ایجاد شده اند. توجه: در بخش بعدی این پیوست چگونگی بارکردن برنامه اسمبلی را در DEBUG نشان خواهیم داد. در این حالت ثبات های قطعه بر طبق پارامترهای برنامه تنظیم شده و ثبات های BX و CX حاوی اندازه برنامه برحسب بایت می باشند.

اگر میدان نام ثبات انتخابی در فرمان R ذکر شود، DEBUG محتوای آن ثبات را نمایش داده و فرصت خواهید داشت مقدار آن را عوض کنید. این در مثال ۳ - الف دیده می شود.

### مثال ۳ - الف: استفاده از ثبات R در نمایش/اصلاح ثبات

(الف) اصلاح محتوای یک ثبات

```
-R CX
CX 0000
:FFFF
-R CX
CX FFFF
:
```

(ب) افزودن مقادیر در چپ همراه با صفر

```
-R AX
AX 0000
:1
-R AX
AX 0001
:21
-R AX
AX 0021
:321
-R AX
AX 0321
:4321
-R AX
AX 4321
:54321
^ Error
```

(پ) ورود داده در بایت بالاتر

```
-R DH
BR Error
-R DX
DX 0000
:4C00
```

بخش (الف) از مثال ۳ - الف نشان می دهد که فرمان R با نام ثبات CX آمده است. سپس DEBUG محتوای ثبات CX را نشان می دهد که 0000 است، پس از آن یک ":" آمده است. در اینجا یک مقدار جدید تایپ شده و DEBUG در انتظار فرمان دیگری با نشانه "-" است. فرمان بعدی تأیید می کند که طبق درخواست CX عوض شده است. این بار در ":" یک بازگشت نورد وارد می شود بطوری که مقدار CX تغییر نمی یابد.

بخش (ب) از مثال ۳ - الف نشان می دهد که اگر کمتر از چهار رقم وارد شود، DEBUG مقادیر را همراه با صفر در چپ وارد خواهد کرد. بخش (پ) نشان می دهد که شما نمی توانید بایت های بالا و پائین را جداگانه با فرمان R دستیابی کنید. اگر هر مقداری به جز 0 تا F را (مانند "2F0G") تایپ کنید، DEBUG یک پیام خطا نمایش خواهد داد و مقدار ثبات بدون تغییر باقی می ماند.

## بخش ۳ - الف : کد کردن و اجرای برنامه‌ها در DEBUG

در چند عنوان بعدی چگونگی ورود دستورالعمل‌ها به زبان اسمبلی را نشان می‌دهیم. هدف از این بخش آشنایی خواننده با بکارگیری DEBUG است و نه توضیح دستورالعمل‌های زبان اسمبلی در مثال‌ها.

### فرمان اسمبل کردن A

فرمان اسمبل کردن برای ورود دستورالعمل‌های اسمبلی به حافظه بکار می‌رود.

A <آدرس شروع>

آدرس شروع ممکن است بصورت یک عدد تفاوت مکان داده شود، که در این حالت فرض بر این است که تفاوت مکان از آدرس شروع در قطعه کد تعریف شده باشد، یا ثبات قطعه هم در آن ذکر گردد. به بیان دیگر، "A100" و "ACS:100" یک نتیجه را فراهم می‌سازند. وقتی این فرمان در نشانه "-" وارد شد، DEBUG شروع به ورود دستورالعمل‌های زبان اسمبلی می‌کند. پس از تایپ یک دستورالعمل که به دنبال آن <return> می‌آید، DEBUG منتظر دستور بعدی خواهد شد. این روند همچنان ادامه می‌یابد. این امر در بخش (الف) از مثال ۴ - الف نشان داده شده است.

قبل از تایپ فرمان‌های مثال ۴ - الف، بدانید که یک تفاوت مهم بین برنامه‌نویسی DEBUG و برنامه‌نویسی زبان اسمبلی این است که DEBUG تمام اعداد را در مبنای شانزده فرض می‌کند، در صورتی که اغلب اسمبلرها فرض می‌کنند که اعداد در مبنای دهدهی می‌باشند، مگر اینکه به دنبال اعداد "H" ذکر شود. بنابراین، مثال‌های زبان اسمبلی در این بخش، پس از اعداد دارای "H" نیستند در حالی که به هنگام استفاده از اسمبلر باید آن را دارا باشند. مثلاً شما ممکن است یک دستورالعمل مانند "MOV AL,3FH" وارد کنید. در یک برنامه به زبان اسمبلی که برای MASM می‌نویسید، این دستور به صورت "MOV AL 3FH" تایپ می‌شود.

مثال ۴ - الف : فرمان‌های اسمبل کردن، عکس اسمبل و اجرا

(الف) فرمان اسمبل کردن

```
-A 100
103D:0100 MOV AX,1
103D:0103 MOV BX,2
103D:0106 MOV CX,3
103D:0109 ADD AX,BX
103D:010B ADD AX,CX
103D:010D INT 3
103D:010E
```

(ب) فرمان عکس اسمبل

```
-U 100 10D
103D:0100 B80100 MOV AX,0001
103D:0103 B80200 MOV BX,0002
103D:0106 B90300 MOV CX,0003
103D:0109 01D8 ADD AX,BX
103D:010B 01C8 ADD AX,CX
103D:010D CC INT 3
```

(پ) فرمان اجرا

```
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0100 NV UP DI PL NZ NA PO NC
103D:0100 B80100 MOV AX,0001
-G
AX=0006 BX=0002 CX=0003 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=010D NV UP DI PL NZ NA PE NC
103D:010D CC INT 3
```

همزمان با ورود دستورات، DEBUG هم آنها را به کد ماشین تبدیل می‌کند. اگر شما دستور را به صورت غلط وارد کنید بطوری که DEBUG نتواند آن را اسمبل کند، پیام خطایی اعلام و سعی مجددی اعلام خواهد شد. مجدداً به خاطر داشته باشید که قطعه کد شما ممکن است به هنگام اجرای مثال ۴-الف دارای شماره متفاوتی باشد. توجه نمایید که هر زمان DEBUG آماده دریافت دستور بعدی شود، تفاوت مکان به آدرس مکان بعدی اصلاح می‌شود. مثلاً، پس از تایپ اولین دستورالعمل در تفاوت مکان 0100، DEBUG آن را به زبان ماشین تبدیل و سپس آنها را در بایت 0100 تا 0102 ذخیره می‌نماید، و کاربر را برای دستور بعدی به تفاوت مکان 0103 راهنمایی می‌کند. توجه: در آدرسی پائین‌تر از 100 چیزی وارد نکنید. اولین 100H (256) بایت برای DOS ذخیره شده و نباید بوسیله برنامه‌های شما مورد استفاده قرارگیرد. به این دلیل مثال‌های این کتاب "A100" را برای شروع اسمبل کردن بکار می‌برند.

### فرمان عکس اسمبل، U: تگاهی بر کد ماشین

فرمان عکس اسمبل کد ماشین را همراه با معادل زبان اسمبلی آنها نشان می‌دهد. فرمان در هر یک از دو قالب زیر داده می‌شود.

<آدرس پایان> <آدرس شروع> U  
 <تعداد بایت L> <آدرس شروع> U

در حالی که دستورات اسمبل، دستورالعمل‌های زبان اسمبلی را از صفحه کلید گرفته و آنها را به کد ماشین تبدیل می‌کند تا در حافظه ذخیره شود، دستور عکس اسمبل عمل عکس را انجام می‌دهد. عکس اسمبل کد ماشین ذخیره شده در حافظه را دریافت و آنها را به صورت دستورالعمل‌های زبان اسمبلی برای نمایش روی مانیتور درمی‌آورد. به بخش (ب) از مثال ۴-الف توجه کنید. فرمان عکس اسمبل برای معکوس کردن کدی که در بخش (الف) با دستور اسمبل وارد شد بکار می‌رود. توجه کنید که هم کد ماشین و هم دستورالعمل‌های اسمبلی نمایش داده می‌شوند. دستور می‌تواند با آدرس شروع و پایان طبق مثال ۴-الف وارد شود، یعنی "U 100 10D" و یا می‌تواند با آدرس شروع و تعداد بایت‌ها به مبنای شانزده وارد گردد. همان فرمان در قالب دوم چنین است "U 100 LD" که به DEBUG می‌گوید تا عکس اسمبل را از CS:100 به تعداد D بایت شروع کند. اگر فرمان U بدون آدرس وارد شود، یعنی <بازگشت> U، آنگاه DEBUG، 32 بایت را از CS:IP نمایش می‌دهد. ورود متوالی فرمان "<بازگشت> U" موجب می‌شود تا DEBUG هر بار 32 بایت را نمایش دهد. این راهی ساده برای مشاهده یک برنامه بزرگ است.

### فرمان اجرا، G

فرمان اجرا به DEBUG دستور می‌دهد تا دستورالعمل‌های بین دو آدرس را اجرا کند. قالب آن به این ترتیب است:

<آدرس (های) پایان> <آدرس شروع> = G

اگر آدرسی داده نشود، DEBUG شروع به اجرای دستورات CS:IP نموده و تا رسیدن به نقطه توقف به اجرا ادامه می‌دهد. این عمل در بخش (پ) از مثال ۴-الف انجام شده است.

قبل از اجرای دستورالعمل‌ها، فرمان R برای تست مقادیر ثبات‌ها بکار رفته است. چون CS:IP به اولین دستور اشاره می‌کند فرمان G وارد شده است، که موجب اجرای دستورات تا "INT3" می‌گردد. در این هنگام ضمن رسیدن به نقطه توقف، DEBUG محتویات ثبات‌ها را نمایش داده و سپس نشانه "-" را به شما باز می‌گرداند. تا 10 آدرس توقف می‌توان وارد کرد. DEBUG اجرا را در اولین نقطه توقف متوقف می‌نماید. این برای برنامه‌هایی که دارای چندین مسیر هستند مفید است.



مثال ۵ - الف : حالت های مختلف دستور GO (اجرا)

ابتدا برنامه اسمبل می شود :

```
-A 100
103D:0100 MOV AX,1
103D:0103 MOV BX,2
103D:0106 MOV CX,3
103D:0109 ADD AX,BX
103D:010B ADD AX,CX
103D:010D INT 3
103D:010E
```

(الف) فرمان اجرا به شکل "G"

```
-G
AX=0006 BX=0002 CX=0003 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=010D NV UP DI PL NZ NA PE NC
103D:010D CC INT 3
```

(ب) فرمان اجرا به شکل "G= آدرس شروع"

```
-G=100
AX=0006 BX=0002 CX=0003 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=010D NV UP DI PL NZ NA PE NC
103D:010D CC INT 3
```

(پ) فرمان اجرا بصورت "آدرس پایان آدرس شروع = G"

```
-G=100 109
AX=0001 BX=0002 CX=0003 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0109 NV UP DI PL NZ NA PE NC
103D:0109 01D8 ADD AX,BX
```

(ت) فرمان اجرا به شکل "آدرس G"

```
-R IP
IP 0109
0100
```

-G 109

```
AX=0001 BX=0002 CX=0003 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0109 NV UP DI PL NZ NA PE NC
103D:0109 01D8 ADD AX,BX
```

در این لحظه خط سوم تخلیه اهمیت خاصی دارد. هدف از این خط نشان دادن مکان، کد ماشین و کد اسمبلی دستور بعدی برای اجرا است. در مثال ۵ - الف به آخرین خط در تخلیه ثبات پس از فرمان G نگاه کنید. به سمت چپ خط سوم، که مقدار CS:IP است نگاه کنید. این مقادیر با آنچه در خطوط قبل آمده یکی هستند. پس از CS:IP کد ماشین و پس از آن دستورالعمل زبان اسمبلی قرار دارد.

بخش (الف) از مثال ۵ - الف مثل بخش (پ) از مثال ۴ - الف است. فرمان اجرا از CS:IP شروع شده و دستورالعمل ها را تا "INT3" اجرا می کند. بخش (ب) آدرس شروع را بدون پایان بدست می دهد، بنابراین DEBUG دستورات را از تفاوت مکان 100 تا "INT3" اجرا می کند. این بصورت "G=CS:100" نیز قابل تایپ است. بخش (پ) هر دو آدرس شروع و پایان را بدست می دهد. از نتایج در ثبات ها می توان دید که از 100 تا 109 اجرا شده است. بخش (ت) فقط آدرس پایان را می دهد. وقتی آدرس شروع جداگانه داده نشود، DEBUG مقدار داخل IP را بکار می برد. قبل از فرمان اجرا از وجود آن اطمینان حاصل کنید.

### فرمان ردیابی T: یک ابزار قوی عیب یابی

فرمان ردیابی اجازه می دهد تا اجرای برنامه شما، به میزان هر بار یک یا چند دستور ردیابی شود تا تأثیر برنامه ها روی ثبات ها و یا واردها تحقیق گردد.

<تعداد دستورالعمل ها > <آدرس شروع > = T

این دستور به برنامه می‌گوید تا اجرای دستورات را از آدرس شروع آغاز نماید. با این وجود DEBUG تعدادی از دستورات را که در میدان دوم معین شده اجرا می‌نماید. اگر میدان دوم داده نشود، مقدار پیش فرض 1 است. اگر آدرس شروعی مشخص نشود، فرمان ردیابی مثل فرمان اجرا عمل می‌کند، یعنی از CS:IP شروع خواهد کرد. اختلاف بین این فرمان و فرمان اجرا این است که ردیابی محتوای ثبات‌ها را پس از هر دستور نمایش می‌دهد در صورتی که فرمان اجرا آن‌ها را تا پایان برنامه نشان نمی‌دهد. اختلاف دیگر این است که آخرین میدان دستور اجرا آدرس توقف است، در صورتی که میدان آخر فرمان ردیابی تعداد دستورات مورد اجرا می‌باشد.

مثال ۶ - الف ردیابی دستورات وارده در بخش (الف) از مثال ۴ - الف را نشان می‌دهد. توجه کنید که چگونه IP پس از هر دستور برای اشاره به دستور بعدی به هنگام شده است. سومین خط نمایش ثبات دستوراتی که بوسیله IP اشاره شد، یعنی دستوری که بعد باید اجرا شود را نشان می‌دهد. ردیابی در یک برنامه اجازه می‌دهد تا آنچه را که بوسیله یک دستور از برنامه اتفاق می‌افتد مشاهده کنید. به مقدار AX پس از هر دستور در مثال ۶ - الف توجه نمایید.

#### مثال ۶ - الف : فرمان ردیابی

```
-T=1005
AX=0001 BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0103 NV UP DI PL NZ NA PO NC
103D:0103 BB0200 MOV BX,0002

AX=0001 BX=0002 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0106 NV UP DI PL NZ NA PO NC
103D:0106 B90200 MOV CX,0003

AX=0001 BX=0002 CX=0003 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0109 NV UP DI PL NZ NA PO NC
103D:0109 01D8 ADD AX,BX

AX=0003 BX=0002 CX=0003 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=010B NV UP DI PL NZ NA PE NC
103D:010B 01C8 ADD AX,CX

AX=0006 BX=0002 CX=0003 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=010D NV UP DI PL NZ NA PE NC
103D:010D CC INT 3
```

ردیابی مشابهی با مثال ۶-الف با فرمان "T5" با فرض IP=0100 امکان‌پذیر است. دستگاه‌های مختلف دستور ردیابی متفاوتی دارند. "T" بدون میدان‌های شروع یا شمارش تنها یک دستور را از CS:IP اجرا می‌کند. اگر میدان اول داده نشود، CS:IP فرض خواهد شد. اگر دومین میدان داده نشود 1 فرض می‌شود.

اگر تعداد زیادی دستورالعمل را ردیابی کنید، ممکن است سریع‌تر از خواندن ثبات، حرکت طوماری به سمت بالا داشته باشید. <ctrl-num lock> برای توقف موقت حرکت طوماری می‌تواند بکار رود. برای تکرار حرکت طوماری، کلیدی را فشار دهید. این عمل نه تنها برای دستور ردیابی کار می‌کند، بلکه با هر دستوری که اطلاعات را روی صفحه نمایش نشان دهد نیز کار خواهد کرد.

مثال ۸ - الف برخی از خطاهای برنامه‌نویسی در انتقال داده به ثبات‌ها را نشان می‌دهد. فرمان اسمبل DEBUG این نوع خطا را وقتی یک دستورالعمل وارد شود گرفته و سعی در اسمبل کردن آن خواهد کرد. در دستور "MOV DS,1200" خطا این است که داده بلائصل (فوری) نمی‌تواند به ثبات قطعه برود. از دیگر خطاهای دستور فوق این است که اندازه اولین و دومین عملوند همخوانی ندارند.

مثال ۷ - الف : انتقال داده به ثبات‌های 8 و 16 بیت

```
A>DEBUG
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
US=103D ES=103D SS=103D CS=103D IP=0100 NV UP DI PL NZ NA PO NC
103D:0100 B664 MOV DH,64
-A 100

103D:0100 MOV AL,3F
103D:0102 MOV BH,04
103D:0104 MOV CX,FFFF
103D:0107 MOV CL,BH
103D:0109 MOV CX,1
103D:010C INT 3
103D:010D
-T=100.5
AX=003F BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0102 NV UP DI PL NZ NA PO NC
103D:0102 B704 MOV BH,04

AX=003F BX=0400 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0104 NV UP DI PL NZ NA PO NC
103D:0104 B9FFFF MOV CX,FFFF

AX=003F BX=0400 CX=FFFF DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0107 NV UP DI PL NZ NA PO NC
103D:0107 88F9 MOV CL,BH

AX=003F BX=0400 CX=FF04 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0109 NV UP DI PL NZ NA PO NC
103D:0109 B90100 MOV CX,0001

AX=003F BX=0400 CX=0001 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=010C NV UP DI PL NZ NA PO NC
103D:010C CC INT 3
.
```

مثال ۸ - الف : خطاهای مرسوم در استفاده از ثبات

```
A>DEBUG
-A 100
103D:0100 MOV AL,FF3
 ^ Error
103D:0100 MOV AX,12345
 ^ Error
103D:0100 MOV DS,1200
 ^ Error
103D:0100 MOV SI,DH
 ^ Error
103D:0100 MOV AX,BH
 ^ Error
103D:0100 MOV AL,BX
 ^ Error
103D:0100
-Q
A>
```

```

A>DEBUG
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=1132 ES=1132 SS=1132 CS=1132 IP=0100 NV UP DI PL NZ NA PO NC
1132:0100 BED548 MOV SI,48D5
-A 100
1132:0100 MOV AL,57
1132:0102 MOV DH,86
1132:0104 MOV DL,72
1132:0106 MOV CX,DX
1132:0108 MOV BH,AL
1132:010A MOV BL,9F
1132:010C MOV AH,20
1132:010E ADD AX,DX
1132:0110 ADD CX,BX
1132:0112 ADD AX,1F35
1132:0115
-U 100 112
1132:0100 B057 MOV AL,57
1132:0102 B686 MOV DH,86
1132:0104 B272 MOV DL,72
1132:0106 89D1 MOV CX,DX
1132:0108 88C7 MOV BH,AL
1132:010A B39F MOV BL,9F
1132:010C B420 MOV AH,20
1132:010E 01D0 ADD AX,DX
1132:0110 01D9 ADD CX,BX
1132:0112 05351F ADD AX,1F35

```

برنامه فوق از مکان CS:IP با 1132:0100 ذخیره می شود. آدرس منطقی مربوط به آدرس فیزیکی 11420(11320+0100) است.

## بخش ۴ - الف : دستکاری داده در DEBUG

در زیر سه فرمان دیگر DEBUG که برای بررسی یا تغییر محتوای حافظه بکار می رود توصیف شده است.

- F فرمان پرکردن که یک بلوک حافظه را با داده پر می کند
- D فرمان تخلیه که محتوای حافظه را روی صفحه نمایش نشان می دهد
- E فرمان ورود که محتوای حافظه را تست/تغییر می دهد

### فرمان پرکردن ، F : پرکردن حافظه با داده

فرمان پرکردن برای پرکردن ناحیه ای از حافظه با ارقام داده است. ترکیب فرمان F به قرار زیر است

- <داده> <آدرس پایان> <آدرس شروع> F
- <داده> <تعداد بایت های L> <آدرس شروع> F

این فرمان برای پرکردن یک بلوک از حافظه مثل صفر کردن ناحیه ای از حافظه، مفید است. معمولاً از این فرمان برای پرکردن ناحیه قطعه داده استفاده خواهید کرد که آدرس های شروع و پایان، آدرس های تفاوت مکان در قطعه داده است. برای پرکردن قطعه دیگر، ثبات باید قبل از تفاوت مکان ذکر شود. مثلاً در زیر اولین فرمان 16 بایت از DS:100 تا DS:10F را با FF پر می کند. دومین فرمان یک بلوک 256 بایتی قطعه کد از CS:100 تا CS:1FF با اسکی 20 (فاصله) پر خواهد شد.

F 100 10F FF

F CS:100 1FF 20



مثال ۱۰ - الف کاربرد فرمان F را نشان می‌دهد. داده می‌تواند رشته‌ای از ارقام باشد که در این صورت DEBUG ناحیه‌ای از حافظه را با آن پرمی‌کند و سپس عمل را مرتباً تکرار خواهد کرد. مثلاً

F 100 L20 00 FF

فرمان فوق موجب می‌شود تا 20H بایت (32 دهدهی) با شروع از DS:100 یک در میان با 00 و FF پر شود.

### فرمان تخلیه، D: بررسی محتویات حافظه

فرمان تخلیه برای بررسی محتویات حافظه بکار می‌رود، قالب دستور به شکل زیر است

D <دستور پایان> <آدرس شروع>

D <تعداد بایت‌های L> <آدرس شروع>

فرمان D می‌تواند با آدرس شروع و پایان وارد شود، و در این صورت تمام بایت‌های بین آن مکان‌ها نشان داده خواهد شد. همچنین می‌تواند با آدرس شروع و تعداد بایت‌ها (به مبنای شانزده) وارد شود که در این حالت از آدرس شروع به همان تعداد نمایش داده می‌شود. اگر آدرس یک تفاوت مکان باشد، DS فرض خواهد شد. فرمان D می‌تواند به تنهایی هم وارد شود، که در اینصورت DEBUG، 128 بایت متوالی را از DS:100 نشان خواهد داد. در نوبت بعد که "D" به تنهایی وارد شود، DEBUG، 128 بایت بعدی را از آخرین محل نمایش، نشان خواهد داد. بدین ترتیب می‌توان در ناحیه وسیعی از حافظه هر بار 128 بایت را ملاحظه کرد.

مثال ۱۰ - الف: پر و تخلیه کردن یک بلوک از حافظه

(الف) فرمان پر و تخلیه

```
A>DEBUG
-F 100 14F 20
-F 150 19F 00
-D 100 19F
103D:0100 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0110 20 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0120 20 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0130 20 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0140 20 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 ..
103D:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 ..
103D:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 ..
103D:0180 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 ..
103D:0190 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 ..
```

(ب) پر و تخلیه کردن مکان‌های منتخبی از حافظه

```
-F 104 10A FF
-D 104 10A
103D:0104 FF FF FF FF-FF FF FF
-D 100 10F
103D:0100 20 20 20 20 FF FF FF FF-FF FF FF 20 20 20 20 20
```

(c) Filling and dumping code segment memory

```
-F CS:100 12F 20
-D CS:100 12F
103D:0100 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0110 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0120 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
```

(پ) پر و تخلیه حافظه قطعه کد

```
-F CS:100 12F 20
-D CS:100 12F
103D:0100 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0110 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
103D:0120 20 20 20 20 20 20 20 20-20 20 20 20 20 20 20
```

مثال ۱۰ - الف فوق، استفاده از فرمان‌های پر و تخلیه کردن را نشان می‌دهد. بخش (الف) دو فرمان پرکردن را برای پرکردن نواحی قطعه داده نشان می‌دهد، که پس از آن تخلیه شده‌اند. بخش (ب) برای نمایش پر و تخلیه ناحیه کوچکی از حافظه لحاظ شده است. در بخش (پ) می‌بینید که چگونه می‌توان حافظه‌ای را در دیگر قطعات پر و خالی کرد. به یاد داشته باشید که مقادیر DS و CS در کامپیوتر شما ممکن است متفاوت باشد. آشنایی کامل با قالب تخلیه DEBUG اهمیت دارد. مثال ۱۱ - الف تمرینات بیشتری را در تخلیه حافظه فراهم می‌کند.

مثال ۱۱ - الف : استفاده از فرمان تخلیه در تست کد ماشین

```
-U 100 112
1132:0100 B057 MOV AL,57
1132:0102 B686 MOV DH,86
1132:0104 B272 MOV DL,72
1132:0106 89D1 MOV BX,DX
1132:0108 88C7 MOV BH,AL
1132:010A B39F MOV BL,9F
1132:010C B420 MOV AH,20
1132:010E 01D0 ADD AX,DX
1132:0110 01D9 ADD CX,BX
1132:0112 05351F ADD AX,1F35
-D CS:100 11F
1132:0100 B0 57 B6 86 B2 72 89 D1-88 C7 B3 9F B4 20 01 D0 0W6.2r.Q.G3.4.P
1132:0110 01 D9 05 35 1F 19 83 3E-E3 45 00 74 12 53 56 BB .Y5L.cE.tSV:
```

مثال ۱۱ - الف برنامه‌ای را نشان می‌دهد که عکس اسمبل شده و قبلاً در DEBUG بار شده است. توجه کنید که کدهای ماشین متناوباً یکی پس از دیگری ذخیره شده‌اند. آشنایی کامل با تخلیه حافظه در DEBUG اهمیت دارد. در زیر یک خط از مثال ۱۱ - الف آمده است.

1132:0100 B0 57 B6 86 B2 72 89 D1-88 C7 B3 9F B4 20 01 D0 0W6.2r.Q.G3.4.P

خط با آدرس اولین بایت نشان داده شده در آن خط آغاز می‌شود، که در این حالت 1132:0100 می‌باشد و در آن 1132 محتوای CS و 0100 متعلق به تفاوت مکان در کد قطعه است. پس از آن، 16 بایت داده نشان داده شده و به دنبال آن اقلام نشان داده شده‌اند. بایت‌های حاوی کاراکترهای اسکی، کاراکترها را نشان خواهند داد. اگر محتوای یک بایت یک کد اسکی نباشد، قابل نمایش نبوده و با "." نمایش داده می‌شود. اولین بایت تفاوت مکان 0100 است، دومین بایت تفاوت مکان 0101، سومین بایت تفاوت مکان 0102 و به همین ترتیب تا آخرین بایت که 010F است.

تفاوت مکان : 100 101 102 103 104 105 106 107-108 109 10A 10B 10C 10D 10E 10F

1132:0100 B0 57 B6 86 B2 72 89 D1-88 C7 B3 9F B4 20 01 D0

آدرس‌های فوق، آدرس منطقی‌اند. آدرس منطقی 1132:0100 فوق مربوط به آدرس فیزیکی 11420 (11320+0100=114200) است.

فرمان ورود E: ورود داده به حافظه

فرمان پرکردن برای پرکردن یک بلوک با یک داده بکار رفت. فرمان ورود می‌تواند لیستی از داده را در بخش معینی از حافظه وارد کند. قالب فرمان E به شکل زیر است:

```
<لیست داده> <آدرس> E
<آدرس> E
```

مثال ۱۲ - الف: استفاده از فرمان E برای ورود داده به حافظه

-E 100 'John Smith'

-D 100 10F

103D:0100 4A 6F 68 6E 20 53 6E 69-74 68 20 20 20 20 20 20 John Smith

(الف) ورود داده با فرمان E

-E 106

103D:0106 6E.6D

-D 100 10F

103D:0100 4A 6F 68 6E 20 53 6D 69-74 68 20 20 20 20 20 20 John Smith

(ب) تغییر داده با فرمان E

(پ) راه دیگری برای تغییر داده با فرمان E بوسیله فشردن کلید فاصله یک بایت هر بار

-E 100

103D:0100 4A. 6F. 68. 6E. 20. 53. 6E.6D

-D 100 10F

103D:0100 4A 6F 68 6E 20 53 6D 69-74 68 20 20 20 20 20 20 John Smith

-E 107

103D:0107 69.-

103D:0106 6E.6D

(ت) راه دیگری برای تغییر داده با فرمان E

بخش (الف) از مثال ۱۲ - الف، ساده‌ترین کاربرد E را نشان داد، که طی آن با ورود آدرس داده به دنبال آن آمده است. این مثال نشان می‌دهد که چگونه داده اسکی می‌تواند در یک یا دو علامت نقل قول باشد. فرمان E یک ویژگی قوی دیگر دارد، و آن توانایی تست و تغییر بایت به بایت حافظه است. اگر فرمان E با یک آدرس خاص و بدون لیست داده وارد شود، DEBUG فرض خواهد کرد که شما مایل به بررسی آن بایت و احتمالاً تغییر آن هستید. پس از آن بایت نمایش داده می‌شود و شما چهار انتخاب دارید:

۱ - می‌توانید یک قلم داده جدید برای آن بایت وارد کنید. DEBUG محتوای قدیمی را با مقدار جدید تایپ شده جایگزین خواهد کرد.

۲ - می‌توانید <برگشت> بزنید، و به این معنی است که مایل به تغییر مقدار نیستید.

۳ - می‌توانید کلید فاصله را بزنید که بدین ترتیب بایت در حال نمایش را بدون تغییر گذاشته و بایت بعدی نشان داده خواهد شد و به شما اجازه تغییر آن را خواهد داد.

۴ - می‌توانید علامت منفی "-" را وارد کنید که بایت در حال نمایش را رها کرده ولی بایت قبلی را نشان خواهد داد و اجازه تغییر را به شما می‌دهد.

به بخش (ب) در مثال ۱۲ - الف توجه نمایید. کاربر می‌خواهد کلمه "Smith" را به "Snith" تغییر دهد. پس از تایپ "E 106" توسط کاربر، DEBUG با محتوای آن بایت، یعنی 6E که اسکی n است و با "." دنبال شده پاسخ خواهد داد. سپس کاربر کد اسکی برای "m" یعنی 6D را تایپ و کلید بازگشت را زده و خواهد دید که آیا داده تخلیه شده اصلاح شده است یا نه. بخش (پ) از مثال ۱۲ - الف راه دیگری برای همان اصلاح را نشان می‌دهد. کاربر از تفاوت مکان حافظه 100 شروع کرده و با ضربه‌های متوالی به خط فاصله تا رسیدن به مکان مورد نظر ادامه می‌دهد. سپس اصلاح را انجام و کلید بازگشت را می‌فشارد.

نهایتاً بخش (ت) راه سوم همان اصلاح را نشان می‌دهد. در این مثال، کاربر بطور اتفاقی آدرس غلطی را وارد کرده است. آدرس یک مکان از محل مورد نیاز برای اصلاح رد شده است. کاربر علامت منفی را فشرده است و بنابراین DEBUG بایت قبلی را در خط بعدی، نمایش خواهد داد. اصلاح برای آن بایت صورت گرفته است. خودتان این مثال را تکرار کنید زیرا فرمان E در اصلاح برنامه‌های آینده شمار بسیار مفید است. از فرمان E برای ورود داده‌های عددی هم استفاده می‌شود:

E 100 23 B4 02 4F

مثال ۱۳ - الف مثالی از ورود کد با فرمان اسمبل، ورود داده با فرمان ورود و اجرای برنامه می‌باشد. این

کاربرد فرمان E در تست و عیب‌یابی برنامه خیلی مفید است. مثال ۱۴ - الف قرارداد منتهی به کوچکتر را در ریزپردازنده‌های 80x86 نشان می‌دهد.

A>DEBUG

مثال ۱۳ - الف : ورود داده و کد و اجرای یک برنامه

```
-A 100
103D:0100 MOV AL,00
103D:0102 ADD AL,[0200]
103D:0106 ADD AL,[0201]
103D:010A ADD AL,[0202]
103D:010E ADD AL,[0203]
103D:0112 ADD AL,[0204]
103D:0116 INT 3
103D:0117
-E DS:0200 25 12 15 1F 2B
-D DS:0200 020F
103D:0200 25 12 15 1F 2B 02 00 E8 51 FF C3 E8 1E F6 74 03 %...+.hQ.Ch.vt.
-G=100 116
AX=0096 BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
CS=103D ES=103D SS=103D CS=103D IP=0116 OV UP DI NG NZ AC PE NC
103D:0116 CC INT 3
```

مثال ۱۴ - الف : چگونگی ذخیره کلمات بوسیله 80x86: منتهی به کوچکتر

(الف) انتقال یک کلمه از حافظه به ثبات

A>DEBUG

```
-D 6820 LF
103D:6820 26 00 EA 27 CF 5B 48 22-0D 00 B8 15 45 00 EA 20 &.'O[H"..8.E.]
-A 100
103D:0100 MOV BX,[6826]
103D:0104 INT 3
103D:0105
-T
AX=0000 BX=2248 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0104 NV UP DI PL NZ NA PO NC
103D:0104 CC INT 3
-D 200 20F
103D:0200 F2 FF 0E 0B 37 B8 FF FF-50 E8 B3 08 83 C4 02 20 R...78..Ph3..D.
-A 100
103D:0100 MOV BX,1234
103D:0103 MOV [200],BX
103D:0107 INT 3
103D:0108
-G=100
AX=0000 BX=1234 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0107 NV UP DI PL NZ NA PO NC
103D:0107 CC INT 3
-D 200 LF
103D:0200 34 12 0E 0B 37 B8 FF FF-50 E8 B3 08 83 C4 02 20 4...78..Ph3..D.
```

(ب) انتقال یک کلمه از ثبات به حافظه

در مثال ۱۴ - الف، بخش (الف)، روش آدرس‌دهی مستقیم برای انتقال دو بایت از تفاوت مکان 6826 به ثبات BX بکار رفت. نگاه به تخلیه نشان می‌دهد که مکان 6826 دارای مقدار 48 بوده و بایت بعدی یعنی 6827 حاوی 22 است. این بایت‌ها به ترتیب بایت پائین به پائین و بایت بالا به بالا به ثبات BX انتقال می‌یابند. محتوای مکان پائین‌تر حافظه 6826، که 48 بود به بایت پائین، یعنی BL منتقل شده است. محتوای مکان بالاتر 6827 که 22 بود نیز به بایت بالاتر BH انتقال یافته است. در بخش (ب)، مقدار 1234H به ثبات BX و سپس در تفاوت مکان 200 ذخیره شده است. توجه کنید که آدرس تفاوت مکان 200 حاوی بایت 34 و آدرس تفاوت مکان بالاتر 201، حاوی بایت بالاتر 12 است.



مثال ۱۷ - الف، نشان داد که چگونه ثبات پرچم قابل بررسی و یا بررسی و سپس تغییر است. هنگامی که فرمان با "F" دنبال شود، به DEBUG فرمان نمایش محتوای ثبات پرچم داده می‌شود. پس از آنکه DEBUG کدهای ثبات پرچم را نشان داد، با یک "-" دیگر در پایان خط کدهای ثبات متوقف می‌گردد. در این نقطه در صورت تمایل به تغییر ثبات پرچم کد آن را می‌توان تایپ کرد و یا اگر مایل به تغییر نبودیم فقط کلید بازگشت را فشار می‌دهیم. کدهای ثبات را می‌توان به هر ترتیبی تایپ نمود.

### تأثیر دستورالعمل‌ها بر بیت‌های پرچم

مثال ۱۸ - الف اثر دستورالعمل ADD را روی ثبات پرچم نشان می‌دهد. ADD در بخش (الف) بصورت بایستی انجام می‌شود. جمع 9C و 64 نتیجه 00 را همراه با رقم نقلی تولید می‌کند. بیت‌های پرچم تأیید کننده چنین جوابی است. توجه کنید که پرچم صفر اکنون ZR است، که به معنی صفر بودن نتیجه می‌باشد. بعلاوه رقم نقلی برقرار شده است، که به معنی وجود رقم نقلی است. ADD در بخش (ب) جمع کلمه می‌باشد. توجه کنید که بیت علامت پس از اجرای دستور ADD در NG برقرار شده است. علت این است که نتیجه، یعنی CAE0 در فرم دودویی‌اش دارای یک 1 در بیت 15 است که همان بیت علامت می‌باشد. چون در اینجا با اعداد بدون علامت سروکار داریم، این عدد را CAE0H مثبت تفسیر می‌کنیم و نه یک عدد منفی. این تفسیر این حقیقت را آشکار می‌سازد که ریزپردازنده با همه داده‌ها به یک شکل برخورد می‌کند. تفسیر داده به برنامه‌نویس مرتبط می‌شود. نهایتاً به جمع بخش (پ) توجه کنید. جمع AAAAH و 5556H نتیجه 10000H را خواهد داد، که BX=0000 را با نقلی خروجی خواهد داد. پرچم صفر بیانگر نتیجه صفر (BX=0000) می‌باشد، در حالی که پرچم نقلی بیانگر وقوع یک رقم نقلی خروجی است.

### فرمان حساب شانزدهی

این فرمان مثل یک ماشین حساب مبنای شانزده متصل است که جمع و تفریق را در مبنای 16 انجام می‌دهد. قالب آن به شکل زیر است:

H <عدد 2> <عدد 1>

وقتی که این فرمان به DEBUG وارد شود، DEBUG جمع آنها را همراه با تفریقشان نمایش خواهد داد. (عدد دوم - عدد اول).

### فرمان رویه

این فرمان ساختاری شبیه به دستور ردیابی دارد:

P <تعداد دستورالعمل‌ها> <آدرس شروع> =

این دستور برای اجرای یک حلقه، فراخوانی، وقفه یا تکرار رشته بصورت یک دستور در عوض چندین دستور بکار می‌رود.

```
A>DEBUG
-A 100
103D:0100 MOV AL,9C
103D:0102 MOV DH,64
103D:0104 ADD AL,DH
103D:0106 INT 3
103D:0107
-T 3
AX=009C BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0102 NV UP DI PL NZ NA PO NC
103D:0102 B664 MOV DH,64
AX=009C BX=0000 CX=0000 DX=6400 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0104 NV UP DI PL NZ NA PO NC
103D:0104 00F0 ADD AL,DH
AX=0000 BX=0000 CX=0000 DX=6400 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0106 NV UP DI PL ZR AC PE CY
103D:0106 CC INT 3
```

مثال ۱۸ - الف: مشاهده تغییرات در ثبات پرچم (الف)

(ب)

```

-A 100
103D:0100 MOV AX,34F5
103D:0103 ADD AX,95EB
103D:0106 INT 3
103D:0107
-T=100 2
AX=34F5 BX=0000 CX=0000 DX=6400 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0103 NV UP DI PL NZ NA PO NC
103D:0103 05EB95 ADD AX,95EB

AX=CAE0 BX=0000 CX=0000 DX=6400 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0106 NV UP DI NG NZ AC PO NC
103D:0106 CC INT 3

```

(پ)

```

-A 100
103D:0100 MOV BX,AAAA
103D:0103 ADD BX,5556
103D:0107 INT 3
103D:0108
-G=100 107
AX=34F5 BX=0000 CX=0000 DX=6400 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0107 NV UP DI PL ZR AC PE CY
103D:0107 CC INT 3

```

در زیر، مثال ۱۹ - الف نشان می‌دهد که چگونه یک برنامه در DEBUG نوشته شده، داده مورد نظر ایجاد و سپس اجرا شده است. این برنامه شامل پرش شرطی است که بر اساس مقدار پرچم صفر پرش می‌کند. همچنین بعضی از تفاوت‌های عمده بین برنامه‌نویسی در DEBUG و برنامه‌نویسی برای یک اسمبلر مانند MASM در این مثال مشخص شده است. ابتدا به دستور JNZ توجه کنید. اگر این برنامه به زبان اسمبلی باشد، دستورالعمل ممکن است به صورت "JNZ LOOP-ADD" نوشته شود که در آن LOOP-ADD به یک خط از برنامه اشاره می‌نماید. در DEBUG فقط JNZ را به آدرس ارجاع می‌دهیم. تفاوت مهم دیگر این است که برنامه زبان اسمبلی دارای قطعات کد و داده جدا از هم است. در مثال ۱۹ - الف، داده مورد آزمایش در تفاوت مکان 0200 آمده بنابراین BX برای تنظیم شده است. زیرا از آن به عنوان اشاره‌گر داده استفاده می‌شود. در برنامه اسمبلی، داده در قطعه داده و دستورالعمل نیز ممکن است بصورت "MOV BX,OFFSET DATA1" باشد که در آن DATA1 پرچسب مربوط به رهنمون داده‌ای است که داده را ذخیره کرده است.

مثال ۱۹ - الف: ردیابی برنامه برای جمع ۵ بایت

```

A>DEBUG
-A 100
103D:0100 MOV CX,05
103D:0103 MOV BX,0200
103D:0106 MOV AL,0
103D:0108 ADD AL,[BX]
103D:010A INC BX
103D:010B DEC CX
103D:010C JNZ 0108
103D:010E MOV [0205],AL
103D:0111 INT 3
103D:0112
-E 0200 25 12 15 1F 2B
-D 0200 020F
103D:0200 25 12 15 1F 2B 9A DE CE-1E F3 20 20 20 20 20 20 %...+.^n.
-G=100 111
AX=0096 BX=0205 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=103D ES=103D SS=103D CS=103D IP=0111 NV UP DI PL ZR NA PE NC
103D:0111 CC INT 3

-D 0200 020F
103D:0200 25 12 15 1F 2B 96 DE CE-1E F3 20 20 20 20 20 20 %...+.^n.

```

```
A>DEBUG
-A 100
103D:0100 MOV SI,0210
103D:0103 MOV DI,0228
103D:0106 MOV CX,6
103D:0109 MOV AL,[SI]
103D:010B MOV [DI],AL
103D:010D INC SI
103D:010E INC DI
103D:010F DEC CX
103D:0110 JNZ 0109
103D:0112 INT 3
103D:0113
-E 0210:25 4F 85 1F 2B C4
-D 0210:022F
103D:0210 25 4F 85 1F 2B C4 43 0C-01 01 01 00 02 FF FF FF %O..+DC.....
103D:0220 FF FF FF FF FF FF FF FF FF-FF FF FF FF 45 0D CA 2A E.J*
-G =100
AX=00C4 BX=0000 CX=0000 DX=0000 SP=CFDE BP=0000 SI=0216 DI=022E
DS=103D ES=103D SS=103D CS=103D IP=0112 NV UP DI PL ZR NA PE NC
103D:0112 CC INT 3
-D 0210:022F
103D:0210 25 4F 85 1F 2B C4 43 0C-01 01 01 00 02 FF FF FF %O..+DC.....
103D:0220 FF FF FF FF FF FF FF FF FF-25 4F 85 1F 2B C4 CA 2A%O..+DJ*
```

## بخش ۷ - الف: فرمان‌های دیگر دستکاری داده

فرمان‌های زیر اغلب برای دستکاری داده در برنامه‌های شما مفیدند.

### فرمان جابجایی، M: کپی داده از یک مکان به مکان دیگر

فرمان جابجایی برای کپی داده از یک مکان به مکان دیگر بکار می‌رود. مکان اصلی بدون تغییر خواهد ماند. قالب دستور به ترتیب زیر است:

M <آدرس مقصد> <آدرس پایان> <آدرس شروع> =  
M <آدرس مقصد> <تعداد بایت‌های L> <آدرس شروع> =

به بیان دیگر، این فرمان یک کپی از داده‌ها را از آدرس شروع تا پایان در آدرس مقصد قرار می‌دهد. بخش (الف) در مثال ۲۱ - الف مثالی از فرمان جابجایی را نشان می‌دهد. این فرمان داده را از مکان 130 تا 13F به مکان 140 کپی می‌کند. نتیجه را با دستور "M 130LF 140" می‌توان بدست آورد.

### فرمان مقایسه، C: تست بلوک‌های داده برای تفاوت‌ها

فرمان مقایسه برای واری کردن دو ناحیه از بایت‌های حافظه و نمایش که حاوی داده‌های متفاوتند، بکار می‌رود. اگر دو بلوک یکسان باشند، DEBUG فقط علامت "-" را نشان خواهد داد. قالب فرمان بدین ترتیب است.

C <آدرس مقایسه> <آدرس پایان> <آدرس شروع> <آدرس مقایسه> <تعداد بایت‌های L> <آدرس شروع> =  
C <آدرس مقایسه> <تعداد بایت‌های L> <آدرس شروع> <آدرس مقایسه> <تعداد بایت‌های L> <آدرس شروع> =

به بیان دیگر، این فرمان داده‌های موجود از آدرس شروع تا پایان را با آدرس مقایسه، مطابقت کرده و هر بایتی که متفاوت باشد را نشان می‌دهد. بخش (ب) در مثال ۲۱ - الف حاوی مثال‌هایی از کاربرد فرمان مقایسه است. اولین دستور از تفاوت مکان 130 تا 134 را با حافظه‌ای که با 140 شروع می‌شود مقایسه می‌نماید. چون تفاوتی پیدا نشده است، DEBUG با علامت "-" پاسخ می‌دهد. فرمان بعدی از تفاوت مکان 130 تا 134 را با آدرس شروع 150 مقایسه کرده و هر پنج مقایسه را چاپ می‌کند چون همه آنها با هم تفاوت دارند.

(الف) فرمان جایجایی

```
-F 130 13F FF
-D 130 15F
103D:0130 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
103D:0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
103D:0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
-M 130 13F 140
-D 130 15F
103D:0130 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
103D:0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
103D:0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(ب) فرمان مقایسه

```
-C 130 134 140
-C 130 134 150
103D:0130 FF 00 103D:0150
103D:0131 FF 00 103D:0151
103D:0132 FF 00 103D:0152
103D:0133 FF 00 103D:0153
103D:0134 FF 00 103D:0154
```

(پ) فرمان جستجو

```
-S 150 15F FF
-S 130 133 FF
103D:0130
103D:0131
103D:0132
103D:0133
```

### فرمان جستجو، S: جستجوی یک بلوک حافظه برای یک قلم داده

فرمان جستجو برای جستجوی داده خاصی از بلوک داده است. اگر آن قلم داده یافت نشود، DEBUG علامت "-" را نشان می‌دهد. در غیر اینصورت همه مکان‌هایی که داده در آن پیدا نشود نمایش داده خواهند شد. قالب دستور چنین است:

```
S <داده> <آدرس پایان> <آدرس شروع> =
S <داده> <تعداد بایت‌های L> <آدرس شروع> =
```

DEBUG برای یافتن داده از آدرس شروع تا پایان را جستجو می‌کند. به بخش (پ) از مثال ۲۱ - الف نگاه کنید. این مثال از آدرس 130 تا 133 را برای یافتن FF و چاپ هر چهار آدرس جستجو می‌کند زیرا هر چهار مکان حاوی FF است. فرمان زیر به نتیجه یکسانی منتهی می‌شود:

"S 130 L4 FF". داده ممکن است لیستی از اقلام داده باشد که در اینصورت DEBUG برای آن الگو جستجو خواهد کرد.

### بخش ۸ - الف: بارکردن و نوشتن برنامه‌ها

فرمان‌های نوشتن و بارکردن زیر برای ذخیره یک برنامه در دیسک و بارکردن برنامه از قبل ذخیره شده از دیسک در DEBUG مورد استفاده قرار می‌گیرند. هر دوی آنها، آشنایی کاملی با مفاهیم پیشرفته DOS را نیاز دارند.

#### فرمان نوشتن، W: ذخیره سازی دستورالعمل‌ها در دیسک

فرمان نوشتن برای ذخیره دستورالعمل‌ها در دیسک بکار می‌رود. قالب آن به شکل زیر است:

```
M <قطاع> <قطعه شروع> <شماره دیسک گردان> <آدرس شروع>
```

نوشتن در قطاع خاصی پیشنهاد نمی‌شود زیرا آشنایی کاملی از روش ذخیره‌سازی در دیسک گردان لازم است. نوشتن در قطاع غلط می‌تواند فهرست دیسک را آسیب رسانده آن را بلامصرف کند. مثال ۲۳-الف چگونگی استفاده از فرمان W را بدون هر پارامتری برای کد ذخیره‌سازی در یک دیسک نشان می‌دهد که در آن فرمان N برای برپایی نام فایل مورد استفاده قرار گرفته است.

### فرمان بارکردن، L: دستورات بارکردن از دیسک

فرمان بارکردن عکس عمل نوشتن است: این فرمان اطلاعات را از آدرس مشخصی در دیسک به حافظه بار می‌کند. قالب آن چنین است

<قطاع> <قطاع شروع> <شماره دیسک گردان> <آدرس شروع> L

پس از بارکردن، ثبات‌های BX و CX اندازه برنامه را برحسب بایت نگه می‌دارند. استفاده از فرمان بارکردن و همه انتخاب‌های آن نیاز به اطلاعات کاملی از دیسک و ذخیره‌سازی دارد و برای دانشجویان مبتدی پیشنهاد می‌شود. با این وجود، فرمان L ممکن است پس از فرمان نام در قالب ساده‌ای که در زیر آمده بکار رود.

### فرمان نام، N: برای بارکردن یک فایل از دیسک

فرمان نام می‌تواند به همراه فرمان بارکردن برای یک برنامه به DEBUG بکار رود.

<نام فایل> -N

-L

فرمان نام فوق نام فایل بارشدنی با فرمان بار را ایجاد می‌کند. راه دیگری در بارکردن یک برنامه در DEBUG هنگام ورود به برنامه DEBUG است:

<نام فایل> -DEBUG

مثال ۲۲ - الف: بارکردن یک برنامه اسمبل شده در DEBUG

(الف) بارکردن با فرمان‌های نام و بارکردن

-N B:\PROGRAMS\PROG1.EXE

-L

(ب) بارکردن با ورود به DEBUG

A>DEBUG B:\PROGRAMS\PROG1.EXE

مثال ۲۳ - الف، ابتدا چگونگی ذخیره کردن کدی که در DEBUG وارد شده را نشان می‌دهد. کد وارده کد مثال ۱۹ - الف است. پس از ورود کد توسط فرمان A، ثبات‌های BX و CX باید برای نگهداری تعداد بایت‌های ذخیره شدنی تنظیم شوند. ثبات CX برای ذخیره ۱۲ بایت با ۱۲ تنظیم می‌شود. در این حالت چون BX بایت بالارته را ذخیره می‌کند باید صفر باشد. فرمان N دیسک گردان و نام فایل بکار رفته را ایجاد می‌کند، آنگاه فرمان W کد را در دیسک می‌نویسد. توجه کنید که توسعه نام فایل باید "com" باشد زیرا فایل‌های "exe" نمی‌توانند به این روش ذخیره شوند. بقیه مثال ۲۳ - الف چگونگی بار شدن کدهای ذخیره شده را در DEBUG نشان می‌دهد. فرمان N فایل‌های مرجوعی را برقرار می‌کند و فرمان بارکردن L کدها را در DEBUG بار می‌کند.



```

C>DEBUG
-A 100
12B0:0100 MOV CX,05
12B0:0103 MOV BX,0200
12B0:0106 MOV AL,0
12B0:0108 ADD AL,[BX]
12B0:010A INC BX
12B0:010B DEC CX
12B0:010C JNZ 0108
12B0:010E MOV [0205],AL
12B0:0111 INT 3
12B0:0112
-R CX
CX 0000
:12
-r
AX=0000 BX=0000 CX=0012 DX=0000 SP=CFDE BP=0000 SI=0000 DI=0000
DS=12B0 ES=12B0 SS=12B0 CS=12B0 IP=0100 NV UP DI PL NZ NA PO NC
12B0:0100 B90500 MOV CX,0005
-NB:EX19.COM
-W
Writing 0012 bytes
-Q

C>DEBUG
-NB:EX19.COM
-L
-R
AX=0000 BX=0000 CX=0012 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=12CC ES=12CC SS=12CC CS=12CC IP=0100 NV UP DI PL NZ NA PO NC
12CC:0100 B90500 MOV CX,0005
-u cs:100 111
12CC:0100 B90500 MOV CX,0005
12CC:0103 B80002 MOV BX,0200
12CC:0106 B000 MOV AL,00
12CC:0108 0207 ADD AL,[BX]
12CC:010A 43 INC BX
12CC:010B 49 DEC CX
12CC:010C 75FA JNZ 0108
12CC:010E A20502 MOV [0205],AL
12CC:0111 CC INT 3

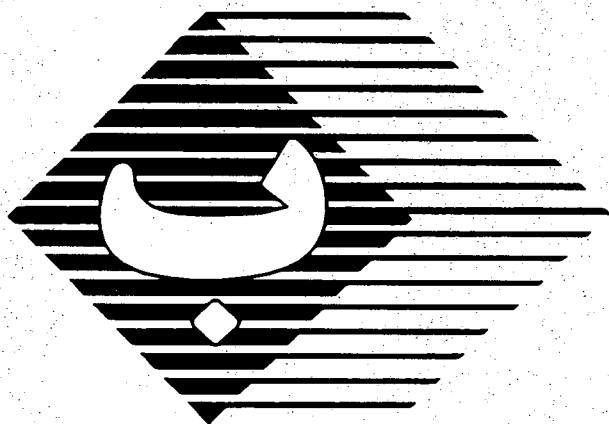
```

جدول الف - ۲ خلاصه فرمان‌های DEBUG

| تابع (Function)        | انتخاب‌های فرمان                                                                           |
|------------------------|--------------------------------------------------------------------------------------------|
| اسمبل (Assemble)       | A <آدرس شروع>                                                                              |
| مقایسه (Compare)       | C <آدرس شروع> <آدرس پایان> <آدرس مقایسه><br>C <آدرس شروع> <تعداد بایت‌های L> <آدرس مقایسه> |
| تخلیه (Dump)           | D <آدرس شروع> <آدرس پایان><br>D <آدرس شروع> <تعداد بایت‌های L>                             |
| ورود (Enter)           | E <آدرس> <لیست داده><br>E <آدرس>                                                           |
| پرکردن (Fill)          | F <آدرس شروع> <آدرس پایان> <داده><br>F <آدرس شروع> <تعداد بایت‌های L> <داده>               |
| اجرا (Go)              | G <آدرس شروع> = <آدرس(های) پایان>                                                          |
| حساب شانزدهی           | H <عدد 1> <عدد 2>                                                                          |
| بارکردن (Load)         | L <آدرس شروع> <دیسک‌گردان> <قطاع شروع> <قطاع‌ها>                                           |
| جابجایی (Move)         | M <آدرس شروع> <آدرس پایان> <مقصد><br>M <آدرس شروع> <تعداد بایت‌های L> <مقصد>               |
| نام (Name)             | N <نام فایل>                                                                               |
| رویه (Procedure)       | P <آدرس شروع> = <تعداد دستورالعمل>                                                         |
| ثبات (Register)        | R <نام ثبات>                                                                               |
| جستجو (Search)         | S <آدرس شروع> <آدرس پایان> <داده><br>S <آدرس شروع> <آدرس پایان> <داده>                     |
| ردیابی (Trace)         | T <آدرس شروع> = <تعداد دستورالعمل‌ها>                                                      |
| عکس اسمبل (Unassemble) | U <آدرس شروع> <آدرس پایان><br>U <آدرس شروع> <تعداد بایت‌های L>                             |
| نوشتن (Write)          | W <آدرس شروع> <دیسک‌گردان> <قطاع شروع> <قطاع‌ها>                                           |

توجه:

- ۱- همه آدرس‌ها و اعداد در مبنای شانزده هستند.
- ۲- فرمان‌ها می‌توانند با حروف کوچک، بزرگ، یا هر دو باشند.
- ۳- Ctrl-C هر فرمانی را متوقف می‌سازد.
- ۴- Ctrl\_num\_Lock خروجی طوماری فرمان را متوقف می‌کند. برای شروع مجدد دوباره یک کلید را فشار دهید.



---

# دستورالعمل‌ها و زمان‌بندی 80x86

---

در اولین بخش از این پیوست، دستورات 8086 را همراه با قالب و عملوندهایشان ارائه کرده و کار هر دستورالعمل را توصیف می‌کنیم. هر کجا لازم باشد مثال‌های برنامه‌نویسی داده شده‌اند. این دستورالعمل‌ها روی هر کامپیوتر 8086 و یا سازگار IBM کار خواهد کرد. دستورالعمل‌های اضافی دیگری برای ریزپردازنده‌های بالاتر (80186 و بالاتر) وجود دارد؛ با این وجود این دستورالعمل‌ها در این لیست داده نشده‌اند. بخش دوم لیستی از شماره پالس‌های ساعت برای هر دستورالعمل در خانواده 80x86 است.



### AAA تنظیم اسکی پس از جمع (ASCII Adjust after Addition)

پرچمها: متأثر: AF و CF: نامشخصها: OF ، SF ، ZF ، PF

قالب: AAA

**عملکرد:** این دستورالعمل پس از یک دستور ADD که دو رقم اسکی را جمع کرده است بکار می‌رود. این تابع جمع اعداد اسکی را بدون حذف نیبل بالاتر "3" ممکن می‌سازد. نتیجه به صورت فشرده نشده BCD در AL و برقراری رقم نقلی در صورت لزوم خواهد بود. این دستور فقط ثبات AL را اصلاح می‌نماید. اگر پرچم رقم نقلی نشانده شود، AH افزایش می‌یابد.

مثال ۱:

```
MOV AL,31H ;AL=31 THE ASCII CODE FOR 1
ADD AL,37H ;ADD 37 (ASCII FOR 7) TO AL; AL=68H
AAA ;AL=08 AND CF=0
```

در مثال فوق، اسکی 1 (31H) به اسکی 7 (37H) اضافه شده است. پس از دستور AAA، AL حاوی 8 در BCD و CF=0 است. مثال زیر جمع اسکی دیگر و سپس تنظیم آن را نشان می‌دهد:

مثال ۲:

```
MOV AL,'9' ;AL=39 ASCII FOR 9
ADD AL,'5' ;ADD 35 (ASCII FOR 5) TO AL THEN AL=6EH
AAA ;NOW AL=04 CF=1
OR AL,30H ;converts result to ASCII
```

### AAD تنظیم اسکی از تقسیم (ASCII Adjust before Division)

پرچمها: متأثر: SF ، ZF ، PF. نامشخصها: OF ، AF ، CF

قالب: AAD

**عملکرد:** این دستور قبل از دستور DIV برای تبدیل دو رقم BCD فشرده نشده در AL و AH به دودویی بکار می‌رود. نام بهتر برای آن می‌تواند تبدیل BCD به دودویی قبل از تقسیم باشد. این تابع تقسیم اعداد اسکی را میسر می‌سازد. قبل از اینکه دستور AAD اجرا شود، باید 3 را از نیبل فوقانی AH و AL حذف کرد.

مثال:

```
MOV AX,3435H ;AX=3435 THE ASCII FOR 45
AND AX,0F0FH ;AX=0405H UNPACKED BCD FOR 45
AAD ;AX=002DH HEX FOR 45
MOV DL,07 ;DL=07
DIV DL ;2DH DIV BY 07 GIVES AL=06,AH=03
OR AX,3030H ;AL=36=QUOTIENT AND AH=33=REMAINDER
```

### AAM تنظیم اسکی پس از ضرب (ASCII Adjust after Multiplication)

پرچمها: متأثر: AF ، CF. نامشخصها: OF ، SF ، ZF ، PF

قالب: AAM

**عملکرد:** مجدداً نام بهتر برای این دستور تنظیم BCD پس از ضرب است. این دستور پس از آن که

دستور MUL دو عدد غیر فشرده BCD را در هم ضرب کرد بکار می‌رود. این دستور AX را از دودویی به BCD غیر فشرده تبدیل می‌نماید. AAM فقط AL را اصلاح می‌کند. هر رقم بزرگتر از 9 در AH ذخیره می‌شود.

مثال:

|     |          |                           |
|-----|----------|---------------------------|
| MOV | AL,'5'   | ;AL=35                    |
| AND | AL,0FH   | ;AL=05 UNPACKED BCD FOR 5 |
| MOV | BL,'4'   | ;BL=34                    |
| AND | BL,0FH   | ;BL=04 UNPACKED BCD FOR 4 |
| MUL | BL       | ;AX=0014H=20 DECIMAL      |
| AAM |          | ;AX=0200                  |
| OR  | AX,3030H | ;AX=3230 ASCII FOR 20     |

### AAS تنظیم اسکی پس از تفریق (ASCII Adjust after Subtraction)

پرچم‌ها: متاثر AF و CF. نامشخص‌ها: OF, SF, ZF, PF.

قالب: AAS

عملکرد: پس از تفریق دو رقم اسکی، این دستور برای تبدیل نتیجه حاصل در AL به BCD فشرده شده بکار می‌رود. تنها AL تنظیم می‌شود و مقدار موجود در AH در صورت 1 بودن رقم نقلی یک واحد کسر خواهد شد.

مثال:

|     |        |                                          |
|-----|--------|------------------------------------------|
| MOV | AL,32H | ;AL=32 ASCII FOR 2                       |
| MOV | DH,37H | ;DH=37 ASCII FOR 7                       |
| SUB | AL,DH  | ;AL-DH=32-37=FBH WHICH IS -5 IN 2'S COMP |
|     |        | ;CF=1 INDICATING A BORROW                |
| AAS |        | ;NOW AL=05 AND CF=1                      |

### ADC جمع با نقلی (Add with Carry)

پرچم‌ها: متاثر: CF, PF, AF, ZF, SF, OF.

قالب: ADC dest, source ; dest = dest + source + CF

عملکرد: اگر قبل از این دستور  $CF = 1$  باشد، پس از اجرای این دستور مبدأ به مقصد و 1 اضافه می‌شود. اگر  $CF = 0$  باشد، منبع با مقصد و 0 جمع می‌گردد. در بسیاری از جمع‌های چندبایتی و چند کلمه‌ای بکار می‌رود.

### ADD جمع علامت دار یا بی علامت (Signed Unsigned ADD)

پرچم‌ها: متاثر: CF, PF, AF, ZF, SF, OF.

قالب: ADC dest, source ; dest = dest + source

عملکرد: عملوند مبدأ را به عملوند مقصد اضافه کرده و نتیجه را در مقصد قرار می‌دهد. هر دو عملوند مبدأ و مقصد باید قابل تطبیق باشند (مثلاً، هر دو اندازه بایت یا اندازه کلمه داشته باشند) و فقط یکی از آنها می‌تواند در حافظه باشد.

### جمع بدون علامت (Unsigned addition)

در جمع اعداد بدون علامت، وضعیت CF, ZF, SF, AF و PF ممکن است تغییر کند ولی CF, ZF و AF مورد استفاده برنامه نویسان است. مهم‌ترین این پرچم‌ها CF است. وقتی که رقم نقلی از D7 در اعمال 8 بیت (در 8 بیت D0-D7) و یا در اعمال 16 بیت از D15 خارج شود مقدار آن 1 خواهد شد.

مثال ۱:

|     |        |                               |
|-----|--------|-------------------------------|
| MOV | BH,45H | ;BH=45H                       |
| ADD | BH,4FH | ;BH=94H (45H+4FH=94H)         |
|     |        | ;CF=0,ZF=0,SF=1,AF=1,and PF=0 |

|     |        |                           |
|-----|--------|---------------------------|
| MOV | AL,FEH | ;AL=FEH                   |
| MOV | DL,75H | ;DL=75H                   |
| ADD | AL,DL  | ;AL=FE+75=73H             |
|     |        | ;CF=1,ZF=0,AF=0,SF=0,PF=0 |

مثال ۲:

|     |           |                               |
|-----|-----------|-------------------------------|
| MOV | DX,126FH  | ;DX=126FH                     |
| ADD | DX,3465H  | ;DX=46D4H (126F+3465=46D4H)   |
|     |           | ;CF=0,ZF=0,AF=1,SF=0,PF=1     |
| MOV | BX,0FFFFH |                               |
| ADD | BX,1      | ;BX=0000 (FFFFH+1=0000)       |
|     |           | ;AND CF=1,ZF=1,AF=1,SF=0,PF=1 |

مثال ۳:

جمع اعداد علامت دار: (signed addition)

در جمع اعداد علامت دار، به وضعیت OF، ZF و SF باید توجه شود. توجه خاصی باید به پرچم سرریز (OF) نمود زیرا این پرچم خطائی را در نتیجه جمع مشخص می نماید. در اعمال اعداد علامت دار دو قاعده برای 1 شدن پرچم سرریز وجود دارد:

- ۱- در یک عمل 8 بیتی اگر یک رقم نقلی از بیت D6 به D7 وجود داشته باشد ولی از D7 رقم نقلی خارج نشود و یا رقم نقلی از D14 به D15 وجود داشته ولی رقم نقلی از D15 به بیرون نرود، سرریز رخ خواهد داد.
- ۲- اگر رقم نقلی از D7 خارج شود ولی رقم نقلی از D6 به D7 در عمل 8 بیت وجود نداشته باشد و یا نقلی از D15 خارج شده ولی رقم نقلی از D14 به D15 در عمل 16 بیتی وجود نداشته باشد سرریز رخ می دهد.

|     |       |                                   |
|-----|-------|-----------------------------------|
| MOV | BL,+8 | ;BL=0000 1000                     |
| MOV | DH,+4 | ;DH=0000 0100                     |
| ADD | BL,DH | ;BL=0000 1100 SF=0,ZF=0,OF=0,CF=0 |

مثال ۴:

توجه کنید که  $SF = D7 = 0$  است زیرا نتیجه مثبت و  $OF = 0$  می باشد. هیچگونه رقم نقلی از D6 به D7 و یا از D7 به بیرون وجود ندارد. چون  $OF = 0$  است، نتیجه صحیح می باشد  $[(+8) + (+4) = (+12)]$ .

|     |        |                                  |
|-----|--------|----------------------------------|
| MOV | AL,+66 | ;AL=0100 0010                    |
| MOV | CL,+69 | ;CL=0100 0101                    |
| ADD | CL,AL  | ;CL=1000 0111 = -121 (INCORRECT) |
|     |        | ;CF=0,SF=1,ZF=0, AND OF=1        |

مثال ۵:

در مثال ۵، نتیجه صحیح  $+135$  است  $[(+66) + (+69) = (+135)]$ ، ولی نتیجه  $-121$  شده است.  $OF = 1$  نشانه ای از خطاست. توجه کنید که  $SF = D7 = 1$  می باشد زیرا نتیجه منفی است؛  $OF = 1$  می باشد زیرا یک نقلی از D6 به D7 رخ داده و  $CF = 0$  است.

|     |        |                             |
|-----|--------|-----------------------------|
| MOV | AL,-12 | ;AL=1111 0100               |
| MOV | BL,+18 | ;BL=0001 0010               |
| ADD | BL,AL  | ;BL=0000 0110 (WHICH IS +6) |
|     |        | ;SF=0,ZF=0,OF=0, AND CF=1   |

مثال ۶:

توجه کنید که  $OF = 0$  است زیرا از D6 به D7 و از D7 به بیرون رقم نقلی وجود دارد.

MOV AH,-30 ;AH=1110 0010  
 MOV DL,+14 ;DL=0000 1110  
 ADD DL,AH ;DL=1111 0000 (WHICH IS -16 AND CORRECT)  
 ;AND SF=1,ZF=0,OF=0, AND CF=0

مثال ۷:

OF = 0 است زیرا رقم نقلی از D7 به بیرون وجود ندارد و هیچ نقلی از D6 به D7 نرفته است.

MOV AL,-126 ;AL=1000 0010  
 MOV BH,-127 ;BH=1000 0001  
 ADD AL,BH ;AL=0000 0011 (WHICH IS +3 AND WRONG)  
 ;AND SF=0,ZF=0 AND OF=1

مثال ۸:

OF = 1 است زیرا یک رقم نقلی از D7 خارج شده ولی از D6 به D7 نقلی وجود ندارد.

### AND AND منطقی

| X | Y | X AND Y |
|---|---|---------|
| 0 | 0 | 0       |
| 0 | 1 | 0       |
| 1 | 0 | 0       |
| 1 | 1 | 1       |

پرچم‌ها: متاثر: PF, ZF, SF, OF = 0, CF = 0

نامشخص‌ها: AF

قالب: AND dest, source

عملکرد: روی عملوندها AND منطقی را بیت به بیت انجام داده

و نتیجه را در مقصد ذخیره می‌کند.

MOV BL,39H ;BL=39  
 AND BL,09H ;BL=09  
 ;39 0011 1001  
 ;09 0000 1001  
 ;--  
 ;09 0000 1001

مثال:

### CALL فراخوانی رویه

پرچم‌ها: بدون تغییر

قالب: انتقال کنترل به رویه CALL proc;

عملکرد: کنترل را به یک رویه منتقل می‌نماید. RET برای بازگشت کنترل به دستور پس از CALL بکار می‌رود. دو نوع CALL وجود دارد: NEAR و FAR. اگر آدرس مقصد در همان قطعه کد باشد، دستور از نوع فراخوانی NEAR است. اگر مقصد در بیرون قطعه کد جاری باشد، از نوع FAR CALL می‌باشد. هر یک از این دو در زیر توصیف شده است.

**NEAR CALL:** اگر فراخوانی به یک رویه نزدیک صورت می‌گیرد (رویه در همان قطعه کدی است که دستور CALL قرار دارد) آنگاه محتوای ثبات IP (که آدرس دستور بعد از CALL است) در پشته درج شده و از SP دو واحد کاسته می‌شود. سپس IP با مقدار جدیدی که تفاوت مکان رویه است بار می‌گردد. در پایان رویه وقتی که RET اجرا شود، IP از پشته خارج شده و بدین وسیله کنترل به دستور بعد از CALL محول می‌شود. سه روش برای نوشتن آدرس رویه NEAR فراخوانده وجود دارد.

CALL proc1  
 ...  
 proc1 PROC NEAR  
 ...  
 RET  
 proc1 ENDP

۱- مستقیم:

۲- غیر مستقیم ثباتی :

CALL [SI] ;transfer control to address in SI

۳- غیر مستقیم حافظه‌ای :

CALL WORD PTR [DI] ;DI points to the address that  
;contains IP address of proc

**FAR CALL** : هنگام فراخوانی یک رویه دور (رویه در قطعه‌ای دیگر نسبت به دستور CALL قرار دارد)، پس از درج شدن CS:IP دستورالعمل بعد از CALL در پشته SP، 4 واحد کسر می‌گردد. آنگاه CS:IP با آدرس قطعه و تفاوت مکان رویه فراخوانده شده بار می‌شود. در درج CS:IP در پشته، CS ابتدا و سپس IP درج می‌شود. وقتی که RETF اجرا شود، CS و IP از پشته بازنشانی شده و اجرا با دستورالعمل بعد از CALL ادامه می‌یابد. روش‌های آدرس دهی زیر پشتیبانی می‌شوند.

۱- مستقیم (ولی خارج قطعه فعلی):

```
CALL proc1
proc1 PROC FAR
...
RETF
proc1 ENDP
```

۲- غیر مستقیم حافظه‌ای :

CALL DWORD PTR [DI] ;transfer control to CS:IP where  
;DI and DI+1 point to location of CS and  
;DI+2 and DI+4 point to location of IP

## CBW تبدیل بایت به کلمه

پرچم‌ها: بدون تغییر

قالب: CBW

عملکرد: D7 (بیت علامت) را در تمام بیت‌های AH کپی می‌کند. بطور گسترده‌ای برای تبدیل بایت علامت دار در AL به کلمه علامت دار برای ممانعت از مسئله سرریز در حساب عدد علامت دار بکار می‌رود.

مثال:

```
MOV AX,0
MOV AL,-5 ;AL=(-5)=FB in 2's complement
;AX = 0000 0000 1111 1011
CBW ;now AX=FFFB
;AX = 1111 1111 1111 1011
```

## CLC پاک کردن پرچم تقلی

پرچم‌ها: متاثر: CF

قالب: CLC

عملکرد: CF را پاک می‌کند (CF = 0).

## CLD پاک کردن پرچم جهت

پرچم‌ها: متاثر: DF

قالب: CLD

عملکرد: DF را پاک می‌کند (DF = 0). در دستورات رشته‌ای اگر DF = 0 باشد، اشاره‌گرها با هر بار

اجرای دستور افزایش می‌یابد. اگر  $DF = 1$  باشد اشاره‌گرها کاهش می‌یابند. بنابراین CLD قبل از دستورات رشته‌ای برای تغییر اشاره‌گرها بکار می‌رود.

### CLI پاک کردن پرچم وقفه

پرچم‌ها: متاثر: IF

قالب: CLI

عملکرد: IF را پاک می‌کند و از آن پس وقفه‌های بخارجی رسیده در ورودی INTR را ممانعت می‌نماید. وقفه‌های دریافتی در ورودی NMI بوسیله این دستور بلوک نمی‌شوند.

### CMC متمم سازی پرچم نقلی

پرچم‌ها: متاثر: CF

قالب: CMC

عملکرد: تغییر CF از 0 به 1 و از 1 به 0

### CMP مقایسه عملوندها

پرچم‌ها: متاثر: CF, PF, AF, ZF, SF, OF

قالب: CMP dest, source; sets flag as if "SUB dest, source"

عملکرد: دو عملوند را با اندازه مساوی مقایسه می‌کند. عملوندهای مبدأ و مقصد تغییری نمی‌یابند. مقایسه را با تفریق عملوند مبدأ از مقصد، و برقراری پرچم‌ها را مثل SUB انجام می‌دهد. پرچم‌های مربوطه بقرار زیرند:

|               | CF | ZF | SF | OF            |
|---------------|----|----|----|---------------|
| dest > source | 0  | 0  | 0  | SF            |
| dest = source | 0  | 1  | 0  | SF            |
| dest < source | 1  | 0  | 1  | inverse of SF |

### CMPSW/CMPSB/CMPS بایت یا رشته کلمه را مقایسه کن

پرچم‌ها: متاثر: CF, PF, AF, ZF, SF, OF

قالب: CMPSx

عملکرد: رشته‌ها را هر بایت یک بار یا یک کلمه یکبار مقایسه می‌نماید. DS:SI برای آدرس دهی اولین عملوند بکار می‌برد؛ ES:DI برای آدرس دهی دومین عملوند بکار گرفته می‌شود. می‌توان آن را با پیشوند REPE یا REPNE برای رشته‌هایی با هر طول بکار برد. عمل مقایسه با تفریق عملوند و برقراری پرچم‌هایی چون SUB صورت می‌گیرد.

### CWD تبدیل کلمه به جفت کلمه

پرچم‌ها: بدون تغییر

قالب: CWD

عملکرد: یک کلمه علامت دار در AX را بوسیله کپی نمودن بیت علامت AX در همه بیت‌های DX به جفت کلمه علامت دار تبدیل می‌کند. اغلب برای جلوگیری از مشکل سرریز در حساب عدد علامت دار بکار می‌رود.

مثال :

```
MOV DX,0
MOV AX,-5
;DX = 0000H
CWD
;DX = FFFFH
```

;AX=(-5)=FFFB in 2's complement

### DAA تنظیم ددهی پس از جمع

پرچمها : متاثر : SF , ZF , AF , PF , CF , OF .

قالب : DAA :

عملکرد : این دستورالعمل پس از جمع BCD اعداد برای تبدیل نتیجه به همان فرم BCD بکار می رود. این دستور 6 واحد به 4 بیت پایین رتبه AL می افزاید بشرطی که این بیت ها بزرگتر از 9 و یا AF = 1 باشد. آنگاه 6 را به 4 بیت مرتبه بالاتر AL خواهد افزود بشرطی که مقدار آن بزرگتر از 9 و یا CF=1 شود.

مثال ۱ :

```
MOV AL,47H ;AL=0100 0111
ADD AL,38H ;AL=47H+38H=7FH. invalid BCD
DAA ;NOW AL=1000 0101 (85H IS VALID BCD)
```

در این مثال، چون نیل پایین تر بزرگتر از 9 است، DAA، عدد 6 را به AL اضافه کرده است. اگر نیل پایین تر کوچکتر از 9 ولی AF = 1 باشد، 6 را به نیل پایین تر اضافه می نماید.

مثال ۲ :

```
MOV AL,29H ;AL=0010 1001
ADD AL,18H ;AL=0100 0001 INCORRECT RESULT
DAA ;AL=0100 0111 A VALID BCD FOR 47H.
```

وضعیت مشابهی برای نیل بالاتر می تواند اتفاق افتد.

مثال ۳ :

```
MOV AL,52H ;AL=0101 0010
ADD AL,91H ;AL=1110 0011 AN INVALID BCD
DAA ;AL=0100 0011 AND CF=1
```

مجدداً نیل بالاتر می تواند کوچکتر از 9 باشد ولی به دلیل CF=1 باید آن را اصلاح کرد.

مثال ۴ :

```
MOV AL,94H ;AL=1001 0100
ADD AL,91H ;AL=0010 0101 INCORRECT RESULT
DAA ;AL=1000 0101 A VALID BCD FOR 85 AND CF=1
```

کلاً ممکن است 6 به هر دو نیل بالا و پائین اضافه شود.

مثال ۵ :

```
MOV AL,54H ;AL=0101 0100
ADD AL,87H ;AL=1101 1011 INVALID BCD
DAA ;AL=0100 0001 AND CF=1 (141 IN BCD)
```

## DAS تنظیم دهنده پس از تفریق

پرچمها: متأثر: SF, ZF, AF, PF, CF. نامشخص: OF

قالب: DAS:

عملکرد: این دستورالعمل پس از تفریق اعداد BCD برای تبدیل نتیجه به BCD بکار می‌رود. اگر چهار بیت پائین تر AL یک عدد بزرگتر از 9 را نشان دهد و یا  $AF=1$  باشد، آنگاه، 6 از نیل پائین تر کسر می‌گردد. اگر 4 بیت بالاتر AL بزرگتر از 9 و یا اگر  $CF=1$  باشد، 6 از نیل بالاتر کسر می‌شود.

مثال:

|     |         |                                    |
|-----|---------|------------------------------------|
| MOV | AL, 45H | ;AL=0100 0101 BCD for 45           |
| SUB | AL, 17H | ;AL=0010 1110 AN INVALID BCD       |
| DAS |         | ;AL=0010 1000 BCD FOR 28(45-17=28) |

برای ملاحظه مثالهای بیشتر مربوط به حساب BCD، DAA ملاحظه شود.

## DEC کاهش

پرچمها: متأثر: SF, ZF, AF, PF. بدون تغییر: CF.

قالب: DEC dest; dest=dest-1

عملکرد: 1 را از عملوند مقصد کم می‌کند. توجه کنید که CF (نقلی/قرض) بدون تغییر می‌ماند حتی اگر مقدار 0000 کم شده و FFFF حاصل گردد.

## DIV تقسیم بدون علامت

پرچمها: نامشخص: SF, ZF, AF, PF, CF, OF

قالب: DIV source; divide AX or DX: AX by source

عملکرد: یک کلمه بدون علامت (AX) را بر یک بایت، یا یک جفت کلمه (DX:AX) را بر کلمه تقسیم می‌نماید. اگر تقسیم یک کلمه بر یک بایت باشد، خارج قسمت در AL و باقیمانده در AH است. اگر جفت کلمه‌ای بر یک کلمه تقسیم شود خارج قسمت در AX و باقیمانده در DX است. تقسیم بر صفر موجب وقفه نوع 0 می‌گردد.

## ESC گذر

پرچمها: بدون تغییر

قالب: ESC

عملکرد: این دستورالعمل استفاده از کمک پردازنده‌ها را (مانند 8087)، که گذرگاه‌های آدرس و داده را با ریزپردازنده به اشتراک می‌گذارند، ممکن می‌سازد. ESC برای تحویل یک دستورالعمل به پردازنده کمکی بکار رفته و معمولاً پردازنده اصلی آن را مثل NOP (هیچکار) تلقی کند.

## HLT توقف

پرچمها: بدون تغییر

قالب: HLT

عملکرد: موجب می‌شود تا ریزپردازنده اجرای دستورالعمل‌ها را متوقف کند. برای خروج از حالت توقف یا یک وقفه (NMI یا INTR) و یا RESET را فعال کنید.

## IDIV تقسیم عدد علامت‌دار

پرچمها: نامشخص: SF, ZF, AF, PF, CF, OF



قالب : IDIV source ; divide AX or DX:AX by source

تابع : این تابع یک کلمه علامت‌دار (AX) را بر یک بایت و یا جفت کلمه (DX:AX) را بر یک کلمه تقسیم می‌کند. اگر کلمه‌ای بر بایت تقسیم شود، خارج قسمت علامت‌دار در AL و باقی مانده علامت‌دار در AH است. اگر جفت کلمه‌ای بر کلمه‌ای تقسیم گردد، خارج قسمت علامت‌دار در AX و باقیمانده علامت‌دار در DX است. تقسیم بر 0 وقفه نوع 0 را موجب می‌شود.

### IMUL ضرب عدد علامت دار

پرچم‌ها : متاثر : CF , OF , نامشخص : PF , AF , ZF , SF

قالب : IMUL source ; AX = source × AL or DX:AX = source × AX

عملکرد : یک بایت علامت‌دار یا کلمه مبدأ را در یک بایت یا کلمه علامت‌دار در AL یا AX ضرب کرده نتیجه را در AX یا DX:AX قرار می‌دهد.

### IN ورود داده از پورت

پرچم‌ها : بدون تغییر

قالب : IN accumulator , port ; input byte or word into AL or AX

عملکرد : یک بایت یا کلمه از یک پورت ورودی که با عملوند دوم مشخص می‌شود به AL یا AX منتقل می‌شود.

۱. مستقیم : آدرس پورت مستقیماً مشخص می‌شود و نمی‌تواند بزرگتر از FF باشد.

مثال ۱:

IN AL,99H ;BRING A BYTE INTO AL FROM PORT 99H

مثال ۲:

IN AX,78H ;BRING A WORD FROM PORT ADDRESSES 78H  
;AND 79H. THE BYTE FROM PORT 78 GOES  
;TO AL AND BYTE FROM PORT 79H TO AH.

۲. غیر مستقیم ثباتی: آدرس پورت در ثبات DX نگهداری می‌شود. بنابراین می‌تواند تا FFFFH باشد.

مثال ۳:

MOV DX,481H ;DX=481H  
IN AL,DX ;BRING THE BYTE TO AL FROM THE PORT  
;WHOSE ADDRESS IS POINTED BY DX

مثال ۴:

IN AX,DX ;BRING A WORD FROM PORT ADDRESS OF  
;POINTED BY DX. THE BYTE FROM PORT  
;DX GOES TO AL AND BYTE FROM PORT  
;DX+1 TO AH.

### INC افزایش

پرچم‌ها : متاثر: PF , AF , ZF , SF , OF بدون تغییر : CF

قالب : INC destination ; dest=dest + 1

تابع : یک واحد به ثبات یا مکان حافظه مشخص شده بوسیله عملوند اضافه می‌کند. توجه کنید که حتی اگر یک واحد به FFFF اضافه شود تا 0000 بدست آید CF متاثر نمی‌شود.

جدول ۱ - ب جدول بردار وقفه

| آدرس منطقی | آدرس فیزیکی | INT# (hex) |
|------------|-------------|------------|
| 0000:0000  | 00000       | INT 00     |
| 0000:0004  | 00004       | INT 01     |
| 0000:0008  | 00008       | INT 02     |
| 0000:000C  | 0000C       | INT 03     |
| 0000:0010  | 00010       | INT 04     |
| 0000:0014  | 00014       | INT 05     |
| .....      | .....       | .....      |
| 0000:03FC  | 003FC       | INT FF     |

## INT وقفه

پرچمها: متاثر IF ، TF

قالب: INT type ; transfer control to INT type

عملکرد: اجرا را به یکی از 256 وقفه می سپارد. آدرس بردار بوسیله شماره نوع معین شده و نمی تواند بزرگتر از FFH (0 تا FF = 256 وقفه) باشد.

مراحل زیر برای وقفه اجرا می شود:

- ۱ - SP دو واحد کسر می شود و پرچمها در پشته ذرج می گردند.
- ۲ - SP دو واحد کسر می گردد و CS در پشته درج می شود.
- ۳ - SP دو واحد کم شده و IP بعد از دستور وقفه در پشته درج می گردد.
- ۴ - شماره نوع را در 4 ضرب می کند تا آدرس جدول بردار بدست آید. با شروع از این آدرس، 2 بایت اول مقدار IP و 2 بایت بعد مقدار CS برای اداره کننده وقفه است (اداره کننده وقفه را روال سرویس وقفه هم می گویند).
- ۵ - IF و TF را پاک می کند.

وقفهها برای جلب توجه ریزپردازنده بکار می روند. در 8086/88 جمعاً 256 وقفه وجود دارد: INT00 ، INT01 ، INT02 ... INTFF. همانطور که در بالا ذکر شد، آدرسی که وقفه به آن پرش می کند همیشه چهار برابر عدد شماره وقفه است. مثلاً INT03 به حافظه 0000CH (0CH = 12 = 03 × 4) پرش خواهد کرد. جدول ۱ - ب بخشی از لیست وقفهها، که عموماً به آن جدول بردار وقفه می گویند را نشان می دهد. هر وقفه ای برنامه مربوط به خود را دارد که به آن روال سرویس وقفه (ISA) می گویند. وقتی وقفه ای رخ می دهد، آدرس CS:IP مربوط به ISR از جدول بردار بازایی می شود. پائین ترین بایت 1024 از RAM (1024 = 4 × 256) برای جدول بردار وقفه کنار گذاشته شده و نباید برای کار دیگری بکار رود.

مثال: آدرس فیزیکی و منطقی جدول بردار مربوط به (الف) INT 14H و (ب) INT38H را پیدا کنید.

حل:

(الف) آدرس فیزیکی INT 14H عبارتست از 00050H-00053H (4 × 14H = 50H). بنابراین آدرس منطقی عبارتست از 0000:0050H تا 0000:0053H.

(ب) آدرس فیزیکی INT 38H عبارتست از 000E0H-000E3H ، آدرس منطقی چنین خواهد شد 0000:00E0H-0000:00E3H

## اختلاف بین دستورات CALL FAR, INT

در زیر برخی از تفاوت های بین دستورالعمل های INT و CALL FAR آمده است:

جدول ۲ - ب سیستم وقفه IBM PC

| وقفه | آدرس منطقی  | آدرس فیزیکی | هدف                              |
|------|-------------|-------------|----------------------------------|
| 0    | 00E3:3072   | 0EA2        | خطای مستقیم                      |
| 1    | 0600:08ED   | 068ED       | تک مرحله (فرمان ردیابی در DEBUG) |
| 2    | F000:E2C3   | FE2C3       | وقفه غیر قابل پوشش               |
| 3    | 0600:08E6   | 068E6       | نقطه توقف                        |
| 4    | 0700:0147   | 00847       | سرریز عدد علامت دار حسابی        |
| 5    | F000:FF54   | FFF54       | چاپ صفحه (BIOS)                  |
| 10   | F000:F065   | FF065       | I/O تصویری (BIOS)                |
| ...  | ...         | ...         | ...                              |
| 21   | relocatable | ---         | فراخوانی تابع DOS                |
| ...  | ...         | ...         | ...                              |

- ۱ - در حالی که CALL به هر مکانی در محدوده آدرس 1 مگابایتی (00000 تا FFFFF) از 8088/86 CPU می تواند پرش کند، "INT nn" به محل خاص در جدول بردار که قبلاً بحث شد پرش می نماید.
  - ۲ - در حالی که CALL بوسیله برنامه نویس در نقطه از پیش تعیین شده بکار می رود، وقفه سخت افزاری در هر لحظه از زمان ممکن است بوقوع پیوندد.
  - ۳ - CALL قابل پوشش نیست، ولی "INT nn" می تواند ممانعت گردد.
  - ۴ - "CALL FAR" قادر است علاوه بر CS:IP، ثابت پرچم FR را هم ذخیره نماید.
  - ۵ - در انتهای رویه ای که فراخوانی شده RETF (بازگشت از FAR) بکار رفته است، در حالی که برای "INT nn" از IRET (بازگشت از وقفه) استفاده می شود.
- 256 وقفه را می توان در دو گروه متفاوت دسته بندی کرد: وقفه های سخت افزاری و نرم افزاری.

#### وقفه های سخت افزاری

ریزپردازنده های 8086/88 دارای دو پایه اند که برای وقفه های سخت افزاری کنار گذاشته شده اند. این دو وقفه INTR (تقاضای وقفه) و NMI (وقفه غیر قابل پوشش) می باشند. گرچه INTR از طریق نرم افزار قابل پیش گیری است، NMI نمی تواند با نرم افزار جلوگیری شود. این وقفه ها از بیرون با قرار گرفتن 5 ولت در پایه های سخت افزاری NMI یا INTR فعال می شوند. اینتل INT02 را به NMI تخصیص داده است. هنگامی که این وقفه فعال شود برنامه به مکان 00008 می پرد تا CS:IP مربوط به روال سرویس وقفه (ISR) را بدست آورد. آدرس های 00008، 00009، 0000A و 0000B حاوی 4 بایت CS:IP است. در جدول بردار مکان خاصی به INTR تخصیص نیافته است زیرا INTR برای گسترش تعداد وقفه های سخت افزاری بکار رفته و باید اجازه دهد تا هر دستور "INT nn" تخصیص نیافته قبلی مورد استفاده قرارگیرد. در IBM PC یک تراشه 8259 PIC (کنترلر وقفه برنامه پذیر) به INTR برای افزایش هشت وقفه سخت افزاری در ریزپردازنده بکاررفته است. کامپیوترهای IBM PC/AT، PS/2، 80286، 80386، 80486 دو تراشه 8259 را بکار می برند تا 15 وقفه سخت افزاری در اختیار باشد.

#### وقفه های نرم افزاری

این وقفه ها، نرم افزاری نام دارند زیرا در نتیجه اجرای یک دستور و نه سخت افزار بیرونی رخ می دهند. به بیان دیگر این وقفه ها با اجرای دستورالعمل "INT nn" مانند فراخوانی DOS، "INT 21H" و یا وقفه تصویری INT 10H اتفاق می افتند. این وقفه ها در هر زمان بوسیله یک برنامه می توانند رخ دهند. بسیاری از وقفه ها در این رده بوسیله سیستم عامل DOS و IBM BIOS بکار می روند تا بسیاری از کارهای اساسی لازم برای سیستم

و کاربر را فراهم سازند. همچنین در میان این گروه از وقفه‌ها توابع از پیش تعریف شده‌ای مربوط به بعضی از وقفه‌ها وجود دارد. این توابع عبارتند از "INT 00" (خطای تقسیم)، "INT 01" (تک مرحله)، "INT 03" (نقطه توقف یا شکست) و "INT 04" (سرریز عدد علامت دار). هر یک از این وقفه‌ها در زیر توصیف شده است و در جدول ۱-ب هم نشان داده‌اند. با نگاه به جدول ۱-ب می‌توان دید که جدا از "INT 00" تا "INT 04"، که توابع از پیش تعریف شده دارند، بقیه وقفه‌ها، یعنی از "INT 05" تا "INT FF" برای پیاده‌سازی وقفه‌های نرم‌افزاری یا سخت‌افزاری می‌توانند بکار روند.

### عملکرد مرتبط با وقفه‌های "INT 00" تا "INT 03"

همانطور که قبلاً ذکر شد، وقفه‌های "INT 00" تا "INT 03" دارای عملکرد از پیش تعیین شده‌ای هستند و نمی‌توانند به طریق دیگری مورد استفاده قرار گیرند. عملکرد هر یک از آنها در زیر توصیف شده است.

#### INT 00 (خطای تقسیم)

این وقفه، که گاهی به آن وقفه شرطی یا استثنایی هم گفته می‌شود، توسط ریزپردازنده به هنگام رخداد واقعه‌ای که قابل مراقبت نیست، مانند تقسیم یک عدد بر صفر، بوقوع می‌پیوندد. "INT 00" به هنگام تلاش ریزپردازنده برای تقسیم عدد بر صفر رخ می‌دهد. در IBM PC و سازگار با آن روال سرویس این وقفه مسئول نمایش پیام "DIVIDE ERROR" روی صفحه است. نمونه‌ای از این گونه برنامه‌ها در زیر آمده است.

```
MOV AL,25 ; put 25 into AL
MOV BL,00 ; put 00 into BL
DIV BL ; divide 25 by 00
```

این وقفه به هنگام بزرگ بودن خارج قسمت و امکان عدم گنجایش آن در ثبات مربوطه به هنگام اجرای دستورالعمل DIV نیز رخ می‌دهد.

#### INT 01 (تک مرحله)

اغلب نیاز به اجرای یک دستورالعمل از برنامه در هر بار و سپس بررسی ثبات (و نیز حافظه‌ها) برای مشاهده آنچه در CPU رخ می‌دهد مشاهده شده است. به این اجرا، تک مرحله می‌گویند. IBM و میکروسافت آن را در برنامه DEBUG، ردیابی (TRACE) می‌خوانند. برای پیاده‌سازی تک مرحله اینتل "INT 01" را برای همین مقصود در نظر گرفت. برای فرمان ردیابی، CPU بعد از اجرای هر دستور بطور خودکار به مکان 00004 پریده و چهار بایت CS:IP مربوط به روال سرویس وقفه را برمی‌دارد. یکی از وظایف این ISR تخلیه محتوای ثبات‌ها در صفحه نمایش است.

#### INT 02 (وقفه پوشش ناپذیر)

این وقفه در PC برای مشخص کردن خطاهای حافظه و تفکیک آن از میان دیگر مسائل است.

#### INT 03 (نقطه توقف)

در مد تک مرحله، می‌توان CPU و حافظه سیستم را پس از اجرای هر دستور واریسی کرد. یک نقطه توقف کار مشابهی را پس از اجرای گروهی از دستورات، و نه یک دستور، اجازه می‌دهند. نقطه توقف در مکان‌های خاصی از برنامه گذاشته می‌شود تا جریان اجرای برنامه را ردیابی کند و سپس نتایج حاصل از دستورات خاصی واریسی شوند. CPU برنامه را تا نقطه توقف اجرا نموده و سپس متوقف می‌شود. می‌توان از یک نقطه توقف به سمت نقطه دیگر پیشروی کرد تا اجرای برنامه کامل شود. به کمک اجرای تک مرحله و نقطه توقف برنامه‌ها را

می توان به سادگی عیب یابی و تست کرد. CPU 8086/88 اینتل از "INT 03" فقط برای پیاده سازی نقطه توقف استفاده کرده است. وقتی که دستورالعمل "INT 03" در برنامه ای جای گیرد، CPU برنامه را تا مواجه شدن با "INT 03" ادامه داده و سپس متوقف می شود. یک نکته جالب درباره این وقفه این است که دستور مذکور یک بایتی است و این برخلاف دستورالعمل های وقفه "INT nn" است که دو بایتی می باشند. بنابراین کاربر می تواند یک کد 1 بایتی را وارد کرده و هنگام اجرا هم آن را حذف کند. کد INT 03 برابر "CC" است.

## تخصیص وقفه های DOS و IBM PC

وقتی که IBM PC ساخته شد، طراحان در IBM مجبور شدند تا 256 وقفه موجود در 8086/88 را با میکروسافت که سازنده سیستم عامل DOS بود هماهنگ نمایند تا تداخل کمتری بین BIOS و DOS پیش آید. نتیجه هماهنگی در تخصیص این وقفه ها به زیر روال های IBM BIOS و فراخوانی تابعی DOS در جدول ۲ - ب نشان داده شده است. جدول تنها بخشی از وقفه های 00 تا FF را لیست کرده است که به همراه آدرس منطقی زیر روال برای هر وقفه، آدرس های فیزیکی آنها و هدف از هر وقفه آورده شده است. باید متذکر شد که بسته به کامپیوتر و ویرایش DOS، بعضی از آدرس های منطقی ممکن است با جدول ۲ - ب تفاوت داشته باشند.

## چگونه می توان جدول بردار را برای هر PC بدست آورد

می توان جدول بردار هر IBM PC/XT ، PC/AT ، PS/2 ، PS/1 یا هر کامپیوتر سازگار با IBM 80x86 را بدست آورده و آدرس منطقی هر وقفه را جستجو کرد. این عمل به کمک فرمان تخلیه DUMP در DEBUG یعنی D 0000:0000 امکان پذیر است.

```
A>debug
-D 0000:0000
0000:0000 E8 56 2B 02 56 07 70 00-C3 E2 00 F0 56 07 70 00
0000:0010 56 07 70 00 54 FF 00 F0-47 FF 00 F0 47 FF 00 F0
```

! توجه: محتوای مکان های حافظه ممکن است بسته به ویرایش DOS متفاوت باشند.

مثال: از تخلیه فوق، CS:IP روال سرویس مربوط به INT 5 را بدست آورید.  
حل:

برای یافتن آدرس "INT 5"، آدرس فیزیکی 00014H (00014H = 5 × 4) را محاسبه کنید. محتوای این مکان ها 00014=54 ، 00015=FF ، 00016=00 و 00017=F0 می باشند. بنابراین CS=F000 و IP=FF54 است.

## INTO وقفه در سرریز

پرچم ها: متاثر: IF ، TF

قالب: INTO :

عملکرد: این دستور اجرا را به یک اداره کننده وقفه که برای سرریز در صورت 1 شدن OF (پرچم سرریز) نوشته شده است منتقل می سازد. اینتل INT4 را برای این منظور در نظر گرفته است. بنابراین، اگر به هنگام اجرای INTO ، OF=1 شود، CPU به مکان حافظه 00010H (10H = 4 × 4) می پرد. محتوای مکان های حافظه 10H ، 11H ، 12H و 13H به عنوان IP و CS رویه اداره کننده بکار می رود. این دستور العمل برای تشخیص سرریز در جمع عدد علامت دار بسیار استفاده می شود. در اعمال علامت دار، OF در دو حالت 1 خواهد شد.

۱ - هر وقت یک رقم نقلی از D6 به D7 در عملیات هشت بیتی وجود داشته باشد، ولی رقم نقلی از D7 خارج نشود. (یا در عملیات ۱۶ بیتی رقم نقلی از D14 به D15 وجود داشته و CF=0 گردد).

۲- وقتی یک رقم نقلی از D7 خارج ولی از D6 به D7 رقم نقلی وجود نداشته باشد (یا در عملیات 16 بیتی از D15 رقم نقلی خارج شده ولی از D14 به D15 رقم نقلی وجود نداشته باشد).

### IRET بازگشت از وقفه

پرچم‌ها: متأثر: CF, PF, AF, ZF, SF, TF, IF, DF, OF

قالب: IRET

عملکرد: در انتهای یک روال سرویس وقفه (اداره کننده وقفه)، این دستور همه پرچم‌ها، CS و IP را با مقداری که قبل از وقفه داشته‌اند بار می‌کند بطوری که اجرا از دستور بعد از دستور INT ادامه یابد. در حالی که برای دستورالعمل CALL، از دستور RET در پایان زیر روال مربوطه استفاده می‌شود، IRET باید در زیر روال مربوط به دستور "INT XX" یا اداره کننده وقفه سخت‌افزاری قرار گیرد.

### دستورالعمل‌های JUMP (پرش)

دستورالعمل‌های زیر مربوط به پرش‌ها (شرطی و غیرشرطی) می‌باشند. آنها برحسب کاربردشان دسته بندی شده‌اند و نه برحسب الفبا.

#### J condition

پرچم‌ها: بدون تغییر

قالب: Jxx target ; jump to target upon condition

عملکرد: در صورت برآورده شدن شرایط برای پرش به آدرس هدف بکار می‌رود. آدرس هدف نمی‌تواند بیش از 128- یا 127+ بایت دور باشد. شرایط بوسیله ثبات پرچم مشخص می‌شود. شرایط پرش به سه گروه دسته‌بندی می‌شود.

(۱) مقادیر پرچم

(۲) مقایسه اعداد بدون علامت

(۳) مقایسه اعداد علامت‌دار

هر یک از موارد فوق در زیر توضیح داده شده‌اند.

۱- پرش شرطی "J condition" که در آن شرط به مقدار پرچم اشاره دارد. وضعیت هر بیت پس از اجرای دستور قبل از پرش در تصمیم‌گیری بکار می‌رود. دستورالعمل‌های "J condition" در زیر، یک شدن یا نشدن پرچم معینی را واریسی می‌کند.

|     |                  |              |
|-----|------------------|--------------|
| JC  | Jump Carry       | jump if CF=1 |
| JNC | Jump No Carry    | jump if CF=0 |
| JP  | Jump Parity      | jump if PF=1 |
| JNP | Jump No Parity   | jump if PF=0 |
| JZ  | Jump Zero        | jump if ZF=1 |
| JNZ | Jump No Zero     | jump if ZF=0 |
| JS  | Jump Sign        | jump if SF=1 |
| JNS | Jump No Sign     | jump if SF=0 |
| JO  | Jump Overflow    | jump if OF=1 |
| JNO | Jump No Overflow | jump if OF=0 |

توجه کنید که برای AF دستورالعمل پرش شرطی وجود ندارد.

۲- پرش شرطی "J condition" که در آن شرط منوط به مقایسه اعداد بی‌علامت است. پس از اجرای دستور مقایسه (CMP dest, source)، CF و ZF نتیجه مقایسه را مشخص می‌کنند:

| CF | ZF |             |
|----|----|-------------|
| 0  | 0  | مبدأ > مقصد |
| 0  | 1  | مبدأ = مقصد |
| 1  | 0  | مبدأ < مقصد |

اگر عملوندهای مقایسه شده بصورت اعداد بدون علامت ارزیابی شده‌اند دستورالعمل‌های "J condition"

زیر بکار می‌روند.

|     |                     |                       |
|-----|---------------------|-----------------------|
| JA  | Jump Above          | jump if CF=0 and ZF=0 |
| JAE | Jump Above or Equal | jump if CF=0          |
| JB  | Jump Below          | jump if CF=1          |
| JBE | Jump Below or Equal | jump if CF=1 or ZF=1  |
| JE  | Jump Equal          | jump if ZF=1          |
| JNE | Jump Not Equal      | jump if ZF=0          |

۳ - پرش شرطی "J condition"، که در آن شرط به مقایسه اعداد علامت‌دار ارجاع داده می‌شود. در حالت مقایسه اعداد علامت‌دار، هرچند که همان دستورالعمل (CMP dest,source) بکار رفته است پرچم‌های بکار رفته برای نتیجه به قرار زیرند:

|              |             |
|--------------|-------------|
| OF=SF یا Z=0 | مبدأ > مقصد |
| ZF=1         | مبدأ = مقصد |
| SF معکوس OF  | مبدأ < مقصد |

در نتیجه دستورالعمل‌های پرش شرطی جداگانه‌ای بکار رفته‌اند که در زیر آمده‌اند:

|     |                       |                       |
|-----|-----------------------|-----------------------|
| JG  | Jump Greater          | jump if ZF=0 or OF=SF |
| JGE | Jump Greater or Equal | jump if OF=SF         |
| JL  | Jump Less             | jump if OF≠SF         |
| JLE | Jump Less or Equal    | jump if ZF=1 or OF≠SF |
| JE  | Jump if Equal         | jump if ZF = 1        |

یک دستورالعمل پرش شرطی دیگر هم وجود دارد

JCXZ ;Jump if CX is Zero. ZF is Ignored.

همه دستورالعمل‌های پرش شرطی، از نوع پرش‌های کوتاه (short) می‌باشند. یعنی آدرس هدف نمی‌تواند بیش از 128- بایت به عقب یا 127+ بایت جلوتر از IP دستورالعمل بعد از پرش باشد. اگر قرار باشد پرش شرطی به آدرس هدفی ورای محدوده 128- تا 127+ برود چه می‌شود؟ راه حل این است که از JMP غیرشرطی مثل زیر استفاده کنیم.

```

 ADD BX,[SI]
 JNC NEXT
 JMP TARGET1
NEXT:
 ...
TARGET1: ADD DI,10
 ...

```

### JMP پرش غیر شرطی

پرچم‌ها: بدون تغییر

قالب: JMP [directives] target ; jump to target address

عملکرد: این دستورالعمل کنترل بدون شرطی را به آدرس جدید منتقل می‌سازد. اختلاف بین JMP و

CALL این است که دستور CALL اجرا را به دستور بعد از خود بازگردانده و ادامه می دهد، در صورتی که JMP برای ادامه باز نمی گردد. آدرس هدف می تواند در قطعه کد جاری باشد که در این صورت به آن پرش نزدیک (near) می گویند، و یا خارج قطعه کد جاری باشد که به آن پرش دور (far) گویند. در هر دسته راه های متعددی برای ارائه آدرس وجود دارد که در زیر آمده است.

#### ۱ - پرش نزدیک (SHORT)

(الف) پرش نزدیک مستقیم: در این پرش، آدرس هدف باید در محدوده 128- تا 127+ بایت از دستور بعد از JMP باشد. این دستور دو بایتی است. اولین بایت کد EBH و دومین بایت مقدار جابجایی علامت دار است که به IP دستور بعد از JMP اضافه می شود تا آدرس هدف بدست آید. رهنمون SHORT به طریق زیر در دستور وارد می شود.

#### JMP SHORT OVER

OVER:

اگر آدرس هدف ورای محدوده 128- تا 127+ بایت باشد و رهنمون SHORT کد شود اسمبلر خطایی را اعلام خواهد کرد.

(ب) پرش مستقیم: این یک دستور 3 بایتی است. بایت اول کد عمل E9H و دو بایت بعدی مقدار تفاوت مکان علامت دار است. تفاوت مکان به IP دستور بعد از JMP اضافه می شود تا آدرس هدف بدست آید. تفاوت مکان می تواند 32768- تا 32767+ باشد. در غیاب رهنمون SHORT، اسمبلر در گذر اول خود این نوع JMP را بکار می برد و سپس در دومین گذر، اگر آدرس هدف در محدوده 128- تا 127+ بایت باشد، دستور NOP، 90H را به عنوان بایت سوم بکار می گیرد. این دلیل کد کردن رهنمون SHORT است به شرطی که آدرس هدف JMP در محدوده SHORT باشد.

(پ) پرش غیر مستقیم ثباتی: در این پرش آدرس هدف در ثبات است:

هر ثبات غیر قطعه ای می تواند بکار رود.

#### JMP DI ; jump to the address found in DI

(ت) پرش غیر مستقیم حافظه ای: در این پرش آدرس هدف در مکانی از حافظه است که بوسیله ثباتی به

آن اشاره شده است:

#### JMP WORD PTR [SI] ; jump to the address found at the address in SI

رهنمون WORD PTR باید کد شود تا مشخص کند که پرش از نوع نزدیک است.

#### ۲ - پرش دور (FAR)

در JMP دور آدرس هدف، خارج از قطعه کد جاری است؛ بنابراین نه فقط مقدار تفاوت مکان بلکه مقدار قطعه آدرس هدف باید داده شود. پرش دور یک دستور 5 بایتی است: کد عمل EAH و 4 بایت برای تفاوت مکان قطعه آدرس هدف. در زیر دو روش کد کردن پرش دور آمده است.

(الف) پرش دور مستقیم: این روش لازم می دارد که هر دو CS و IP به هنگام شوند. یک راه برای انجام آن

استفاده از رهنمون LABEL است:

#### JMP TARGET2

TARGET2 LABEL FAR  
ENTRY: ...

این دقیقاً همان چیزی است که IBM در BIOS کامپیوتر IBM PC/XT هنگام بوت انجام می دهد. وقتی PC روشن می شود CPU 8088/86 شروع به اجرای آدرس FFFF:0000H می نماید. IBM از یک پرش دور



برای رفتن به F000:E05BH مطابق زیر استفاده می‌کند:

```

0000 EA5BE000F0 ;CS=FFFF and IP=0000
 JMP RESET
 ;CS=F000
 ORG 0E05BH
 LABEL FAR

E05B RESET
E05B START:
E05B
E05C CLI
 ...

```

رهنمون‌های EXTRN و PUBLIC نیز می‌توانند برای هدفی مشابه بکار روند.

(ب) پرش دور غیرمستقیم حافظه‌ای: آدرس هدف (CS:IP) در مکانی از حافظه است که بوسیله ثبات مشخص می‌شود:

```
JMP DWORD PTR [BX]
```

رهنمون‌های DWORD و PTR باید برای مشخص شدن پرش دور بکار روند.

### LAHF AH را از پرچم‌ها بار کن

پرچم‌ها: بدون تغییر

LAHF: قالب

عملکرد: 8 بیت پائین رتبه ثبات پرچم را در AH بار می‌کند.

### LDS ثبات قطعه داده را بار کن

پرچم‌ها: بدون تغییر

LDS dest, source ; load dest and DS starting at source : قالب

عملکرد: محتوای دو مکان از حافظه را که بوسیله مبدأ مشخص می‌شود در مقصد بار می‌کند و نیز DS را با محتوای دو مکان بعدی بار می‌نماید. این برای دستیابی به قطعه داده جدید و تفاوت مکان آن مفید است.

مثال: مکان‌های حافظه را همراه با محتوایشان چنین فرض کنید.

```

;DS:1200=(46)
;DS:1201=(10)
;DS:1202=(38)
;DS:1203=(82)
LDS DI,[1200] ;now DI=1046 and DS=8238.

```

### LEA آدرس مؤثر را بار کن

پرچم‌ها: بدون تغییر

LEA dest, source ; dest = OFFSET source : قالب

عملکرد: آدرس مؤثر یک عملوند حافظه مستقیم را در مقصد (16 بیت) بار می‌کند.

```

ORG 0100H
DATA DB 34,56,87,90,76,54,13,29
...
;to access the sixth element
LEA SI,DATA+5 ;SI=100H+5=105 THE EFFECTIVE ADDRESS
MOV AL,[SI] ;GET THE SIXTH ELEMENT

```

```

;if BX=2000H and SI=3500H
LEA DX,[BX][SI]+100H
;DX=effective address=2000+3500+100=5600H

```

دو دستورالعمل زیر دو راه مختلف را برای انجام عملی مشابه نشان می دهند.

```

MOV SI,OFFSET DATA ;advantage: executes faster
LEA SI,DATA

```

### LES ثبات قطعه اضافی را بار کن

پرچمها: بدون تغییر

قالب: LES dest, source; load dest and ES starting at source

عملکرد: محتوای دو مکان حافظه مشخص شده بوسیله مبدأ در مقصد (یک ثبات) بار می شود و نیز ES را با محتوای دو مکان حافظه بعدی بار می نماید. برای دستیابی به یک قطعه جدید و تفاوت مکان آن مفید است. این دستور مشابه LDS است با این تفاوت که ES و تفاوت مکان آن بار شده است.

### LOCK پیشوند قفل گذرگاه سیستم

پرچمها: بدون تغییر

قالب: LOCK; used as a prefix before instruction

عملکرد: در ریز کامپیوترهایی که بیش از یک پردازنده دارند برای ممانعت از بدست گرفتن کنترل گذرگاه سیستم در حین اجرای یک دستور توسط پردازنده دیگر بکار می رود.

### LODS/LODSB /LODSW بار کردن بایت یا رشته کلمه

پرچمها: بدون تغییر

قالب: LODSx

عملکرد: AL یا AX را با یک بایت یا یک کلمه از مکان حافظه ای که با DS:SI مشخص شده بار می کند و سپس SI برای اشاره به مکان بعدی افزایش می یابد به شرطی که DF=0 باشد. اگر DF=1 باشد آنگاه SI یک واحد کم شده و به مکان بعدی اشاره خواهد داشت. بسته به اینکه یک بایت و یا رشته کلمه بار شود، 1 یا 2 واحد از SI افزایش یا کاهش می یابد.

### LOOP حلقه تا CX=0

پرچمها: بدون تغییر

قالب: LOOPx target; DEC CX, jump to target if CX not 0

عملکرد: CX را یک واحد کم می کند، سپس به تفاوت مکان مشخص شده بوسیله عملوند پرش می نماید به شرطی که CX صفر نباشد، در غیر اینصورت با دستور بعد از LOOP ادامه می دهد. این دستور معادل است با:

```

DEC CX
JNZ target

```

### LOOPE/LOOPNZ حلقه اگر مساوی یا نتیجه صفر

پرچمها: بدون تغییر

**قالب :** LOOPx target : DEC CX, jump to target if CX≠0 and ZF=1

**عملکرد :** CX را یک واحد کم می‌نماید، سپس به مکان مشخص شده با عملوند، اگر ZF=1 CX≠0 باشد پرش می‌کند، در غیر اینصورت با دستور بعد از LOOP ادامه می‌دهد. به بیان دیگر فقط هنگامی از حلقه خارج می‌شود که CX برابر صفر و یا ZF=0 باشد.

**مثال :** فرض کنید که تعداد 200H مکان حافظه از تفاوت مکان 1680H باید حاوی 55H باشند. می‌خواهیم با استفاده از LOOPE ببینیم کدام یک از این مکان‌ها حاوی 55H نیست.

|           |          |                                                                                                       |
|-----------|----------|-------------------------------------------------------------------------------------------------------|
| MOV       | CX,200   | ;SET UP THE COUNTER                                                                                   |
| MOV       | SI,1680H | ;SET UP THE POINTER                                                                                   |
| BACK: CMP | [SI],55H | ;COMPARE THE 55H WITH MEM LOCATION<br>POINTED AT BY SI                                                |
| INC       | SI       | ;INCREMENT THE POINTER                                                                                |
| LOOPE     | BACK     | ;CONTINUE THE PROCESS UNTIL CX=0 OR<br>ZF=0. IN OTHER WORDS EXIT IF ONE<br>LOCATION DOES NOT HAVE 55H |

### LOOPNE/LOOPNZ حلقه اگر CX غیر صفر و ZF مساوی صفر

**پرچم‌ها :** بدون تغییر

**قالب :** LOOPxx target : DEC CX, then jump if CX and ZF not zero and ZF=0

**عملکرد :** CX را یک واحد کاهش می‌دهد، سپس به مکانی که با عملوند مشخص شده پرش می‌نماید، به شرطی که CX صفر نباشد و ZF صفر باشد در غیر اینصورت با دستور بعد از حلقه ادامه می‌دهد. به بیان دیگر هنگامی از حلقه خارج می‌شود که CX=0 یا ZF=1 باشد.

**مثال :** فرض کنید که دمای روز برای 30 روز آخر در مکان‌های حافظه‌ای با تفاوت مکان 1200H ذخیره شده باشد. LOOPE می‌تواند برای یافتن اولین روز که دمای 90 درجه دارد بکار رود.

|            |          |                     |
|------------|----------|---------------------|
| MOV        | CX,30    | ;SET UP THE COUNTER |
| MOV        | DI,1200H | ;SET UP THE POINTER |
| AGAIN: CMP | [DI],90  |                     |
| INC        | DI       |                     |
| LOOPNE     | AGAIN    |                     |

### MOV جابجایی

**پرچم‌ها :** بدون تغییر

**قالب :** MOV dest, source ; copy source to dest

**عملکرد :** کلمه یا بایتی را از یک ثبات، مکان حافظه، یا عدد بلافاصل (فوری) به یک ثبات یا مکان حافظه دیگر منتقل می‌نماید. مبدأ و مقصد باید هم اندازه باشند و هر دو نمی‌توانند مکان حافظه باشند.

### MOVS/MOVSX/MOVSX جابجایی بایت یا رشته کلمه

**پرچم‌ها :** بدون تغییر

**قالب :** MOVsx :

**عملکرد :** بایت یا کلمه را از مکان حافظه اشاره شده با DS:SI به مکان حافظه اشاره شده با ES:DI جابجا می‌نماید. اگر DF=0 باشد، هر دو اشاره‌گر افزایش می‌یابند، در غیر اینصورت، کاهش می‌یابند. SI و DI به میزان یک یا دو واحد، بسته به اینکه بایت یا رشته کلمه باشد، افزایش/کاهش می‌یابند. هنگامی که با پیشوند REP بکار رود، CX هر بار تا صفر شدن کاهش می‌یابد.

## MUL ضرب بدون علامت

پرچم‌ها: متأثر: CF, OF, PF, AF, ZF, SF. نامشخص: AF

قالب: MUL source; AX = source × AL or DX; AX = source

عملکرد: یک بایت یا کلمه بدون علامت را در بایت یا کلمه‌ای بدون علامت در AL یا AX ضرب کرده و نتیجه را در AX یا DX ذخیره می‌کند.

## NEG (Negate) متمم دو

پرچم‌ها: متأثر: CF, PF, AF, ZF, SF, OF

قالب: NEG dest; negates operand

عملکرد: متمم 2 عملوند را بدست می‌آورد. بطور مؤثری بیت علامت عملوند را عوض می‌کند. این دستورالعمل فقط باید با اعداد علامت‌دار بکار برده شود.

## NOP هیچکار

پرچم‌ها: بدون تغییر

قالب: NOP

عملکرد: هیچ کاری انجام نمی‌دهد. گاهی اوقات برای تنظیم تأخیرها در اتلاف پالس‌های ساعت بکار برده می‌شود. IP را به هنگام می‌کند تا به دستور بعد از NOP اشاره شود.

## NOT NOT منطقی

پرچم‌ها: بدون تغییر

قالب: NOT dest = 1's complement of dest

عملکرد: عملوند را با متمم 1 جایگزین می‌کند. هر بیت معکوس می‌شود.

## OR OR منطقی

پرچم‌ها: متأثر: PF, ZF, SF, OF=0, CF=0

نامشخص: AF

قالب: OR dest, source; dest = dest OR source

عملکرد: OR منطقی را روی بیت‌های دو عملوند اجرا می‌کند و عملوند مقصد را با نتیجه جایگزین می‌نماید. اغلب برای 1 کردن یک بیت بکار می‌رود.

| X | Y | X OR Y |
|---|---|--------|
| 0 | 0 | 0      |
| 0 | 1 | 1      |
| 1 | 0 | 1      |
| 1 | 1 | 1      |

## OUT خروج بایت یا کلمه

پرچم‌ها: بدون تغییر

قالب: OUT dest, acc; transfer acc to port dest

عملکرد: یک بایت یا کلمه را از AL یا AX به یک پورت خروجی مشخص شده با اولین عملوند منتقل می‌سازد.

۱ - مستقیم: آدرس پورت مستقیماً مشخص شده و نمی‌تواند بزرگتر از FFH باشد.

### مثال ۱:

```
OUT 68H,AL ;SEND OUT A BYTE FROM AL TO PORT 68H
یا
OUT 34H,AX;SEND OUT A WORD FROM AX TO PORT
;ADDRESSES 34H AND 35H. THE BYTE
;FROM AL GOES TO PORT 34H AND
;THE BYTE FROM AH GOES TO PORT 35H
```

۲ - غیرمستقیم ثباتی: آدرس پورت بوسیله ثبات DX مشخص می‌گردد. بنابراین می‌تواند تا FFFFH

باشد.

### مثال ۲:

```
MOV DX,64B1H ;DX=64B1H
OUT DX,AL ;SENT OUT THE BYTE IN AL TO THE PORT
;WHOSE ADDRESS IS POINTED TO BY DX
or
OUT DX,AX ;SEND OUT A WORD FROM AX TO PORT
;ADDRESS POINTED TO DX. THE BYTE
;FROM AL GOES TO PORT DX AND AND BYTE
;FROM AH GOES TO PORT DX+1.
```

## POP خارج کردن کلمه

پرچم‌ها: بدون تغییر

قالب: POP dest; dest = word off top of stack

عملکرد: کلمه اشاره شده بوسیله اشاره گر پشته را در ثبات یا مکان حافظه مشخص شده توسط عملوند کپی می‌نماید و SP را 2 واحد افزایش می‌دهد.

## POPF خارج کردن پرچم‌ها از پشته

پرچم‌ها: OF, DF, IF, TF, SF, ZF, AF, PF, CF

قالب: POPF

عملکرد: بیت‌هایی که قبلاً با دستور PUSHF در پشته درج شده‌اند را در ثبات پرچم قرار می‌دهد. سپس اشاره گر 2 واحد اضافه می‌شود.

## PUSH درج کلمه

پرچم‌ها: بدون تغییر

قالب: PUSH source; PUSH source onto stack

عملکرد: کلمه مبدأ را در پشته درج و SP را 2 واحد کم می‌کند.

## PUSHF درج پرچم‌ها در پشته

پرچم‌ها: بدون تغییر

قالب: PUSHF

عملکرد: SP را 2 واحد کم کرده و محتوای ثبات پرچم را در پشته درج می‌نماید.

## RCL/RCR چرخش به چپ از طریق نقلی و چرخش به راست از طریق نقلی

پرچمها: بدون تغییر

قالب:  $PCx\ dest ; dest = dest\ rotate\ right/left$

عملکرد: بیت‌های عملوند را به چپ یا راست می‌چرخانند. بیت‌های چرخیده به خارج عملوند به داخل CF می‌چرخند و CF نیز به انتهای مقابل کلمه یا بایت می‌چرخد. توجه: "n" باید 1 یا CL باشد.

## RET برگشت از رویه

پرچمها: بدون تغییر

قالب:  $RET [n] ; return\ from\ procedure$

عملکرد: برای بازگشت از رویه‌ای که قبلاً بوسیله دستور CALL به آن وارد شده بکار می‌رود. IP از پشته بارگیری می‌شود و پشته 2 واحد افزایش می‌یابد. اگر رویه FAR باشد، RETF (بازگشت دور) بکار می‌رود، و علاوه بر IP و CS از پشته باز شده و مجدداً SP دو واحد افزایش می‌یابد. دستور RET می‌تواند با عددی دنبال باشد که پس از افزایش SP با آن جمع شود. این برای گذر از هر پارامتری است که به قطعه برنامه فراخوان محول شده باشد.

## ROL/ROP چرخش به چپ و چرخش به راست

پرچمها: متأثر: CF ، OF

قالب:  $ROX\ dest , n ; rotate\ dest\ right/left\ n\ bit\ position$

عملکرد: بیت‌های یک کلمه یا بایت را به انتهای مقابل کلمه یا بایت می‌چرخانند. توجه: "n" باید 1 یا CL باشد.

## AH SAHF را در ثبات پرچم ذخیره کن

پرچمها: متأثر: SF ، ZF ، AF ، PF ، CF نامشخص: AF

قالب: SAHF

عملکرد: AH را در 8 بیت پائین رتبه ثبات پرچم کپی می‌نماید.

## SAL/SAR جابجایی حسابی چپ/جابجایی حسابی راست

پرچمها: متأثر: PF ، ZF ، SF ، OF

قالب:  $SAX\ dest , n ; shift\ signed\ dest\ left/right\ n\ bit\ position$

عملکرد: یک کلمه یا بایت را به چپ یا راست جابجا می‌کند. جابجایی SAR/SAL حسابی برای اعداد علامت دار بکار می‌رود. در SAL، ضمن جابجایی بیت به بیت عملوند، LSB با 0ها پر می‌شود و MSB در CF کپی می‌گردد، در SAR، یا نقل مکان به راست هر بیت، LSB در CF کپی و بیت تخلیه شده با بیت علامت (MSB) پر می‌شود. اساساً SAL/SAR مقصد را به ازای بیت نقل مکان در توانی از 2 ضرب/تقسیم می‌نماید. توجه: "n" باید 1 یا CL باشد.

## SBB تفریق با قرض

پرچمها: متأثر: CF ، PF ، AF ، ZF ، SF ، OF

قالب:  $SBB\ dest , source ; dest = dest - source$

عملکرد: عملوند مبدأ را از مقصد کم کرده و نتیجه را در مقصد جایگزین می‌کند. اگر  $CF=1$  باشد 1 را از

نتیجه کسر می نماید در غیر اینصورت مثل SUB عمل خواهد کرد.

### SCAS/SCASB/SCASW پویش بایت یا رشته کلمه

پرچمها: متأثر: CF, PF, AF, ZF, SF, OF

قالب: SCASx

عملکرد: یک رشته از داده را که بوسیله ES:DI به آنها اشاره می شود و برای مقداری که در AL یا AX قرار دارد جستجو می کند اغلب با پیشوند REPE/REPNE بکار می رود. اگر DF صفر باشد، آدرس اضافه می شود، در غیر اینصورت کاهش می یابد.

### SHL/SHR جابجایی چپ/جابجایی راست

پرچمها: متأثر: CF, PF, ZF, SF, OF نامشخص: AF

قالب: SHx dest, n; shift unsigned dest left/right n bit position

عملکرد: اینها دستورات جابجایی منطقی هستند که برای اعداد بدون علامت بکار می روند، بدین معنی که با بیت علامت هم مثل داده برخورد می شود. در SHR ضمن نقل مکان بیت به بیت عملوند به سمت راست و کپی شدن آن در CF، بیت های خالی، در عوض بیت علامت در SAR، با 0ها پر می شوند. در SHL با نقل مکان یا شیفت به چپ بیت ها، MSB در CF کپی شده و بیت های خالی با 0 پر می شوند که دقیقاً مثل SAL است. در واقع SAL، SHL دو نماد مختلف برای یک کد عمل می باشند. SHL/SHR در اصل ضرب/تقسیم مقصد را در توانی از 2 برای هر نقل مکان انجام می دهند.  
توجه: "n" باید 1 یا CL باشد.

### STC نشاندن پرچم نقلی

پرچمها: متأثر: CF

قالب: STC

عملکرد: نشاندن CF در 1

### STD نشاندن پرچم جهت

پرچمها: متأثر: DF

قالب: STD

عملکرد: DF را در 1 می نشانند. بطور گسترده ای با دستورالعمل های رشته ای بکار می رود. همانطور که در دستورالعمل های رشته بیان شده است، اگر DF=1 باشد، اشاره گر ها کاهش می یابند.

### STI نشاندن پرچم وقفه

پرچمها: متأثر: IF

قالب: STI

عملکرد: IF را در 1 می نشانند، و به وقفه سخت افزاری تشخیص وقفه را از طریق پایه INTR در CPU اجازه می دهد.

### STOS/STOSB/STOSW ذخیره بایت یا رشته کلمه

پرچمها: بدون تغییر

### قالب STOSx :

**عملکرد :** یک بایت را از AX یا AL به مکانی که بوسیله ES:DI اشاره می شود کپی کرده و DI را برای اشاره به عنصر بعدی بهنگام می نماید. اگر DF=0 باشد اشاره گر DI اضافه می شود، در غیر این صورت کاهش می یابد.

### SUB تفریق

**پرچم ها :** متاثر: OF ، SF ، ZF ، AF ، PF ، CF

**قالب :** SUB dest, source ; dest = dest - source

**عملکرد :** مبدأ را از مقصد کم کرده و نتیجه را در مقصد قرار می دهد. رقم نقلی و پرچم صفر را بر طبق زیر می نشانند:

|             | ZF | CF |                                |
|-------------|----|----|--------------------------------|
| مبدأ > مقصد | 0  | 0  | نتیجه مثبت است                 |
| مبدأ = مقصد | 0  | 1  | نتیجه صفر است                  |
| مبدأ < مقصد | 1  | 0  | نتیجه منفی و به شکل متمم 2 است |

مراحل تفریق که در سخت افزار داخلی CPU انجام می شود به شرح زیر است.

- ۱ - متمم 2 مبدأ را بدست می آورد.
  - ۲ - آن را به مقصد اضافه می کند.
  - ۳ - نقلی را معکوس و پرچم ها را بر طبق آن تغییر می دهد.
- با این دستور عملوند مبدأ بدون تغییر باقی می ماند.

### TEST تست بیت ها

**پرچم ها :** متاثر: OF ، SF ، ZF ، PF ، CF. نامشخص: AF

**قالب :** TEST dest, source; performs dest AND source

**عملکرد :** AND منطقی را روی دو عملوند انجام می دهد، پرچم ها را تنظیم می کند ولی محتوای مبدأ و مقصد بدون تغییر می مانند. در حالی که دستور AND محتوای مقصد و بیت های پرچم را عوض می کرد، دستور TEST فقط بیت های پرچم را عوض می کند.

**مثال :** فرض کنید که D0 و D1 از پورت 27 شرایط A و B را به ترتیب مشخص کنند که تحت شرایط فوق هر دو آنها در سطح بالا و یا فقط یکی از آنها در سطح بالاست. دستور به طریق زیر بکار می رود.

```
IN AL,PORT_27
TEST AL,0000 0001B ;CHECK THE CONDITION A
JNZ CASE_A ;JUMP TO INDICATE CONDITION A
TEST AL,0000 0010B ;CHECK FOR CONDITION B
JNZ CASE_B ;JUMP TO INDICATE CONDITION B
.... ;THERE IS AN ERROR SINCE NEITHER
.... ; A OR B HAS OCCURRED.
CASE_A:
....
CASE_B:
```



## WAIT بردن پردازنده به حالت انتظار

پرچمها: بدون تغییر

قالب: WAIT

عملکرد: موجب می شود تا ریزپردازنده وارد یک حالت بیکار شود تا یک وقفه خارجی رخ دهد. این اغلب به خاطر همگام کردن با پردازنده ای دیگر و یا یک وسیله خارجی است.

## XCHG تعویض

پرچمها: بدون تغییر

قالب: XCHG dest, source ; swaps dest and source

عملکرد: محتوای دو ثبات یا یک ثبات و یک مکان حافظه را با هم عوض می کند.

## XLAT ترجمه

پرچمها: بدون تغییر

قالب: XLAT

عملکرد: محتوای AL را با محتوای جدول نظاره (نگرش) که آدرسش با AL داده می شود، عوض می کند. BX باید با آدرس شروع جدول نظاره بار شده و عنصر مورد ترجمه باید قبل از اجرای این دستور در AL باشد. AL به عنوان تفاوت مکان در جدول تبدیل بکار می رود. اغلب برای ترجمه داده از یک قالب به قالب دیگر، مانند ASCII به EBCDIC بکار می رود.

## OR XOR انحصاری

پرچمها: متأثر: PF, ZF, SF, OF = 0, CF = 0

نامشخص: AF

قالب: XOR dest, source

عملکرد: OR انحصاری منطقی را روی بیت های دو عملوند انجام داده و نتیجه را در مقصد قرار می دهد. "XOR AX,AX" موجب پاک شدن AX می گردد.

| X | Y | X XOR Y |
|---|---|---------|
| 0 | 0 | 0       |
| 0 | 1 | 1       |
| 1 | 0 | 1       |
| 1 | 1 | 0       |

## بخش ۲ - ب: زمان بندی دستورالعمل

در این بخش از پیوست، تعداد پالس های ساعت را برای همه دستورالعمل های ریزپردازنده های 8086، 286، 386، 486 اینتل فراهم می کنیم. تعداد پالس به این معنی است که یک دستور چند عدد پالس برای اجرا لازم دارد. این اطلاعات از کتابخانه راهنمای اینتل استخراج شده است. تعداد پالس های ساعت برای هر دستور با این فرض ذکر شده است که قبلاً دستور وارد CPU شده است. تعداد پالس های ساعت می تواند برحسب طراحی سخت افزار حافظه سیستم متغیر باشد. وقتی که تعداد پالس ها را برای یک CPU مفروض محاسبه می کنید به نکات زیر توجه نمایید.

۱ - در محاسبه سیکل های ساعت کل برای 8086/88 باید تعداد پالس اضافی مربوط به آدرس مؤثر EA را که در جدول ۳ - ب آمده است به آن افزود.

۲ - در محاسبه زمان لازم برای ریزپردازنده های 8086، 286 و 386SX، گذرگاه داده خارجی باید به حساب آید. بعلاوه اینکه آدرس آیا آدرس عملوند زوج است یا فرد. برای کاهش زمان لازم جهت برداشت داده از حافظه این

CPU، نیاز به قرارگرفتن داده در مرز آدرس زوج دارند. اگر آدرس‌ها روی مرزهای زوج نباشند، یک جریمه 4 پالس ساعتی اضافی برای برداشت عملوند 16 بیتی در نظر گرفته می‌شود. به مثال‌های زیر با فرض  $DS = 25000H$  و  $BX = 3000H$  نگاه کنید.

`MOV AX,[BX]` ; total clocks = 10+5

چون آدرس فیزیکی  $25000H + 3000H = 28000H$  است، و یک آدرس زوج می‌باشد، و عرض گذرگاه داده در 8086 دارای عرض 16 بیتی است، محتوای مکان‌های حافظه  $28000H$  و  $28001H$  به داخل CPU در یک سیکل حافظه برده می‌شوند. در تمام ریزپردازنده‌های 8086 بایت پائین رتبه به آدرس پائین و بایت بالا رتبه به آدرس بالا می‌رود. در مثال فوق محتوای مکان حافظه  $28000H$  به ثبات AL و  $28001H$  به AH می‌رود. اگر  $BX = 3005H$  باشد، آدرس فیزیکی  $25000H + 3005H = 28005H$  خواهد شد که یک آدرس فرد است و پالس ساعت لازم به دلیل 4 پالس جریمه برای داده نامنطبق برابر است با:

`MOV AX,[BX]` ; total clocks = 10+4+5

در دستورالعمل فوق، محتوای مکان‌های حافظه  $28005H$  به AL و  $28006H$  به AH جابجا می‌شوند. در واقع روشی که 8086 بر اساس آن حافظه را دستیابی می‌کند به این ترتیب است که در اولین سیکل حافظه، 16 بیت داده را از  $28004H$  و  $28005H$  روی گذرگاه داده D0-D15 قرار داده و سپس 16 بیت داده از مکان‌های  $28006H$  و  $28007H$  را در دومین سیکل حافظه با استفاده از گذرگاه داده 16 بیتی برداشت می‌نماید. به بیان دیگر، مکان‌های حافظه  $28004H$ ،  $28005H$ ،  $28006H$  و  $28007H$  استفاده شده‌اند ولی محتوای مکان‌های  $28004H$  و  $28007H$  چشم‌پوشی شده‌اند. به این دلیل داده باید کلمه (16 بیتی) منطبق بر گذرگاه داده 16 بیتی ریزپردازنده باشد. اگر آدرس در 8086 فرد باشد چه می‌شود؟ باز هم فقط یک سیکل حافظه متشکل از 4 پالس ساعت طول می‌کشد. مثلاً در `MOV AH,[BX]`، با  $BX = 3005H$  و  $DS = 25000H$ ، محتوای مکان‌های حافظه  $28004H$  و  $28005H$  در یک سیکل حافظه دستیابی می‌شوند، ولی فقط محتوای آدرس  $28005H$  به ثبات AH برده می‌شود.

۳- در ریزپردازنده 8088، زمان لازم برای اجرای یک دستورالعمل می‌تواند نسبت به 8086 متفاوت باشد زیرا گذرگاه داده در 8088 فقط 8 بیتی است. بنابراین عملوند 16 بیتی نیاز به دو سیکل حافظه (هر یک 4 سیکل ساعت) خواهد داشت تا عملوند را به داخل یا خارج ریزپردازنده بفرستد. مثلاً

`MOV AX,[BX]` ; total clocks = 10+5

`MOV AX,[BX]` ; total clocks = 10+4+5

۴- برای پرش‌های شرطی و دستورات LOOP، اولین عدد، تعداد پالس‌های ساعت است به شرطی که پرش موفقیت آمیز باشد (انجام شود) و دومین عدد برای هنگامی است که پرش صورت نمی‌گیرد (noj = no Jump). مثلاً ستون 8086 برای دستور JNZ دارای "16, noj4" برای تعداد پالس ساعت است. 16 تعداد پالس‌های ساعت برای حالتی است که پرش رخ دهد. اگر پرشی بوقوع نپیوندد، تعداد پالس ساعت 4 است.

۵- تعداد پالس‌های ساعت برای SX 80386 یکسان است به جز اینکه برای دستیابی به عملوندهای 32 بیتی، 2 پالس ساعت باید اضافه شود زیرا گذرگاه داده در SX 80386، 16 بیت و زمان سیکل حافظه در آن 2 پالس ساعت است.

۶- اگر عملوند 16 بیتی هم‌ردیف نباشد، دو پالس ساعت اضافی برای 80286 و SX 80386 لازم است و نیز برای 386 هم اگر عملوند 32 بیت در مرز 32 بیت هم‌ردیف نباشد دو پالس مذکور مورد نیاز است. بحث شماره ۲ در فوق ملاحظه شود.

۷- تعداد پالس‌های ساعت مفروض برای 80486 برای مواردی است که عملوند در حافظه نهان تراشه 486 است، در

غیراینصورت پالس‌های اضافی باید به عنوان جبران نیافتن در حافظه نهان لحاظ شود. برای لیستی از جریمه‌های نیافتن به کتاب مرجع برنامه‌نویسی ریزپردازنده 486 مراجعه شود.

۸- زمان‌بندی دستور PM (مد انحصاری) برای مواردی است که CPU به وضعیت حفاظت شده سوئیچ کرده باشد.

۹- "m" (اغلب در دستورات 286 و 386 دیده شده است) تعداد مؤلفه‌های مربوط به دستور مورد اجرای بعدی است. مقدار m متغیر است، زیرا اندازه دستور واقع در آدرس مقصد می‌تواند متغیر باشد. عملاً m می‌تواند متوسط 2 را داشته باشد.

۱۰- "n" نمایشگر تعداد تکرار یک دستور مفروض است.

۱۱- به دلیل پیشرفت دائمی ریزپردازنده‌های 286/386/486، تعداد کل پالس‌های ساعت نمی‌تواند در یک برنامه مفروض 100% صحیح باشد. به این دلیل، مرز خطای 10%، هنگام محاسبات تعداد پالس ساعت کل، باید در یک برنامه مفروض لحاظ شود.

۱۲- با هر نسل جدید 80x86، دستورالعمل‌های جدید اضافه می‌شوند، بنابراین برای این دستورات جدید تعداد پالسی در نسل‌های قبلی وجود ندارد. این نکته با "--" نشان داده شده است.

جدول ۳- ب سیکل‌های ساعت برای آدرس مؤثر

| CLK | عملوند        | روش آدرس دهی     |
|-----|---------------|------------------|
| 6   | label         | مستقیم           |
| 5   | [BX]          | غیرمستقیم ثباتی  |
| 5   | [SI]          |                  |
| 5   | [DI]          |                  |
| 5   | [BP]          |                  |
| 9   | [BX]+disp     | پایه نسبی        |
| 9   | [BP]+disp     |                  |
| 9   | [DI]+disp     | اندیسی نسبی      |
| 9   | [SI]+disp     |                  |
| 7   | [BX][SI]      | پایه اندیسی      |
| 7   | [BX][DI]      |                  |
| 8   | [BP][SI]      |                  |
| 8   | [BP][DI]      |                  |
| 11  | [BX][SI]+disp | پایه اندیسی نسبی |
| 11  | [BX][DI]+disp |                  |
| 12  | [BP][SI]+disp |                  |
| 12  | [BP][DI]+disp |                  |

توجه:

در این زمان‌ها لغو قطعه فرض نشده است. اگر لغو قطعه بکار رفت 2 سیکل ساعت باید اضافه شود. خلاصه‌ای از سیکل‌های ساعت برای دستورات ریزپردازنده‌های مختلف اینتل در جدول ۴- ب داده شده است.

| <u>Code</u>         | <u>Description</u>                                      | <u>8086</u> | <u>80286</u> | <u>80386</u> | <u>80486</u> |
|---------------------|---------------------------------------------------------|-------------|--------------|--------------|--------------|
| AAA                 | ASCII adjust for addition                               | 8           | 3            | 4            | 3            |
| AAD                 | ASCII adjust for division                               | 60          | 14           | 19           | 14           |
| AAM                 | ASCII adjust for multiplication                         | 83          | 16           | 17           | 15           |
| AAS                 | ASCII adjust for subtraction                            | 8           | 3            | 4            | 3            |
| ADC                 | Add with carry                                          |             |              |              |              |
|                     | reg to reg                                              | 3           | 2            | 2            | 1            |
|                     | mem to reg                                              | 9+EA        | 7            | 6            | 2            |
|                     | reg to mem                                              | 16+EA       | 7            | 7            | 3            |
|                     | immed to reg                                            | 4           | 3            | 2            | 1            |
|                     | immed to mem                                            | 17+EA       | 7            | 7            | 3            |
|                     | immed to acc                                            | 4           | 3            | 2            | 1            |
| ADD                 | Addition                                                |             |              |              |              |
|                     | reg to reg                                              | 3           | 2            | 2            | 1            |
|                     | mem to reg                                              | 9+EA        | 7            | 6            | 2            |
|                     | reg to mem                                              | 16+EA       | 7            | 7            | 3            |
|                     | immed to reg                                            | 4           | 3            | 2            | 1            |
|                     | immed to mem                                            | 17+EA       | 7            | 7            | 3            |
|                     | immed to acc                                            | 4           | 3            | 2            | 1            |
| AND                 | Logical AND                                             |             |              |              |              |
|                     | reg to reg                                              | 3           | 2            | 2            | 1            |
|                     | mem to reg                                              | 9+EA        | 7            | 6            | 2            |
|                     | reg to mem                                              | 16+EA       | 7            | 7            | 3            |
|                     | immed to reg                                            | 4           | 3            | 2            | 1            |
|                     | immed to mem                                            | 17+EA       | 7            | 7            | 3            |
|                     | immed to acc                                            | 4           | 3            | 2            | 1            |
| ARPL                | Adjust RPL (requested<br>privilege level)               |             |              |              |              |
|                     | reg to reg                                              | --          | 10           | 20           | 9            |
|                     | reg to mem                                              | --          | 11           | 21           | 9            |
| BOUND               | Check array bounds                                      | --          | 13noj        | 10noj        | 7noj         |
| BSF                 | Bit scan forward                                        |             |              |              |              |
|                     | reg to reg                                              | --          | --           | 10+3n        | 6/42         |
|                     | mem to reg                                              | --          | --           | 10+3n        | 7/43         |
| BSR                 | Bit scan reverse                                        |             |              |              |              |
|                     | reg to reg                                              | --          | --           | 10+3n        | 6/103        |
|                     | mem to reg                                              | --          | --           | 10+3n        | 7/104        |
| BSWAP               | Byte swap                                               | --          | --           | --           | 1            |
| BT                  | Bit test                                                |             |              |              |              |
|                     | reg to reg                                              | --          | --           | 3            | 3            |
|                     | reg to mem                                              | --          | --           | 12           | 8            |
|                     | immed to reg                                            | --          | --           | 3            | 3            |
|                     | immed to mem                                            | --          | --           | 6            | 3            |
| BTC/<br>BTR/<br>BTS | Bit test complement/<br>Bit test reset/<br>Bit test set |             |              |              |              |
|                     | reg to reg                                              | --          | --           | 6            | 6            |
|                     | reg to mem                                              | --          | --           | 13           | 13           |
|                     | immed to reg                                            | --          | --           | 6            | 6            |
|                     | immed to mem                                            | --          | --           | 8            | 8            |

| Code    | Description                                | 8086     | 80286   | 80386   | 80486 |
|---------|--------------------------------------------|----------|---------|---------|-------|
| CALL    | Call a procedure                           |          |         |         |       |
|         | intra-segment direct                       | 19       | 7+m     | 7+m     | 3     |
|         | intra-segment indirect through register    | 16       | 7+m     | 7+m     | 5     |
|         | intra-segment indirect through memory      | 21+EA    | 11+m    | 10+m    | 5     |
|         | inter-segment direct                       | 28       | 13+m    | 17+m    | 18    |
|         | 486: to same level                         |          |         |         | 20    |
|         | 486: thru Gate to same level               |          |         |         | 35    |
|         | 486: to inner level, no parameters         |          |         |         | 69    |
|         | 486: to inner level, x parameter (d) words |          |         |         | 77+4x |
|         | 486: to TSS                                |          |         |         | 37+TS |
|         | 486: thru Task Gate                        |          |         |         | 38+TS |
|         | inter-segment direct PM                    | --       | 26+m    | 34+m    |       |
|         | inter-segment indirect                     | 37+EA    | 16+m    | 22+m    | 17    |
|         | 486: to same level                         |          |         |         | 20    |
|         | 486: thru Gate to same level               |          |         |         | 35    |
|         | 486: to inner level, no parameters         |          |         |         | 69    |
|         | 486: to inner level, x parameter (d) words |          |         |         | 77+4x |
|         | 486: to TSS                                |          |         |         | 37+TS |
|         | 486: thru Task Gate                        |          |         |         | 38+TS |
|         | inter-segment indirect PM                  | --       | 29+m    | 38+m    |       |
| CBW     | Convert byte to word                       | 2        | 2       | 3       | 3     |
| CDQ     | Convert double to quad                     | --       | --      | 2       |       |
| CLC     | Clear carry flag                           | 2        | 2       | 2       | 2     |
| CLD     | Clear direction flag                       | 2        | 2       | 2       | 2     |
| CLI     | Clear interrupt flag                       | 2        | 3       | 3       | 5     |
| CLTS    | Clear task switched flag                   | --       | 2       | 5       | 7     |
| CMC     | Complement carry flag                      | 2        | 2       | 2       | 2     |
| CMP     | Compare                                    |          |         |         |       |
|         | reg to reg                                 | 3        | 2       | 2       | 1     |
|         | mem to reg                                 | 9+EA     | 6       | 6       | 2     |
|         | reg to mem                                 | 9+EA     | 7       | 5       | 2     |
|         | immed to reg                               | 4        | 3       | 2       | 1     |
|         | immed to mem                               | 10+EA    | 6       | 5       | 2     |
|         | immed to acc                               | 4        | 3       | 2       | 1     |
| CMPS/   | Compare string/                            |          |         |         |       |
| CMPSB/  | Compare byte string/                       |          |         |         |       |
| CMPSW   | Compare word string                        |          |         |         |       |
|         | not repeated                               | 22       | 8       | 10      | 8     |
|         | REPE/REPNE CMPS/CMPSB/CMPSW                | 9+22/rep | 5+9/rep | 5+9/rep | 7+7c  |
| CMPXCHG | Compare and exchange                       |          |         |         |       |
|         | reg with reg                               | --       | --      | --      | 6     |
|         | reg with mem                               | --       | --      | --      | 7/10  |
| CWD     | Convert word to doubleword                 | 5        | 2       | 2       | 3     |
| CWDE    | Convert word to extended double            | --       | --      | 3       | 3     |
| DAA     | Decimal adjust for addition                | 4        | 3       | 4       | 2     |
| DAS     | Decimal adjust for subtraction             | 4        | 3       | 4       | 2     |

| Code  | Description                 | 8086         | 80286     | 80386     | 80486 |
|-------|-----------------------------|--------------|-----------|-----------|-------|
| DEC   | Decrement by 1              |              |           |           |       |
|       | 16-bit reg                  | 3            | 2         | 2         | 1     |
|       | 8-bit reg                   | 3            | 2         | 2         | 1     |
|       | memory                      | 15+EA        | 7         | 6         | 3     |
| DIV   | Unsigned division           |              |           |           |       |
|       | 8-bit reg                   | 80-90        | 14        | 14        | 16    |
|       | 16-bit reg                  | 144-162      | 22        | 22        | 24    |
|       | double                      | --           | --        | --        | 40    |
|       | 8-bit mem                   | (86-96)+EA   | 17        | 17        | 16    |
|       | 16-bit mem                  | (150-168)+EA | 25        | 25        | 24    |
|       | double                      | --           | --        | --        | 40    |
| ENTER | Make stack frame            |              |           |           |       |
|       | W,0                         | --           | 11        | 10        | 14    |
|       | W,1                         | --           | 15        | 12        | 17    |
|       | dw,db                       | --           | 12+4(n-1) | 15+4(n-1) | 17+3n |
| ESC   | Escape                      |              |           |           |       |
|       | reg                         | 2            | 9-20      | varies    |       |
|       | mem                         | 8+EA         | 9-20      | varies    |       |
| HLT   | Halt                        | 2            | 2         | 5         | 4     |
| IDIV  | Integer division            |              |           |           |       |
|       | 8-bit reg                   | 101-112      | 17        | 19        | 19    |
|       | 16-bit reg                  | 165-184      | 25        | 27        | 27    |
|       | 32-bit reg                  | --           | --        | 43        | 43    |
|       | 8-bit mem                   | (107-118)+EA | 20        | 22        | 20    |
|       | 16-bit mem                  | (171-190)+EA | 28        | 30        | 28    |
|       | 32-bit reg                  | --           | --        | 46        | 44    |
| IMUL  | Integer multiplication      |              |           |           |       |
|       | 8-bit reg                   | 80-98        | 13        | 9-14      | 13-18 |
|       | 16-bit reg                  | 128-154      | 21        | 9-22      | 13-26 |
|       | 32-bit reg                  | --           | --        | 9-38      | 13-42 |
|       | 8-bit mem                   | (86-104)+EA  | 16        | 12-17     | 13-18 |
|       | 16-bit mem                  | (134-160)+EA | 24        | 12-25     | 13-26 |
|       | 32-bit reg                  | --           | --        | 12-41     | 13-42 |
|       | immed to 16-bit reg         | --           | 21        | 9-34      | 13-18 |
|       | immed to 32-bit reg         | --           | 21        | 9-38      | 13-18 |
|       | reg to reg (byte)           | --           | --        | 9-38      | 13-18 |
|       | reg to reg (word)           | --           | --        | 9-38      | 13-26 |
|       | reg to reg (dword)          | --           | --        | 9-38      | 13-42 |
|       | mem to reg (byte)           | --           | --        | 12-25     | 13-18 |
|       | mem to reg (word)           | --           | --        | 12-25     | 13-26 |
|       | mem to reg (dword)          | --           | --        | 12-41     | 13-42 |
|       | reg with imm to reg (byte)  | --           | --        | 9-14      | 13-18 |
|       | reg with imm to reg (word)  | --           | --        | 9-22      | 13-26 |
|       | reg with imm to reg (dword) | --           | --        | 9-38      | 13-42 |
|       | mem with imm to reg (byte)  | --           | --        | 12-17     | 13-18 |
|       | mem with imm to reg (word)  | --           | --        | 12-25     | 13-26 |
|       | mem with imm to reg (dword) | --           | --        | 12-41     | 13-42 |

| Code                           | Description                                                           | 8086     | 80286     | 80386        | 80486   |
|--------------------------------|-----------------------------------------------------------------------|----------|-----------|--------------|---------|
| IN                             | Input from I/O port<br>fixed port                                     | 10       | 5         | 12           | 14      |
|                                | variable port through DX                                              | 8        | 5         | 13           | 14      |
| INC                            | Increment by 1                                                        |          |           |              |         |
|                                | 16-bit reg                                                            | 3        | 2         | 2            | 1       |
|                                | 8-bit reg                                                             | 3        | 2         | 2            | 1       |
|                                | mem                                                                   | 15+EA    | 7         | 6            | 3       |
| INS/<br>INSB/<br>INSW/<br>INSD | Input from port to string<br>Input byte<br>Input word<br>Input double | --       | 5         | 15           | 17      |
|                                | PM                                                                    | --       | --        | 9,29         | 10-32   |
|                                | REP INS/INSB/INSW                                                     | --       | 5+4/rep   | 13+6/rep     |         |
|                                | REP INS/INSB/INSW PM                                                  | --       | --        | (7,27)+6/rep |         |
| INT                            | Interrupt                                                             |          |           |              |         |
|                                | type=3                                                                | 52       | 23+m      | 33           |         |
|                                | type=3 PM                                                             | --       | (40,78)+m | 59,99        |         |
|                                | type3                                                                 | 51       | 23+m      | 37           |         |
|                                | type3 PM                                                              | --       | (40,78)+m | 59,99        |         |
| INTO                           | Interrupt if overflow                                                 |          |           |              |         |
|                                | interrupt taken                                                       | 53       | 24+m      | 35           |         |
|                                | interrupt not taken                                                   | 4        | 3         | 3            |         |
|                                | PM                                                                    | --       | (40,78)+m | 59,99        |         |
| INVD                           | Invalidate data cache                                                 | --       | --        | --           | 4       |
| INVLPG                         | Invalidate TLB entry                                                  | --       | --        | --           | 12/11   |
| IRET                           | Return from interrupt                                                 | 32       | 17+m      | 22           | 15      |
|                                | PM                                                                    | --       | (31,55)+m | 38,82        | 36      |
| IRETD                          | Return from interrupt double                                          | --       | --        | 22           | 20      |
|                                | PM                                                                    | --       | --        | 38,82        | 36      |
| JA/<br>JNBE                    | Jump if above/<br>Jump if not below or equal                          | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3    | 3,noj 1 |
| JAE/<br>JNB                    | Jump if above or equal/<br>Jump if not below/                         | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3    | 3,noj 1 |
| JNA                            | Jump if not above                                                     |          |           |              |         |
| JCXZ                           | Jump if CX is zero                                                    | 18,noj 6 | 8+m,noj 4 | 9+m,noj 5    | 8,noj 5 |
| JECXZ                          | Jump if ECX is zero                                                   | --       | --        | --           | 8,noj 5 |
| JE/<br>JZ                      | Jump if equal/<br>Jump if zero                                        | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3    | 3,noj 1 |
| JG/<br>JNLE                    | Jump if greater<br>Jump if not less, or equal                         | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3    | 3,noj 1 |
| JGE/<br>JNL                    | Jump if greater or equal/<br>Jump if not less                         | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3    | 3,noj 1 |
| JL/<br>JNGE                    | Jump if less/<br>Jump if not greater, or equal                        | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3    | 3,noj 1 |
| JLE/<br>JNG                    | Jump if less or equal/<br>Jump if not greater                         | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3    | 3,noj 1 |

| Code  | Description                             | 8086     | 80286     | 80386     | 80486   |
|-------|-----------------------------------------|----------|-----------|-----------|---------|
| JMP   | Jump                                    |          |           |           |         |
|       | intra segment direct short              | 15       | 7+m       | 7+m       | 3       |
|       | intra segment direct                    | 15       | 7+m       | 7+m       | 3       |
|       | inter segment direct                    | 15       | 11+m      | 12+m      | 17      |
|       | PM                                      | --       | 23+m      | 27+m      | 18      |
|       | intra segment indirect through memory   | 18+EA    | 11+m      | 10+m      | 5       |
|       | intra segment indirect through register | 11       | 7+m       | 7+m       | 5       |
|       | inter segment indirect                  | 24+EA    | 15+m      | 12+m      | 8       |
|       | PM                                      | --       | 26+m      | 27+m      | 18      |
|       | direct inter segment                    |          |           |           | 17      |
|       | 486: to same level                      |          |           |           | 19      |
|       | 486: thru call gate to same level       |          |           |           | 32      |
|       | 486: thru TSS                           |          |           |           | 42+TS   |
|       | 486: thr Task Gate                      |          |           |           | 43+TS   |
|       | indirect inter segment                  |          |           |           | 13      |
|       | 486: to same level                      |          |           |           | 18      |
|       | 486: thru call gate to same level       |          |           |           | 31      |
|       | 486: thru TSS                           |          |           |           | 41+TS   |
|       | 486: thr Task Gate                      |          |           |           | 42+TS   |
| JNE/  | Jump if not equal/                      | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3 | 3,noj 1 |
| JNZ   | Jump if not zero                        |          |           |           |         |
| JNO   | Jump if not overflow                    | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3 | 3,noj 1 |
| JNP/  | Jump if not parity/                     | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3 | 3,noj 1 |
| JPO   | Jump if parity odd                      |          |           |           |         |
| JNS   | Jump if not sign                        | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3 | 3,noj 1 |
| JO    | Jump if overflow                        | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3 | 3,noj 1 |
| JP/   | Jump if parity/                         | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3 | 3,noj 1 |
| JPE   | Jump if parity even                     |          |           |           |         |
| JS    | Jump if sign                            | 16,noj 4 | 7+m,noj 3 | 7+m,noj 3 | 3,noj 1 |
| LAHF  | Load AH from flags                      | 4        | 2         | 2         | 3       |
| LAR   | Load access rights                      |          |           |           |         |
|       | reg to reg                              | --       | 14        | 15        | 11      |
|       | mem to reg                              | --       | 16        | 16        | 11      |
| LDS/  | Load pointer using DS/                  |          |           |           |         |
| LES   | Load pointer using ES                   | 16+EA    | 7         | 7         | 6       |
|       | PM                                      | --       | 21        | 22        | 12      |
| LFS/  | Load far pointer                        |          |           |           |         |
| LGS/  |                                         |          |           |           |         |
| LSS   |                                         | --       | --        | 7         | 6/12    |
|       | PM                                      | --       | --        | 22-25     |         |
| LEA   | Load effective address                  | 2+EA     | 3         | 2         | 2,noj 1 |
| LEAVE | High level procedure exit               | --       | 5         | 4         | 5       |
| LGDT  | Load global descriptor table            | --       | 11        | 11        | 11      |
| LIDT  | Load interrupt desc. table              | --       | 12        | 11        | 11      |
| LLDT  | Load local desc. table                  |          |           |           |         |
|       | reg                                     | --       | 17        | 20        | 11      |
|       | mem                                     | --       | 19        | 24        | 11      |
| LMSW  | Load machine status word                |          |           |           |         |
|       | reg                                     | --       | 3         | 10        | 13      |
|       | mem                                     | --       | 6         | 13        | 13      |



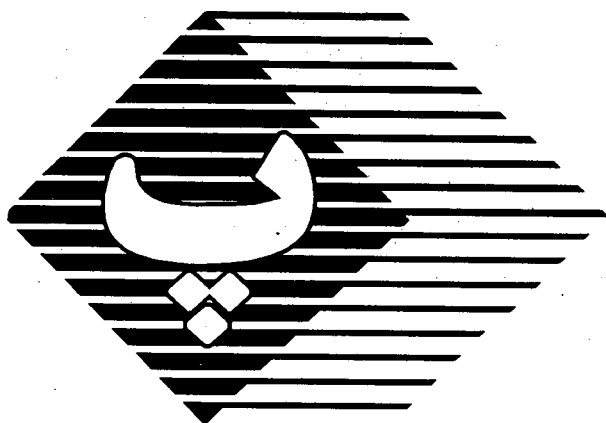
| Code    | Description           | 8086     | 80286     | 80386   | 80486    |
|---------|-----------------------|----------|-----------|---------|----------|
| LOCK    | Lock bus              | 2        | 0         | 0       | 1        |
| LODS/   | Load string/          |          |           |         |          |
| LODSB/  | Load byte string/     |          |           |         |          |
| LODSW   | Load word string      |          |           |         |          |
|         | not repeated          | 12       | 5         | 5       | 5        |
|         | repeated              | 9+13/rep |           |         | 7+4c     |
| LOOP    | Loop                  | 17,noj 5 | 8+m,noj 4 | 11+m    | 7,noj 6  |
| LOOPE/  | Loop if equal/        |          |           |         |          |
| LOOPZ   | Loop if zero          | 18,noj 6 | 8+m,noj 4 | 11+m    | 9,noj 6  |
| LOOPNE/ | Loop if not equal/    |          |           |         |          |
| LOOPNZ  | Loop if not zero      | 19,noj 5 | 8+m,noj 4 | 11+m    | 9,noj 6  |
| LSL     | Load segment limit    |          |           |         |          |
|         | reg to reg            | --       | 14        | 20,25   | 10       |
|         | mem to reg            | --       | 16        | 21,26   | 10       |
| LTR     | Load task register    |          |           |         |          |
|         | reg                   | --       | 17        | 23      | 20       |
|         | mem                   | --       | 19        | 27      | 20       |
| MOV     | Move                  |          |           |         |          |
|         | acc to mem            | 10       | 3         | 2       | 1        |
|         | mem to acc            | 10       | 5         | 4       | 1        |
|         | reg to reg            | 2        | 2         | 2       | 1        |
|         | mem to reg            | 8+EA     | 5         | 4       | 1        |
|         | reg to mem            | 9+EA     | 3         | 2       | 1        |
|         | immed to reg          | 4        | 2         | 2       | 1        |
|         | immed to mem          | 10+EA    | 3         | 2       | 1        |
|         | reg to SS/DS/ES       | 2        | 2         | 2       | 3/9      |
|         | reg to SS/DS/ES PM    | --       | 17        | 18      |          |
|         | mem to SS/DS/ES       | 8+EA     | 5         | 5       | 3/9      |
|         | mem to SS/DS/ES PM    | --       | 19        | 19      | 3        |
|         | segment reg to reg    | 2        | 2         | 2       | 3        |
|         | segment reg to mem    | 9+EA     | 3         | 2       | 3        |
|         | control reg to reg    | --       | --        | 6       | 4        |
|         | reg to control reg 0  | --       | --        | 10      | 16       |
|         | reg to control reg 2  | --       | --        | 4       | 4        |
|         | reg to control reg 3  | --       | --        | 5       | 4        |
|         | debug reg 0-3 to reg  | --       | --        | 22      | 10       |
|         | debug reg 6-7 to reg  | --       | --        | 14      | 10       |
|         | reg to debug reg 0-3  | --       | --        | 22      | 11       |
|         | reg to debug reg 6-7  | --       | --        | 16      | 11       |
|         | test reg to reg       | --       | --        | 12      | 3,4      |
|         | reg to test reg       | --       | --        | 12      | 6,4      |
| MOVS/   | Move string/          |          |           |         |          |
| MOVSB/  | Move byte string/     |          |           |         |          |
| MOVSW   | Move word string      |          |           |         |          |
|         | not repeated          | 18       | 5         | 7       | 7        |
|         | REP MOVS/MOVS/MOVSW   | 9+17/rep | 5+4/rep   | 8+4/rep | 12+3/rep |
| MOVSX   | Move with sign-extend |          |           |         |          |
|         | reg to reg            | --       | --        | 3       | 3        |
|         | mem to reg            | --       | --        | 6       | 3        |
| MOVZX   | Move with zero-extend |          |           |         |          |
|         | reg to reg            | --       | --        | 3       | 3        |
|         | mem to reg            | --       | --        | 6       | 3        |

| Code                               | Description                 | 8086         | 80286   | 80386        | 80486 |
|------------------------------------|-----------------------------|--------------|---------|--------------|-------|
| MUL                                | Unsigned multiplication     |              |         |              |       |
|                                    | 8-bit reg                   | 70-77        | 13      | 9-14         | 13/18 |
|                                    | 16-bit reg                  | 118-133      | 21      | 9-22         | 13/26 |
|                                    | double                      | --           | --      | 9-38         | 13/42 |
|                                    | 8-bit mem                   | (76-83)+EA   | 16      | 12-17        | 13/18 |
|                                    | 16-bit mem                  | (124-139)+EA | 24      | 12-25        | 13/26 |
| NEG                                | double                      | --           | --      | 12-41        | 13/42 |
|                                    | Negate                      |              |         |              |       |
|                                    | reg                         | 3            | 2       | 2            | 1     |
| NOP                                | mem                         | 16+EA        | 7       | 6            | 3     |
|                                    | No operation                | 3            | 3       | 3            | 3     |
| NOT                                | Logical NOT                 |              |         |              |       |
|                                    | reg                         | 3            | 2       | 2            | 1     |
| OR                                 | mem                         | 16+EA        | 7       | 6            | 3     |
|                                    | Logical OR                  |              |         |              |       |
|                                    | reg to reg                  | 3            | 2       | 2            | 1     |
|                                    | mem to reg                  | 9+EA         | 7       | 6            | 2     |
|                                    | reg to mem                  | 16+EA        | 7       | 7            | 3     |
|                                    | immed to acc                | 4            | 3       | 2            | 1     |
| OUT                                | immed to reg                | 4            | 3       | 2            | 1     |
|                                    | immed to mem                | 17+EA        | 7       | 7            | 3     |
|                                    | Output to I/O port          |              |         |              |       |
|                                    | fixed port                  | 10           | 3       | 10           | 16    |
|                                    | fixed port PM               | --           | --      | 4,24         | 11,31 |
|                                    | variable port               | 8            | 3       | 11           | 16    |
| OUTS/<br>OUTSB/<br>OUTSW/<br>OUTSD | variable port PM            | --           | --      | 5,25         | 10,30 |
|                                    | Output string to port/      |              |         |              |       |
|                                    | Output byte                 |              |         |              |       |
|                                    | Output word                 |              |         |              |       |
|                                    | Output double               | --           | 5       | 14           | 17    |
|                                    | PM                          | --           | --      | 8,28         | 10,32 |
| POP                                | REP OUTS/OUTSB/OUTSW        | --           | 5+4/rep | 12+5/rep     |       |
|                                    | REP OUTS/OUTSB/OUTSW PM     | --           | --      | (6,26)+5/rep |       |
|                                    | Pop word off stack          |              |         |              |       |
| POPA/<br>POPAD<br>POPF<br>POPFD    | reg                         | 8            | 5       | 4            | 4     |
|                                    | segment reg                 | 8            | 5       | 7            | 3/9   |
|                                    | segment reg PM              | --           | 20      | 21           | 9     |
|                                    | memory                      | 17+EA        | 5       | 5            | 6     |
| PUSH                               | Pop all                     |              |         | 9            |       |
|                                    | Pop all double              | --           | 19      | 24           | 9     |
|                                    | Pop flags off stack         | 8            | 5       | 5            | 9     |
|                                    | Pop flags off stack double  | --           | --      | 5            |       |
| PUSHA<br>PUSHF/<br>PUSHD           | Push word onto stack        |              |         |              |       |
|                                    | reg                         | 11           | 3       | 2            | 4     |
|                                    | segment reg: ES/SS/CS       | 10           | 3       | 2            | 3     |
|                                    | segment reg: FS/GS          | --           | --      | 2            | 3     |
|                                    | memory                      | 16+EA        | 5       | 5            | 4     |
|                                    | immed                       | --           | 3       | 2            | 1     |
| PUSHA<br>PUSHF/<br>PUSHD           | Push All                    | --           | 17      | 18           | 11    |
|                                    | Push flags onto stack       |              |         |              |       |
| PUSHA<br>PUSHF/<br>PUSHD           | Push double flag onto stack | 10           | 3       | 4            | 4     |
|                                    |                             |              |         |              |       |

| <u>Description</u>                                                                              | <u>8086</u>  | <u>80286</u> | <u>80386</u> | <u>80486</u> |
|-------------------------------------------------------------------------------------------------|--------------|--------------|--------------|--------------|
| Rotate left through carry/<br>Rotate right through carry/<br>reg with single-shift              | 2            | 2            | 9            | 3            |
| reg with variable-shift                                                                         | 8+4/bit      | 5+n          | 9            | 8/30         |
| mem with single-shift                                                                           | 15+EA        | 7            | 10           | 4            |
| mem with variable-shift                                                                         | 20+EA+4/bit  | 8+n          | 10           | 9/31         |
| immed to reg                                                                                    | --           | 5+n          | 9            | 8/30         |
| immed to mem                                                                                    | --           | 8+n          | 10           | 9/31         |
| Return from procedure/<br>Return far/<br>Return near                                            |              |              |              |              |
| intrasegment                                                                                    | 16           | 11+m         | 10+m         | 5            |
| intrasegment with constant                                                                      | 20           | 11+m         | 10+m         | 5            |
| intersegment                                                                                    | 26           | 15+m         | 18+m         | 18           |
| intersegment PM                                                                                 | --           | 25+m,55      | 32+m,62      | 13           |
| intersegment with constant                                                                      | 25           | 15+m         | 18+m         | 33           |
| intersegment w/constant PM                                                                      | --           | 25+m,55      | 32+m,68      | 17           |
| 486: imm. to SP                                                                                 | --           | --           | --           | 14           |
| 486: to same level                                                                              | --           | --           | --           | 17           |
| 486: to outer level                                                                             | --           | --           | --           | 33           |
| Rotate left<br>Rotate right                                                                     |              |              |              |              |
| reg with single-shift                                                                           | 2            | 2            | 3            | 3            |
| reg with variable-shift                                                                         | 8+4/bit      | 5+n          | 3            | 3            |
| mem with single-shift                                                                           | 15+EA        | 7            | 7            | 4            |
| mem with variable-shift                                                                         | 20+EA+4/bit  | 8+n          | 7            | 4            |
| immed to reg                                                                                    | --           | 5+n          | 3            | 2            |
| immed to mem                                                                                    | --           | 8+n          | 7            | 4            |
| Store AH into flags                                                                             | 4            | 2            | 3            | 2            |
| Shift arithmetic left/<br>Shift arithmetic right/<br>Shift logical left/<br>Shift logical right |              |              |              |              |
| reg with single-shift                                                                           | 2            | 2            | 3            | 3            |
| reg with variable-shift                                                                         | 8+4/bit      | 5+n          | 3            | 3            |
| mem with single-shift                                                                           | 15+EA        | 7            | 7            | 4            |
| mem with variable-shift                                                                         | 20+EA +4/bit | 8+n          | 7            | 4            |
| immed to reg                                                                                    | --           | 5+n          | 3            | 2            |
| immed to mem                                                                                    | --           | 8+n          | 7            | 4            |
| Subtract with borrow                                                                            |              |              |              |              |
| reg from reg                                                                                    | 3            | 2            | 2            | 1            |
| mem from reg                                                                                    | 9+EA         | 7            | 7            | 2            |
| reg from mem                                                                                    | 16+EA        | 7            | 6            | 3            |
| immed from acc                                                                                  | 4            | 3            | 2            | 1            |
| immed from reg                                                                                  | 4            | 3            | 2            | 1            |
| immed from mem                                                                                  | 17+EA        | 7            | 7            | 3            |
| Scan string/<br>Scan byte string/                                                               |              |              |              |              |

| <u>Code</u>              | <u>Description</u>                                       | <u>8086</u> | <u>80286</u> | <u>80386</u> | <u>80486</u> |
|--------------------------|----------------------------------------------------------|-------------|--------------|--------------|--------------|
| SCASW                    | Scan word string<br>not repeated                         | 15          | 7            | 7            | 6            |
|                          | REPE/REPNE SCAS/SCASB/SCASW                              | 9+15/rep    | 5+8/rep      | 5+8/rep      | 7+5/rep      |
| SET                      | Set conditionally                                        |             |              |              |              |
|                          | reg                                                      | --          | --           | 4            | 4 or 3       |
|                          | mem                                                      | --          | --           | 5            | 3 or 4       |
| SGDT                     | Store global descrpt. table                              | --          | 11           | 9            | 10           |
| SIDT                     | Store interrupt desc. table                              | --          | 12           | 9            | 10           |
| SLDT                     | Store local desc. table                                  |             |              |              |              |
|                          | reg                                                      | --          | 2            | 2            | 2            |
|                          | mem                                                      | --          | 3            | 2            | 3            |
| SHLD/<br>SHRD            | Shift left double precision/<br>Shift right double       |             |              |              |              |
|                          | reg to reg                                               | --          | --           | 3            | 2            |
|                          | mem to mem                                               | --          | --           | 7            | 3            |
|                          | reg by CL                                                | --          | --           | 3            | 3            |
|                          | mem by CL                                                | --          | --           | 7            | 4            |
| SMSW                     | Store machine status word                                |             |              |              |              |
|                          | reg                                                      | --          | 2            | 10           | 2            |
|                          | mem                                                      | --          | 3            | 3            | 3            |
|                          | mem PM                                                   | --          | --           | 2            |              |
| STC                      | Set carry flag                                           | 2           | 2            | 2            | 2            |
| STD                      | Set direction flag                                       | 2           | 2            | 2            | 2            |
| STI                      | Set interrupt flag                                       | 2           | 2            | 3            | 5            |
| STOS/<br>STOSB/<br>STOSW | Store string/<br>Store byte string/<br>Store word string |             |              |              |              |
|                          | not repeated                                             | 11          | 3            | 4            | 5            |
|                          | REP STOS/STOSB/STOSW                                     | 9+10/rep    | 4+3/rep      | 5+5/rep      | 7+4/rep      |
| STR                      | Store task register                                      |             |              |              |              |
|                          | reg                                                      | --          | 2            | 2            | 2            |
|                          | mem                                                      | --          | 3            | 2            | 3            |
| SUB                      | Subtraction                                              |             |              |              |              |
|                          | reg from reg                                             | 3           | 2            | 2            | 1            |
|                          | mem from reg                                             | 9+EA        | 7            | 7            | 2            |
|                          | reg from mem                                             | 16+EA       | 7            | 6            | 3            |
|                          | immed from acc                                           | 4           | 3            | 2            | 1            |
|                          | immed from reg                                           | 4           | 3            | 2            | 1            |
|                          | immed from mem                                           | 17+EA       | 7            | 7            | 3            |
| TEST                     | Test                                                     |             |              |              |              |
|                          | reg with reg                                             | 3           | 2            | 2            | 1            |
|                          | mem with reg                                             | 9+EA        | 6            | 5            | 2            |
|                          | immed with acc                                           | 4           | 3            | 2            | 1            |
|                          | immed with reg                                           | 5           | 3            | 2            | 1            |
|                          | immed with mem                                           | 11+EA       | 6            | 5            | 2            |
| VERR                     | Verify read                                              |             |              |              |              |
|                          | reg                                                      | --          | 14           | 10           | 11           |
|                          | mem                                                      | --          | 16           | 11           | 11           |
| VERW                     | Verify write                                             |             |              |              |              |
|                          | reg                                                      | --          | 14           | 15           | 11           |
|                          | mem                                                      | --          | 16           | 16           | 11           |

| <u>Code</u>    | <u>Description</u>               | <u>8086</u> | <u>80286</u> | <u>80386</u> | <u>80486</u> |
|----------------|----------------------------------|-------------|--------------|--------------|--------------|
| WAIT           | Wait while TEST pin not asserted | 4           | 3            | 6            | 1-3          |
| WBINVD         | Write-back invalid data cache    | --          | --           | --           | 5            |
| XADD           | Exchange and add                 | --          | --           | --           | 3            |
|                | reg with reg                     | --          | --           | --           | 4            |
| XCHG           | Exchange                         |             |              |              |              |
|                | reg with acc                     | 3           | 3            | 3            | 3            |
|                | reg with mem                     | 17+EA       | 5            | 5            | 5            |
|                | reg with reg                     | 4           | 3            | 3            | 3            |
| XLAT/<br>XLATB | Translate                        | 11          | 5            | 5            | 4            |
| XOR            | Logical exclusive OR             |             |              |              |              |
|                | reg with reg                     | 3           | 2            | 2            | 1            |
|                | mem with reg                     | 9+EA        | 7            | 7            | 2            |
|                | reg with mem                     | 16+EA       | 7            | 6            | 3            |
|                | immed with acc                   | 4           | 3            | 2            | 1            |
|                | immed with reg                   | 4           | 3            | 2            | 1            |
|                | immed with mem                   | 17+EA       | 7            | 7            | 3            |



---

# رهنمون‌های اسمبلر و قوانین نام‌گذاری

---

این پیوست از دو بخش تشکیل شده است. بخش اول بعضی از رهنمون‌های پرمصرف در برنامه‌نویسی زبان اسمبلی 80x86 را توصیف می‌نماید. در بخش دوم قوانین زبان اسمبلی و محدودیت‌ها برای نام و برجسب‌ها بحث شده و فهرست کلمات رزرو تهیه شده است.

## بخش ۱- پ: رهنمون‌های اسمبلر

رهنمون‌ها، یا آنچه گاهی به آن شبه عمل یا شبه دستورالعمل هم می‌گویند، برای کمک به اسمبلر در ترجمه برنامه‌های زبان اسمبلی به زبان ماشین بکار می‌رود. بر خلاف دستورات ریزپردازنده‌ها، رهنمون‌ها هیچ کد عملی تولید نمی‌کنند؛ بنابراین هیچ مکان حافظه‌ای بوسیله رهنمون‌ها به‌نگام آماده بودن ویرایش exe برنامه اسمبلی اشغال نمی‌گردد. بطور خلاصه رهنمون‌ها جهت رابه برنامه اسمبلر برای تولید کد ماشین نشان می‌دهند. دستورالعمل‌ها به صورت کد ماشین درمی‌آیند تا جهت را به‌نگام اجرا برای CPU معین کنند. در زیر بعضی رهنمون‌های اسمبلر 80x86 آمده است. این رهنمون‌ها برای سادگی در مراجعه، به صورت الفبا مرتب شده‌اند.

## ASSUME

رهنمون ASSUME بوسیله اسمبلر برای مرتبط ساختن نام قطعه به ثبات قطعه بکار می‌رود. این رهنمون برای دستوراتی که باید آدرسی را از ترکیب تفاوت مکان با ثبات قطعه محاسبه کنند لازم است. یک رهنمون ASSUME می‌تواند همه ثبات‌های قطعه را داشته باشد. مثلاً

```
ASSUME CS:name1, DS:name2, SS:name3, ES:name4
```

که name1، name2 و غیره نام قطعات هستند. نتیجه مشابهی با تخصیص یک ASSUME برای هر ثبات حاصل می‌شود:

```
ASSUME CS:name1
ASSUME DS:name2
ASSUME SS:name3
ASSUME ES:nothing
ASSUME nothing
```

از کلمه کلیدی "nothing" می‌توان برای حذف رهنمون ASSUME استفاده کرد.

## DB (تعریف بایت)

رهنمون DB برای تخصیص حافظه افزایشی با اندازه بایت بکار می‌رود. به مثال‌های زیر توجه کنید:

```
DATA1 DB 23
DATA2 DB 45,97H,10000011B
DATA3 DB 'The planet Earth'
```

در DATA1، یک بایت با مقدار اولیه 23 تعریف شده است. DATA2 متشکل از چند مقدار به دهمی (45)، شانزده شانه‌دهی (97H) و دودویی (10000011B) است. نهایتاً در DATA3، رهنمون DB برای تعریف کاراکترهای اسکی بکار رفته است. رهنمون DB معمولاً برای تعریف داده اسکی هم بکار می‌رود. در همه مثال‌های فوق، مکان آدرس بوسیله اسمبلر معین می‌گردد. ما می‌توانیم آدرس تفاوت مکان خاصی را با استفاده از رهنمون ORG تعریف کنیم.

## DD (تعریف جفت کلمه)

برای تخصیص حافظه 4 بایتی (32 بیت)، رهنمون DD بکار می‌رود. چون عملوندهای کلمه‌ای 16 بیت عرض دارند (2 بایت)، در اسمبلرهای 80x86 یک جفت کلمه، 4 بایت خواهد بود.

```
VALUE1 DD 4563F57H
RESULT DD ? ;RESERVE 4-BYTE LOCATION
DAT4 DD 25000000
```

باید متذکر شد که مقادیر تعریف شده با رهنمون DD بوسیله اسمبلر بصورت بایت پایین در آدرس پایین و بایت بالا در آدرس بالا ذخیره می‌شود. این قرارداد را منتهی به کوچکتر می‌نامند. مثلاً، فرض کنید که تفاوت مکان 0020 به VALUE1 در مثال فوق تخصیص یابد، هر بایت به ترتیب زیر در حافظه قرار می‌گیرد:

```
DS:20=(57)
DS:21=(3F)
DS:22=(56)
DS:23=(04)
```

## DQ (تعریف چهار کلمه)

برای تخصیص حافظه در فرم 8 بایت، رهنمون DQ بکار می‌رود. در 80x86 هر کلمه بصورت 2 بایت تعریف می‌شود. بنابراین یک چهارکلمه، 8 بایت است.

DAT\_64B DQ 5677DD4EE4FF45AH  
 DAT8 DQ 10000000000000

### DT (تعریف ده بایت)

برای تخصیص داده BCD فشرده، یعنی ده بایت در هر بار، از رهنمون DT استفاده می‌شود. این رهنمون بطور گسترده‌ای برای تخصیص حافظه به اعداد BCD بکار می‌رود.

DATA DT 399977653419974

توجه کنید که برای مشخص کردن عدد مبنای شانزده، H وجود ندارد. این از خصوصیات ویژه رهنمون DT است. در دیگر رهنمون‌ها (DB، DW، DD، DQ) اگر H وجود نداشته باشد. فرض بر این است که داده دهدهی است و بوسیله اسمبلر به مبنای شانزده تبدیل خواهد شد. به خاطر داشته باشید که قرارداد منتهی به کوچکتر برای قراردادن بایت‌ها در حافظه بکار می‌رود، و در آن بایت کم‌ارزستر به آدرس پایین و بایت با ارزشتر به آدرس بالا می‌رود. DT برای قراردادن داده دهدهی بهنگام قراردادن "d" پس از عدد هم می‌تواند بکار رود.

DATA DT 65535d ;stores hex FFFF in a 10-byte location

### DUP (تکرار)

رهنمون DUP می‌تواند برای تکرار مجموعه‌ای از داده‌ها به تعداد معینی در عوض نوشتن‌های مکرر بکار رود.

|       |    |                   |                               |
|-------|----|-------------------|-------------------------------|
| DATA1 | DB | 20 DUP (99)       | ;DUPLICATE 99 20 TIMES        |
| DATA2 | DW | 6 DUP (5555H)     | ;DUPLICATE 5555H 6 TIMES      |
| DATA3 | DB | 10 DUP (?)        | ;RESERVE 10 BYTES             |
| DATA4 | DB | 5 DUP (5 DUP (0)) | ;25 BYTES INITIALIZED TO ZERO |
| DATA5 | DB | 10 DUP (00,FFH)   | ;20 BYTES ALTERNATE 00, FF    |

### DW (تعریف کلمه)

برای تخصیص حافظه بصورت 2 بایت (16 بیت)، رهنمون DW بکار می‌رود. در خانواده 80x86 یک کلمه بصورت 16 بیت تعریف می‌گردد.

DATAW\_1 DW 5000  
 DATAW\_2 DW 7F6BH

مجدداً برای قرارگرفتن بایت‌ها در حافظه از قرارداد منتهی به کوچکتر استفاده شده است که در آن بایت کم اهمیت به آدرس پایین و با اهمیت به آدرس بالا می‌رود.

### END

هر برنامه‌ای باید نقطه ورود داشته باشد. برای یافتن آن نقطه ورود، اسمبلر بر رهنمون END متکی است. برچسب نقطه ورود و پایان باید باهم تطابق داشته باشند.

```
HERE: MOV AX,DATASEG ;ENTRY POINT OF THE PROGRAM
 ...
 ...
 END HERE ;EXIT POINT OF THE PROGRAM
```

اگر چندین مدول وجود داشته باشد، فقط یکی از آنها می‌تواند نقطه ورود داشته باشد و نام آن ورودی باید با نام جلو رهنمون END یکی باشد.



از برنامه اصلی،

```
EXTRN PROG1:NEAR

MAIN_PRO: ... MOV AX,DATASG ;THE ENTRY POINT
 MOV DS,AX
 ...
 CALL PROG1
 ...
 END MAIN_PRO ;THE EXIT POINT
```

از برنامه PROG1،

```
PROG1 PUBLIC PROG1
 PROC
 ...
 RET ;RETURN TO THE MAIN MODULE
PROG1 ENDP
 END ;NO LABEL IS GIVEN
```

در مورد برنامه‌های فوق به نکات زیر توجه کنید:

- ۱- نقطه ورود باید با یک نام مشخص شود، در مثال فوق نقطه ورودی با MAIN-PRO مشخص شده است.
- ۲- نقطه خروج باید بوسیله همان نام نقطه ورودی یعنی MAIN-PRO شناسایی شود.
- ۳- چون یک برنامه مفروض فقط می‌تواند یک نقطه ورودی و نقطه خروجی داشته باشد، تمام مدول‌های فراخوانده شده (از برنامه‌های اصلی یا از زیرمدول‌ها) باید رهنمون END را بدون چیزی بعد از آن داشته باشند.

ENDP (رهنمون PROC ملاحظه شود)

ENDS (رهنمون‌های STRUCT و SEGMENT ملاحظه شوند)

EQU (برابر)

برای تخصیص یک مقدار ثابت به یک نام از رهنمون EQU استفاده می‌شود. اسمبلر هر وقوع نام را با مقدار متناسب به آن جایگزین می‌نماید.

```
FIX_VALU EQU 1200
PORT_A EQU 60H
COUNT EQU 100
MASK_1 EQU 00001111B
```

بر خلاف رهنمون‌های داده‌ای همچون DB، DW و غیره، EQU هیچ حافظه‌ای را اشغال نمی‌کند، بنابراین در هر جا و هر زمان قابل تعریف است و حتی می‌تواند در قطعه کد هم تعریف شود.

EVEN

رهنمون EVEN اختصاص حافظه را به شروع از آدرس زوج و امی دارد. این رهنمون به این علت مفید است که ریزپردازنده‌های 8086، 286 و 386SX بهنگام دسترسی به عملوند دویابیتی در یک مکان فرد زمان زیادتری لازم دارند. استفاده از رهنمون EVEN اسمبلر را به تخصیص یک آدرس زوج به متغیر هدایت می‌کند.

```
DATA_1 ORG 0020H
 DB 34H
 EVEN
DATA_2 DW 7F5BH
```

در زیر محتوای مکان‌های حافظه دیده می‌شوند:

DS:0020 = (34)

DS:0021 = (?)

DS:0022 = (5B)

DS:0023 = (7F)

توجه کنید که رهنمون EVEN موجب شده است تا از مکان حافظه DS:0021 چشم‌پوشی شود و مقدار DATA\_2 از مکان آدرس زوجی شروع گردد.

### EXTRN (بیرونی)

رهنمون EXTRN برای تشخیص متغیرها و نام‌های معینی در یک مدول است که بوسیله مدول دیگری تعریف شده است. در غیاب رهنمون EXTRN، اسمبلر بدنبال تعریف خواهد گشت و هنگامی که نتواند آن را بیابد خطایی را اعلام خواهد کرد. قالب این رهنمون چنین است.

**EXTRN name1:typea [,name2:typeb]**

که type هنگامی که name به یک رویه اشاره کند NEAR یا FAR می‌باشد و اگر name به یک متغیر داده اشاره کند BYTE، WORD، DWORD، QWORD و TBYTE خواهد بود.  
از برنامه اصلی:

```
EXTRN PROG1:NEAR
PUBLIC DATA1

...
MAIN_PRO MOV AX,DATASG ;THE ENTRY POINT
 MOV DS,AX
...
 CALL PROG1
...
 END MAIN_PRO ;THE EXIT POINT
```

PROG1 در فایل متفاوتی قرار دارد:

```
EXTRN DATA1:WORD
PUBLIC PROG1
PROC
...
 MOV BX,DATA1
...
 RET
PROG1 ENDP
 END
```

;RETURN TO THE MAIN MODULE

توجه کنید که رهنمون EXTRN در رویه اصلی برای شناسایی PROG1 بعنوان یک رویه NEAR بکار رفته است. این عمل لازم است زیرا PROG1 در آن مدول تعریف نشده است. در همین ارتباط، PROG1 در مدولی که آمده بعنوان PUBLIC تعریف شده است. EXTRN در مدول PROG1، برای اعلام این مطلب که عملوند DATA1 با اندازه WORD در مدول دیگری تعریف شده، بکار رفته است. در نتیجه DATA1 بعنوان PUBLIC در مدول فراخوان اعلام گشته است.

## GROUP

رهنمون GROUP موجب می شود قطعات نام گرفته در یک قطعه 64K بایت بهم متصل شوند. همه قطعات فهرست شده در رهنمون GROUP باید در یک 64K جای شوند. این رهنمون برای ترکیب قطعات هم نوع یا رده های متفاوتی از قطعات می تواند بکار رود. مثلاً

SMALL\_SYS GROUP DTSEG,STSEG,CDSEG

رهنمون ASSUME باید تغییر یابد تا ثبات های قطعه به گروه اشاره کنند

ASSUME CS:SMALL\_SYS,DS:SMALL\_SYS,SS:SMALL\_SYS

GROUP در فایل لیست، فهرست می شود:

قطعات و گروه ها

| Name            | Length | Align | Combine | Class |
|-----------------|--------|-------|---------|-------|
| SMALL_SYS ..... | GROUP  |       |         |       |
| STSEG .....     | 0040   | PARA  | NONE    |       |
| DTSEG .....     | 0024   | PARA  | NONE    |       |
| CDSEG .....     | 005A   | PARA  | NONE    |       |

## INCLUDE

وقتی که گروهی از ماکروها نوشته شده و در فایل های جداگانه ذخیره شده اند، رهنمون INCLUDE می تواند برای آوردن آنها به فایلی دیگر بکار رود. در برنامه (فایل .lst)، این ماکروها بوسیله نماد C (یا "+") در بعضی از ویرایش های MASM شناسایی می شوند. این شناسایی قبل از هر دستور برای درک این مطلب که در فایل فعلی بوسیله رهنمون INCLUDE کپی شده اند، صورت می گیرد.

## LABEL

رهنمون LABEL اجازه می دهد تا یک متغیر مفروض با نام یا نام های چندگانه اشاره شود. این اغلب برای تعریف چندگانه یک متغیر یا نام بکار می رود. قالب رهنمون LABEL چنین است.

name LABEL type

که type ممکن است BYTE، WORD، DWORD و QWORD باشد. مثلاً نام متغیر DATA1 بعنوان یک کلمه تعریف می شود و لازم است طبق آنچه در زیر آمده بصورت دوبایت دستیابی شود:

|        |              |           |
|--------|--------------|-----------|
| DATA_B | LABEL        | BYTE      |
| DATA1  | DW           | 25F6H     |
| MOV    | AX,DATA1     | ;AX=25F6H |
| MOV    | BL,DATA_B    | ;BL=F6H   |
| MOV    | BH,DATA_B +1 | ;BH=25H   |

در زیر LABEL برای دستیابی به اقلام داده 32 بیتی در بخش های 16 بیتی بکار رفته است.

|         |                |           |
|---------|----------------|-----------|
| DATA_16 | LABEL          | WORD      |
| DATDD_4 | DD             | 4387983FH |
| ...     |                |           |
| MOV     | AX,DATA_16     | ;AX=983FH |
| MOV     | DX,DATA_16 + 2 | ;DX=4387H |

در زیر کاربرد آن در دستور JMP برای رفتن به قطعه کد متفاوت دیده می شود

```

 JMP PROG_A
PROG_A LABEL FAR
INITI: MOV AL,12H
 OUT PORT,AL

```

در برنامه فوق آدرس اختصاص یافته به نام PROG\_A و INITI دقیقاً یکی است. عمل مشابهی بطریق زیر حاصل می شود:

**JMP FAR PTR INITI**

## LENGTH

عملگر LENGTH تعداد اقلام تعریف شده با عملوند DUP را بازمی گرداند. رهنمون SIZE را بعنوان مثال ملاحظه کنید.

## OFFSET

برای دستیابی آدرس تفاوت مکان متناسب به یک متغیر یا نام از رهنمون OFFSET استفاده می شود. رهنمون OFFSET در مثال زیر برای بدست آوردن آدرس تفاوت مکان که بوسیله اسمبلر به DATA1 اختصاص یافته بکار رفته است.

```

DATA1 ORG 5600H
 DW 2345H
 MOV SI,OFFSET DATA1 ;SI=OFFSET OF DATA1 = 5600H

```

توجه کنید که این نتیجه یکسانی را با "LEA SI, DATA1" دارد.

## ORG (مبدأ)

رهنمون ORG برای تخصیص یک آدرس تفاوت مکان برای متغیر یا نام بکار می رود. مثلاً برای قراردادن متغیر DATA1 و آدرس تفاوت مکان شروع 0020، می توان نوشت:

```

DATA1 ORG 0020H
 DW 41F2H

```

این موجب می شود تا آدرس های تفاوت مکان 0020 و 0021 دارای محتوای (F2) = 0020H و (41) = 0021H گردند.

## PAGE

رهنمون PAGE برای چاپ فایل ".lst" چاپ در قالبی خاص بکار می رود. قالب رهنمون PAGE چنین است:

**PAGE [lines],[columns]**

لیست پیش فرض ها (یعنی که رهنمون PAGE کد نشده است) دارای 66 خط در هر صفحه با حداکثر 80 کاراکتر در هر خط است. این مقادیر می توانند با رهنمون "PAGE 60,132" به 60 خط و 132 کاراکتر در هر خط

تغییر یابند. محدوده خطوط از 10 تا 255 و ستون‌ها از 60 تا 132 است. یک رهنمون PAGE بدون شماره موجب قطع صفحه می‌شود.

## PROC و ENDP (رویه و پایان رویه)

اغلب گروهی از دستورالعمل‌های زبان اسمبلی در یک رویه ترکیب می‌شوند و بنابراین قابل فراخوانی بوسیله مدول دیگر می‌باشند. رهنمون‌های PROC و ENDP برای تشخیص شروع و پایان رویه بکار می‌روند. برای یک رویه مفروض نام اختصاص یافته به PROC باید دقیقاً با نام ENDP یکی باشد.

```
name1 PROC [attribute]
```

```
name1 ENDP
```

برای خصیصه PROC دو انتخاب وجود دارد: NEAR یا FAR. اگر خصیصه‌ای داده نشود، پیش‌فرض NEAR است. وقتی یک رویه NEAR فراخوانی شود، فقط IP ذخیره می‌شود زیرا CS متعلق به رویه خوانده شده، همانند CS در برنامه فراخوان است. اگر یک رویه FAR فراخوانی شود، هر دو IP و CS ذخیره می‌شوند زیرا قطعه کد رویه فراخوانده شده با برنامه فراخوان متفاوت است.

## PTR (اشاره‌گر)

رهنمون PTR برای مشخص کردن اندازه عملوند بکار می‌رود. انتخاب‌های اندازه عبارتند از BYTE، WORD، DWORD و QWORD. این رهنمون به طرق مختلف مورد استفاده قرار می‌گیرد که مهمترین آنها در زیر آمده است.

۱- PTR برای لغو رهنمون داده قبلی بکار می‌رود.

```
DATA1 DB 23H,7FH,99H,0B2H
DATA2 DW 67F1H
DATA3 DD 22229999H

MOV AX, WORD PTR DATA1 ;AX=7F23
MOV BX, WORD PTR DATA1 + 2 ;BX,B299H
```

گرچه DATA1 در آغاز بصورت DB تعریف شده بود ولی می‌توان آن را با رهنمون WORD PTR دستیابی کرد.

```
MOV AL, BYTE PTR DATA2 ;AL=F1H
```

در کد فوق، توجه دارید که DATA2 بصورت WORD تعریف شده بود ولی به کمک BYTE PTR بصورت BYTE دستیابی شد. اگر آن را بصورت "MOV AL,DATA2" بنویسیم خطا تولید می‌شود زیرا اندازه عملوندها همخوانی ندارد.

```
MOV AX, WORD PTR DATA3 ;AX=9999H
MOV DX, WORD PTR DATA3 + 2 ;DX=2222H
```

DATA3 بصورت عملوند 4 بیتی تعریف شده بود ولی ثبات‌ها فقط 2 بایت عرض دارند. رهنمون WORD PTR مشکل را حل کرده است.

۲- رهنمون PTR برای مشخص کردن اندازه یک رهنمون برای کمک به ترجمه دستورالعمل توسط اسمبلر بکار می‌رود.

INC [DI] ;will cause an error

این دستورالعمل به معنی افزایش محتوای مکان حافظه (ها) می باشد که به آن بوسیله [DI] اشاره شد. اسمبلر چگونه می فهمد که محتوای عملوند یک بایتی، کلمه، یا جفت کلمه است؟ چون نمی داند کدامیک است خطایی تولید خواهد کرد. برای اصلاح آن، رهنمون PTR را برای مشخص کردن اندازه عملوند بکار برید.

INC BYTE PTR [SI] ;increment a byte pointed by SI  
or  
INC WORD PTR [SI] ;increment a word pointed by SI  
or  
INC DWORD PTR [SI] ;increment a doubleword pointed by SI

۳- رهنمون PTR می تواند مقصد یک دستور پرش را پیدا کند. انتخاب های مقصد FAR و NEAR است. رهنمون LABEL برای یافتن و دستیابی به نتیجه یکسان ملاحظه شود.

JMP FAR PTR INTI ;ensures that it will be a 5-byte instruction  
...  
INITI: MOV AX,1200

See the LABEL directive to find out how it can be used to achieve the same result.

## PUBLIC

برای آگاه کردن اسمبلر از اینکه نام یا سمبلی بوسیله مدول های دیگر ارجاع می شود، آن را با رهنمون PUBLIC علامت می زنند. اگر مدولی به متغیری در خارج خودش ارجاع می دهد، آن متغیر با EXTRN اعلام می گردد. در این رابطه، در مدولی که متغیر تعریف شده است، آن متغیر باید با PUBLIC مشخص شود تا به دیگر مدول ها ارجاع داده شود. رهنمون EXTRN را برای کاربرد EXTRN و PUBLIC ملاحظه نمایید.

## SEG (آدرس قطعه)

عملگر SEG برای دستیابی آدرس قطعه که نام در آن تعریف شده بکار می رود.

DATA1 DW 2341H  
...  
MOV AX,SEG DATA1 ;AX=SEGMENT ADDRESS OF DATA1

این برخلاف رهنمون OFFSET است که آدرس تفاوت مکان را در عوض قطعه دستیابی می نماید.

## ENDS و SEGMENT

در تعریف قطعه کامل این دو رهنمون بکار می روند تا شروع و پایان قطعه را بیان نمایند. آنها برای یک قطعه مفروض باید نام یکسانی داشته باشند. به مثال زیر توجه کنید.

DATSEG SEGMENT  
DATA1 DB 2FH  
DATA2 DW 1200  
DATA3 DD 99999999H  
DATSEG ENDS

انتخاب‌های متعددی در رابطه با رهنمون SEGMENT مطابق زیر وجود دارد:

**name1 SEGMENT [align] [combine] [class]**

**name1 ENDS**

**انطباق:** وقتی چندین مدول اسمبل شده بهم ارتباط داده شوند این رهنمون شروع قطعه را معین می‌نماید. انتخاب‌های متعددی وجود دارد، از جمله PARA (بایت = 16 paragraph)، WORD و BYTE. اگر PARA انتخاب شود، قطعه از آدرس مبنای شانزده قابل تقسیم بر 10H آغاز می‌شود. PARA انطباق پیش‌فرض است. در این انطباق، اگر یک قطعه مدولی در 00024H پایان یافت، قطعه بعد از آدرس 00030H شروع شده و فاصله 00025H تا 0002F بدون استفاده خواهد ماند. اگر WORD اختیار گردد، قطعه به شروع از مرز کلمه واداشته می‌شود. در انطباق BYTE، قطعه از بایت بعدی شروع و هیچ حافظه‌ای تلف نمی‌شود. انتخاب PAGE هم وجود دارد که قطعات را در مرز 100H (256) بایت منطبق می‌سازد. در حالی که همه این انتخاب‌ها بوسیله بسیاری از اسمبلرها مانند MASM و TASM پشتیبانی می‌شوند، انتخاب دیگری هم وجود دارد که طراحی سیستم را بوسیله اسمبلر مجاز می‌دارد. این انتخاب AT است. انتخاب AT به برنامه اجازه تخصیص آدرس فیزیکی را می‌دهد. مثل سوزاندن یک برنامه در یک ROM با شروع از آدرس فیزیکی F0000 کد چنین است:

**ROM\_CODE SEGMENT AT F000H**

چون انتخاب AT به برنامه نویس اجازه تخصیص یک آدرس فیزیکی را می‌دهد، به دلیل تداخل با مسئولیت مدیریت حافظه DOS، بسیاری از اسمبلرها مانند MASM انتخاب AT را مجاز نمی‌دانند.

**COMBINE TYPE:** این انتخاب برای گردآوری همه قطعات مشابه برای ایجاد یک قطعه بزرگتر بکار می‌رود. در میان انتخاب‌های رایج می‌توان از PUBLIC و STACK نام برد. PUBLIC در تعاریف قطعه کد هنگام ارتباط بیش از یک مدول بکار می‌رود. این رهنمون همه قطعات کد انواع مدول‌ها را در یک قطعه کد بزرگتر یکپارچه می‌کند. اگر فقط یک قطعه داده وجود داشته و آنهم متعلق به مدول اصلی باشد نیازی به تعریف آن بصورت PUBLIC نیست زیرا هیچ مدول دیگری دارای قطعه داده ترکیب شدنی با آن نیست. با این وجود، اگر دیگر مدول‌ها دارای قطعات داده متعلق به خود باشند، پیشنهاد می‌شود آنها را با PUBLIC برای تولید یک قطعه داده مشترک بهنگام ارتباط، یکپارچه کرد. در غیاب آن ارتباط دهنده فرض خواهد کرد که هر قطعه خصوصی است و با دیگر قطعات مشابه ترکیب نخواهند شد (کد با کد و داده با داده). چون فقط یک قطعه پشته وجود دارد، و متعلق به مدول اصلی است، نیازی به تعریف آن بعنوان PUBLIC نیست. انتخاب STACK فقط برای تعریف قطعه پشته بکار رفته و برای ارتباط دهنده مشخص می‌کند که پشته تعریف شده بوسیله کاربر را با پشته سیستم برای ایجاد یک پشته در کل برنامه ترکیب کند. این پشته‌ای است که بهنگام اجرای برنامه بکار می‌رود. (وقتی که در واقع CPU برنامه را اجرا می‌کند).

**CLASS NAME:** به ارتباط دهنده می‌فهماند که همه قطعات هم‌رده باید بوسیله LINKER کنار هم قرار گیرند. چهار نام رده‌ای که عمدتاً مورد استفاده‌اند عبارتند از 'CODE'، 'DATA'، 'STACK' و 'EXTRA'. وقتی این خصیصه در تعریف قطعه بکار رود، باید در درون یک جفت ویرگول بالا قرار گیرد تا بوسیله ارتباط دهنده قابل تشخیص باشد.

## SHORT

در پرش مستقیم مانند "JMP POINT\_A" اسمبلر مجبور به انتخاب قالب 2 بایتی یا 3 بایتی است. در قالب 2 بایتی، یک بایت کد عمل و بایت دوم مقدار عدد علامت دار جابجایی است و به IP دستور که بلافاصله بعد از

JMP آمده است اضافه می شود. این جابجایی می تواند هر کجا در میان 128- تا 127+ باشد. عدد منفی به معنی پرش به عقب و عدد مثبت پرش به جلو است. در قالب 3 بایتی بایت اول کد عمل و دو بایت بعدی مقدار عدد علامت دار جابجایی است که می تواند در محدوده 32768- تا 32767+ باشد. هنگام اسمبل کردن یک برنامه، اسمبلر در برنامه دو مرور انجام می دهد. وظایف خاصی در اولین مرور انجام می شود و بقیه به مرور دوم محول می گردند. در اولین مرور اسمبلر کد را برای 3 JMP بایتی در نظر می گیرد. پس از تکمیل اولین مرور، آدرس هدف را خواهد دانست و در دومین مرور آن را در محلش قرار می دهد. اگر آدرس هدف پرش کوتاهی را مشخص کند (کمتر از 128) آخرین بایت را با NOP پر می کند. برای آگاهی اسمبلر از اینکه آدرس هدف بیش از 128 بایت نیست رهنمون SHORT را می توان بکار برد. هنگام استفاده از رهنمون SHORT مطمئن باشید که پرش 2 بایتی است و نه 3 بایتی با یک بایت بعنوان کد NOP. JMP دو بایتی یک بایت حافظه کمتر نیاز داشته و سریع تر اجرا می گردد.

## SIZE

عملگر SIZE تعداد کل بایت های اشغالی بوسیله یک نام را باز می گرداند. سه رهنمون LENGTH، SIZE و TYPE تا حدی بهم مربوطند. در زیر توصیف هر کدام با استفاده از داده های تعریف شده در قطعه داده ارائه شده اند.

|       |    |             |
|-------|----|-------------|
| DATA1 | DQ | ?           |
| DATA2 | DW | ?           |
| DATA3 | DB | 20 DUP (?)  |
| DATA4 | DW | 100 DUP (?) |
| DATA5 | DD | 10 DUP (?)  |

TYPE اجازه می دهد تا رهنمون بکار رفته برای یک متغیر مفروض با تهیه تعداد بایت ها طبق جدول زیر دانسته شود:

| بایت ها |    |
|---------|----|
| 1       | DB |
| 2       | DW |
| 4       | DD |
| 8       | DQ |
| 10      | DT |

مثلاً:

|     |               |       |
|-----|---------------|-------|
| MOV | BX,TYPE DATA2 | ;BX=2 |
| MOV | DX,TYPE DATA1 | ;DX=8 |
| MOV | AX,TYPE DATA3 | ;AX=1 |
| MOV | CX,TYPE DATA5 | ;CX=4 |

وقتی که DUP برای تعریف تعداد وارده های یک متغیر مفروض بکار رود می توان از رهنمون LENGTH برای بدست آوردن آن استفاده کرد.

|     |                 |                       |
|-----|-----------------|-----------------------|
| MOV | CX,LENGTH DATA4 | ;CX=64H (100 DECIMAL) |
| MOV | AX,LENGTH DATA3 | ;AX=14H (20 DECIMAL)  |
| MOV | DX,LENGTH DATA5 | ;DX=0A (10 DECIMAL)   |

اگر متغیر تعریف شده دارای DUP نباشد LENGTH برابر 1 فرض می شود.

|     |                 |       |
|-----|-----------------|-------|
| MOV | BX,LENGTH DATA1 | ;BX=1 |
|-----|-----------------|-------|

SIZE برای تعیین تعداد کل بایت های اختصاص یافته به یک متغیر که با رهنمون DUP تعریف شده بکار می رود. در واقع رهنمون SIZE اساساً ضرب TYPE در LENGTH را فراهم می سازد.



```
MOV DX, SIZE DATA4 ;DX=C8H=200 (100 x 2=200)
MOV CX, SIZE DATA5 ;CX=28H=40 (4 x 10=40)
```

## STRUC (ساختار)

رهنمون STRUC شروع یک تعریف ساختار را مشخص می‌نماید. این تعریف با رهنمون ENDS پایان می‌پذیرد که برچسب آن با برچسب STRUC یکی است. گرچه از رهنمون ENDS یکسانی برای پایان قطعه و پایان ساختار استفاده می‌شود، اسمبلر از متن می‌فهمد که کدام یک چه معنی می‌دهد. ساختار مجموعه‌ای از انواع داده هاست که می‌تواند بطور جمعی بوسیله نام ساختار یا بصورت منفرد بوسیله برچسب‌های انواع داده‌ها در ساختار دستیابی شود. نوع ساختار در ابتدا باید تعریف شود، آنگاه و سپس متغیرها در قطعه داده بر اساس نوع ساختار قابل استقرارند. با نگاهی به مثال زیر، رهنمون‌های داده بین STRUC و ENDS اعلام می‌نمایند که AREA\_ASC مانند چه ساختاری است. هیچ حافظه‌ای به این تعاریف ساختار اختصاص نمی‌یابد. بلافاصله، در زیر تعریف ساختار، برچسب ASC\_INPUT قرار دارد که از نوع ASC\_INPUT است. حافظه به متغیر ASC\_INPUT اختصاص یافته است. در قطعه کد دقت کنید که ASC\_INPUT می‌تواند کاملاً یا با بخشی از مؤلفه‌اش دستیابی شود. بعنوان یک واحد کامل بصورت "MOV DX, OFFSET ASC\_INPUT" دستیابی شود. بخش‌های جزئی‌اش بوسیله نام متغیر و بدنبال آن یک فاصله، سپس نام مؤلفه‌اش دستیابی می‌شود. مثلاً "MOV BL, ASC\_INPUT. ACT\_LEN" میدان واقعی ASC\_INPUT را دستیابی می‌کند.

؛از قطعه داده

```
;from the data segment:
ASC_AREA STRUC ;defines struc for string input
MAX_LEN DB 6 ; maximum length of input string
ACT_LEN DB ? ; actual length of input string
ASC_NUM DB 6 DUP (?) ; input string
ASC_AREA ENDS ;end struc definition
ASC_INPUT ASC_AREA <> ;allocates memory for struc
```

```
GET_ASC: ...
MOV AH, 0AH ;از قطعه کد
MOV DX, OFFSET ASC_INPUT
INT 21H

...
MOV SI, OFFSET ASC_INPUT.ASC_NUM ;SI points to ASCII num
MOV BL, ASC_INPUT.ACT_LEN ;BL holds string length
```

## TITLE

رهنمون TITLE به اسمبلر فرمان می‌دهد تا تیتیر برنامه را در بالای صفحه فایل اول چاپ کند. آنچه پس از شبه دستور می‌آید به برنامه‌نویس بستگی دارد ولی معمولاً نام برنامه را بلافاصله پس از TITLE می‌نویسند و سپس توضیح خلاصه‌ای از عملکرد برنامه می‌آید. هر آنچه که پس از شبه دستور TITLE می‌آید نباید بیش از 60 کاراکتر اسکی (حرف، عدد، فاصله، نقطه) باشد.

## TYPE

عملگر TYPE تعداد بایت‌های ذخیره شده برای داده نام‌گذاری شده را برمی‌گرداند. برای مثالی از کاربرد آن به رهنمون SIZE مراجعه شود.

## بخش ۲ - پ: قوانین برچسب‌ها و نام‌های رزرو

برچسب‌ها در زبان اسمبلی 80x86 برای MASM 5.1 و بالاتر باید از قوانین زیر تبعیت کند:

- ۱- نام‌ها می‌توانند متشکل باشند از:  
کاراکترهای الفبایی: A-Z و a-z  
ارقام: 0-9  
کاراکترهای خاص: "?", ".", "@", "\_", "\$".
  - ۲- نام‌ها باید با یک حرف یا کاراکتر خاص آغاز شود. نام‌ها نمی‌توانند با یک رقم شروع شوند.
  - ۳- نام‌ها می‌توانند تا 31 کاراکتر طول داشته باشند.
  - ۴- کاراکتر خاص "." فقط می‌تواند بعنوان اولین کاراکتر بکار رود.
  - ۵- حروف بزرگ و کوچک یکسانند. با "NAME1"، "Name1"، "name1" یکسان برخورد می‌شود.  
برنامه‌های زبان اسمبلی پنج نوع برچسب یا نام دارند:
- ۱- برچسب‌های کد، نام‌های نمادین را به دستورالعمل‌ها می‌دهند بطوری که دیگر دستورات می‌توانند به آنها ارجاع دهند (مثل پرش).
  - ۲- برچسب‌های رویه که یک نام را به رویه‌ای منتسب می‌کنند.
  - ۳- برچسب‌های قطعه که یک نام را به قطعه تخصیص می‌دهند.
  - ۴- برچسب‌های داده که نام‌ها را به اقلام داده تخصیص می‌دهند.
  - ۵- برچسب‌هایی که بوسیله رهنمون LABEL ایجاد می‌شوند.

### برچسب‌های کد

این برچسب‌ها بوسیله یک جفت نقطه دنبال می‌شود و نوع NEAR را دارا هستند. این امر دیگر دستورالعمل‌ها را قادر می‌سازد در قطعه کد به دیگر دستورات ارجاع دهند. برچسب‌ها در همان خط دستورالعمل واقعند:

```
ADD_LP: ... ;label is on same line as the instruction
 ...
 ...
 LOOP ADD_LP
```

or on a line by themselves:

```
ADD_LP: ... ;label is on a line by itself
 ADD AL,[BX] ;ADD_LP refers to this instruction
 ...
 ...
 LOOP ADD_LP
```

### برچسب‌های رویه

این برچسب‌ها یک نام نمادین را به یک رویه اختصاص می‌دهند. برچسب می‌تواند NEAR یا FAR باشد. وقتی تعریف کامل قطعه بکار رود، نوع پیش‌فرض NEAR است. وقتی که تعریف ساده قطعه بکار رود، نوع برای فشرده یا مدل کوچک، NEAR است ولی برای مدل‌های متوسط، بزرگ و خیلی بزرگ، FAR است. برای اطلاعات بیشتر در مورد رویه‌ها به PROC در بخش ۱- پ مراجعه کنید.

### برچسب‌های قطعه

این برچسب‌ها به قطعات، نام نمادین می‌دهند. نام باید در رهنمون‌های SEGMENT و ENDS یکی باشند. SEGMENT را در بخش ۱- پ ملاحظه کنید. مثال:

|        |         |   |
|--------|---------|---|
| DAT_SG | SEGMENT |   |
| SUM    | DW      | ? |
| DAT_SG | ENDS    |   |

### برچسب‌های داده

این برچسب‌ها به اقلام داده، نام نمادین می‌دهند. این برچسب، امکان دستیابی به آنها را بوسیله دستورالعمل‌ها فراهم می‌سازد. رهنمون‌های DB، DW، DD، DQ و DT برای اختصاص داده بکار رفته‌اند.

|       |    |     |
|-------|----|-----|
| DATA1 | DB | 43H |
| DATA2 | DB | F2H |
| SUM   | DW | ?   |

### برچسب‌های تعریف شده با رهنمون LABEL

رهنمون LABEL می‌تواند یک برچسب را تعریف کند. در بخش ۱- پ به LABEL مراجعه کنید.

### نام‌های رزرو

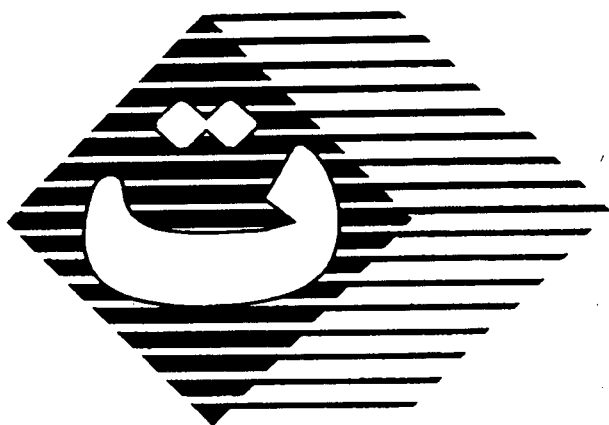
در زیر فهرستی از کلمات رزرو در برنامه نویسی زبان اسمبلی 80x86 ارائه شده است. این کلمات نمی‌توانند بوسیله کاربر برای برچسب‌ها و نام‌های متغیرها بکار روند.  
نام‌های ثابت‌ها:

|    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|
| AH | AL | AX | BH | BL | BP | BX | CH | CL | CS | CX | DH |
| DI | DL | DS | DX | ES | SI | SP | SS |    |    |    |    |

### دستورالعمل‌ها:

|       |        |        |       |       |      |
|-------|--------|--------|-------|-------|------|
| AAA   | AAD    | AAM    | AAS   | ADC   | ADD  |
| AND   | CALL   | CBW    | CLC   | CLD   | CLI  |
| CMC   | CMP    | CMPS   | CWD   | DAA   | DAS  |
| DEC   | DIV    | ESC    | HLT   | IDIV  | IMUL |
| IN    | INC    | INT    | INTO  | IRET  | JA   |
| JAE   | JB     | JBE    | JCXZ  | JE    | 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 |
| LDS   | LEA    | LES    | LOCK  | LODS  | LOOP |
| LOOPE | LOOPNE | LOOPNZ | LOOPZ | MOV   | MOVS |
| MUL   | NEG    | NIL    | NOP   | NOT   | OR   |
| OUT   | POP    | POPF   | PUSH  | PUSHF | RCL  |
| RCR   | REP    | REPE   | REPNE | REPZ  | REPZ |
| RET   | ROL    | ROR    | SAHF  | SAL   | SAR  |
| SBB   | SCAS   | SHL    | SHR   | STC   | STD  |
| STI   | STOS   | SUB    | TEST  | WAIT  | STG  |
| XLAT  | XOR    |        |       |       | XCHG |

|            |   |   |         |   |          |   |           |         |         |  |
|------------|---|---|---------|---|----------|---|-----------|---------|---------|--|
| \$         | * | + | -       | / | =        | ? | []        |         |         |  |
| ALIGN      |   |   | ASSUME  |   | BYTE     |   | COMM      | COMMENT | DB      |  |
| DD         |   |   | DF      |   | DOSSEG   |   | DQ        | DS      | DT      |  |
| DW         |   |   | DWORD   |   | DUP      |   | ELSE      | END     | ENDIF   |  |
| ENDM       |   |   | ENDS    |   | EQ       |   | EQU       | EVEN    | EXITM   |  |
| EXTRN      |   |   | FAR     |   | FWORD    |   | GE        | GROUP   | GT      |  |
| HIGH       |   |   | IF      |   | IFB      |   | IFDEF     | IFDIF   | IFE     |  |
| IFIDN      |   |   | IFNB    |   | IFNDEF   |   | IF1       | IF2     | INCLUDE |  |
| INCLUDELIB |   |   | IRP     |   | IRPC     |   | LABEL     | LE      | LENGTH  |  |
| LINE       |   |   | LOCAL   |   | LOW      |   | LT        | MACRO   | MASK    |  |
| MOD        |   |   | NAME    |   | NE       |   | NEAR      | NOTHING | OFFSET  |  |
| ORG        |   |   | PAGE    |   | PROC     |   | PTR       | PUBLIC  | PURGE   |  |
| QWORD      |   |   | RECORD  |   | REPT     |   | REPTRD    | SEG     | SEGMENT |  |
| SHORT      |   |   | SIZE    |   | STACK    |   | STRUC     | SUBTTL  | TBYTE   |  |
| THIS       |   |   | TITLE   |   | TYPE     |   | WIDTH     | WORD    |         |  |
| .186       |   |   | .286    |   | .286P    |   | .287      | .386    | .386P   |  |
| .387       |   |   | .8086   |   | .8087    |   | .ALPHA    | .CODE   | .CONST  |  |
| .CREF      |   |   | .DATA   |   | .DATA?   |   | .ERR      | .ERR1   | .ERR2   |  |
| .ERRB      |   |   | .ERRDEF |   | .ERRDIF  |   | .ERRE     | .ERRIDN | .ERRNB  |  |
| .ERRNDEF   |   |   | .ERRNZ  |   | .FARDATA |   | .FARDATA? | .LALL   | .LFCOND |  |
| .LIST      |   |   | .MODEL  |   | %OUT     |   | .RADIX    | .SALL   | .SEQ    |  |
| .SFCOND    |   |   | .STACK  |   | .TFCOND  |   | .TYPE     | .XALL   | .XREF   |  |
| .XLIST     |   |   |         |   |          |   |           |         |         |  |



---

## لیست وقفه DOS 21H

---

این پیوست بسیاری از وقفه‌های DOS 21H را که عمدتاً برای ورودی، خروجی، فایل و مدیریت حافظه بکار رفته‌اند توصیف می‌کند. ابتدا کلامی چند درباره مدیریت فایل تحت DOS بیان می‌شود. دو روش برای دستیابی به فایل در DOS وجود دارد. یکی روشی به نام شناسه فایل و دیگر FCB یا بلوک کنترل فایل است. این جملات بطور مشروح در زیر تعریف شده‌اند. توابع 0FH تا 28H برای دستیابی فایل‌ها از FCB استفاده می‌کنند. توابع 39H تا 62H از شناسه فایل استفاده می‌نمایند. فراخوانی‌های شناسه‌ای بسیار قوی‌تر و کاربرد ساده‌تری دارند. با این وجود فراخوانی‌های FCB سازگاری را برای DOS تا ویرایش 1.10 حفظ نمودند. FCBها محدودیت‌های دیگری هم دارند و آن ارجاع به فایل‌های جاری است، در حالی که شناسه‌ها به هر فایلی در هر لیستی (فهرستی) می‌توانند اشاره کنند. فراخوانی‌های FCB بلوک کنترل فایل را برای انجام هر کاری در فایل از آن استفاده می‌نمایند. شناسه‌ها از یک رشته ASCIIZ (در زیر تعریف شده) برای بازکردن، ایجاد، پاک کردن یا نام‌گذاری مجدد یک فایل استفاده می‌کنند و برای تقاضای I/O از یک شناسه فایل استفاده می‌نمایند. در لیست وقفه اصطلاحاتی وجود دارند که برای خوانندگان ناآشناست. کتاب‌های راهنمای DOS جزئیات کاملی از مدیریت فایل فراهم می‌سازند، اما در زیر چند نکته کلیدی آورده شده است.

### رشته ASCIIZ

این رشته متشکل از هر ترکیبی از کاراکترهای اسکی است که با یک بایت از صفرهای دودویی (00H) خاتمه می‌یابد. در فراخوانی‌های وقفه DOS 21H برای مشخص کردن نام فایل یا مسیر بکار می‌رود. در زیر مثالی از یک رشته ASCIIZ که در قطعه داده یک برنامه تعریف شده است دیده می‌شود.

```
NAME_1 DB 'C:\PROGRAMS\SYSTEM_A\PROGRAM5.ASM',0
```

## فهرست

DOS محل نام فایل را با فهرست دنبال می‌کند. هر دیسک می‌تواند به یک یا چند فهرست تقسیم‌بندی شود. لیست فهرست هر فایل در فهرست، تعداد بایت‌ها در فایل، تاریخ و زمان ایجاد فایل و دیگر اطلاعاتی را که DOS برای دستیابی به آن فایل نیاز دارد فراهم می‌سازد. فرمان "DIR" از DOS فهرست دیسک‌گردان جاری را برای ردیابی نشان می‌دهد.

## ناحیه انتقال دیسک DTA

این در واقع یک ناحیه بافری است که DOS برای نگهداری داده در خواندن یا نوشتن بوسیله فراخوانی FCB بکار می‌برد. این ناحیه می‌تواند با برنامه شما در هر مکانی از قطعه داده ایجاد شود. فراخوانی 1AH به DOS مکان DTA را می‌گوید. هر بار فقط یک DTA فعال است.

## جدول تخصیص فایل FAT

هر دیسک جدول تخصیص فایلی دارد که اطلاعات را درباره دسته‌ها در دیسک می‌دهد. هر دیسک به تعدادی قطاع تقسیم شده که در دسته‌هایی گروه بندی شده‌اند. اندازه قطاع و دسته‌ها در انواع دیسک‌ها متغیر است. برای هر دسته در دیسک، FAT دارای کدی است که در دسترس بودن، رزرو شدن یا به عنوان دسته خراب علامت خوردن یک دسته در حال بکارگیری بوسیله فایل را مشخص می‌نماید. DOS از این اطلاعات برای ذخیره و بازیابی فایل‌ها استفاده می‌کند.

## بلوک کنترل فایل FCB

به هر فایل باز یک FCB اختصاص می‌یابد. این بلوک متشکل از 37 داده است که اطلاعات لازم در مورد فایل، مانند دیسک‌گردان، نام فایل و توسعه، اندازه فایل برحسب بایت، و تاریخ و ساعت ایجاد آن را در اختیار می‌گذارد. همچنین شماره‌های بلوک و رکورد را ذخیره می‌نماید. این شماره‌ها به عنوان اشاره‌گر در هنگام نوشتن یا خواندن فایل بکار می‌روند. DOS INT 21H فراخوانی 0FH تا 28H از FCB‌ها برای دستیابی به فایل‌ها استفاده می‌کنند. تابع 0FH برای بازکردن فایل، 16H برای ایجاد فایل جدید بکار می‌روند. توابع 14H-28H اعمال خواندن/نوشتن را روی فایل انجام می‌دهند، و بالاخره 10H برای بستن فایل بکار می‌رود. معمولاً اطلاعات نام فایل با تابع 29H (parse filename) ایجاد می‌شود، و سپس آدرس FCB در DS:DX قرار گرفته و برای دستیابی به فایل بکار می‌رود.

## شناسه فایل

فراخوان تابع DOS، 3CH تا 62H از شناسه فایل استفاده می‌نمایند. وقتی که یک فایل برای وسیله‌ای با یکی از این توابع ایجاد یا باز شود، شناسه فایل آن برمی‌گردد. از آن پس شناسه فایل برای ورود، خروج، بستن فایل و غیره بکار می‌رود. DOS چند شناسه از پیش تعریف شده دارد که قابل استفاده بوسیله هر برنامه زبان اسمبلی است. آنها قبل از بکارگیری نیازی به باز شدن ندارند.

| مقدار شناسه | اشاره دارد به                            |
|-------------|------------------------------------------|
| 0000        | وسیله ورودی استاندارد (مثل صفحه کلید)    |
| 0001        | وسیله خروجی استاندارد (مثل مانیتور)      |
| 0002        | خطای وسیله خروجی استاندارد (مثل مانیتور) |
| 0003        | وسیله استاندارد کمکی                     |
| 0004        | وسیله چاپ استاندارد (PTR1)               |

## PSP پیشوند قطعه برنامه

PSP یک ناحیه 256 بایتی از حافظه است که بوسیله DOS برای هر برنامه ذخیره می‌شود. این ناحیه برای برنامه ذخیره اطلاعات مشترک بین برنامه و DOS بکار می‌رود.

- 00 برنامه را پایان می دهد**  
ثبات های فراخوانده شده اضافی  
 CS = آدرس قطعه PSP  
 (پیشوند قطعه برنامه)  
ثبات های نتیجه  
 هیچ  
 توجه: فایل ها باید قبلاً بسته شوند در غیر این صورت داده از دست می رود.
- 01 ورودی کلید با انعکاس**  
ثبات های فراخوانده شده اضافی  
 هیچ  
ثبات های نتیجه  
 AL = کاراکتر ورودی  
 توجه: Ctrl-break را جستجو می کند
- 02 کاراکتر خروجی به مانیتور**  
ثبات های فراخوانده شده اضافی  
 DL = کاراکتر مورد نمایش  
ثبات های نتیجه  
 هیچ
- 03 ورودی غیرهمزمان از وسیله کمکی (وسیله سریال)**  
ثبات های فراخوانده شده اضافی  
 هیچ  
ثبات های نتیجه  
 AL = کاراکتر ورودی
- 04 خروجی کاراکتر غیرهمزمان**  
ثبات های فراخوانده شده اضافی  
 DL = کاراکتری که باید خارج شود  
ثبات های نتیجه  
 هیچ
- 05 کاراکتر خروجی که باید چاپ شود**  
ثبات های فراخوانده شده اضافی  
 DL = کاراکتری که باید چاپ شود  
ثبات های نتیجه  
 هیچ
- 06 کنسول I/O**  
ثبات های فراخوانده شده اضافی  
 DL = OFFH به شرط ورودی، یا کاراکتر  
ثبات های نتیجه  
 0H=AL اگر کاراکتری موجود  
 نباشد  
 = کاراکتر ورودی، اگر ورودی  
 موفقیت آمیز باشد  
 توجه: اگر ورودی وجود دارد، ZF پاک شده و AL کاراکتر را داراست. ZF اگر ورودی وجود داشته  
 ولی کاراکتری نباشد.
- 07 ورود صفحه کلید بدون انعکاس**  
ثبات های فراخوانده شده اضافی  
 هیچ  
ثبات های نتیجه  
 AL = کاراکتر ورودی  
 توجه: به دنبال Ctrl-break نمی گردد.
- 08 ورود صفحه کلید بدون انعکاس**  
ثبات های فراخوانده شده اضافی  
 هیچ  
ثبات های نتیجه  
 AL = کاراکتر ورودی  
 توجه: به دنبال Ctrl-break می گردد.

09

## رشته خروجی

ثبات های اضافی

ثبات های فراخوانده شده اضافی

DS:DX = آدرس رشته

هیچ

توجه: کاراکترها را از آدرس تا مواجهه با \$(ASCII36) نمایش می دهد

0A

## رشته ورودی

ثبات های نتیجه

ثبات های فراخوانده شده اضافی

DS:DX = آدرسی که رشته ذخیره می شود

هیچ

توجه: حداکثر اندازه رشته در بایت 1 مشخص می شود و DOS اندازه واقعی رشته را در بایت 2 قرار می دهد. رشته از بایت 3 شروع می شود

0B

## بدست آوردن وضعیت کلمه صفحه کلید

ثبات های نتیجه

ثبات های فراخوانده شده اضافی

AL = 100 اگر هیچ کاراکتری منتظر نیست

هیچ

= 0FFH اگر کاراکتر منتظر است

توجه: Ctrl-break را واریسی می کند

0C

## بافر باز نشان ورودی و تابع ورودی صفحه کلید را فرامی خواند

ثبات های نتیجه

ثبات های فراخوانده شده اضافی

AL = شماره تابع صفحه کلید

هیچ

0AH, 08H, 07H, 06H, 01H

توجه: این تابع منتظر می ماند تا یک کاراکتر تایپ شود.

0D

## باز نشانیدن دیسک

ثبات های نتیجه

ثبات های فراخوانده شده اضافی

هیچ

هیچ

توجه: بافرهای فایل DOS را خالی می کند ولی فایل ها را نمی بندد

0E

## نشانیدن دیسک گردان پیش فرض

ثبات های نتیجه

ثبات های فراخوانده شده اضافی

AL = تعداد دیسک گردانهای

DL = کد دیسک گردان

منطقی در سیستم

(A=0, B=1, C=2 و غیره)

0F

## باز کردن فایل

ثبات های نتیجه

ثبات های فراخوانده شده اضافی

AL=100 اگر موفقیت هست

DS:DX = آدرس FCB

= 0FFH اگر فایل پیدا نشد

توجه: فهرست جاری را برای فایل جستجو می کند. اگر پیدا نشد، FCB را پر می کند

10

## بستن فایل

ثبات های نتیجه

ثبات های فراخوانده شده اضافی

AL=100 اگر موفقیت هست

DS:DX = آدرس FCB

= 0FFH اگر فایل پیدا نشد

توجه: همه بافرها را تخلیه می کند. اگر فایلی اصلاح شد فهرست را به هنگام می کند.



11

## جستجو برای تطبیق نام فایل

ثبات های فراخوانده شده اضافی

DS:DX = آدرس FCB

ثبات های نتیجه

AL=100 اگر تطبیق پیدا شد.

FFH=10 اگر تطبیق وجود ندارد.

توجه: نام فایل می تواند حاوی "?" و "\*" باشند.

12

## جستجو برای تطبیق بعدی

ثبات های فراخوانده شده اضافی

DS:DX = آدرس FCB

ثبات های نتیجه

AL=100 اگر تطبیق پیدا شد.

FFH=10 اگر تطبیقی نبود

توجه: این فراخوانی هنگامی صورت می گیرد که فراخوانی قبلی 11H و 12H با موفقیت همراه بوده است.

13

## پاک کردن فایل

ثبات های فراخوانده شده اضافی

DS:DX = آدرس FCB

ثبات های نتیجه

AL=100 اگر فایل (ها) پاک شدند.

FFH=10 اگر پاک نشدند.

توجه: همه فایل ها را در فهرست فایل تطبیق شده پاک می کند، به شرطی که از نوع فقط خواندنی نباشند فایل ها قبل از پاک شدن باید بسته شوند.

14

## خواندن ترتیبی

ثبات های فراخوانده شده اضافی

DS:DX = آدرس FCB باز شده

ثبات های نتیجه

AL=100 اگر خواندن موفق است

H=101 اگر انتهای فایل است و داده خوانده

نشده.

H=102 اگر DTA برای حفظ رکورد خیلی

کوچک است.

H=103 اگر بخشی از رکورد خوانده شده و

انتهای فایل فرارسیده است.

توجه: اشاره گر فایل، اشاره گر بلوک و اشاره گر رکورد FCB بطور خودکار بوسیله DOS بهنگام می شوند.

15

## نوشتن ترتیبی

ثبات های فراخوانده شده اضافی

DS:DX = آدرس FCB باز شده

ثبات های نتیجه

H=100 اگر نوشتن با موفقیت باشد.

H=101 اگر دیسک پر است.

H=102 اگر DTA برای نگهداری رکورد

خیلی کوچک است.

توجه: اشاره گر فایل، اشاره گر بلوک و اشاره گر رکورد FCB بطور خودکار بوسیله DOS بهنگام می شوند. رکورد تا یک دسته یا فایل بسته نشود، عملاً نوشته نمی شود.

16

## ایجاد/بازکردن فایل

ثبات های فراخوانده شده اضافی

DS:DX = آدرس FCB باز نشده

ثبات های نتیجه

H=100 اگر با موفقیت

FFH=0 اگر ناموفق

توجه: اگر فایل از قبل وجود دارد به طول 0 کوتاه می شود

|    |                                                                                                                                                                                                                                                                                                       |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 17 | نام‌گذاری مجدد فایل (ها)<br><u>ثبات‌های فراخوانده شده اضافی</u><br>FCB = DS:DX<br><u>ثبات‌های نتیجه</u><br>00H=AL اگر فایل (ها) نام‌گذاری مجدد شود.<br>0FFH= اگر فایل پیدا نشد یا نام جدید از قبل وجود دارد.<br>توجه: نام قدیم در محل نام FCB قرار دارد؛ نام جدید در مکان اندازه (تفاوت مکان 16) است. |
| 18 | رزرو                                                                                                                                                                                                                                                                                                  |
| 19 | بدست آوردن دیسک‌گردان پیش‌فرض<br><u>ثبات‌های فراخوانده شده اضافی</u><br>هیچ<br><u>ثبات‌های نتیجه</u><br>0H=AL برای دیسک‌گردان A<br>1H= برای دیسک‌گردان B<br>2H= برای دیسک‌گردان C                                                                                                                     |
| 1A | مشخص کردن DTA (آدرس انتقال دیسک)<br><u>ثبات‌های فراخوانده شده اضافی</u><br>DTA=DS:DX<br><u>ثبات‌های نتیجه</u><br>هیچ<br>توجه: تنها یک DTA در هر لحظه می‌تواند در جریان باشد. این تابع باید قبل از خواندن، نوشتن و جستجوی فهرست FCB فراخوانی شود                                                       |
| 1B | بدست آوردن FAT (جدول تخصیص فایل) برای دیسک‌گردان پیش‌فرض<br><u>ثبات‌های فراخوانده شده اضافی</u><br>هیچ<br><u>ثبات‌های نتیجه</u><br>AL= تعداد قطاع در هر دسته<br>CX= تعداد بایت‌ها در قطاع<br>DX= تعداد دسته در دیسک<br>DS:BX FAT id                                                                   |
| 1C | بدست آوردن (جدول تخصیص فایل) برای هر دیسک‌گردان<br><u>ثبات‌های فراخوانده شده اضافی</u><br>DL= کد دیسک‌گردان<br>0 برای A<br>1 برای B<br>2 برای C<br><u>ثبات‌های نتیجه</u><br>AL= تعداد قطاع در هر دسته<br>CX= تعداد بایت‌ها در قطاع<br>DX= تعداد دسته در دیسک<br>DS:BX FAT id                          |
| 1D | رزرو                                                                                                                                                                                                                                                                                                  |
| 1E | رزرو                                                                                                                                                                                                                                                                                                  |
| 1F | رزرو                                                                                                                                                                                                                                                                                                  |
| 20 | رزرو                                                                                                                                                                                                                                                                                                  |
| 21 | خواندن تصادفی                                                                                                                                                                                                                                                                                         |

ثبات‌های فراخوانده شده اضافی  
DS:DX = آدرس FCB باز شده

ثبات‌های نتیجه  
AL=00H اگر خواندن موفق است  
01H= اگر پایان فایل است ولی خواندن ناموفق  
02H= اگر DTA برای رکورد خیلی کوچک است  
03H= اگر انتهای فایل است ولی بخشی خوانده شده

توجه: رکورد اشاره شده با بلوک جاری میدان‌های رکورد را به DTA می‌خواند.

22

### نوشتن تصادفی

ثبات‌های فراخوانده شده اضافی  
DS:DX = آدرس FCB باز شده

ثبات‌های نتیجه  
AL=00H اگر نوشتن موفق

01= اگر دیسک پر است  
02= اگر برای رکورد کوچک است

توجه: از DTA در رکوردی که بوسیله بلوک جاری و میدان‌های رکورد اشاره می‌شود، می‌نویسد.

23

### بدست آوردن اندازه فایل

ثبات‌های فراخوانده شده اضافی  
DS:DX = آدرس FCB باز نشده رکوردها

ثبات‌های نتیجه  
AL=00H اگر فایل پیدا شد، عدد

در میدان رکورد FCB تصادفی  
(تفاوت مکان 0021H) قرار می‌گیرد.  
0FFH= اگر تطبیق پیدا نشد

توجه: FCB قبل از وقفه باید حاوی اندازه رکورد باشد.

24

### ایجاد میدان رکورد تصادفی

ثبات‌های فراخوانده شده اضافی  
DS:DX = آدرس FCB باز شده

ثبات‌های نتیجه  
هیچ

توجه: این میدان رکورد - تصادفی (تفاوت مکان 0021H) را در FCB ایجاد می‌کند. این تابع قبل از سوئیچ کردن از پردازش ترتیبی به تصادفی بکار می‌رود.

25

### ایجاد بردار وقفه

ثبات‌های فراخوانده شده اضافی  
DS:DX = آدرس FCB باز شده

ثبات‌های نتیجه  
هیچ

AL= شماره وقفه ماشین

توجه: این برای تغییر روش سرویس دهی سیستم به وقفه‌ها بکار می‌رود.

26

### ایجاد یک PSP جدید (پیشوند قطعه برنامه)

ثبات‌های فراخوانده شده اضافی  
DX= آدرس قطعه PSP جدید

ثبات‌های نتیجه  
هیچ

توجه: ویرایش DOS 2.0 و بالاتر پیشنهاد می‌کنند از آن استفاده نکنید بلکه از سرویس 4B (exec) استفاده نمایید.

## خواندن بلوک تصادفی

ثبات‌های فراخوانده شده اضافی

DS:DX = آدرس FCB باز شده

CX = تعداد رکوردهای خواندن

ثبات‌های نتیجه

AL=100H اگر خواندن موفق

H=01H اگر انتهای فایل و داده خوانده

نشده

DTA اگر 02H= برای بلوک کوچک است

EOF اگر 03H= و بلوک جزیی خوانده شد

CX = تعداد رکوردهای خوانده شده واقعی

توجه: قبل از وقفه، رکورد FCB تصادفی و میدان اندازه رکورد را ایجاد می‌کند. DOS رکورد تصادفی بلوک جاری و میدان‌های رکورد جاری را پس از خواندن به‌نگام می‌سازد.

## نوشتن بلوک تصادفی

ثبات‌های فراخوانده شده اضافی

DS:DX = آدرس FCB باز شده

CX = تعداد رکوردها برای نوشتن

DTA اگر 02= برای بلوک کوچک است

CX = تعداد رکوردهایی که واقعاً نوشته شده

توجه: قبل از وقفه، رکورد FCB تصادفی و میدان اندازه رکورد را ایجاد می‌نماید. DOS رکورد تصادفی، بلوک جاری و میدان‌های رکورد جاری را پس از نوشتن به‌نگام می‌سازد. اگر قبل از وقفه CX=0 باشد چیزی در فایل نوشته نمی‌شود و فایل که بوسیله رکورد تصادفی و میدان‌های اندازه رکورد محاسبه می‌شود کوچک یا بزرگ خواهد شد.

## تجزیه نام فایل

ثبات‌های فراخوانده شده اضافی

DS:DX = آدرس FCB فرمان خط

ES:DI = آدرس FCB

باز نشده و فرمت نشده

0=Bit 1 اگر جداکننده‌های پیشرو چشم پوشی

شوند؛ در غیراینصورت قطع پویش رخ نمی‌دهد.

1=Bit 1 دیسک‌گردان ID در FCB تنها هنگامی

عوض می‌شود که دیسک‌گردان در خط فرمان

مشخص شده باشد.

2=Bit 1 نام فایل فقط هنگامی عوض می‌شود که

نام فایل در خط فرمان ذکر شده باشد

3=Bit 3 توسعه فقط هنگامی تغییر می‌کند که در خط

فرمان توسعه مشخص شده باشد

توجه: خط فرمان برای یک نام فایل Parsed می‌شود، سپس یک FCB باز نشده در DS:SI ایجاد می‌گردد. اگر نام مسیرها مشخصند فرمان نباید بکار رود.

## بدست آوردن تاریخ سیستم

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

CX = سال 1980-2099

DH = ماه (1-12)

DL = روز (1-31)

AL = کد روز هفته (0=یکشنبه، ....

.....6=شنبه)

## AH

### عملکرد

2B

#### تنظیم تاریخ سیستم

ثبات‌های فراخوانده شده اضافی

CX = سال (1980-2099)

DH = ماه (1-12)

DL = روز (1-31)

ثبات‌های نتیجه

AL=00H اگر تاریخ تنظیم است

=0FFH اگر تاریخ نامعتبر است

2C

#### بدست آوردن زمان سیستم

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

CH = ساعت (0-23)

CL = دقیقه (0-59)

DH = ثانیه (0-59)

DL = صدم ثانیه (0-99)

توجه: قالب برگشتی می‌تواند در محاسبات بکار رود ولی قابل تبدیل به قالب چاپ شدنی نیز هست

2D

#### تنظیم زمان سیستم

ثبات‌های فراخوانده شده اضافی

CH = ساعت (0-23)

CL = دقیقه

DH = ثانیه

DL = صدم ثانیه

ثبات‌های نتیجه

AL=00H اگر فرمان تنظیم است

=0FFH اگر زمان نامعتبر

2E

#### سوئیچ تأیید نشاندن/بازنشان (بازنشانندن)

ثبات‌های فراخوانده شده اضافی

AL=0 برای خاموش کردن تأیید

=1 برای روشن کردن تأیید

ثبات‌های نتیجه

هیچ

توجه: اگر تأیید روشن است، هر بار که داده در دیسک بنویسد DOS تأیید خواهد کرد. فراخوانی وقفه 54H تنظیم سوئیچ تأیید را بدست می‌آورد.

2F

#### بدست آوردن DTA (ناحیه انتقال دیسک)

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

ES:BX = آدرس DTA

30

#### بدست آوردن شماره ویرایش

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

AL = شماره ویرایش بالاتر

AH = شماره ویرایش پایین‌تر

31

#### پایان داده فرآیند و مقیم ماندن (KEEP process)

ثبات‌های فراخوانده شده اضافی

AL = کد دودویی بازگشت

ثبات‌های نتیجه

هیچ

DX = اندازه حافظه در پاراگراف

توجه: این فراخوانی وقفه فرآیند جاری را پایان داده و سعی در قراردادن اندازه حافظه در پاراگراف در بلوک تخصیص اولیه می‌کند، ولی تخصیص بلوک‌های دیگر را رها نمی‌سازد. کد بازگشتی در AL بوسیله فرآیند اولیه با استفاده از وقفه 21 فراخوانی 4DH است.

## کنترل Ctrl-break

33

ثبات‌های فراخوانده شده اضافی  
 ثبات‌های نتیجه  
 00=AL برای بدست آوردن حالت تست Ctrl-break  
 00=DL اگر تست ctrl-break خاموش است

01= برای اصلاح حالت تست Ctrl-break  
 00=DL برای خاموش کردن تست  
 01= برای روشن کردن تست  
 توجه: وقتی که تست Ctrl-break خاموش است، DOS زمان تست ورودی Ctrl-break را کاهش می‌دهد. وقتی که روشن باشد، DOS در اغلب عملیات Ctrl-break را تست می‌کند.

34

## بدست آوردن آدرس بردار وقفه

35

ثبات‌های فراخوانده شده اضافی  
 ثبات‌های نتیجه  
 =AL شماره وقفه  
 =ES:BX آدرس سرویس وقفه

## بدست آوردن فضای خالی دیسک

36

ثبات‌های فراخوانده شده اضافی  
 ثبات‌های نتیجه  
 =DL کد دیسک‌گردان  
 (=0 پیش فرض  
 A=1، B=2 و غیره)  
 =AX FFFFH اگر کد دیسک‌گردان نامعتبر  
 = قطع در هر دسته اگر معتبر  
 =BX تعداد دسته‌های موجود  
 =CX بایت در هر قطعه  
 =DX کل دسته‌ها در دیسک‌گردان

37

## اطلاعات متعلق به کشور

38

ثبات‌های فراخوانده شده اضافی  
 ثبات‌های نتیجه  
 =DS:DX آدرس بلوک 32 بایت از حافظه  
 =AL کد تابع

## ایجاد زیرفهرست (MKDIR)

39

ثبات‌های فراخوانده شده اضافی  
 ثبات‌های نتیجه  
 =DS:DX آدرس ASCII نام مسیر  
 زیرفهرست جدید  
 =AX 13 اگر مسیر پیدا نشد  
 =11 اگر ناموفق  
 =15 اگر دستیابی نفی شد

## حذف فهرست (RMDIR)

3A

ثبات‌های فراخوانده شده اضافی  
 ثبات‌های نتیجه  
 =DS:DX آدرس ASCII نام مسیر  
 زیرفهرست جدید  
 =AX 13 اگر مسیر پیدا نشد  
 =15 اگر فهرست خالی نباشد  
 =15 اگر دیسک‌گردان نامعتبر  
 توجه: فهرست جاری حذف نمی‌شود

3B

## تغییر زیر فهرست جاری (CHDIR)

ثبات‌های نتیجه  
پرچم نقلی = 10 اگر موفق

ثبات‌های فراخوانده شده اضافی  
DS:DX = آدرس نام مسیر ASCIIZ  
زیر فهرست جدید

= 1 اگر ناموفق  
AX = 3 اگر مسیر پیدا نشد

3C

## ایجاد فایل

ثبات‌های نتیجه  
پرچم نقلی = 10 اگر موفق  
= 1 اگر ناموفق  
AX = شناسه اگر موفق  
= 3 اگر مسیر پیدا نشد  
= 5 اگر دستیابی نفی شد

ثبات‌های فراخوانده شده اضافی  
DS:DX = آدرس ASCIIZ مسیر و نام فایل  
CX = خصیصه فایل

توجه: یک فایل جدید را اگر وجود نداشته باشد ایجاد می‌نماید، در غیر اینصورت طول فایل را صفر می‌کند. فایل را برای نوشتن یا خواندن باز می‌کند. یک شناسه 16 بیتی در AX باز می‌گردد، به شرطی که ایجاد موفقیت آمیز باشد.

3D

## باز کردن فایل

ثبات‌های نتیجه  
پرچم نقلی = 10 اگر موفق  
= 1 اگر ناموفق  
AX = شناسه فایل شانزده بیتی اگر موفق  
= 1 اگر شماره تابع نامعتبر  
= 2 اگر فایل پیدا نشد  
= 3 اگر مسیر پیدا نشد  
= 4 اگر شناسه موجود نیست  
= 5 اگر دستیابی نفی شد  
= 10C اگر کد دستیابی نامعتبر

ثبات‌های فراخوانده شده اضافی  
DS:DX = آدرس ASCIIZ مسیر و نام فایل

## خلاصه پرچم AL

7 6 5 4 3 2 1 0 (بیت‌ها)

0 0 0  
0 0 1  
0 1 0  
0

0 0 0  
0 0 1  
0 1 0  
0 1 1  
1 0 0

0

1

## نتیجه

باز برای خواندن

باز برای نوشتن

باز برای خواندن/نوشتن

رزرو

به دیگران دستیابی سازگار می‌دهد

دستیابی خواندن/نوشتن به دیگران نفی شد

دستیابی نوشتن به دیگران نفی

دستیابی خواندن به دیگران نفی

دستیابی کامل به دیگران بده

فایل موروثی بوسیله فرآیند دیگر

فایل اختصاصی برای فرآیند جاری

## فایل بسته

ثبات‌های فراخوانده شده اضافی

BX = شناسه فایل

ثبات‌های نتیجه

پرچم نقلی = 0 اگر موفق

= 1 اگر ناموفق

AX = 6 اگر شناسه نامعتبر و یا فایل باز نیست

توجه: همه بافرهای داخلی قبل از بسته شدن خالی شوند.

## خواندن از فایل یا از وسیله

ثبات‌های فراخوانده شده اضافی

DS:DX = آدرس بافر

BX = شناسه فایل

CX = تعداد بایت‌های خواندنی

ثبات‌های نتیجه

پرچم نقلی = 0 اگر موفق

= 1 اگر ناموفق

AX = تعداد بایت خوانده شده

= 5 اگر دستیابی نفی شد

= 6 اگر فایل باز نیست یا شناسه نامعتبر

توجه: وقتی که از وسیله استاندارد خوانده شود (صفحه کلید)، یک خط از متن خوانده خواهد شد. این خواندن بدون توجه به CX انجام می‌شود.

## نوشتن در فایل یا وسیله

ثبات‌های فراخوانده شده اضافی

DS:DX = آدرس بافر

BX = شناسه فایل

CX = تعداد بایت‌های نوشتنی

ثبات‌های نتیجه

پرچم نقلی = 0 اگر موفق

= 1 اگر ناموفق

AX = تعداد بایت‌های نوشتنی اگر موفق

= 5 اگر دستیابی نفی شد

= 6 اگر فایل باز نیست یا شناسه نامعتبر

توجه: اگر پرچم نقلی پاک باشد و AX کمتر از CX باشد، یک قسمتی از رکورد نوشته شده یا دیسک بر یا خطایی مشابه رخ داده است.

## پاک کردن فایل

ثبات‌های فراخوانده شده اضافی

DS:DX = آدرس ASCIIZ مسیر و نام فایل

ثبات‌های نتیجه

پرچم نقلی = 0 اگر موفق

= 1 اگر ناموفق

AX = 2 اگر فایل پیدا نشد

= 5 اگر دستیابی نفی شد

توجه: این تابع برای فایلی که فقط خواندنی است نمی‌تواند بکار رود. ابتدا خصیصه فایل را با استفاده از وقفه 21 فراخوانی 43H، به 0 تغییر دهید. در نام کاراکتر عام نمی‌تواند بکار رود. این تابع با پاک کردن ورودی فهرست برای فایل کار می‌کند.

## انتقال اشاره‌گر فایل (LSEEK)

ثبات‌های فراخوانده شده اضافی

BX = شناسه فایل

CX:DX = تفاوت مکان

AL = 0 برای انتقال بایت‌های تفاوت

مکان اشاره‌گر از شروع فایل

= 1 برای انتقال بایت‌های تفاوت مکان

اشاره‌گر از محل فعلی

= 2 برای انتقال بایت‌های تفاوت مکان از انتهای فایل

توجه: برای تعیین اندازه فایل با AX = 2 و تفاوت مکان = 0 فراخوانی کنید.

ثبات‌های نتیجه

پرچم نقلی = 0 اگر موفق

= 1 اگر ناموفق

AX = 1 اگر شماره تابع ناموفق

= 6 اگر فایل باز نیست یا شناسه معتبر

AX = DX = تفاوت مکان مطلق از شروع

فایل اگر موفق



## تنظیم یا بدست آوردن مد فایل (CHMOD)

ثبات‌های فراخوانده شده اضافی

DS:DX = آدرس مشخص گر فایل

AL=0H برای بدست آوردن خصیصه

01H= برای تنظیم

CX= خصیصه به شرط تنظیم

= کد خصیصه به شرط بدست آوردن

(به قسمت زیر مراجعه شود)

ثبات‌های نتیجه

پرچم نقلی = 0 اگر موفق

= 1 اگر ناموفق

CX= خصیصه جاری به شرط تنظیم

AX= 1 اگر شماره تابع نامعتبر

= 2 اگر فایل پیدا نشد

= 13 اگر فایل موجود نیست یا مسیر پیدا نشد

= 15 اگر خصیصه قابل تعویض نباشد

## بیت‌های کد خصیصه

رزرو

رزرو

آرشیو

فهرست (با 43H تنظیم نکنید؛

از توسعه FCB استفاده کنید)

سیستم

پنهان

فقط خواندنی

7 6 5 4 3 2 1 0

0

0

x

0

x

x

x

## کنترل وسیله I/O (IOCTL)

ثبات‌های فراخوانده شده اضافی

AL=0H بدست آوردن اطلاعات وسیله

ثبات‌های نتیجه

AX = تعداد بایت‌های انتقالی اگر

CF=0

در غیر اینصورت = کد خطا

01H= برای تنظیم اطلاعات وسیله

02H= خواندن کاراکتر وسیله به بافر

03H= نوشتن کاراکتر بافر به وسیله

04H= خواندن بلوک بافر به وسیله

05H= نوشتن بلوک بافر به وسیله

06H= تست وضعیت ورودی

07H= تست وضعیت خروجی

08H= تست اگر بلوک وسیله قابل تعویض

09H= تست اگر دیسک‌گردان محلی یا دوردست

0AH= تست اگر شناسه محلی یا دوردست

0BH= برای تغییر اشتراک شماره تلاش

0CH= کنترل I/O وسیله کاراکتر

0DH= کنترل I/O وسیله بلوک

0EH= نقشه را برای دیسک‌گردان منطقی بدست آور

0FH= نقشه را برای دیسک‌گردان منطقی ایجاد کن

DS:DX = بافر داده

BX = شناسه فایل؛ CX = تعداد بایت‌ها

45

## تکرار شناسه فایل (DUP)

ثبات‌های فراخوانده شده اضافی  
 BX = شناسه فایل باز شده

## ثبات‌های نتیجه

پرچم نقلی = 10 اگر موفق

= 11 اگر ناموفق

= 14 اگر شناسه موجود نیست

= 16 اگر شناسه نامعتبر یا باز نشده

توجه: دو شناسه به توالی کار خواهند کرد. مثلاً اگر اشاره گر فایل یک شناسه انتقال یافت دیگری هم منتقل می‌شود.

46

## واداشتن کپی یک شناسه (FORCDUP)

ثبات‌های فراخوانده شده اضافی

BX = اولین شناسه فایل

CX = دومین شناسه فایل

## ثبات‌های نتیجه

پرچم نقلی = 10 اگر موفق

= 11 اگر ناموفق

AX = 14 اگر شناسه موجود نیست

= 16 اگر شناسه نامعتبر یا باز نشده

توجه: اگر فایل ارجاع شده با CX باز باشد؛ ابتدا بسته خواهد شد. دومین شناسه فایل به نقطه یکسانی در شناسه فایل اول واداشته می‌شود. دو شناسه به توالی کار خواهند کرد؛ مثلاً اگر اشاره گر فایل از یک شناسه انتقال یابد دیگری هم انتقال می‌یابد

47

## بدست آوردن فهرست جاری

ثبات‌های فراخوانده شده اضافی

DL = کد دیسک گردان

(پیش فرض = 0، 1=A، ....)

DS:SI آدرس بافر 64 بایتی

## ثبات‌های نتیجه

پرچم نقلی = 10 اگر موفق

= 11 اگر ناموفق

DS:SI مشخص گر مسیر

AX = OFH اگر مشخص گر دیسک گردان

نامعتبر

توجه: نام مسیر بازگشته شامل اطلاعات دیسک گردان یا بارکردن نیست.

48

## تخصیص حافظه

ثبات‌های فراخوانده شده اضافی

BX = تعداد پاراگراف

## ثبات‌های نتیجه

پرچم نقلی = اگر موفق

= اگر ناموفق

AX = اگر موفق، اشاره به بلوک

= 7 اگر بلوک‌های کنترل حافظه تخریب

شده‌اند

= 18 اگر حافظه ناکافی

BX = اندازه بزرگترین بلوک موجود اگر

ناموفق

49

## آزاد کردن تخصیص حافظه

ثبات‌های فراخوانده شده اضافی

ES = آدرس قطعه بلوک آزاد شده

## ثبات‌های نتیجه

پرچم نقلی = 10 اگر موفق

= 11 اگر ناموفق

AX = 17 اگر بلوک‌های کنترل حافظه تخریب

شده‌اند

= 19 اگر آدرس بلوک حافظه نامعتبر در ES

توجه: حافظه تخصیص یافته را با 48H آزاد می‌کند.

4A

## اصلاح تخصیص حافظه (SETBLOCK)

ثبات‌های نتایج  
ثبات‌های فراخوانده شده اضافی  
 ES = آدرس قطعه بلوک  
 BX = اندازه بلوک جدید درخواستی در پاراگراف  
 ثبات‌های نتایج  
 پرچم نقلی 0 = اگر موفق  
 1 = اگر ناموفق  
 BX = اندازه حداکثر بلوک موجود اگر ناموفق  
 AX = 7 اگر بلوک‌های کنترل حافظه از بین رفته است  
 8 = اگر حافظه کافی نیست  
 9 = اگر آدرس بلوک حافظه نامعتبر در ES  
 توجه: بصورت دینامیکی حافظه اختصاص یافته در فراخوانی قبل از وقفه 21 تابع 48H را افزایش یا کاهش می‌دهد.

4B

## بار و اجرای برنامه (EXEC)

ثبات‌های نتایج  
ثبات‌های فراخوانده شده اضافی  
 DS:SI = آدرس ASCIIZ مسیر و نام فایل  
 برای بار کردن  
 ES:BX = آدرس بلوک پارامتر  
 AL = 0 برای بار و اجرا  
 3 = برای بار ولی نه اجرا

4C

## اتمام فرآیند (EXIT)

ثبات‌های نتایج  
ثبات‌های فراخوانده شده اضافی  
 AL = کد دودویی بازگشتی  
 توجه: یک فرآیند را پایان می‌بخشد، و کنترل را به فرآیند اصلی یا DOS بازمی‌گرداند. کد بازگشت می‌تواند در AL بازگردانده شود

4D

## بدست آوردن کد بازگشت از یک زیرفرآیند (WAIT)

ثبات‌های نتایج  
ثبات‌های فراخوانده شده اضافی  
 AL = کد بازگشت  
 AH = 00 اگر اتمام معمولی  
 01 = اگر اتمام با Ctrl-Break  
 02 = اگر با خطای بحرانی وسیله  
 03 = اگر با فراخوانی وقفه 21 تابع 31H  
 توجه: کد ارسالی از طریق وقفه 21 تابع 4CH را بازمی‌گرداند. کد فقط یکبار می‌تواند بازگردد.

4E

## جستجو برای اولین تطبیق (FIND FIRST)

ثبات‌های نتایج  
ثبات‌های فراخوانده شده اضافی  
 DS:DI = آدرس مشخصات فایل  
 CX = خصیصه بکار رفته در جستجو  
 ثبات‌های نتایج  
 پرچم نقلی 0 = اگر موفق  
 1 = اگر ناموفق  
 AX = کد خطا  
 توجه: نام فایل باید حاوی یک یا چند کاراکتر جانشین باشد. قبل از این فراخوانی، یک فراخوانی قبلی به وقفه 21 تابع 1AH باید آدرس DTA را برقرار کند. اگر نام فایل منطبقی یافت شد، DTA فعلی با مقادیر زیر بار می‌شود.  
 بایت‌های 0-20: بوسیله DOS برای فراخوانی‌های جستجو بعدی رزرو شده است.

- 21: خصیصه یافت شده  
 22-23: زمان فایل  
 24-25: تاریخ فایل  
 26-27: اندازه فایل (کلمه کم ارزشتر)  
 28-29: اندازه فایل (کلمه با ارزشتر)  
 30-42: مشخصات فایل ASCIIZ

### جستجو برای تطبیق نام فایل بعدی (FIND NEXT)

4F

ثبات های فراخوانده شده اضافی  
 هیچ  
ثبات های نتیجه  
 پرچم نقلی = 10 اگر موفق  
 1 اگر ناموفق  
 AX = کد خطا

توجه: DTA فعلی باید بوسیله وقفه 21H تابع 4EH یا 4F پر شود. DTA همانطور که در وقفه 21 تابع 4E گفته شد پر می شود.

50 رزرو

51 رزرو

52 رزرو

53 رزرو

### بدست آوردن حالت تایید

54

ثبات های فراخوانده شده اضافی  
 هیچ  
ثبات های نتیجه  
 AL = 100 اگر تایید خاموش  
 101 اگر تایید روشن

توجه: حالت پرچم تایید از طریق وقفه 21 تابع 2EH عوض می شود

55 رزرو

### تغییر نام فایل

56

ثبات های فراخوانده شده اضافی  
 DS:DX = آدرس ASCIIZ  
 مشخصات نام فایل قدیم  
 ES:DI = آدرس ASCIIZ  
 مشخصات نام فایل جدید  
ثبات های نتیجه  
 پرچم نقلی = 10 اگر موفق  
 1 اگر ناموفق  
 AX = 12 اگر فایل پیدا نشود  
 13 اگر مسیر یا فایل پیدا نشود  
 15 اگر دستیابی نفی شود  
 11H اگر وسیله متفاوتی در نام جدید

توجه: اگر مشخصات دیسک گردانی بکار رفت، در مشخصات نام فایل قدیم و جدید باید یکسان باشد. با این وجود نام فهرست ممکن است متفاوت باشد تا اجازه انتقال و نام گذاری مجدد را در یک نوبت میسر سازد.

## بدست آوردن / تنظیم تاریخ و زمان

|                              |                           |
|------------------------------|---------------------------|
| ثبات های فراخوانده شده اضافی | ثبات های نتیجه            |
| 00=AL برای بدست آوردن        | پرچم نقلی = 0 اگر موفق    |
| 01= برای تنظیم               | 1= اگر ناموفق             |
| BX= شناسه فایل               | CX= زمان بدست آوردن       |
| CX= زمان تنظیم               | DX= تاریخ بدست آوردن      |
| DX= تاریخ تنظیم              | AX= 1 اگر کد تابع نامعتبر |
|                              | 16= اگر شناسه نامعتبر     |

توجه: فایل قبل از وقفه باید باز باشد. قالب تاریخ و زمان به ترتیب زیر است:

|                                     |                              |
|-------------------------------------|------------------------------|
| زمان:                               | تاریخ                        |
| بیت ها 0BH-0FH ساعت (0-23)          | بیت ها 09H-0FH سال (از ۱۹۸۰) |
| 05H-0AH دقیقه (0-59)                | 05H-08H ماه (0-12)           |
| 00H-04H تعداد 2 ثانیه افزایش (0-29) | 00H-04H روز (0-31)           |

## استراتژی تخصیص بدست آوردن / تنظیم

|                                |                             |
|--------------------------------|-----------------------------|
| ثبات های فراخوانده شده اضافی   | ثبات های نتیجه              |
| 00=AL برای بدست آوردن استراتژی | پرچم نقلی = 0 اگر موفق      |
| 01= برای تنظیم استراتژی        | 1= اگر ناموفق               |
| BX= استراتژی اگر تنظیم         | AX= استراتژی اگر بدست آوردن |
| 100 اگر اولین fit              | = کد خطا اگر تنظیم          |
| 101 اگر بهترین fit             |                             |
| 102 اگر آخرین fit              |                             |

## بدست آوردن اطلاعات خطای توسعه یافته

|                              |                         |
|------------------------------|-------------------------|
| ثبات های فراخوانده شده اضافی | ثبات های نتیجه          |
| 00=BX                        | AX= کد خطای توسعه یافته |
|                              | (جدول ۱ - ت ملاحظه شود) |
|                              | BH= رده خطا             |
|                              | BL= راه حل پیشنهادی     |
|                              | CH= مکان خطا            |

احتیاط! این تابع محتوای ثبات های CL، DX، SI، DI، BP، DS و ES را از بین می برد. کدهای خطا یا ویرایش های آینده DOS تغییر خواهد کرد.

## ایجاد فایل موقت

|                              |                          |
|------------------------------|--------------------------|
| ثبات های فراخوانده شده اضافی | ثبات های نتیجه           |
| DS:DX= آدرس مسیر ASCIIZ      | پرچم نقلی = 0 اگر موفق   |
| CX= خصیصه فایل               | 1= اگر ناموفق            |
| 101 اگر فقط خواندنی          | AX= شناسه اگر ناموفق     |
| 102 اگر پنهان، 104 اگر سیستم | = کد خطا اگر ناموفق      |
|                              | DS:DX= آدرس خصیصیات مسیر |
|                              | ASCIIZ اگر موفق          |

توجه: فایلی که با این تابع وقفه تولید می شود و به هنگام پایان برنامه پاک نمی شود.

5B

## ایجاد فایل جدید

ثبات‌های فراخوانده شده اضافی

DS:DX = آدرس ASCIIZ مشخصات فایل

CX = خصیصه فایل

100 اگر معمولی،

101 اگر فقط خواندنی

102 اگر پنهان

104 اگر سیستم

ثبات‌های نتیجه

پرچم نقلی = 10 اگر موفق

1 اگر ناموفق

AX = شناسه اگر موفق

= کد خطا اگر ناموفق

توجه: این تابع مشابه وقفه 21 تابع 3CH کار می‌کند؛ با این وجود اگر فایل قبلاً وجود داشته باشد تابع ناموفق است، در صورتی که تابع 3CH طول فایل را صفر می‌کند.

5C

## دستیابی رکورد کنترل

ثبات‌های فراخوانده شده اضافی

AL = 00 برای قفل

01 برای غیر قفل

BX = شناسه فایل

CX:DX = ناحیه تفاوت مکان

SI:DI = طول ناحیه

توجه: رکوردها را در سیستم‌هایی که چند وظیفه‌ای یا شبکه را پشتیبانی کنند قفل یا غیر قفل می‌کند.

5D

## رزرو

5E 00

## بدست آوردن نام ماشین

ثبات‌های فراخوانده شده اضافی

DS:SI = آدرس بافر

ثبات‌های نتیجه

پرچم نقلی = 10 اگر موفق

1 اگر ناموفق

CH = 10 اگر نام تعریف نشده

10 اگر نام تعریف شده

CL = تعداد NETBIOS اگر موفق

DS:DX = آدرس معرف اگر موفق

AX = کد خطا

توجه: 15 بایت رشته ASCIIZ معرف کامپیوتر را بازمی‌گرداند.

5E 02

## تنظیم نصب چاپگر

ثبات‌های فراخوانده شده اضافی

BX = اندیس لیست تغییر مسیر

CX = طول رشته نصب

DS:SI = آدرس رشته نصب

ثبات‌های نتیجه

پرچم نقلی = 10 اگر موفق

1 اگر ناموفق

AX = کد خطا

توجه: این تابع رشته‌ای را مشخص می‌کند که مقدم بر همه فایل‌های ارسال شده به چاپگر شبکه از گروه محلی در LAN است. شبکه‌های مایکروسافت باید در حال کار باشند تا این تابع بکار رود.

5E 03

## بدست آوردن نصب چاپگر

ثبات‌های فراخوانده شده اضافی

BX = اندیس لیست تغییر مسیر

ES:DI = آدرس بافر

ثبات‌های نتیجه

پرچم نقلی = 10 اگر موفق

1 اگر ناموفق

AX = کد خطا

CX = طول رشته نصب

ES:DI = تنظیم رشته اگر موفق

5F 02

## بدست آوردن لیست تغییر مسیر

ثبات‌های فراخوانده شده اضافی

BX = اندیس لیست تغییر مسیر

DS:SI = آدرس بافر نام وسیله 16 بیتی

ES:DI = آدرس بافر نام شبکه 128 بیتی

ثبات‌های نتیجه

پرچم نقلی = 0 اگر موفق

1 اگر ناموفق

BH = پرچم وضعیت وسیله

بیت 1 = 0 اگر وسیله معتبر

1 = اگر وسیله نامعتبر

BL = نوع وسیله

CX = مقدار پارامتر

DS:SI = آدرس ASCII نام وسیله محلی

ES:DI = آدرس ASCII نام شبکه

AX = پرچم خطا

5F 03

## تغییر مسیر وسیله

ثبات‌های فراخوانده شده اضافی

BL = نوع وسیله

03 چاپگر

04 دیسک‌گردان

ثبات‌های نتیجه

پرچم وقفه = 0 اگر موفق

1 اگر ناموفق

AX = کد خطا

CX = مقدار فراخوانده

DS:SI = آدرس ASCII نام وسیله محلی

ES:DI = آدرس ASCII نام شبکه

توجه: هنگام کار LAN عمل می‌کند، این تابع اجازه افزودن وسایل را به مسیر تغییر لیست شبکه فراهم می‌سازد.

5F 04

## حذف تغییر مسیر

ثبات‌های فراخوانده شده اضافی

DS:SI = آدرس ASCII نام وسیله محلی

ثبات‌های نتیجه

پرچم نقلی = 0 اگر موفق

1 اگر ناموفق

AX = کد خطا

توجه: هنگام کار LAN عمل می‌کند، این تابع اجازه حذف وسایل را از لیست تغییر مسیر شبکه فراهم می‌سازد.

رژرو

60

رژرو

61

## بدست آوردن PSP (پیشوند قطعه برنامه)

62

ثبات‌های فراخوانده شده اضافی

ثبات‌های نتیجه

BX = آدرس PSP

هیچ

خلاصه‌ای از کدهای خطای IBM در جدول ۱ - ت داده شده است.

جدول ۱ - ت اطلاعات کد خطای توسعه یافته

| کد | خطا                                | کد    | خطا                                 |
|----|------------------------------------|-------|-------------------------------------|
| 1  | شماره تابع نامعتبر                 | 36    | سرریز بافر اشتراکی                  |
| 2  | فایل پیدا نشد                      | 37-49 | رزرو                                |
| 3  | مسیر پیدا نشد                      | 50    | تقاضا شبکه پشتیبانی نشد             |
| 4  | چندین فایل باز                     | 51    | کامپیوتر دوردست در جریان نیست       |
| 5  | دستیابی نفی شد                     | 52    | نام تکراری در شبکه                  |
| 6  | شناسه نامعتبر                      | 53    | نامه شبکه پیدا نشد                  |
| 7  | بلوک‌های کنترل حافظه تخریب شده‌اند | 54    | شبکه مشغول                          |
| 8  | حافظه ناکافی                       | 55    | وسيله شبکه وجود ندارد               |
| 9  | آدرس بلوک حافظه نامعتبر            | 56    | از محدوده فرمان net BIOS فراتر شد   |
| 10 | محیط نامعتبر                       | 57    | خطای سخت‌افزاری آداپتور شبکه        |
| 11 | فرمت نامعتبر                       | 58    | پاسخ غیر صحیح از شبکه               |
| 12 | کد دستیابی نامعتبر                 | 59    | خطای غیرمنتظره از شبکه              |
| 13 | داده نامعتبر                       | 60    | آداپتور دوردست ناسازگار             |
| 14 | واحد ناشناخته                      | 61    | صف چاپ پر                           |
| 15 | دیسک‌گردان نامعتبر                 | 62    | فضای کافی برای فایل چاپ موجود نیست  |
| 16 | سعی در حذف فهرست جاری              | 63    | فایل چاپ پاک شده است                |
| 17 | همان وسیله نیست                    | 64    | نام شبکه پیدا نشد                   |
| 18 | فایل دیگری نیست                    | 65    | دستیابی نفی شد                      |
| 19 | سعی در نوشتن در دیسک حفاظت شده     | 66    | نوع وسیله شبکه غیر صحیح             |
| 20 | واحد ناشناخته                      | 67    | نام شبکه پیدا نشد                   |
| 21 | دیسک‌گردان آماده نیست              | 68    | از حد نام شبکه فراتر شد             |
| 22 | فرمان ندانسته                      | 69    | از حد زمانی net BIOS فراتر شد       |
| 23 | خطای داده (CRC)                    | 70    | مکث موقت                            |
| 24 | طول ساختار تقاضای بد               | 71    | تقاضای شبکه پذیرفته نشد             |
| 25 | خطای جستجو                         | 72    | تغییر مسیر دیسک یا چاپ نگه‌داشته شد |
| 26 | نوع محیط ناشناخته                  | 73-79 | رزرو                                |
| 27 | قطاع پیدا نشد                      | 80    | فایل موجود است                      |
| 28 | چاپگر بدون کاغذ                    | 81    | رزرو                                |
| 29 | خطای نوشتن                         | 82    | نمی‌توان وارده فهرست ساخت           |
| 30 | خطای خواندن                        | 83    | عدم موفقیت در INT24                 |
| 31 | خرابی کلی                          | 84    | چندین تغییر مسیر                    |
| 32 | تخطی از اشتراک                     | 85    | تغییر مسیر کمی                      |
| 33 | تخطی از قفل                        | 86    | مجوز نامعتبر                        |
| 34 | تغییر نامعتبر دیسک                 | 87    | پارامتر نامعتبر                     |
| 35 | FCB در دسترس نیست                  | 88    | خطای وسیله شبکه                     |



بدست آوردن اطلاعات کلید فشرد

05

| <u>ثبات های نتیجه</u>     | <u>ثبات های فراخوانده شده اضافی</u> |
|---------------------------|-------------------------------------|
| AX = وضعیت دکمه           | BX = دکمه 0 برای چپ                 |
| بیت 0 - دکمه چپ           | 1 برای راست                         |
| بیت 1 - دکمه راست         | 2 برای مرکز                         |
| بیت 2 - دکمه وسط          |                                     |
| = 0 اگر بالا؛ 1 اگر پائین |                                     |
| BX = تعداد فشار دکمه      |                                     |
| CX = مکان افقی            |                                     |
| DX = مکان عمودی           |                                     |

توجه: این تابع وضعیت همه دکمه ها را همراه با تعداد فشردن های مشخص شده در BX به هنگام فراخوانی مشخص می کند، مکان اشاره گر ماوس به پیکسل ها را داده و مکان آخرین فشار دکمه را نمایش می دهد.

بدست آوردن اطلاعات کلید آزاد شده

06

| <u>ثبات های نتیجه</u>           | <u>ثبات های فراخوانده شده اضافی</u> |
|---------------------------------|-------------------------------------|
| AX = وضعیت دکمه                 | BX = دکمه: 0 برای چپ؛ 1 برای راست   |
| بیت 0 - دکمه چپ                 | 2 برای وسط                          |
| 1 - دکمه راست                   |                                     |
| 2 - برای دکمه وسط               |                                     |
| = 0 اگر بالا؛ 1 اگر پائین       |                                     |
| BX = تعداد دفعات آزاد کردن دکمه |                                     |
| CX = مکان افقی                  |                                     |
| DX = مکان عمودی                 |                                     |

توجه: این تابع وضعیت همه دکمه را همراه با تعداد دفعات آزادسازی مشخص شده در BX را هنگام فراخوانی معین می کند. مکان اشاره گر ماوس به پیکسل داده شده و مکان آخرین آزاد سازی را نشان می دهد.

تنظیم محدوده افقی برای اشاره گر ماوس

07

| <u>ثبات های نتیجه</u> | <u>ثبات های فراخوانده شده اضافی</u> |
|-----------------------|-------------------------------------|
| هیچ                   | CX = حداقل مکان افقی                |
|                       | DX = حداکثر مکان افقی               |

توجه: این تابع محدوده افقی را (برحسب پیکسل) برای اشاره گر ماوس تنظیم می کند. پس از این فراخوانی، ماوس در این محدوده دیده خواهد شد.

تنظیم محدوده عمودی برای اشاره گر ماوس

08

| <u>ثبات های نتیجه</u> | <u>ثبات های فراخوانده شده اضافی</u> |
|-----------------------|-------------------------------------|
| هیچ                   | CX = حداقل مکان عمودی               |
|                       | DX = حداکثر مکان عمودی              |

توجه: این تابع محدوده عمودی را برای اشاره گر ماوس (برحسب پیکسل) تنظیم می کند. پس از فراخوانی، ماوس در این محدوده ها ظاهر می گردد.

10

## تنظیم اشاره گر ماوس در ناحیه انحصاری

ثبات های نتیجه

هیچ

ثبات های فراخوانده شده اضافی

CX = مختصات افقی چپ بالا

DX = مختصات عمودی چپ بالا

SI = مختصات افقی راست پائین

DI = مختصات عمودی راست پائین

توجه: این تابع ناحیه ای را تعریف می کند که در آن اشاره گر ماوس نمایش داده نمی شود. ناحیه انحصاری می تواند با فراخوانی توابع 00 یا 01 منتفی شود.

24

## بدست آوردن اطلاعات ماوس

ثبات های نتیجه

BH = بخش بزرگتر ویرایش

BL = بخش کوچکتر ویرایش

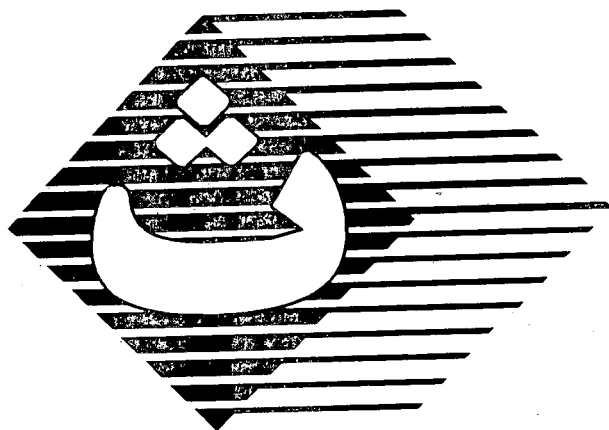
CH = نوع ماوس

CL = شماره IRQ

هیچ

توجه: این تابع شماره ویرایش (مثلاً ویرایش 7.5 : BH=7 ، BL=5)

نوع ماوس: برای گذرگاه؛ 2 برای سریال؛ 3 برای Inport؛ 4 برای PS/2؛ 5 برای HP ؛ IRQ برای PS/2؛ در غیر اینصورت 2، 3، 4، 5 یا 7.



# وقفه‌های BIOS

این پیوست مهمترین وقفه‌های BIOS را پوشش می‌دهد. INT10H منحصرأً برای برنامه‌نویسی گرافیک بکار می‌رود. INT11H آرایش وسایل و INT 12H اندازه حافظه را باز می‌گرداند. INT14H برای تبادل اطلاعات غیرهمزمان بکار می‌رود. دو تابع INT15H نیز آورده شده‌اند: یکی برای آغاز انتظار و دیگری برای اندازه حافظه توسعه یافته. INT16H برای صفحه کلید و INT17H برای چاپگر می‌باشند. INT1AH تایمر و RTC را اداره می‌کند.

## بخش ۱- ت: INT 10H فراخوانی تابع تصویر

| عملکرد                                                                | AH |
|-----------------------------------------------------------------------|----|
| برقراری مد تصویر                                                      | 00 |
| ثبات‌های فراخوانده شده اضافی                                          |    |
| AL = مد تصویر                                                         |    |
| جدول ۲- ت را برای لیستی از مدهای تصویر موجود و تعریف آنها ملاحظه کنید |    |
| تنظیم نوع مکان‌نما                                                    | 01 |
| ثبات‌های فراخوانده شده اضافی                                          |    |
| CH = خط شروع مکان‌نما                                                 |    |
| (بیت‌های ۰-۴)                                                         |    |
| CL = خط پایان مکان‌نما                                                |    |
| (بیت‌های ۰-۴)                                                         |    |

توجه: همه بیت‌های دیگر باید در صفر تنظیم شوند. چشمک مکان‌نما سخت‌افزاری کنترل می‌شود.

02

## تنظیم محل مکان نما

ثبات های فراخوانده شده اضافی

BH = شماره صفحه

DH = سطر

DL = ستون

ثبات های نتیجه

هیچ

توجه: هنگام استفاده از مد گرافیک، BH باید صفر شود. مختصات گوشه سمت چپ بالا (0,0) خواهد بود.

03H

## خواندن محل مکان نما و اندازه آن

ثبات های فراخوانده شده اضافی

BH = شماره صفحه

ثبات های نتیجه

CH = خط شروع مکان نما

CL = خط پایان مکان نما

DH = سطر

DL = ستون

توجه: وقتی مد گرافیک بکار می رود، BH باید صفر شود

04H

## خواندن مکان قلم نوری

ثبات های فراخوانده شده اضافی

هیچ

ثبات های نتیجه

AH = 0 اگر قلم نوری فعال نشده

= 1 اگر قلم نوری فعال شده

BX = ستون پیکسل

CH = سطر پیکسل (مدهای 04H-06H)

CX = سطر پیکسل (مدهای 0DH-13H)

DH = سطر کارا کتر

DL = ستون کارا کتر

05H

## انتخاب صفحه نمایش فعال

ثبات های فراخوانده شده اضافی

AL = شماره صفحه

ثبات های نتیجه

هیچ

(جدول ۱ - ت ملاحظه شود)

جدول ۱ - ت: نمایش صفحات برای مدها و آداپتورهای متفاوت

| مد  | صفحات | آداپتور          |
|-----|-------|------------------|
| 00H | 0-7   | CGA EGA MCGA VGA |
| 01H | 0-7   | CGA EGA MCGA VGA |
| 02H | 0-3   | CGA              |
|     | 0-7   | EGA MCGA VGA     |
| 03H | 0-3   | CGA              |
|     | 0-7   | EGA MCGA VGA     |
| 07H | 0-7   | EGA VGA          |
| 0DH | 0-7   | EGA VGA          |
| 0EH | 0-3   | EGA VGA          |
| 0FH | 0-1   | EGA VGA          |
| 10H | 0-1   | EGA VGA          |

سایر ترکیبات مد - آداپتور فقط یک صفحه را پشتیبانی می کنند.

06

## حرکت طوماری پنجره به سوی بالا

ثبات‌های نتیجه

هیچ

ثبات‌های فراخوانده شده اضافی

AL = تعداد خطوط حرکت طوماری

BH = خصیصه نمایش

CH = مختصات y از سمت چپ بالا

CI = مختصات x از چپ بالا

DH = مختصات y از راست پائین

DL = مختصات x از راست پائین

توجه: اگر  $AL=0$  باشد، تمام پنجره تاریک است. در غیراینصورت صفحه نمایش به تعداد خطوط در AL به سمت بالا حرکت طوماری خواهد داشت. خطوطی که از بالای صفحه خارج می‌شوند از دست خواهند رفت. خطوط تاریک بر طبق خصیصه در BH از پائین وارد می‌شوند.

07

## حرکت طوماری پنجره به سوی پائین

ثبات‌های نتیجه

ثبات‌های فراخوانده شده اضافی

AL = تعداد خطوط حرکت طوماری

BH = خصیصه نمایش

CH = مختصات y از سمت چپ بالا

CL = مختصات x از سمت چپ بالا

DH = مختصات y از سمت راست پائین

DL = مختصات x از سمت راست پائین

08

## خواندن کاراکتر و خصیصه در محل مکان نما

ثبات‌های نتیجه

ثبات‌های فراخوانده شده اضافی

AH = بایت خصیصه

BH = صفحه نمایش

AL = کد اسکی کاراکتر

09

## نوشتن کاراکتر و خصیصه در محل مکان نما

ثبات‌های نتیجه

ثبات‌های فراخوانده شده اضافی

هیچ

AL = کد اسکی کاراکتر

BH = صفحه نمایش

BL = خصیصه

CX = تعداد کاراکترهای نوشتن

توجه: محل مکان نما را بهنگام نمی‌کند. برای تنظیم مکان نما از وقفه 10 تابع 2 استفاده کنید.

0A

## نوشتن کاراکتر در محل مکان نما

ثبات‌های نتیجه

ثبات‌های فراخوانده شده اضافی

هیچ

AL = کد اسکی کاراکتر

BH = صفحه نمایش

BL = رنگ گرافیک

CX = تعداد کاراکترهای نوشتن

توجه: کاراکتر (ها) را با خصیصه تصویری موجود بنویسید، محل مکان نما را بهنگام نمی‌کند. از وقفه 10 تابع 2 برای تنظیم محل مکان نما استفاده کنید.

**AH****عملکرد****0B****تنظیم جعبه رنگ**ثبات های نتیجه

هیچ

ثبات های فراخوانده شده اضافی

BH=00H تنظیم مرز یا رنگ زمینه

01=تنظیم جعبه رنگ

BL=جعبه رنگ/رنگ

توجه: اگر BH=00H و در مد متن باشد، این تابع رنگ مرز را فقط تنظیم می کند. اگر BH=00H و در مد گرافیک، تابع زمینه و رنگ مرز را تنظیم می نماید. اگر BH=01H، این تابع جعبه رنگ را انتخاب خواهد کرد. در گرافیک چهار رنگ 200 x 320، جعبه رنگ 0 و 1 در دسترسند:

**رنگ پیکسل ها برای جعبه رنگ 0 و 1**

| پیکسل | جعبه رنگ 0  | جعبه رنگ 1 |
|-------|-------------|------------|
| 0     | زمینه       | زمینه      |
| 1     | سبز         | آبی        |
| 2     | قرمز        | بنفش       |
| 3     | قهوه ای/زرد | سفید       |

**0C****نوشتن پیکسل**ثبات های نتیجه

هیچ

ثبات های فراخوانده شده اضافی

AL=مقدار پیکسل

CX=ستون پیکسل

DX=سطر پیکسل

BH=صفحه

توجه: مختصات و مقدار پیکسل به مد تصویر جاری بستگی دارند. تنظیم بیت 7 از AL موجب می شود تا مقدار AL با مقدار پیکسل XOR شود.

**0D****خواندن پیکسل**ثبات های نتیجه  
AL=مقدار پیکسلثبات های فراخوانده شده اضافی

CX=ستون پیکسل

DX=سطر پیکسل

BH=صفحه

**0E****خروج کاراکتر TTY**ثبات های نتیجه

هیچ

ثبات های فراخوانده شده اضافی

AL=کاراکتر

BH=صفحه

BL=رنگ پیش زمینه

توجه: کاراکتر را برای نمایش می نویسد و محل مکان نما را بهنگام می کند. مد TTY حداقل پردازش کاراکتر را داراست. کدهای ASCII برای رنگ، پسر، تعویض خط، برگشت بصورت عمل مناسبی ترجمه می شوند.

0F

بدست آوردن مد تصویر

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

AH = عرض صفحه نمایش بر حسب کاراکتر

AL = مد تصویر

BH = صفحه نمایش

توجه: جدول ۲ - ث برای ملاحظه مدهای ممکن تصویر ملاحظه شود.

10 00

زیر تابع 00H: تنظیم ثبات جعبه با رنگ مربوطه

ثبات‌های فراخوانده شده اضافی

00H=AL

BH = رنگ

CL = ثبات جعبه رنگ

(0FH تا 00H)

ثبات‌های نتیجه

هیچ

10 01

زیر تابع 01H: تنظیم رنگ محدوده

ثبات‌های فراخوانده شده اضافی

01H=AL

BH = رنگ محدوده (مرز)

ثبات‌های نتیجه

هیچ

10 02

زیر تابع 02H: تنظیم جعبه رنگ و محدوده

ثبات‌های فراخوانده شده اضافی

02H=AL

ES:DX = آدرس لیست رنگ

ثبات‌های نتیجه

هیچ

13

نوشتن رشته

ثبات‌های فراخوانده شده اضافی

AL = مد نوشتن

00 = خصیصه در BL

مکان نما حرکت نکرده است

01H = خصیصه در BL مکان نما حرکت کرده است

02H = خصیصه کاراکتر را دنبال می‌کند،

مکان نما حرکت نکرده است

03H = خصیصه کاراکتر را دنبال می‌نماید،

مکان نما حرکت کرده است

ES:BP = آدرس رشته

CX = تعداد کاراکتر

DH = مکان اولیه سطر

DL = مکان اولیه ستون

BH = صفحه

ثبات‌های نتیجه

هیچ

توجه: برای 00H و 01H، رشته فقط متشکل است از کاراکترها، که همه با خصیصه موجود در BL نمایش داده می‌شوند. برای 02H و 03H داده با خصیصه‌ها (کاراکتر، خصیصه، کاراکتر، خصیصه و غیره) ذخیره می‌شود.

جدول ۲- ث مدهای تصویر تعریف آنها

| AL  | Pixels   | کاراکترها | جعبه متن/گرافیک کاراکتر | رنگها | آداپتور | حداکثر صفحات | بافر شروع |
|-----|----------|-----------|-------------------------|-------|---------|--------------|-----------|
| 00H | 320x200  | 40 x 25   | 8x8 text                | 16 *  | CGA     | 8            | B8000h    |
|     | 320x350  | 40 x 25   | 8x14 text               | 16 *  | EGA     | 8            | B8000h    |
|     | 360x400  | 40 x 25   | 9x16 text               | 16 *  | VGA     | 8            | B8000h    |
|     | 320x400  | 40 x 25   | 8x16 text               | 16 *  | MCGA    | 8            | B8000h    |
| 01H | 320x200  | 40 x 25   | 8x8 text                | 16    | CGA     | 8            | B8000h    |
|     | 320x350  | 40 x 25   | 8x14 text               | 16    | EGA     | 8            | B8000h    |
|     | 360x400  | 40 x 25   | 9x16 text               | 16    | VGA     | 8            | B8000h    |
|     | 320x400  | 40 x 25   | 8x16 text               | 16    | MCGA    | 8            | B8000h    |
| 02H | 640x200  | 80 x 25   | 8x8 text                | 16 *  | CGA     | 8            | B8000h    |
|     | 640x350  | 80 x 25   | 8x14 text               | 16 *  | EGA     | 8            | B8000h    |
|     | 720x400  | 80 x 25   | 9x16 text               | 16 *  | VGA     | 8            | B8000h    |
|     | 640x400  | 80 x 25   | 8x16 text               | 16 *  | MCGA    | 8            | B8000h    |
| 03H | 640x200  | 80 x 25   | 8x8 text                | 16    | CGA     | 8            | B8000h    |
|     | 640x350  | 80 x 25   | 8x14 text               | 16    | EGA     | 8            | B8000h    |
|     | 720x400  | 80 x 25   | 9x16 text               | 16    | VGA     | 8            | B8000h    |
|     | 640x400  | 80 x 25   | 8x16 text               | 16    | MCGA    | 8            | B8000h    |
| 04H | 320x200  | 40 x 25   | 8x8 graph               | 4     | CGA     | 1            | B8000h    |
|     | 320x200  | 40 x 25   | 8x8 graph               | 4     | EGA     | 1            | B8000h    |
|     | 320x200  | 40 x 25   | 8x8 graph               | 4     | VGA     | 1            | B8000h    |
|     | 320x200  | 40 x 25   | 8x8 graph               | 4     | MCGA    | 1            | B8000h    |
| 05H | 320x200  | 40 x 25   | 8x8 graph               | 4 *   | CGA     | 1            | B8000h    |
|     | 320x200  | 40 x 25   | 8x8 graph               | 4 *   | EGA     | 1            | B8000h    |
|     | 320x200  | 40 x 25   | 8x8 graph               | 4 *   | VGA     | 1            | B8000h    |
|     | 320x200  | 40 x 25   | 8x8 graph               | 4 *   | MCGA    | 1            | B8000h    |
| 06H | 640x200  | 80 x 25   | 8x8 graph               | 2     | CGA     | 1            | B8000h    |
|     | 640x200  | 80 x 25   | 8x8 graph               | 2     | EGA     | 1            | B8000h    |
|     | 640x200  | 80 x 25   | 8x8 graph               | 2     | VGA     | 1            | B8000h    |
|     | 640x200  | 80 x 25   | 8x8 graph               | 2     | MCGA    | 1            | B8000h    |
| 07H | 720x350  | 80 x 25   | 9x14 text mono          |       | MDA     | 8            | B0000h    |
|     | 720x350  | 80 x 25   | 9x14 text mono          |       | EGA     | 4            | B0000h    |
|     | 720x400  | 80 x 25   | 9x16 text mono          |       | VGA     | 8            | B0000h    |
| 08H | reserved |           |                         |       |         |              |           |
| 09H | reserved |           |                         |       |         |              |           |
| 0AH | reserved |           |                         |       |         |              |           |
| 0BH | reserved |           |                         |       |         |              |           |
| 0CH | reserved |           |                         |       |         |              |           |
| 0DH | 320x200  | 40 x 25   | 8x8 graph               | 16    | EGA     | 2/4          | A0000h    |
|     | 320x200  | 40 x 25   | 8x8 graph               | 16    | VGA     | 8            | A0000h    |
| 0EH | 640x200  | 80 x 25   | 8x8 graph               | 16    | EGA     | 1/2          | A0000h    |
|     | 640x200  | 80 x 25   | 8x8 graph               | 16    | VGA     | 4            | A0000h    |
| 0FH | 640x350  | 80 x 25   | 9x14 graph mono         |       | EGA     | 1            | A0000h    |
|     | 640x350  | 80 x 25   | 8x14 graph mono         |       | VGA     | 2            | A0000h    |
| 10H | 640x350  | 80 x 25   | 8x14 graph              | 4     | EGA     | 1/2          | A0000h    |
|     | 640x350  | 80 x 25   | 8x14 graph              | 16    | VGA     | 2            | A0000h    |
| 11H | 640x480  | 80 x 30   | 8x16 graph              | 2     | VGA     | 1            | A0000h    |
|     | 640x480  | 80 x 30   | 8x16 graph              | 2     | MCGA    | 1            | A0000h    |
| 12H | 640x480  | 80 x 30   | 8x16 graph              | 16    | VGA     | 1            | A0000h    |
| 13H | 320x200  | 40 x 25   | 8x8 graph               | 256   | VGA     | 1            | A0000h    |
|     | 320x200  | 40 x 25   | 8x8 graph               | 256   | MCGA    | 1            | A0000h    |

\* color burst off



## بخش ۲- ث: INT 11H - تعیین تجهیزات

عملکرد

AH

بدست آوردن آرایش تجهیزات

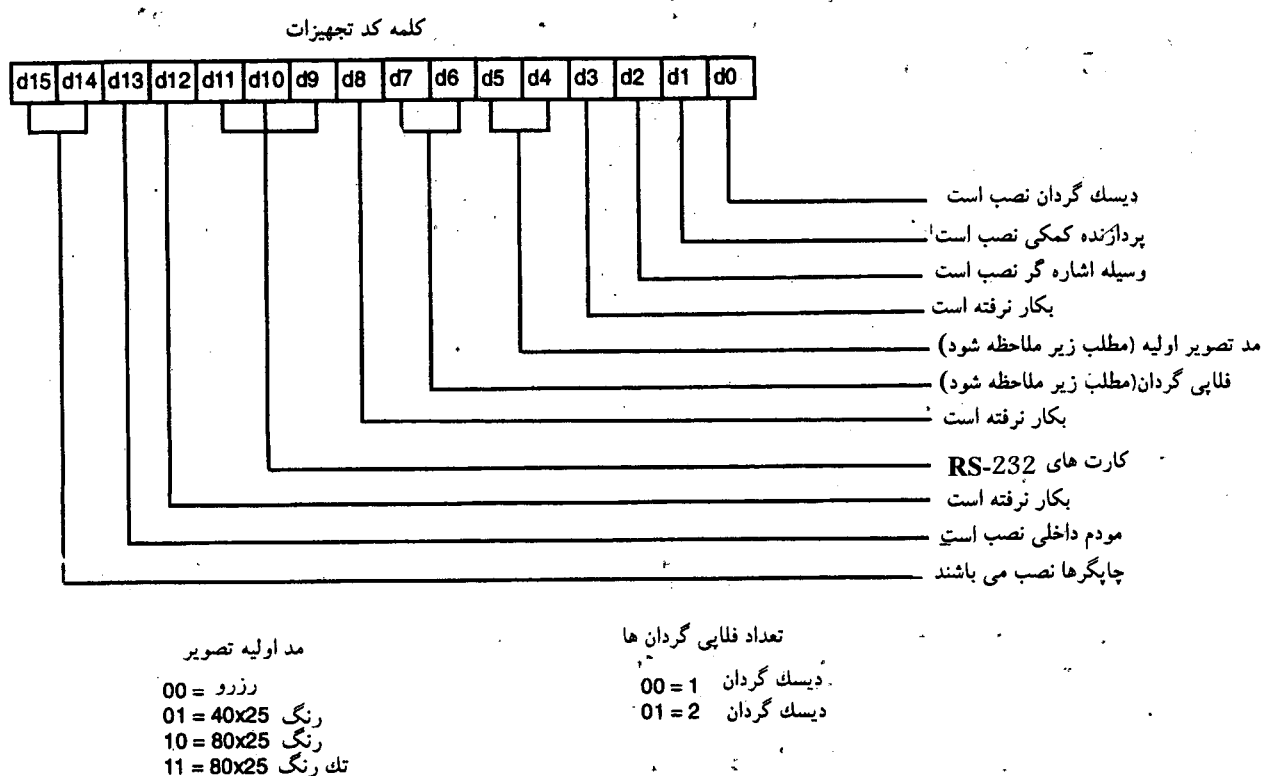
ثبات های نتیجه

ثبات های فراخوانده شده اضافی

هیچ AX = کد دستگاه (شکل زیر ملاحظه شود)

هیچ

توجه: ناحیه داده BIOS ، 40:10 بنابر کلمه کد که تجهیزات اختیاری متصل به سیستم را نشان می دهد، طی مرحله POST تنظیم می شود.



## بخش ۳- ث: INT 14H - تعیین اندازه حافظه

بدست آوردن اندازه حافظه متعارف

ثبات های نتیجه

ثبات های فراخوانده شده اضافی

AX = اندازه حافظه (KB)

هیچ

توجه: مقدار حافظه متعارف موجود را برای DOS و برنامه کاربردی فراهم می آورد

## بخش ۴ - ث: INT 14H - تبادل داده غیرهمزمان

| عملکرد                                    | AH                                              |
|-------------------------------------------|-------------------------------------------------|
| مقدار دهی اولیه به پورت COM               | 00                                              |
| <u>ثبات‌های نتیجه</u>                     | <u>ثبات‌های فراخوانده شده اضافی</u>             |
| AH = وضعیت پورت (به مطالب زیر مراجعه شود) | AL = پارامتر (به مطلب زیر توجه شود)             |
| AL = وضعیت مودم (به مطالب زیر مراجعه شود) | DX = شماره پورت (0 اگر COM1، 1 اگر COM2 و غیره) |

تذکر ۱: بایت پارامتر در AL به طریق زیر تعریف می‌شود

| مفهوم                              | 7 6 5 4 3 2 1 0 |
|------------------------------------|-----------------|
| میزان باود (110=000، 150=001)      | x x x           |
| 1200=100 600=011 300=010           |                 |
| 9600=111 4800=110 2400=101         |                 |
| توازن (01=فرد، 11=زوج، x0=هیچکدام) | x x             |
| بیت‌های توقف (1=0، 2=1)            | x               |
| طول کلمه (10=7بیت، 11=8بیت)        | x x             |

تذکر ۲: وضعیت پورت در AH طبق جدول زیر تعریف می‌شود

| مفهوم                   | 7 6 5 4 3 2 1 0 |
|-------------------------|-----------------|
| وقت تمام                | 1               |
| ثبات جابجایی ارسال خالی | 1               |
| ثبات نگهداری ارسال خالی | 1               |
| توقف شناسایی شد         | 1               |
| خطای قاب شناسایی شد     | 1               |
| خطای توازن شناسایی شد   | 1               |
| خطای عقب ماندن          | 1               |
| داده رسیده آماده        | 1               |

تذکر ۳: وضعیت مودم که در AL بازگشته به صورت زیر تعریف می‌شود:

| مفهوم                                 | 7 6 5 4 3 2 1 0 |
|---------------------------------------|-----------------|
| سیگنال خط دریافتی شناسایی شد          | 1               |
| نشانه رنگ                             | 1               |
| DSP (مجموعه داده آمار)                | 1               |
| CTS (پاک برای ارسال)                  | 1               |
| تغییر در سیگنال خط دریافتی شناسایی شد | 1               |
| لبه انتهایی زنگ شناسایی شد            | 1               |
| تغییر در وضعیت DSR                    | 1               |
| تغییر در وضعیت CTS                    | 1               |

نوشتن کاراکتر در پورت COM

ثبات‌های فراخوانده شده اضافی

AL = کاراکتر

DX = شماره پورت (0 اگر COM1

1 اگر COM2، و غیره)

ثبات‌های نتیجه

بیت 7 از AH = 0 اگر موفق

بیت 7 از AH = 1 اگر ناموفق

بیت 0-6 از AH = وضعیت اگر موفق

AL = کاراکتر

توجه: بایت وضعیت در AH، بیت‌های 0-6 پس از فراخوانی به قرار زیر است

6 5 4 3 2 1 0

1

1

1

1

1

1

1

مفهوم

ثبات جابجایی ارسال، خالی

ثبات نگهداری ارسال، خالی

توقف شناسایی شد

خطای قاب شناسایی شد

خطای توازن شناسایی شد

خطای عقب ماندن

داده رسیده آماده

خواندن کاراکتر از پورت COM

ثبات‌های فراخوانده شده اضافی

DX = شماره پورت (0 اگر COM1،

1 اگر COM2 و غیره)

ثبات‌های نتیجه

بیت 7 از AH = 0 اگر موفق، 1 اگر ناموفق

بیت 0-6 از AH = وضعیت اگر موفق

AL = کاراکتر خوانده شده

توجه: بایت وضعیت در AH بیت‌های 1-4 پس از خواندن به قرار زیر است

4 3 2 1

1

1

1

1

مفهوم

توقف شناسایی شد

خطای قاب شناسایی شد

خطای توازن شناسایی شد

خطای عقب ماندن شناسایی شد

وضعیت خواندن پورت COM

ثبات‌های فراخوانده شده اضافی

DX = شماره پورت (0 اگر COM1،

1 اگر COM2 و غیره)

ثبات‌های نتیجه

AH = وضعیت پورت

AL = وضعیت مودم

نتیجه: وضعیت پورت و وضعیت مودم که در AH و AL بازگشته‌اند دارای قالب یکسانی با INT14H

تابع 00H هستند که در بالا تشریح شد.

## پورت COM توسعه یافته

ثبات‌های فراخوانده شده اضافی

00H=AL (توقف) 01H (بدون توقف)

DX= شماره پورت (0 اگر COM1، 1 اگر COM2)

BH= توازن

00H= هیچ

01H= فرد

02H= زوج

03H= پایداری توازن فرد

04H= پایداری توازن زوج

BL= بیت‌های توقف

00H= (یک بیت توقف)

01H= (1.5 بیت برای کلمه 5 بیت)

01H= (2 بیت برای &lt; کلمه 5 بیت)

CH= طول کلمه

00H= 5 بیت

01H= 6 بیت

02H= 7 بیت

03H= 8 بیت

CL= میزان باود

00H= 110 باود

01H= 150 باود

03H= 600 باود

04H= 1200 باود

05H= 2400 باود

06H= 4800 باود

07H= 9600 باود

08H= 19200 باود

## کنترل پورت COM توسعه یافته

ثبات‌های فراخوانده شده اضافی

00H=AL (خواندن ثبات کنترل)

01H= (نوشتن در ثبات کنترل)

DX= شماره پورت (0 اگر COM1، 1 اگر COM2، و غیره)

1 اگر COM2، و غیره)

BL= ثبات کنترل مودم (جدول ۹-۱۴ ملاحظه شود)

ثبات‌های نتیجه

اگر زیر تابع خوانده شود

BL= ثبات کنترل مودم

اگر زیر تابع نوشته شود

AL= وضعیت مودم (شکل ۹-۶)

AH= وضعیت خط (شکل ۹-۱۵)

بیت‌های 5-7: رزرو

بیت 4: حلقه

بیت 3=OUT2

بیت 2=OUT1

بیت 1=RTS

بیت 0=DTR

توجه: زیر تابع 00H=AL محتوای ثبات کنترل مودم را در BL بازمی‌گرداند. زیر تابع 01H=AL محتوای BL را در ثبات کنترل BL می‌نویسد و محتوای ثبات وضعیت خط را در AL و AH قرارداده و بازمی‌گرداند.

## بخش ۵ - ث: INT 15H - سرویس‌های سیستم

| <u>AH</u> | <u>عملکرد</u>                                                                                                                                                                                                           |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 86H       | انتظار                                                                                                                                                                                                                  |
|           | <p><u>ثبات‌های فراخوانده شده اضافی</u><br/>CX:DX زمان انتظار به ms</p> <p><u>ثبات‌های نتیجه</u><br/>CF=0 برای انتظار موفق<br/>CF=1 اگر انتظار انجام نشود</p> <p>توجه: دوره انتظار همیشه مضربی از 976 میلی ثانیه است</p> |

|     |                                                                                                                                                                                                                                                                                                                                                                  |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 88H | تعیین اندازه حافظه توسعه یافته                                                                                                                                                                                                                                                                                                                                   |
|     | <p><u>ثبات‌های فراخوانده شده اضافی</u><br/>هیچ</p> <p><u>ثبات‌های نتیجه</u><br/>AX = حافظه توسعه یافته (KB)</p> <p>توجه: مقدار حافظه توسعه یافته نصب شده را به KB باز می‌گرداند، این حافظه که بوسیله POST معین می‌شود از آدرس 100000H شروع می‌شود. اگر مدیریت حافظه DOS کنترل را به عهده دارد، حتی اگر حافظه توسعه یافته هم داشته باشد، 0 بازگردانده می‌شود.</p> |

## بخش ۶ - ث: INT 16H - صفحه کلید

| <u>AH</u> | <u>عملکرد</u>                                                                                                                                                                                                                                                                                                                               |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 00H       | خواندن صفحه کلید                                                                                                                                                                                                                                                                                                                            |
|           | <p><u>ثبات‌های فراخوانده شده اضافی</u><br/>هیچ</p> <p><u>ثبات‌های نتیجه</u><br/>AH = کد پویش کلید<br/>AL = کاراکتر اسکی</p> <p>توجه: یک کاراکتر را از بافر صفحه کلید خوانده و اشاره گر head آن را بهنگام می‌کند</p>                                                                                                                         |
| 01H       | وضعیت صفحه کلید را بدست آور                                                                                                                                                                                                                                                                                                                 |
|           | <p><u>ثبات‌های فراخوانده شده اضافی</u><br/>هیچ</p> <p><u>ثبات‌های نتیجه</u><br/>اگر کلیدی به انتظار نیست<br/>ZF=1<br/>اگر کلید در انتظار است<br/>ZF=0</p> <p>AH = کد پویش کلید<br/>AL = کاراکتر اسکی</p> <p>توجه: اگر کلیدی به انتظار است، کد پویش و کاراکتر در AH و AL باز می‌گردند، ولی اشاره گر سر از بافر صفحه کلید بهنگام نمی‌شود.</p> |

## وضعیت جابجایی را بدست آور

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

AL = بایت وضعیت

بیت 7: کلید Insert فشرده شد.

بیت 6: کلید Caps lock فشرده شد.

بیت 5: کلید Num lock فشرده شد.

بیت 4: کلید Scroll lock فشرده شد.

بیت 3: ALT فشرده شد.

بیت 2: ctrl فشرده شد.

بیت 1: Left shift فشرده شد.

بیت 0: Right shift فشرده شد.

توجه: بایت وضعیت صفحه کلید که در AL بازمی‌گردد نشانگر فشردگی کلیدهای خاص فوق است.  
بیت 1 به معنی فشردگی کلید مربوطه است.

## تنظیم میزان تکرار کد کلید

ثبات‌های فراخوانده شده اضافی

05H = AL

BH = تأخیر تکرار (به اطلاعات زیر مراجعه شود)

= میزان تکرار (به اطلاعات زیر مراجعه شود)

توجه: میزان تکرار پذیرفته شده کلیدزنی را تنظیم می‌کند. مقدار تأخیر در BH می‌تواند 00H (برای 250)، 01H (برای 500)، 02H (برای 750)، یا 03H (برای 1000) باشد. همه مقادیر به میلی‌ثانیه هستند. میزان تکرار در BL تعداد کاراکترها در ثانیه است. انتخاب‌ها عبارتند از

## خواندن کلید توسعه یافته

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

AH = کد پوش کلید

AL = کاراکتر اسکی

توجه: به جای INT 16H تابع 00H بکار رفته و برای شناسایی F11 و F12 و دیگر کلیدهای توسعه صفحه کلید بکار می‌رود. پس از خواندن اشاره گر سر از بافر صفحه کلید بهنگام می‌شود.

## وضعیت صفحه کلید توسعه یافته

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

اگر کلید به انتظار نیست

ZF=1

اگر کلیدی به انتظار است

ZF=0

AH = کد پوش کلید

AL = کاراکتر اسکی

توجه: این تابع به جای INT 16H تابع 01H بکار می‌رود بطوری که برنامه‌ها بتوانند کلیدهای توسعه F11 و F12 را شناسایی کنند. اگر کلیدی به انتظار است کد پوش و کاراکتر در AH و AL بازمی‌گردند، ولی اشاره گر سر صفحه کلید اصلاح نمی‌شود.

## AH

### عملکرد

12H

وضعیت جابجایی کلید توسعه یافته

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

AL = وضعیت جابجایی

بیت 7: Insert قفل است

بیت 6: Caps Lock قفل است

بیت 5: Num Lock قفل است

بیت 4: Scroll Lock قفل است

بیت 3: ALT فشرده است

بیت 2: Ctrl فشرده است

بیت 1: Left Shift فشرده است

بیت 0: Right Shift فشرده است

AH = وضعیت جابجایی توسعه یافته

بیت 7: SysRq فشرده است

بیت 6: Caps Lock فشرده است

بیت 5: Num Lock فشرده است

بیت 4: Scroll Lock فشرده است

بیت 3: Right Alt فشرده است

بیت 2: Right Ctrl فشرده است

بیت 1: Left Ctrl فشرده است

بیت 0: Left Ctrl فشرده است

توجه: بایت‌های وضعیت کلیدهای بازگشته در AL و AH بیانگر فشرده شدن کلیدهای خاصی هستند.

اگر بیت = 1، کلید فشرده شده است

## بخش ۷- ث: INT 17H - چاپگر

## AH

### عملکرد

00H

کاراکتر چاپگر

ثبات‌های فراخوانده شده اضافی

AL = کاراکتر چاپ

DX = چاپگر (0 برای LPT1، 1 برای LPT2

2 برای LPT3)

ثبات‌های نتیجه

AH = وضعیت

بیت 7: چاپگر غیر مشغول

بیت 6: تصدیق چاپگر

بیت 5: بدون کاغذ

بیت 4: انتخاب چاپگر

بیت 3: خطای I/O

بیت 1-2: رزرو

بیت 0: زمان اتمام چاپگر

01H

مقداردهی اولیه پورت چاپگر

ثبات‌های فراخوانده شده اضافی

DX = چاپگر (0 برای LPT1، 1 برای LPT2

2 برای LPT3)

ثبات‌های نتیجه

AH = وضعیت (تابع 00H ملاحظه شود)

02H

خواندن وضعیت

ثبات‌های فراخوانده شده اضافی

DX = چاپگر (0 برای LPT1، 1 برای LPT2

2 برای LPT3)

ثبات‌های نتیجه

AH = وضعیت

(تابع 00H ملاحظه شود)

## بخش ۸-ث: INT 1AH - تایمر و سرویس‌های ساعت بلادرنگ

| عملکرد                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | AH  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| <p><b>خواندن شمارشگر زمان تایمر سیستم</b></p> <p><u>ثبات‌های فراخوانده شده اضافی</u></p> <p>هیچ</p> <p><u>ثبات‌های نتیجه</u></p> <p>CX = بخش بالای شمارش<br/> DX = بخش پایین شمارش<br/> AL = 0 اگر 24 ساعت از آخرین خواندن نگذشته است<br/> &lt; 10 اگر 24 ساعت از آخرین خواندن گذشته است.</p> <p>توجه: این تابع تعداد تیک‌ها از نیمه شب را بازمی‌گرداند. یک ثانیه حدود 1802 تیک است. وقتی تعداد تیک‌های مشخص‌کننده 24 ساعت گذشت، AL یک واحد افزایش یافته و تعداد تیک‌ها صفر می‌گردد. فراخوانی این تابع موجب می‌شود تا AL صفر شود به نحوی که تشخیص 24 ساعت فقط هر روز یکبار میسر است.</p> | 00H |
| <p><b>تنظیم شمارشگر زمان تایمر سیستم</b></p> <p><u>ثبات‌های فراخوانده شده اضافی</u></p> <p>CX = بخش بالای تعداد تیک<br/> DX = بخش پایین تعداد تیک</p> <p><u>ثبات‌های نتیجه</u></p> <p>هیچ</p>                                                                                                                                                                                                                                                                                                                                                                                            | 01H |
| <p><b>خواندن زمان ساعت بلادرنگ</b></p> <p><u>ثبات‌های فراخوانده شده اضافی</u></p> <p>هیچ</p> <p><u>ثبات‌های نتیجه</u></p> <p>CH = ساعت<br/> CL = دقیقه<br/> DL = 01 برای صرفه‌جویی روشنایی انتخابی<br/> = 00 برای عدم انتخاب فوق<br/> CF = 0 اگر ساعت کار می‌کند در غیر اینصورت 1 =</p> <p>توجه: ساعت، دقیقه و ثانیه در قالب BCD بازگردانده می‌شوند. این تابع برای بدست آوردن زمان در تراشه CMOS زمان/تاریخ بکار می‌رود.</p>                                                                                                                                                             | 02H |
| <p><b>تنظیم زمان ساعت بلادرنگ</b></p> <p><u>ثبات‌های فراخوانده شده اضافی</u></p> <p>CH = ساعت<br/> CL = دقیقه<br/> DL = 01 برای انتخاب صرفه‌جویی نور روز<br/> = 00 برای غیرانتخاب</p> <p>توجه: ساعت، دقیقه و ثانیه در قالب BCD می‌باشند. این تابع برای تنظیم زمان در تراشه CMOS زمان/تاریخ بکار می‌رود.</p>                                                                                                                                                                                                                                                                              | 03  |



## AH

### عملکرد

04H

خواندن تاریخ ساعت بلادرنگ

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

CH = قرن (19 یا 20)

CH = سال

DH = ماه

DL = روز

CF = 10 اگر ساعت کار کند، در غیر این صورت

1 =

توجه: قرن، سال، ماه و روز در قالب BCD می‌باشند. این تابع برای بدست آوردن تاریخ در تراشه CMOS زمان/تاریخ بکار می‌رود.

05H

تنظیم تاریخ ساعت بلادرنگ

ثبات‌های فراخوانده شده اضافی

هیچ

ثبات‌های نتیجه

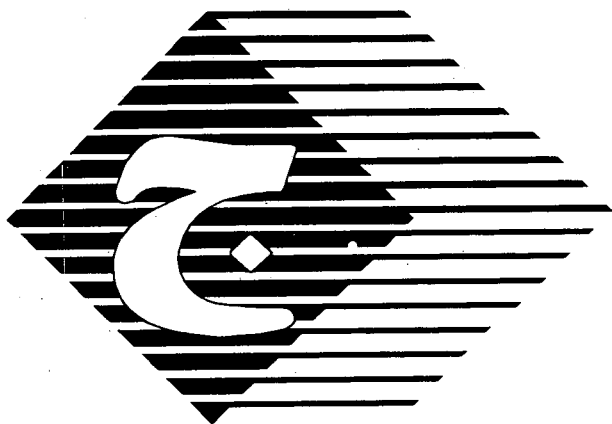
CH = قرن (19 یا 20)

CL = سال

DH = ماه

DL = روز

توجه: قرن، سال، ماه و روز در قالب BCD می‌باشند. این تابع برای تنظیم تاریخ در تراشه CMOS زمان/تاریخ بکار می‌رود.



## کدهای اسکی

| Ctrl | Dec | Hex | Ch | Code | Dec | Hex | Ch | Dec | Hex | Ch | Dec | Hex | Ch |
|------|-----|-----|----|------|-----|-----|----|-----|-----|----|-----|-----|----|
| ^@   | 0   | 00  |    | NUL  | 32  | 20  | !  | 64  | 40  | @  | 96  | 60  | `  |
| ^A   | 1   | 01  | ⊙  | SOH  | 33  | 21  | !" | 65  | 41  | A  | 97  | 61  | a  |
| ^B   | 2   | 02  | ●  | STX  | 34  | 22  | #" | 66  | 42  | B  | 98  | 62  | b  |
| ^C   | 3   | 03  | ♥  | ETX  | 35  | 23  | #  | 67  | 43  | C  | 99  | 63  | c  |
| ^D   | 4   | 04  | ♦  | EOT  | 36  | 24  | \$ | 68  | 44  | D  | 100 | 64  | d  |
| ^E   | 5   | 05  | +  | ENQ  | 37  | 25  | %  | 69  | 45  | E  | 101 | 65  | e  |
| ^F   | 6   | 06  | ▲  | ACK  | 38  | 26  | &  | 70  | 46  | F  | 102 | 66  | f  |
| ^G   | 7   | 07  | ●  | BEL  | 39  | 27  | '  | 71  | 47  | G  | 103 | 67  | g  |
| ^H   | 8   | 08  | □  | BS   | 40  | 28  | (  | 72  | 48  | H  | 104 | 68  | h  |
| ^I   | 9   | 09  | ○  | HT   | 41  | 29  | )  | 73  | 49  | I  | 105 | 69  | i  |
| ^J   | 10  | 0A  | ■  | LF   | 42  | 2A  | *  | 74  | 4A  | J  | 106 | 6A  | j  |
| ^K   | 11  | 0B  | σ  | VT   | 43  | 2B  | +  | 75  | 4B  | K  | 107 | 6B  | k  |
| ^L   | 12  | 0C  | ♀  | FF   | 44  | 2C  | ,  | 76  | 4C  | L  | 108 | 6C  | l  |
| ^M   | 13  | 0D  | ♪  | CR   | 45  | 2D  | -  | 77  | 4D  | M  | 109 | 6D  | m  |
| ^N   | 14  | 0E  | ♪  | SO   | 46  | 2E  | .  | 78  | 4E  | N  | 110 | 6E  | n  |
| ^O   | 15  | 0F  | *  | SI   | 47  | 2F  | /  | 79  | 4F  | O  | 111 | 6F  | o  |
| ^P   | 16  | 10  | ▶  | DLE  | 48  | 30  | 0  | 80  | 50  | P  | 112 | 70  | p  |
| ^Q   | 17  | 11  | ◀  | DC1  | 49  | 31  | 1  | 81  | 51  | Q  | 113 | 71  | q  |
| ^R   | 18  | 12  | ‡  | DC2  | 50  | 32  | 2  | 82  | 52  | R  | 114 | 72  | r  |
| ^S   | 19  | 13  | ‡  | DC3  | 51  | 33  | 3  | 83  | 53  | S  | 115 | 73  | s  |
| ^T   | 20  | 14  | ‡  | DC4  | 52  | 34  | 4  | 84  | 54  | T  | 116 | 74  | t  |
| ^U   | 21  | 15  | \$ | NAK  | 53  | 35  | 5  | 85  | 55  | U  | 117 | 75  | u  |
| ^V   | 22  | 16  | -  | SYN  | 54  | 36  | 6  | 86  | 56  | V  | 118 | 76  | v  |
| ^W   | 23  | 17  | ‡  | ETB  | 55  | 37  | 7  | 87  | 57  | W  | 119 | 77  | w  |
| ^X   | 24  | 18  | ↑  | CAN  | 56  | 38  | 8  | 88  | 58  | X  | 120 | 78  | x  |
| ^Y   | 25  | 19  | ↓  | EM   | 57  | 39  | 9  | 89  | 59  | Y  | 121 | 79  | y  |
| ^Z   | 26  | 1A  | →  | SUB  | 58  | 3A  | :  | 90  | 5A  | Z  | 122 | 7A  | z  |
| ^[   | 27  | 1B  | ←  | ESC  | 59  | 3B  | ;  | 91  | 5B  | [  | 123 | 7B  | {  |
| ^[ \ | 28  | 1C  | ←  | FS   | 60  | 3C  | <  | 92  | 5C  | \  | 124 | 7C  |    |
| ^]   | 29  | 1D  | →  | GS   | 61  | 3D  | =  | 93  | 5D  | ]  | 125 | 7D  | }  |
| ^^   | 30  | 1E  | ▲  | RS   | 62  | 3E  | >  | 94  | 5E  | ^  | 126 | 7E  | ~  |
| ^-   | 31  | 1F  | ▼  | US   | 63  | 3F  | ?  | 95  | 5F  | -  | 127 | 7F  | ○  |



# فلوچارت‌ها، شبه‌کد و اسمبلرها

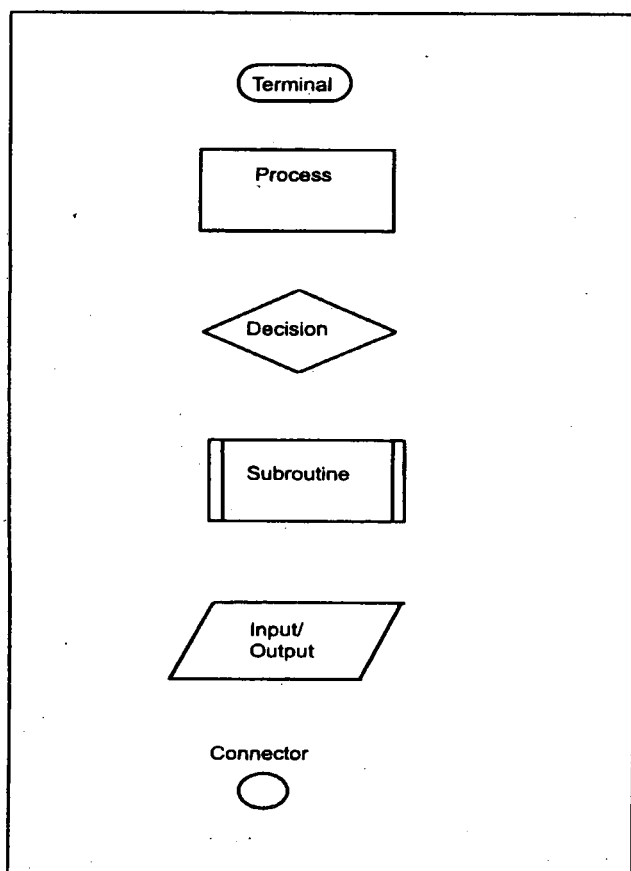
## بخش ۱- (چ) : فلوچارت‌ها و شبه‌کد

### فلوچارت‌ها

اگر قبلاً درسی از برنامه‌نویسی را گرفته‌اید، احتمالاً با فلوچارت آشنا هستید. فلوچارت‌ها نمادهای گرافیکی را برای ارائه انواع متفاوت عملیات برنامه بکار می‌برند. این نمادها بیکدیگر وصل می‌شود تا یک فلوچارت برای نمایش جریان اجرای برنامه بوجود آید. انواع الگوهای فلوچارت برای کمک به ترسیم سریع و تمیز سمبل‌ها در دسترس می‌باشند.

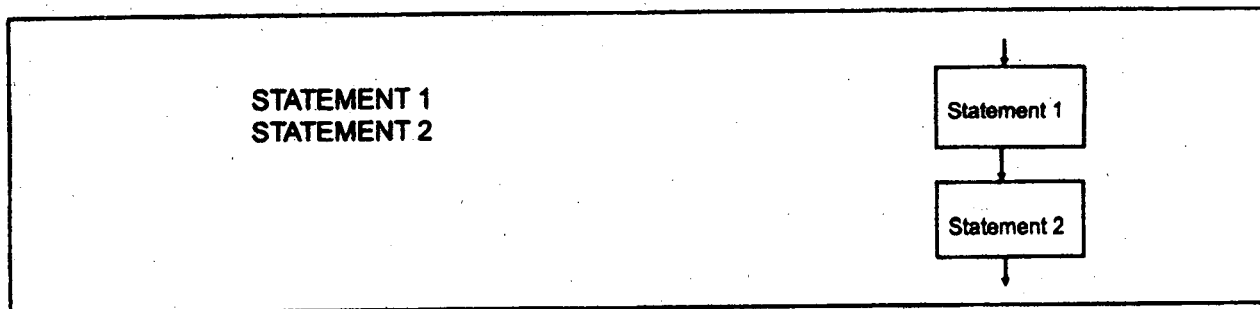
### شبه‌کد

ترسیم فلوچارت یک عمل استاندارد در طول دهه‌های گذشته بوده است. با این وجود بعضی‌ها با محدودیت‌هایی در استفاده از فلوچارت، مانند محدودیت نوشتن در هر جعبه کوچک، مشکل بودن ترسیم تصاویر بزرگ از آنچه که برنامه انجام می‌دهد مواجه



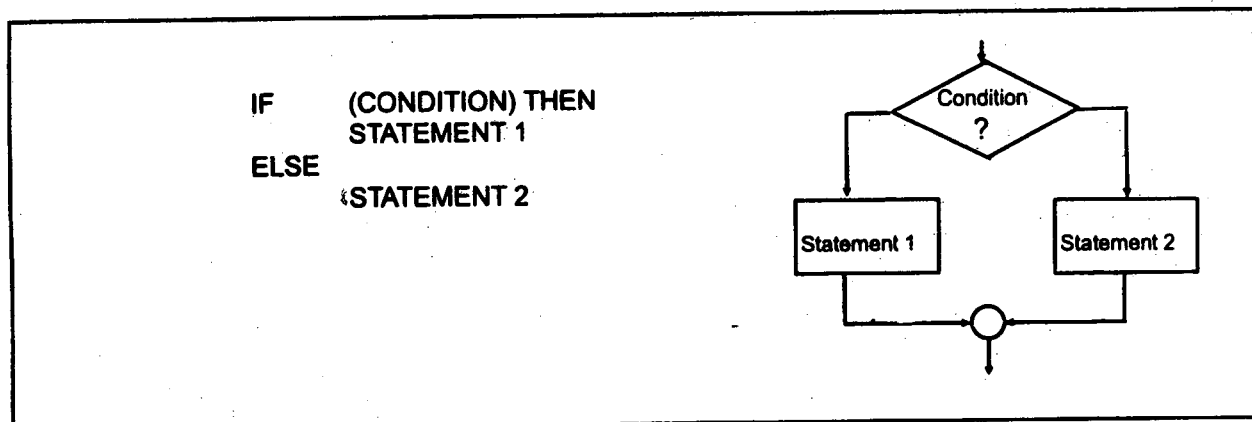
شکل ۱- (چ). نمادهای رایج فلوچارت

می‌شوند. روش جایگزین در کاربرد فلوچارت شبه کد است که شامل توصیف‌های نوشتاری خلاصه از جریان کد می‌باشد. شکل‌های ۲- (چ) تا ۶ (چ) فلوچارت و شبه کدهایی را که برای ساختارهای کنترل بکار می‌روند نشان می‌دهند. آنگاه چند مثال از فلوچارت و شبه کد را برای سه برنامه در کتاب نشان می‌دهیم. برنامه‌نویسی ساخت یافته از سه نوع ساختار کنترل برنامه استفاده می‌کند: توالی، کنترل و تکرار. توالی در واقع اجرای دستورات یکی پس از دیگری است. شکل ۲- (چ) نشان می‌دهد که چگونه توالی در فرم شبه کد و فلوچارت نمایش داده می‌شود.

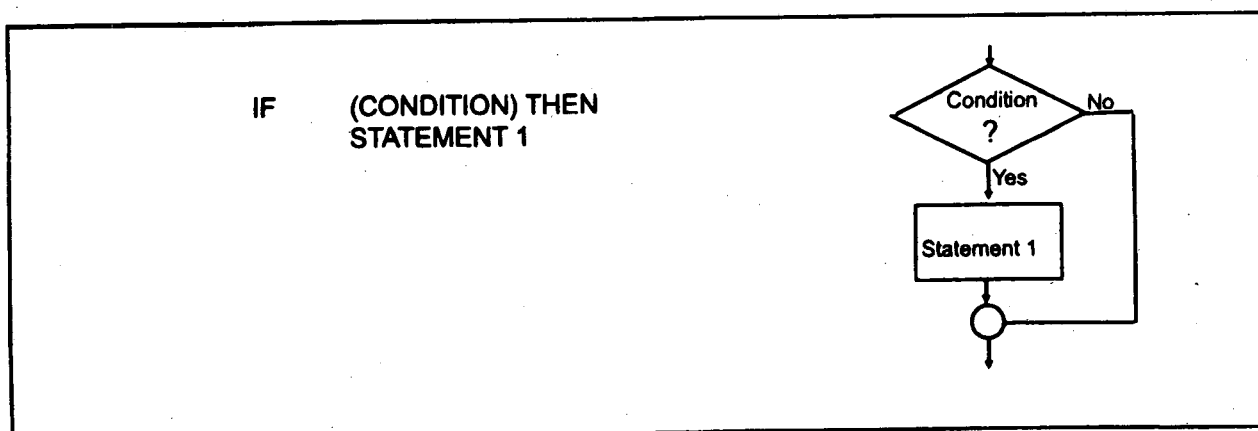


شکل ۲- (چ). شبه کد SEQUENCE در برابر فلوچارت

شکل ۳- (چ) و ۴- (چ) دو ساختار کنترل برنامه را نشان می‌دهد: ساختارهای IF-THEN و IF-THEN-ELSE در هر دو شبه کد و فلوچارت

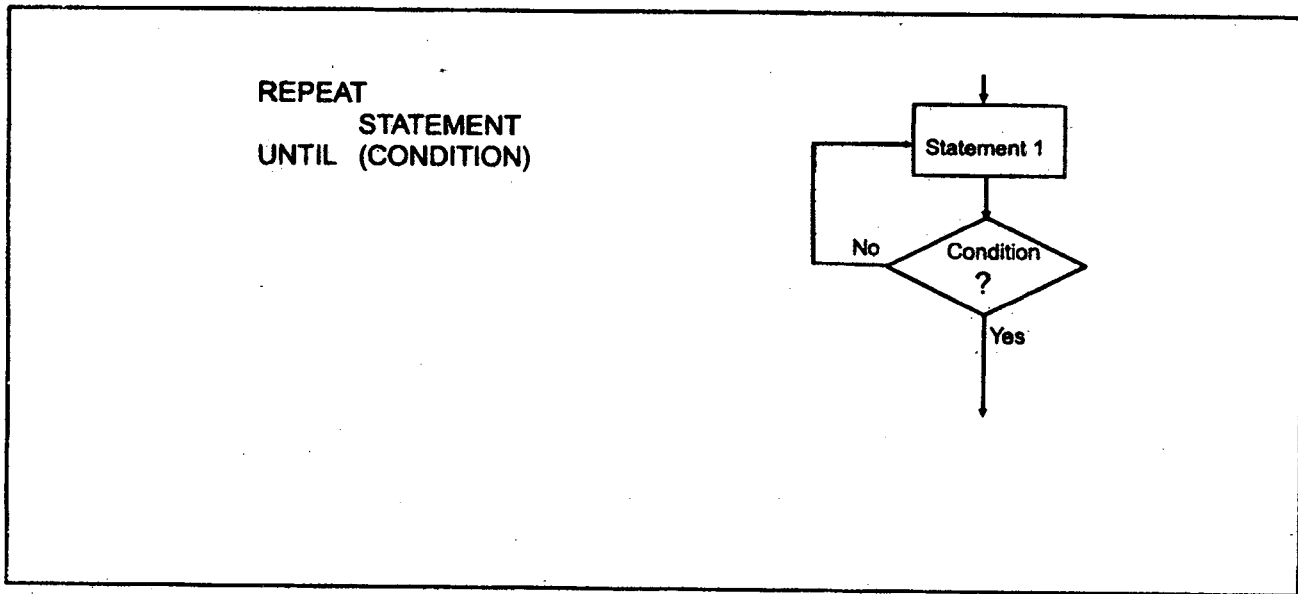


شکل ۳- (چ). شبه کد IF-THEN-ELSE در برابر فلوچارت



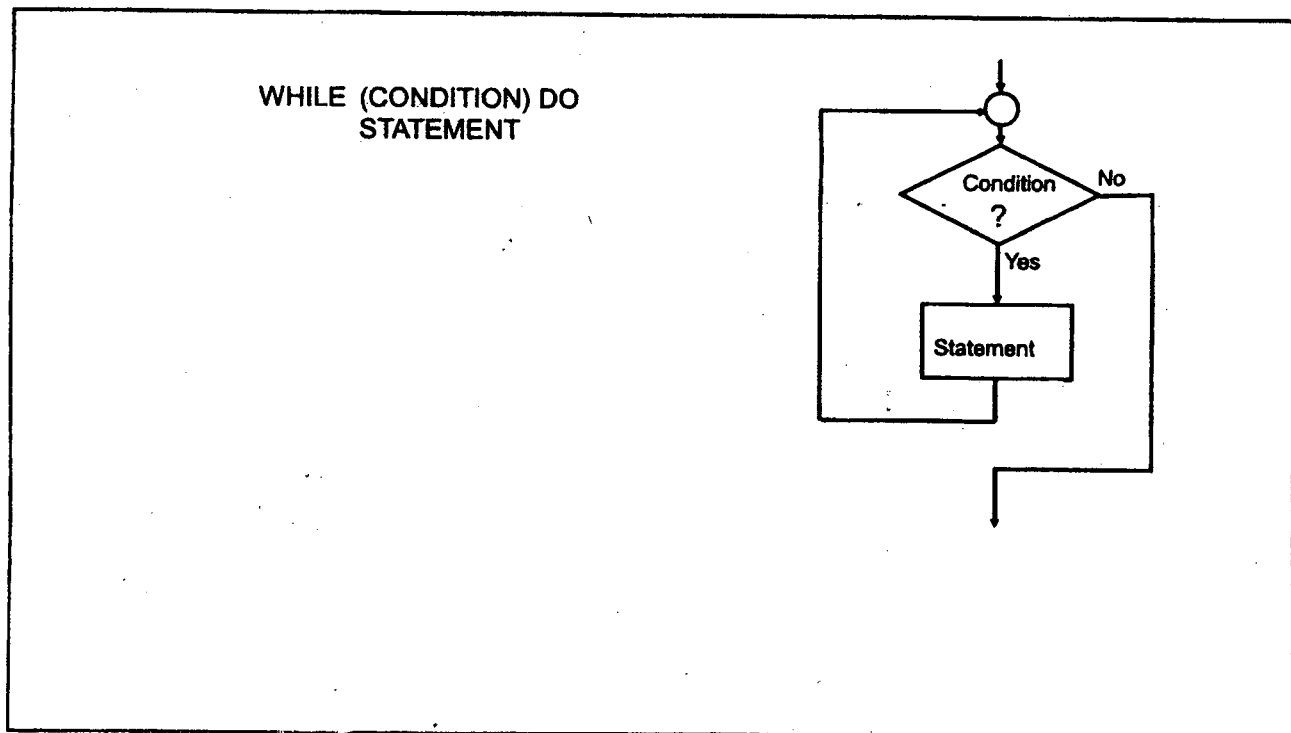
شکل ۴- (چ). شبه کد IF-THEN در برابر فلوچارت

در شکل های ۲- (چ) تا ۶- (چ)، عبارت "Statement" می تواند یک عبارت یا گروهی از عبارات را مشخص کند.

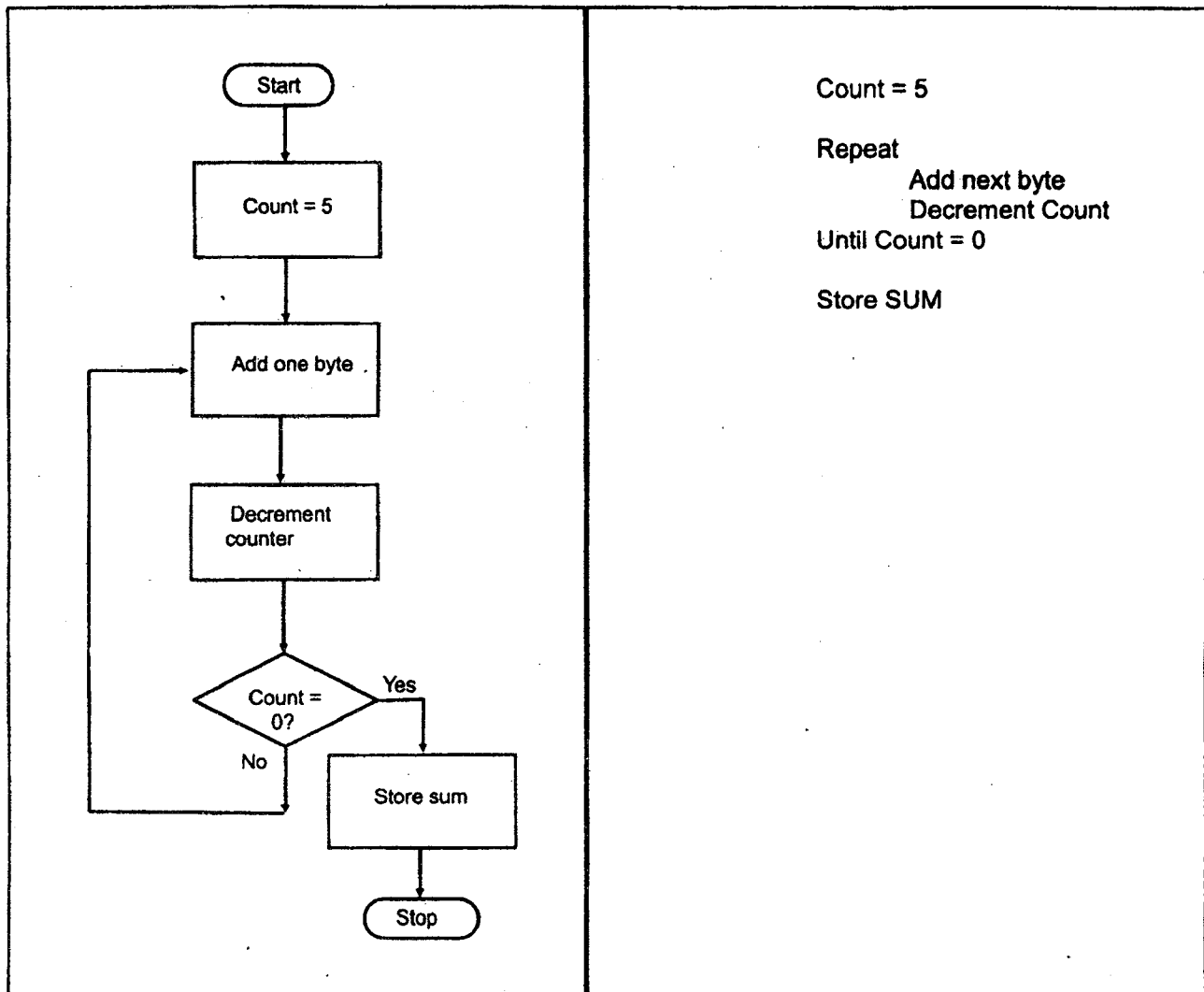


شکل ۵- (چ). شبه کد REPEAT UNTIL در برابر فلوچارت

شکل های ۵- (چ) و ۶- (چ) دو ساختار کنترل تکرار را نشان می دهند:  
 WHILE DO و REPEAT UNTIL. هر دو ساختار اجرای متوالی یک عبارت یا گروهی از عبارات را نشان می دهند. تفاوت بین آنها این است که ساختار REPEAT UNTIL همیشه عبارت را حداقل یک بار تکرار می کند و پس از هر تکرار شرط را چک می نماید، در صورتی که WHILE DO هرگز عبارت را اجرا نمی کند زیرا شرط در آغاز هر تکرار چک شده است.



شکل ۶- (چ). شبه کد WHILE DO در برابر فلوچارت

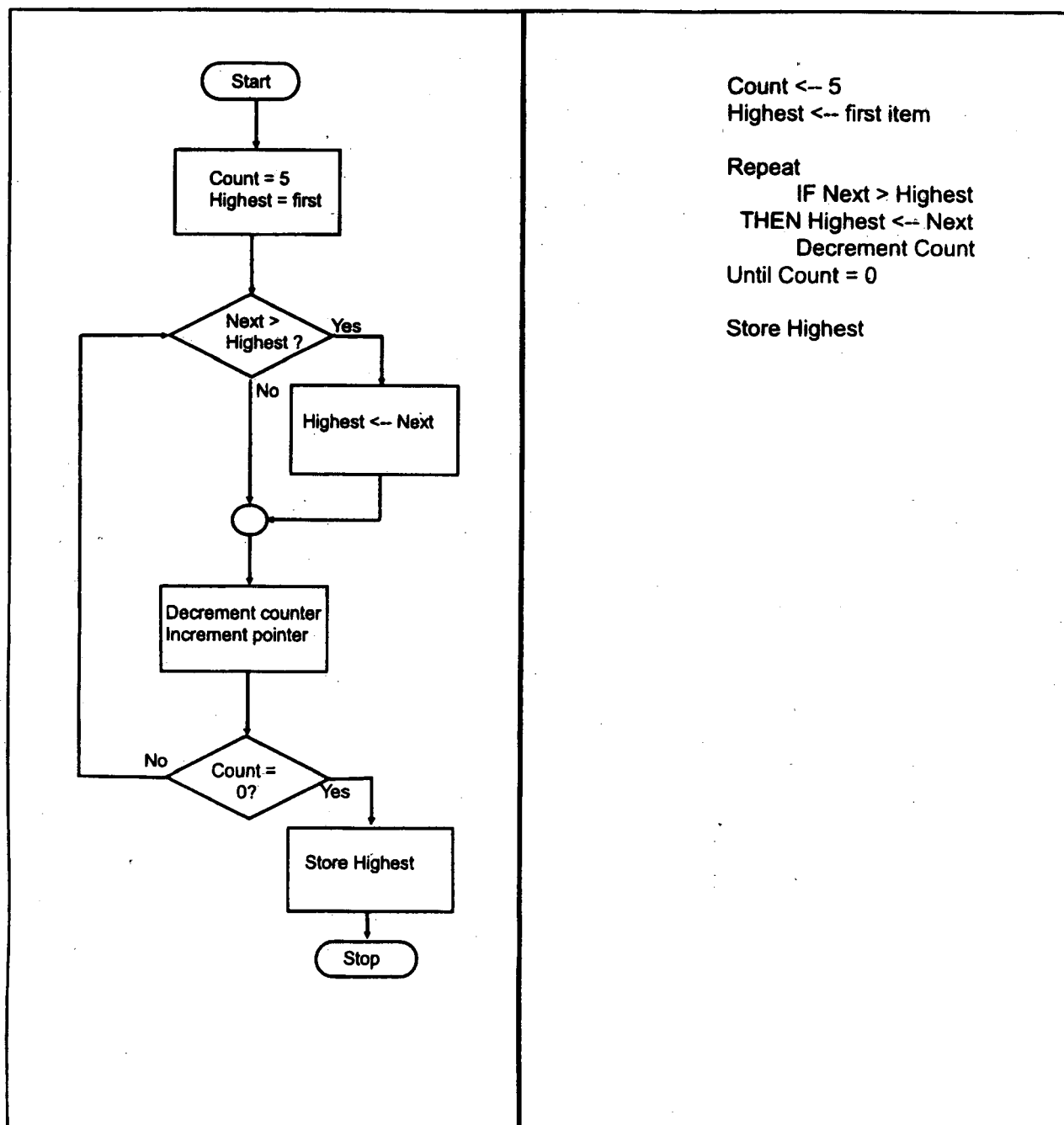


فلوچارت برنامه ۲-۱

شبه کد برنامه ۲-۱

این مثال فلوچارت را در برابر شبه کد برنامه ۲-۱ نشان می‌دهد. در این مثال، اطلاعاتی بیش از آنچه کسی پیدا کند داده شده است. مثلاً مراحل مقدار دهی اولیه و کاهش شمارنده‌ها داده شده است. برنامه‌نویسی دیگر ممکن است چنین مراحل را در فلوچارت یا شبه دستور ذکر کند. توجه کنید که اعمالی همچون مقداردهی اولیه و ثبات قطعه داده در رویه اصلی در فلوچارت یا شبه کد ذکر نشده است.

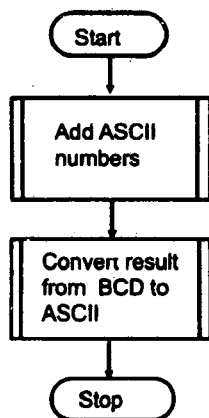
بیاد آورید این مطلب که هدف از فلوچارت‌ها یا شبه دستور نمایش جریان برنامه و کار آنست و نه دستورالعمل‌های زبان اسمبلی که اهداف برنامه را برآورده می‌سازند. گاهی اوقات شبه دستورات در لایه‌ها نوشته می‌شوند، بطوری که لایه یا سطح خارج، جریان برنامه و سطوح بعدی جزئیات بیشتری از وظیفه منتسب به آن را نشان می‌دهد.



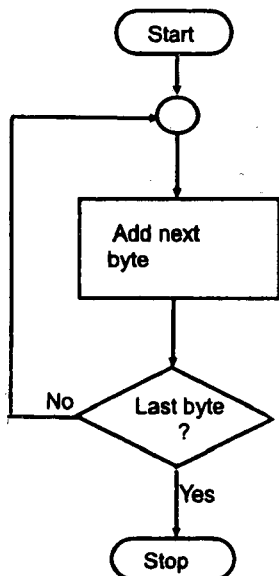
فلوچارت برنامه ۳-۳

شبه کد برنامه ۳-۳

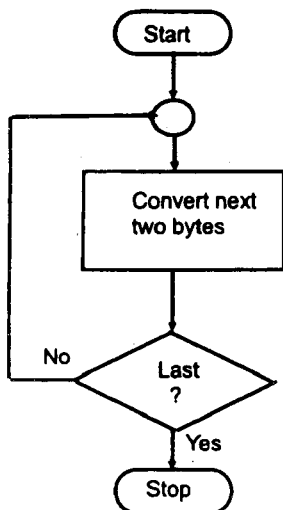
در بالا فلوچارت در برابر شبه کد برای برنامه ۳-۳ نشان داده شده است. صفحات بعد فلوچارت را در برابر شبه کد برای برنامه ۳-۷ و ۷-۱ نشان می دهد.



CALL add subroutine  
CALL convert subroutine



Repeat  
Add next byte  
Until all bytes are added

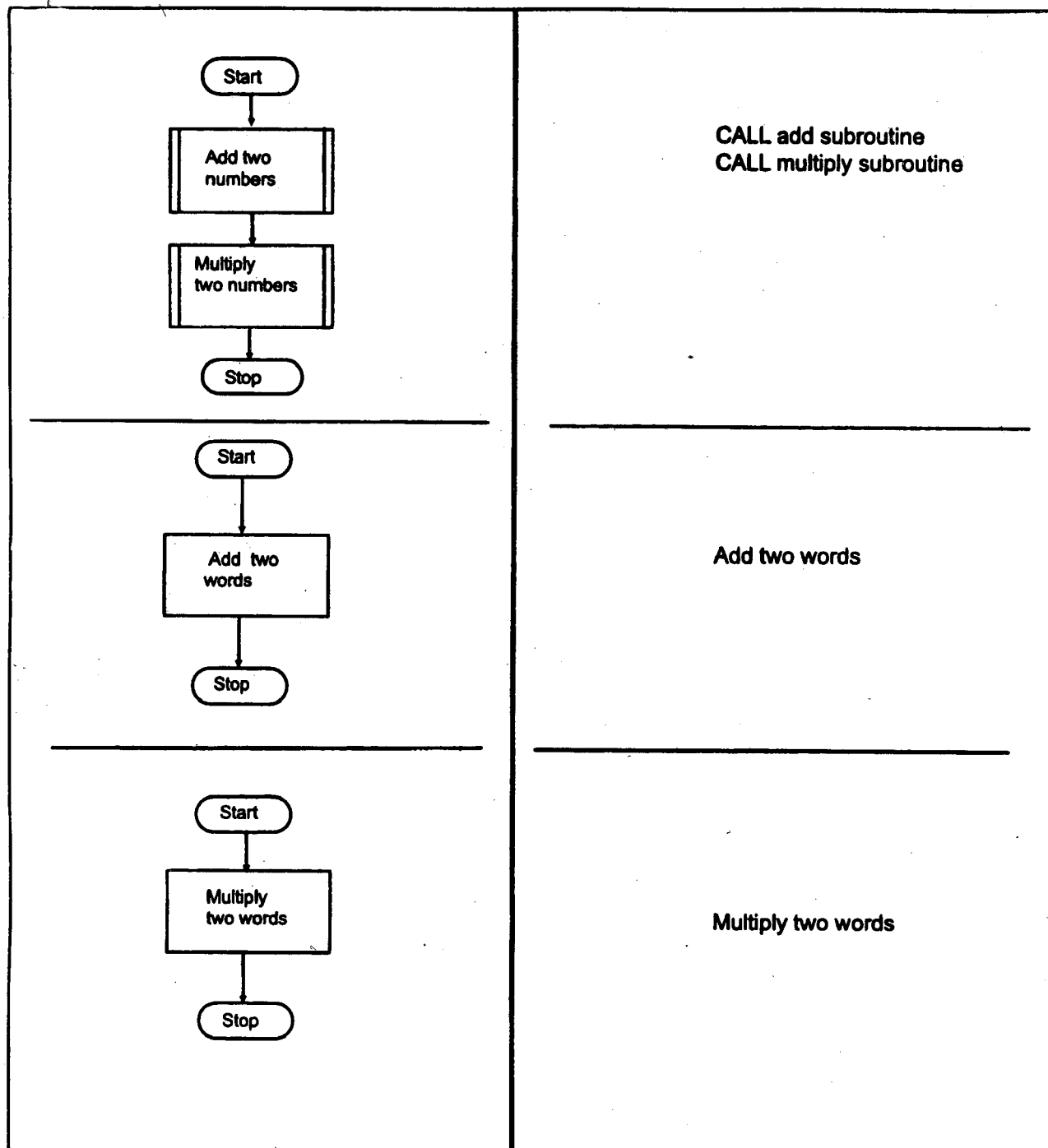


Repeat  
Convert next byte  
Until all bytes are converted

فلوچارت برای برنامه ۳-۷

شبه کد برای برنامه ۳-۷





فلوچارت برای برنامه ۱-۷

شبه کد برای برنامه ۱-۷

## بخش ۲ - (چ): اسمبلر

بخش ۲-۲ نشان داد که چگونه یک برنامه به زبان اسمبلی، اسمبل، پیوند و اجرا میشود. این کتاب کاربرد اسمبلر MASM را بدلیل سادگی استفاده از آن و در دسترس قرار داشتنش ارائه نمود. با این وجود این بخش برای کسانی است که Code view از میکروسافت و یا اسمبلر TASM از بورلند را بکار می‌برند.

### بکارگیری Codeview از میکروسافت

در زیر بعضی از فرمان‌های Code view نشان داده شده است. توصیف کامل Code view در کتب راهنمای میکروسافت یافت می‌شود. اطلاعات موجود در اینجا برای شما کافی است.

### نصب برنامه برای Code view

برای اینکه اطلاعات سمبلیکی از Code view در دسترس باشد، برنامه زبان اسمبلی باید با انتخاب /ZI، اسمبل و با انتخاب /CO پیوند شود. با استفاده از این انتخاب‌ها می‌توانید کد مبدأ در Code view را مشاهده کنید. Code view با فایل‌هایی که بدون انتخاب‌های اسمبل / پیوند فوق ایجاد شده باشند هم قابل استفاده است.

### وارد کردن Code view

در زیر چگونگی ورود به Code view برای PROG1.EXE را نشان می‌دهد. Code view برای هر نوع فایل ".exe" تولید شده با C، پاسکال یا دیگر کامپایلرها هم قابل استفاده است. شکل ۷-۷ (چ) نمونه‌ای از صفحه نمایش Code view را نشان می‌دهد. این صفحه نمایش بهنگام اجرای برنامه ۱-۲ در Code view حاصل شده است. به نشانه "<" که در انتهای صفحه نمایش Code view ظاهر می‌شود توجه کنید. در زیر لیستی از بعضی فرمان‌های رایج که در این نشانه تایپ می‌شوند آمده است. بسیاری از این فرمان‌ها می‌توانند از منوی موجود که با ماوس انتخاب می‌شوند برداشته شوند.

|                                                         |                                                                                   |
|---------------------------------------------------------|-----------------------------------------------------------------------------------|
| r                                                       | مقادیر ثبات در پائین صفحه نمایش را، مشابه با آنچه DEBUG انجام می‌داد نشان می‌دهد. |
| a                                                       | اسمبل کردن دستورات در برنامه                                                      |
| db                                                      | بایت‌های حافظه را نشان می‌دهد.                                                    |
| aw                                                      | کلمات حافظه را نشان می‌دهد.                                                       |
| eb                                                      | بایت‌ها را در حافظه وارد می‌کند.                                                  |
| g                                                       | تا رسیدن به نقطه توقف اجرا می‌کند (go)                                            |
| p                                                       | خط مبدأ جاری را اجرا می‌نماید (از روال‌ها و وقفه‌ها چشم می‌پوشد)                  |
| q                                                       | خارج شود                                                                          |
| t                                                       | ردیابی خط مبدأ جاری (اجرا)، (روال‌ها و وقفه‌های چشم‌پوشی نمی‌شوند)                |
| w                                                       | نگاه کن (متغیر یا عبارت)                                                          |
| در زیر بعضی از کلیدهای تابعی رایج در Code view آمده است |                                                                                   |
| F2                                                      | خاموش و روشن شدن نمایش ثبات                                                       |
| F3                                                      | رفت و برگشت بین اسمبلی و لیست مبدأ                                                |
| F4                                                      | رفت و برگشت بین خروجی و نمایش برنامه                                              |
| F5                                                      | اجرا تا توقف                                                                      |
| F6                                                      | مکان نما بین برنامه و فرمان ">" رفت و آمد کند                                     |
| F8                                                      | تک مرحله                                                                          |
| F9                                                      | توقف                                                                              |

| File View Search Run Watch Options Language Calls Help   F8=Trace F5=Go |        |      |                    |             |      |
|-------------------------------------------------------------------------|--------|------|--------------------|-------------|------|
| 4B35:0000                                                               | B8344B | MOV  | AX,4B34            | AX=         | 6DEE |
| 4B35:0003                                                               | 8ED8   | MOV  | DS,AX              | BX=         | 0000 |
| 4B35:0005                                                               | B90500 | MOV  | CX,0005            | CX=         | 0000 |
| 4B35:0008                                                               | BB0000 | MOV  | BX,0000            | DX=         | 0000 |
| 4B35:000B                                                               | B000   | MOV  | AL,00              | SP=         | 0200 |
| 4B35:000D                                                               | 0207   | ADD  | AL,Byte Ptr [BX]   | BP=         | 0000 |
| 4B35:000F                                                               | 43     | INC  | BX                 | SI=         | 0000 |
| 4B35:0010                                                               | 49     | DEC  | CX                 | DI=         | 0000 |
| 4B35:0011                                                               | 75FA   | JNZ  | 000D               | DS=...      | 4836 |
| 4B35:0016                                                               | B44C   | MOV  | AH,4C              | ES=...      | 4823 |
| 4B35:0018                                                               | CD21   | INT  | 21                 | FS=...      | 0000 |
| 4B35:001A                                                               | 5A     | POP  | DX                 | GS=...      | 0000 |
| 4B35:001B                                                               | 17     | POP  | SS                 | SS=...      | 0837 |
| 4B35:001C                                                               | 2C06   | SUB  | AL,06              | CS=...      | 4B35 |
| 4B35:001E                                                               | A31740 | MOV  | Word Ptr [4017],AX | IP=0000002C |      |
| 4B35:0021                                                               | 06     | PUSH | ES                 | NV UP       |      |
| 4B35:0022                                                               | E717   | OUT  | 17,AX              | EI NG       |      |
|                                                                         |        |      |                    | NZ AC       |      |
|                                                                         |        |      |                    | PE NC       |      |
| >t                                                                      |        |      |                    |             |      |
| >t                                                                      |        |      |                    |             |      |
| >t                                                                      |        |      |                    |             |      |
| >                                                                       |        |      |                    | DS:0004     |      |
|                                                                         |        |      |                    | 00000000    |      |

شکل ۷- (ج). صفحه نمایش نمونه‌ای از Code view

### نمایش و ورود داده

داده‌ها می‌توانند در قالب DB یا DW نمایش داده شوند. این دو فرمان داده را در مبنای 16 نمایش می‌دهند. Code view نمایش داده را در دهدهی با استفاده از فرمان DU اجازه می‌دهد. عددی در دنباله فرمان D به Code view محل شروع نمایش را متذکر می‌شود. با ورود مکرر فرمان D در قطعه داده یک به یک پیش می‌رویم.

>DB 0 36                      قطعه داده را از تفاوت مکان 0 تا 36 نمایش می‌دهد  
 >DB 0 36                      معادل دهدهی بایت‌های 0 تا 36 را نشان می‌دهد  
 >DW 0 36                      کلمات را از تفاوت مکان 0 تا 36 نشان می‌دهد

توجه کنید که اعداد وارده فوق به دهدهی هستند و نه شانزده شانزدهی. مثلاً، "DB 0 36" 36 بایت را تخلیه می‌کند که معادل 24 شانزده شانزدهی است. داده می‌تواند بصورت بایت‌های مبنای شانزده (EB) یا کلمات (EW) و یا صحیح دهدهی (EI) هم وارد می‌شود. عددی که بدنبال فرمان E می‌آید تفاوت مکان شروع وارد کردن داده را نشان می‌دهد. مثال‌هایی از این مورد در زیر آمده است.

>DB 16 0 0 0                      سه بایت 0 را از تفاوت مکان 16 وارد می‌کند.  
 >EW 16 0 0 0                      سه کلمه 0 را از تفاوت مکان 16 وارد می‌نماید  
 >EA 16 32 32 32                      سه فاصله اسکی را از تفاوت مکان 16 وارد می‌کند  
 >EI 0 -10 10 20                      اعداد صحیح (20, 10, -10) را در تفاوت مکان صفر وارد می‌کند.

### فرمان‌های Watch

فرمان‌های Watch ابزارهای مفیدی در Code view هستند. آنها بعنوان فرمان روی خط فرمان تایپ و یا در

بالای صفحه تصویر Code view ایجاد می‌شوند. فرمان‌های Watch یک جعبه در بالای صفحه نمایش ایجاد می‌کنند و شما می‌توانید ضمن اجرای برنامه تغییر داده را ملاحظه کنید. چند مثال می‌تواند به چگونگی استفاده از فرمان‌های Watch کمک کند.

- >DW DS:0 L 16 این فرمان قطعه داده را از تفاوت مکان 0 تا 16 نشان می‌دهد.
- >W? DATA\_1 این فرمان یک قلم داده با نام DATA-1 را نشان می‌دهد.
- >WP ? SUM > 0 این موجب می‌شود تا زمانی که داده SUM بزرگتر از 0 است مشاهده شود.

شبه فرمان فوق بعضی از مفاهیم فرمان Watch را نشان می‌دهد. اولین دستور نشان می‌دهد که فرمان Watch برای تخلیه مقادیر خاص تفاوت مکان قطعه داده بکار رفته است. دومین فرمان برای مشاهده یک مقدار خاص بکار رفته است. این فرمان فقط هنگامی کار می‌کند که برنامه با انتخاب‌های قبلی اسمبل و پیوند شده باشد. فرمان سوم چگونگی ایجاد نقاط توقف در برنامه را نشان می‌دهد. اگر فرمان اجرا ">G" پس از این فرمان وارد شود، برنامه تا زمانی که SUM بزرگتر از صفر است و یا به انتهای برنامه برسد، به اجرا ادامه خواهد داد.

### نکات کمکی

۱- ضمن اجرای برنامه صفحه بسمت بالا حرکت طوماری می‌کند. ممکن است در بعضی مواقع شما مایل به مشاهده کدی باشید که از صفحه بیرون رفته است. در این حال، فرمان U باید بکار رود. مثلاً اگر فرمان U0 را برای عکس اسمبل کردن از تفاوت مکان 0 بکار ببرید، استفاده از آن موجب می‌شود شما به شروع برنامه باز گردید.

۲- اگر شما سعی در نمایش قطعه داده قبل از نصب آن در برنامه بکنید "MOV DS:AX"، با اشکال مواجه خواهید شد. بنابراین در برنامه به فرم تک مرحله وارد شده و دستورات سیستم‌داری اولیه را قبل از مشاهده قطعه داده ملاحظه کنید.

### استفاده از Turbo debugger بورلند

اکنون بعضی از فرمان‌های اصلی Turbo debugger را نشان می‌دهیم. توصیف کامل Turbo debugger می‌تواند در کتب راهنمای بورلند ملاحظه شود. اطلاعاتی که در اینجا فراهم شده است برای عیب‌یابی برنامه‌های ساده کفایت می‌کند.

### نصب یک برنامه برای Turbo debugger

برنامه زبان اسمبلی باید با انتخاب /ZI اسمبل شود

```
C>TASM /ZI TEST.ASM
```

و با /V پیوند زده شود:

```
C>TLINK /V TEST1.OBJ
```

توجه کنید که بهنگام استفاده از برنامه‌های اسمبلی و پیوند، فرمان‌های TASM و TLINK می‌باشند و نه MASM و LINK. استفاده از انتخاب‌های فوق امکان می‌دهد تا کد مبدأ در Turbo debugger ملاحظه شود ولی debugger می‌تواند با فایل‌های ایجاد شده با انتخاب‌های اسمبل/پیوند نیز بکار رود.

### ورود به Turbo debugger

در زیر چگونگی ورود به Turbo debugger برای TEST.EXE را نشان می‌دهد.

```
C>TD TEST
```

به آدرس های تفاوت مکان بکار رفته در زیر توجه کنید.

```
.MODEL SMALL
.STACK64H
.DATA
DATA_1 DB 4,6,2,3,7,8,6,9
 ORG 10H
SUM_B DB ?
 ORG 18H
DATA_2 DW 1222H,1333H,1191H,7869H
 ORG 28H
SUM_W DW ?
.CODE
MOV AX,@DATA
MOV DS,AX
```

با استفاده از DEBUG می توانید ببینید که آدرس تفاوت مکان 10H متعلق به SUMB پذیرفته نشده است. این مطلب برای آدرس های تفاوت مکان دیگر مانند 18H و 28H نیز صحیح است. دلیل این است که MASM سعی در بالا بردن کارایی تخصیص حافظه دارد. بنابراین در بکارگیری تعریف قطعه جدید قویاً پیشنهاد می شود آدرس تفاوت مکانی را به متغیری نسبت ندهید. این پیشنهاد برای برنامه های واقعی، بدون توجه به اینکه تعریف قطعه قدیمی و یا جدید را بکار می برید، نیز صادق است. تنها دلیلی که برای اجرای آن در این کتاب بوده است. این است که میدان های داده را برای وضوح بیشتر DEBUG از هم تفکیک نمائیم.

در تعریف قطعه کامل، آدرس پایه برای محاسبه یک آدرس OFFSET (تفاوت مکان) همان آدرس قطعه داده است. با تعریف قطعه ساده شده، آدرس پایه، آدرس قطعه برای همه قطعات در گروه، DGROUP می باشد. به دستور زیر نگاه کنید.

```
MOV AX,OFFSET NUM_1
```

اگر عبارت فوق در برنامه ای با تعریف قطعه کامل قرار داشته باشد، تفاوت مکان NUM-1 نسبت به شروع قطعه داده است. با این وجود، اگر قطعه فوق در برنامه ای با تعریف قطعه ساده شده، باشد، تفاوت مکان NUM-1 نسبت به شروع DGROUP است. برای کسب اطلاعات بیشتر در مورد DGROUP به بخش بعد مراجعه کنید. تفاوت دیگر بین تعریف قطعه کامل و ساده شده، این است که بعضی پیش فرض ها در این دو فرق می کنند. مثلاً پیش فرض تطابق در تعریف قطعه کامل PARA است ولی در تعریف قطعه ساده شده با مدل SMALL، برای قطعات داده و کد، WORD می باشد و برای قطعه پشته PARA است. اگر قرار باشد این پیش فرض ها لغو شوند، از تعریف قطعه کامل باید استفاده شود.

### استفاده از DGROUP

گروه عبارت از قطعات انتخابی مرتبط با یک آدرس شروع می باشد. آنها در داخل یک قطعه 64K قرار دارند. این در مواردی مفید است که بیش از یک قطعه به یک ثابت قطعه در هنگام اجرا مرتبط باشد. گروه DGROUP، هنگام استفاده از تعریف قطعه ساده شده بطور خودکار تولید می شود. DGROUP شامل قطعات DATA و STACK است ولی قطعه CODE نیست. می توان DGROUP را بعنوان یک گروه داده خودکار تصور کرد. وقتی تعریف قطعه ساده شده بکار رود، آدرس اختصاص یافته با دستور که به حافظه ای در قطعه داده دسترسی پیدا می کند آدرس DGROUP خواهد بود و نه آدرس قطعه داده.



فهرست کتابهای کامپیوتری که در این انتشارات به چاپ رسیده است:

|                                        |                                                      |                                               |
|----------------------------------------|------------------------------------------------------|-----------------------------------------------|
| ۱ - زبان اسمبلی                        | محمد علی مزیدی<br>جانیس مزیدی                        | دکتر قدرت سپیدنام<br>دکتر حسین دلدار          |
| ۲ - مدارهای واسط                       | محمد علی مزیدی<br>جانیس مزیدی                        | دکتر قدرت سپیدنام                             |
| ۳ - میکروکنترلر ۸۰۵۱                   | محمد علی مزیدی<br>جانیس مزیدی                        | دکتر قدرت سپیدنام                             |
| ۴ - میکروکنترلر ۸۰۵۱                   | آی. اسکات مکنزی                                      | مهندس حمیدرضا رضایی نیا<br>مهندس پیمان دربندی |
| ۵ - طراحی دیجیتال                      | موریس مانو                                           | دکتر قدرت سپیدنام                             |
| ۶ - معماری کامپیوتر                    | موریس مانو                                           | دکتر قدرت سپیدنام                             |
| ۷ - طراحی و تحلیل مدار منطقی دیجیتال   | ویکتور - پی نلسون<br>اچ. تروی نیکل<br>بیل دی. کارول  | دکتر قدرت سپیدنام                             |
| ۸ - کامپایلرها (اصول، روشها و ابزارها) | اهو<br>سدی<br>المن                                   | دکتر حسین دلدار<br>مهندس محمد مهدی سالخورده   |
| ۹ - ساختمان داده‌ها در پاسکال          | الیس هورویتز<br>سارتج ساهنی                          | ملیحه مهدی پور<br>سهیل صالحی                  |
| ۱۰ - ساختمان داده‌ها به زبان ++C       | الیس هورویتز<br>سارتج ساهنی<br>سوزان آندرسون<br>فرید | امیر علیخانزاده                               |
| ۱۱ - ساختمان داده‌ها به زبان C         | الیس هورویتز<br>سارتج ساهنی<br>سوزان آندرسون - فرید  | امیر علیخانزاده                               |
| ۱۲ - بانک اطلاعاتی                     | دیت                                                  | امیر علیخانزاده                               |
| ۱۲ - پاسکال                            | الیوت ب. کافمن                                       | لیدا جواهر قلم<br>فرنگیس شاکری                |
| ۱۳ - توربو پاسکال                      | الیوت ب. کافمن                                       | سپهر صالحی                                    |

# **The 80x86 IBM PC and Compatible Computers**

**Assembly Language  
Programming on the IBM PC,  
PS, and Compatibles**

**Second Edition**

**- Muhammad Ali Mazidi  
- Janice Gillispie Mazidi**



مرکز پخش :

مشهد: چهارراه دکترا انتشارات خراسان تلفن (نمابر) ۸۱۱۸۳۱