

بسمه تعالی

زبان ماشین و برنامه سازی سیستم

قسمت اول جزوه

دانشگاه آزاد اسلامی واحد تهران شمال

مدرس: مهندس کریمی

فهرست مطالب

فصل اول: مقدمه

فصل دوم: آشنایی با سخت افزار و ویژگیهای آن

۱-۲- قطعات سخت افزاری یک سیستم کامپیوتری

cpu -

ram -

rom- نرم افزار BIOS و جداول گلایف

- حافظه های جانبی

- تجهیزات ورودی و خروجی

- سایر تجهیزات

۲-۲- مفهوم ۰ و ۱ در سیستم های کامپیوتری

- نحوه تبدیل داده های مختلف (عدد، کاراکتر، رشته، تصویر، صدا) به ۰ و ۱

- نحوه تبدیل دستورات به ۰ و ۱

۳-۲- اساس کار یک cpu

۴-۲- آشنایی با گذرگاه ها

۵-۲- شمای کلی یک پردازنده 8086

۶-۲- مفهوم خط لوله گی (pipelining)

۷-۲- مفهوم گذرگاه های multiplex شده

۸-۲- مشخصات تکنیکی پردازنده های شرکت intel

فصل سوم: مبناهای عددی

۱-۳- اعداد صحیح

۱-۱-۳- اعداد صحیح بدون علامت

۲-۱-۳- اعداد صحیح علامتدار

۳-۱-۳- مزایای مکمل ۲

۲-۳- اعداد اعشاری

۱-۲-۳- ممیز ثابت

۱-۲-۳- ممیز شناور

۳-۳- نمایش اعداد بدون علامت بصورت BCD

فصل چهارم: زبان اسمبلی ۸۰۸۶

۱-۴- مفهوم ثبات و آشنایی با ثباتهای پردازنده های 8086

۲-۴- مفهوم پردازنده های n بیتی و آشنایی با ثباتهای پردازنده های ۳۲ بیتی

۳-۴- ساختار کلی برنامه ها در زبان اسمبلی

۴-۴- یک برنامه نمونه در اسمبلی و شرح دستورات آن

۵-۴- روش نوشتن ، ترجمه و اجرای یک برنامه در زبان اسمبلی

۶-۴- دستورات mul , div

۷-۴- دستورات مقایسه و پرش

۴-۸- حل چند مثال

فصل پنجم: مفهوم آدرسها در زبان اسمبلی

۵-۱- آدرسهای نسبی، مطلق و واقعی

۵-۲- روشهای آدرس دهی

فصل ششم: وقفه ها در زبان اسمبلی

فصل هفتم : مفهوم پشته و روش استفاده از آن

فصل هشتم: زیربرنامه ها و ماکروها در زبان اسمبلی

فصل نهم: بررسی چند دستور و چند مثال

فصل اول - مقدمه

• تعریف یک سیستم کامپیوتری:

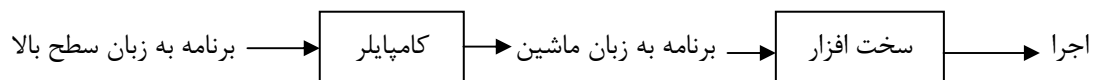
- ۱- پردازش انجام می دهد.
 - ۲- از اجزای الکترونیکی تشکیل شده است.
 - ۳- دارای حافظه است.
 - ۴- قابل برنامه ریزی است.
- هر سیستم کامپیوتری دارای دوبرخ است: سخت افزار (Hardware) و نرم افزار (Software)
- سخت افزار:** قطعات ظاهری و فیزیکی یک سیستم کامپیوتری
- نرم افزار:** برنامه هایی که بر روی سخت افزار قرار گرفته و اجرا می شوند.

• **تعریف زبان (language):** مجموعه ای از علائم و نشانه ها که بر اساس یک سری قوانین صرفی یا لغوی (lex)، نحوی یا گرامری (syntax) و معنایی یا مفهومی (semantic) در کنار هم قرار می گیرند و برای برقراری ارتباط بین دو یا چند نفر استفاده می شوند.

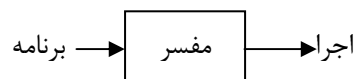
• **تعریف زبانهای برنامه نویسی (programming languages):** زبانهایی هستند که برای برقراری ارتباط بین برنامه نویسان و سیستم های کامپیوتری استفاده می شوند.

ابزارهای اجرای برنامه های نوشته شده در زبان های برنامه سازی سطح بالا: کامپایلرها و مفسرها

۱- **کامپایلرها:** کامپایلر در واقع یک فایل exe تولید می کند که برنامه به زبان ماشین (1,0) است و می تواند توسط سخت افزار اجرا شود.



۲- **مفسرها:** مفسر مستقیماً دستورات برنامه را اجرا می کند.

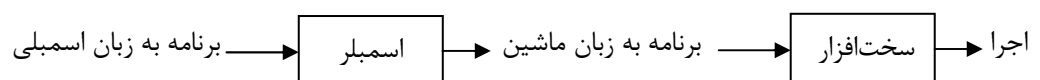


• **هدف از برنامه نویسی:** صحبت کردن با کامپیوتر (انتقال خواسته های خود به کامپیوتر، حل مسائل به کمک کامپیوتر)

• **مفهوم برنامه سازی سیستم (system programming):** برنامه های سیستمی برنامه هایی هستند که بدون استفاده از توابع کتابخانه ای زبان، با سیستم عامل ارتباط برقرار کنند و یا بدون استفاده از توابع کتابخانه ای سیستم عامل با سخت افزار ارتباط برقرار کنند.

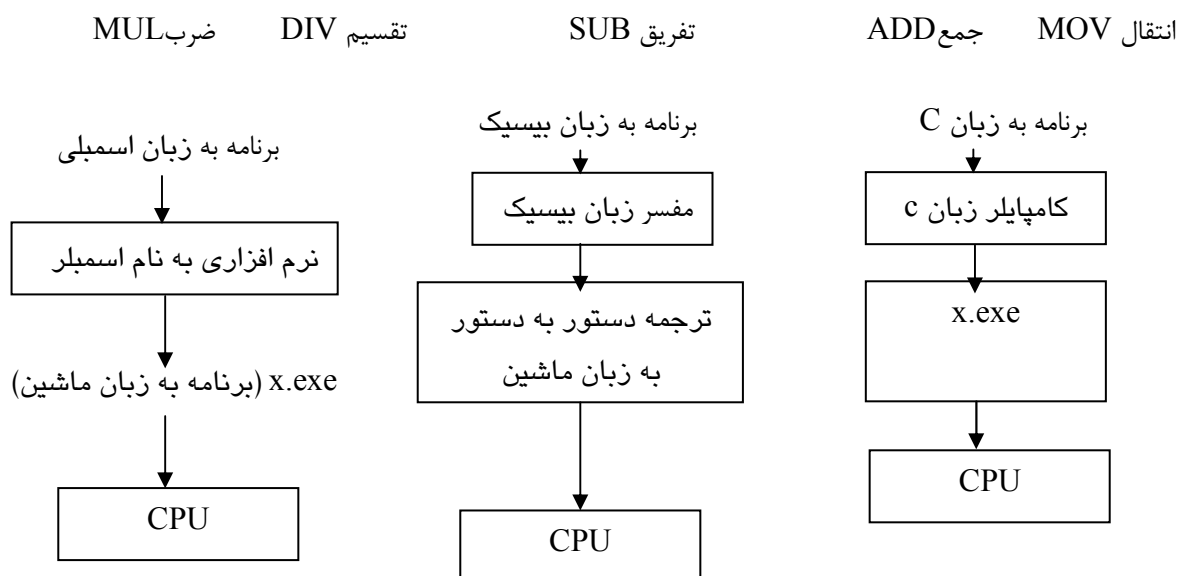
• **تعریف زبان ماشین (machine lang.) و زبان اسمبلی (assembly lang.):**

زبان ماشین زبانی است که دستورات آن به سخت افزار وابسته بوده و به ازای هر دستور یک مدار سخت افزاری در داخل cpu وجود دارد. هر کدام از دستورات زبان ماشین دارای یک کد عددی بوده که بصورت مبنای ۲ یعنی ۰ و ۱ در داخل حافظه و یا cpu قرار می گیرند. زبان اسمبلی همان زبان ماشین است ولی به جای ۰ کدهای ۱ از کلمات انگلیسی به عنوان نام دستورات استفاده می گردد. نرم افزار اسمبلر دستورات اسمبلی را به کدهای زبان ماشین ترجمه می کند.



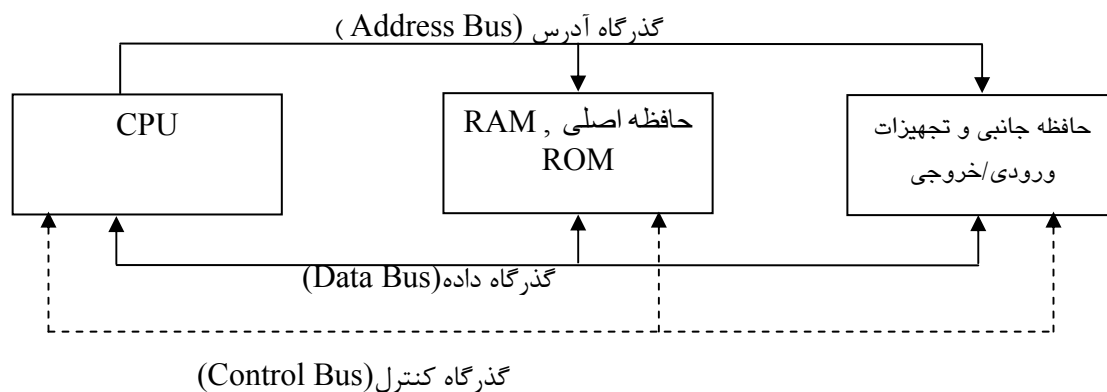
نکته: سخت افزار فقط و فقط دستورات زبان ماشین را متوجه می شود و اجرا می کند. زبان ماشین به معنی زبان سخت افزار است.

زبان ماشین مجموعه ای از دستورات است که برای هر دستور یک مدار منطقی در داخل CPU قرار گرفته است:



فصل دوم: آشنایی با سخت افزار و ویژگیهای آن

۱-۲- قطعات سخت افزاری یک سیستم کامپیوتری در معماری وان نیومن



- پردازنده یا پردازشگر (CPU)

CPU (Central Processing Unit) مهمترین و اصلی ترین بخش از یک سیستم کامپیوتری می باشد که همانند مغز انسان بوده و وظیفه پردازش و اجرای دستورات را بر عهده دارد. وظیفه دیگر cpu کنترل سایر تجهیزات می باشد. هر cpu از دو قسمت واحد کنترل (CU) و واحد محاسبه و مقایسه (ALU) تشکیل یافته است.

مشخصات تجاری یک پردازنده:

- نام شرکت سازنده: مثلاً INTEL و AMD
- مدل CPU: ۸۰۸۶، ۸۰۲۸۶، ...، pentium، pentium4 و ...
- سرعت اجرای دستورات برحسب HZ (در مورد کامپیوترهای بزرگ واحد سرعت MIPS (Milion Instruction Per Second) می باشد.
- سرعت BUS (سرعت نقل و انتقال داده ها) بر حسب واحد MHZ
- میزان حافظه cache (حافظه نهان): حافظه داخلی cpu

پردازنده های RISC (Reduced I.S.C.) ، CISC (Complicated Instruction Set Computers)

و ZISC (Zero I.S.C.)

CISC: تعداد دستورات زیاد و پیچیده، هزینه بالاتر، بیشتر مناسب برای کامپیوترهای شخصی

RISC: کاهش تعداد دستورات و در عوض بالا رفتن توان CPU، بیشتر مناسب کامپیوترهای صنعتی و بزرگ

ZISC: بر مبنای تطبیق الگو و بر اساس ایده شبکه های عصبی کار می کنند و مبتنی بر مجموعه دستورات (instruction set) نیستند.

امروزه پردازنده های dual core (دو هسته ای) و Quad core (چهار هسته ای) وجود دارند که یک پردازنده در هر لحظه می تواند در نقش دو یا ۴ پردازنده عمل نماید.

حافظه : حافظه برای ذخیره اطلاعات به کار می رود.

ram (random access memory):

- جهت ذخیره اطلاعات بصورت موقت بکار می رود و با خاموش شدن کامپیوتر اطلاعات آن هم پاک می گردد.
- از نظر ظاهری یک برد الکترونیکی با تعدادی IC می باشد که بر روی برد اصلی کامپیوتر نصب می گردد.
- مشخصات تجاری یک ram عبارتند از:
 - شرکت سازنده (مثلا ,lg,Samsung,gel,kingstone,apacer)
 - ظرفیت (مثلا ,1GB,2GB,1GB,512MB,256MB,128MB)
 - سرعت BUS (مثلا ,1066MHZ,133MHZ)
 - نوع معماری (SD RAM, DDR1,DDR2,...)

rom (read only memory):

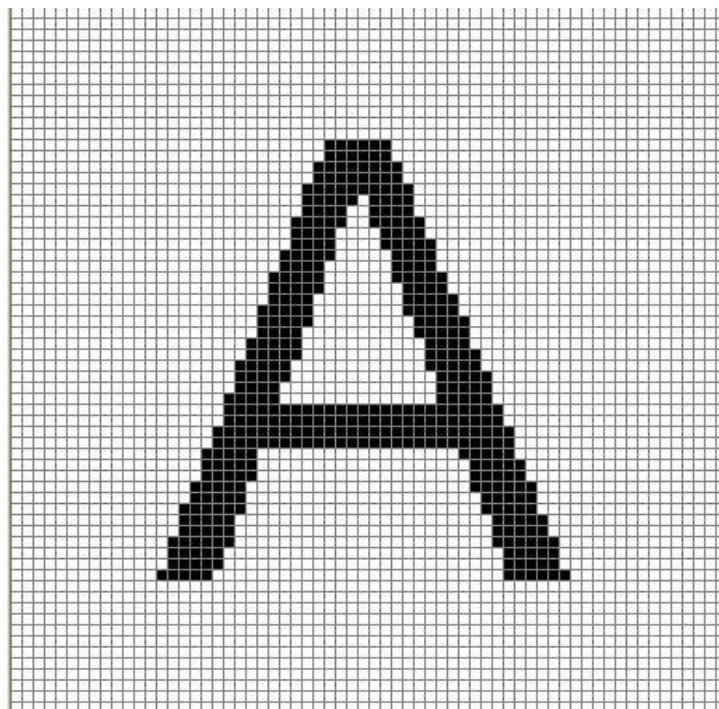
- اطلاعات آن فقط خواندنی بوده و قابل تغییر نمی باشد.
- این حافظه بصورت یک IC بوده و به عنوان بخشی از مادربرد محسوب می شود.
- اطلاعات خاص مورد نیاز سیستم مانند جداول گلیف (glyph) و نرم افزار BIOS توسط شرکت سازنده مادربرد در این حافظه قرار داده می شود و به روشهای خاصی امکان UPDATE کردن آنها وجود دارد.

نرم افزار BIOS: اولین نرم افزاری است که با روشن شدن کامپیوتر شروع به اجرا می کند.

بصورت سخت افزاری CPU بطور خودکار در شروع کار کامپیوتر به سراغ اولین دستور BIOS می رود.

وظایف BIOS:

- ۱- تست قطعات مختلف سخت افزاری سیستم بر مبنای تنظیمات انجام شده توسط کاربر (با روشن شدن کامپیوتر با زدن دکمه delete یا F2 وارد این تنظیمات می شویم)
 - ۲- کپی کردن برخی اطلاعات از جمله جداول گلیف و توابع موبوط به وقفه ها در RAM
 - ۳- جستجوی حافظه های جانبی جهت پیدا کردن سیستم عامل و بار کردن هسته آن در RAM (ترتیب جستجوی حافظه های جانبی توسط کاربر قابل تنظیم است) اگر سیستم عاملی یافت نشد، صدور پیغام خطا
- جداول گلیف:** حاوی نقشه بیتی جهت مشخص شدن شکل ظاهری کاراکترها



HARD DISK, FLOPPY DISK, FLASH MEMORY, CD ROM, DVD ROM, TAPE, PUNCH CARDS

این حافظه‌ها می‌توانند اطلاعات را به مدت زمان طولانی در خود ذخیره کنند. نسبت به حافظه‌های اصلی دارای حجم بیشتر، سرعت و قیمت پایینتری هستند.

– تجهیزات ورودی و خروجی: صفحه کلید، ماوس، چاپگر، مانیتور، رسام، میکروفون، اسکنر، بلندگو و ...

– سایر تجهیزات: کارت‌های مختلفی مانند شبکه، مودم و ...

۲-۲- مفهوم ۰ و ۱ در سیستم‌های کامپیوتری

هر دستور زبان ماشین دارای یک کد عددی می‌باشد که با صفر و یک نمایش داده می‌شود. مثال: 10001110

کارخانه سازنده CPU، این کدها را تعبیه کرده و ما فقط می‌توانیم از آنها استفاده کنیم.

سوال: چرا 0 و 1؟

جواب: کامپیوترهایی که داریم اگر بخواهند با ارقام ۰ تا ۹ کار کنند (کامپیوترهای آنالوگ یا رقمی) ساخت این کامپیوترها هزینه بالایی دارند زیرا باید ۱۰ سطح ولتاژ داشته باشیم و تشخیص این سطوح ولتاژ نیاز به مدارهای زیادی دارد برای همین کامپیوتر به صورت دیجیتال (دو سطحی) ساخته شده است که سیگنالهای رد و بدل شده در آن دو سطح پایین و بالا دارد. مثلاً (0V, 12V) یا (5V, -5V). اگر کامپیوتر آنالوگ باشد باید در خانه‌های حافظه بتوانیم ارقام ۰ تا ۹ را ذخیره کنیم. ولی در کامپیوترهای دیجیتال به راحتی با استفاده از ترانزیستور که می‌تواند در دو حالت قطع و شارژ قرار گیرد، اعداد 0 و 1 در داخل یک خانه از حافظه ذخیره می‌شوند.

0: نشان دهنده حالت قطع ترانزیستور

1: نشان دهنده حالت شارژ ترانزیستور است.

با دیدن داده 10001110 یاد هشت ترانزیستور در کنار هم می‌افتیم.

0: سطح ولتاژ پایین

1: سطح ولتاژ بالا

نحوه تبدیل داده‌های مختلف (عدد، کاراکتر، رشته، تصویر، صدا) به ۰ و ۱:

هر نوع اطلاعاتی را اعم از اعداد، کاراکترها، متون، تصاویر، صداها، بوها و ... را می‌توان با 0 و 1 نمایش داد:

اعداد: اعداد با استفاده از تبدیل شدن به معنای 2 ذخیره می‌شوند.

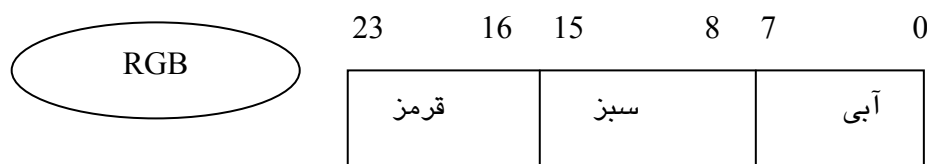
متن‌ها و کاراکترها: هر متن از تعدادی کاراکتر تشکیل می‌شود. به ازای هر کاراکتر یک کد عددی در نظر می‌گیریم و در واقع به جای ذخیره

کاراکتر، کد مربوطه ذخیره می‌گردد.

معروفترین کدگذاری برای کاراکترها: قبل از سال ۲۰۰۰ کدگذاری اسکی، بعد از سال ۲۰۰۰: کدگذاری unicode

تصاویر: در ذخیره تصاویر، رنگها را کدگذاری می کنیم. (در برخی از سیستمهای گرافیکی امروزی مثلا 16 میلیون رنگ داریم) مثلا در یکی از حالتهای گرافیکی، در صفحه خروجی کلاً 480×640 نقطه رنگ داریم، به ازای هر نقطه یک عدد به عنوان شماره رنگ ذخیره می کنیم که یک عدد ۳ بایتی است.

شماره ۲۴ بیتی برای رنگ ها در سیستم کدگذاری RGB: هر رنگی را می توان از ترکیب سه رنگ اصلی RED, GREEN و BLUE بدست آورد. در این سیستم کدگذاری برای هر کدام از این سه رنگ یک بایت در نظر گرفته می شود.



برای هر نقطه یک عدد ۳ بایتی را ذخیره می کنیم.

حافظه مورد نیاز: بایت $480 \times 640 \times 3$

صداها:

صدا = فرکانس (لحن صدا و زیر و بم بودن آن) + دامنه (میزان بلندی صدا)

صدا توسط میکروفرن به جریان الکتریکی تبدیل شده و توسط بلندگو دوباره به صدایی دیگر که بلندتر است تبدیل می شود.

مثال: یک موسیقی ۵ دقیقه ای را می خواهیم ذخیره کنیم، باید در هر لحظه فرکانس و دامنه آن را ذخیره کرده هر چه لحظات را نزدیک به هم بگیریم کیفیت صدا بهتر خواهد بود. به عنوان مثال در هر ثانیه ۱۰۰ نمونه می گیریم.

هر نمونه هم فرکانس دارد و هم دامنه که هر کدام یک عدد صحیح بدون علامت ۲ بایتی هستند. پی هر نمونه ۴ بایت حافظه لازم دارد.

تعداد نمونه ها: $5 \times 60 \times 100 = 30000$

کل حافظه لازم: کیلو بایت ۱۲۰ = بایت $30000 \times 4 = 120000$

برای پخش صدا عکس این قضیه اتفاق می افتد دو عدد فرکانس و دامنه را گرفته به بلندگو می دهیم و همان صدای قبلی را پخش می کنیم.

- میکروفرن ها یک پرده مرتعش دارند که فرکانس ارتعاش آن با فرکانس صدایی که به آن برخورد می کند برابر است.

- دامنه، میزان بالا و پایین رفتن صفحه و فرکانس، تعداد حرکت آن در ثانیه است.

- نحوه تبدیل دستورات به ۱۰۰

هر دستور زبان ماشین دارای یک کد عددی است که توسط شرکت سازنده CPU مشخص گردیده است. اگر این کد عددی را به مبنای ۲ ببریم نشان دهنده کد باینری آن دستور می باشد.

۲-۳- اساس کار یک cpu

CPU در حقیقت یک IC است که مانند هر مدار الکترونیکی تنها با سیگنال ها سرو کار دارد هر عملی که توسط CPU خواهد انجام شود

باید کد آن دستور از طریق پایه های CPU به آن منتقل شود.

همانطور که می دانید برنامه ها بصورت فایل های exe بر روی حافظه های جانبی قرار دارند. با اجرای فایل exe که در داخل آن دستورات زبان ماشین نوشته شده، دستورات برنامه در حافظه RAM بار می شوند (کار قسمت Loader سیستم عامل، بار کردن برنامه از Hard به RAM است) از داخل RAM دستورات برنامه یکی یکی به داخل CPU واکنشی (Fetch) می شوند و پس از رمزگشایی (Decode)، اجرا می شوند (Execute). نتایج، داخل ثبات های داخل CPU و یا در RAM ذخیره می شود. CPU یک شمارنده دستور به نام PC دارد که آدرس دستوری را در RAM نشان می دهد که باید واکنشی شود. CPU دستورات را رمزگشایی می کند تا بفهمد که دستور چیست و چه عملوندهایی دارد و سپس عملوندها به داخل CPU واکنشی می شوند و در نهایت دستور اجرا می شود.

۲-۴- آشنایی با گذرگاه ها

گذرگاه (Bus):

محلی که اطلاعات رد و بدل می شود، عملاً به طور فیزیکی Mother Board اطلاعات را منتقل می کند. تعدادی پایه CPU و RAM و وسایل جانبی به گذرگاه آدرس و تعدادی گذرگاه داده متصل است.

گذرگاه داده: محلی برای تبادل داده ها

گذرگاه داده خارجی:

یک گذرگاه دو طرفه است که برای نقل و انتقال داده ها به کار می رود هر چقدر پهنای باند (تعداد بیت ها) بیشتری داشته باشد سرعت نقل و انتقال داده ها بیشتر می شود. توسط گذرگاه داده اطلاعات خوانده یا نوشته می شود (دوطرفه است).

می خواهیم اطلاعاتی را به حجم ۲۰ بیت بخوانیم وقتی D.B، ۱۶ بیتی (۲ بیت) است داده ها ۲ بیت ۲ بایت خوانده می شوند، ۸۰۸۶ آن را در ۱۰ مرحله اما Pentium آنها را در سه مرحله می خواند، در نتیجه سرعت خواندن و نوشتن اطلاعات خیلی متفاوت است. اهمیت گذرگاه داده به قدری است که وقتی گفته می شود یک پردازنده n بیتی است منظور گذرگاه داده آن می باشد.

- گذرگاه داده در بالا بردن کارایی سیستم بسیار اهمیت دارد.

گذرگاه داده داخلی: برای نقل و انتقال اطلاعات در داخل CPU به کار می رود (بین ثبات ها) در بیشتر مواقع گذرگاه داده داخلی و گذرگاه داده خارجی تعداد بیت های یکسانی دارند. در واقع اندازه گذرگاه داده داخلی برابر با اندازه ظرفیت ثبات های داخلی CPU است.

گذرگاه آدرس:

گذرگاه آدرس یک طرفه است، هنگام خواندن یا نوشتن اطلاعات از حافظه و وسایل جانبی، آدرسی که می خواهیم اطلاعات از آن خوانده یا در آن نوشته شود توسط CPU برای حافظه یا وسایل جانبی فرستاده می شود.

- هر خانه از حافظه RAM و ROM و هر کدام از تجهیزات جانبی دارای یک آدرس منحصر به فرد هستند. جهت خواندن یا نوشتن اطلاعات باید آدرس مربوطه از CPU به همه حافظه ها و تجهیزات ارسال شود تا مشخص شود عمل خواندن یا نوشتن از کدام Device صورت پذیرد.

گذرگاه آدرس یک طرفه از CPU به سایر تجهیزات می باشد.

کل فضای آدرس $2^{20}=1MB$ 20 بیت : A.B

$$A.B : 2^{32} = 4GB \text{ بیت}$$

یعنی مجموع RAM و ROM حدود 4GB می تواند باشد.

اندازه A.B بر سرعت اجرای دستورات تأثیری ندارد.

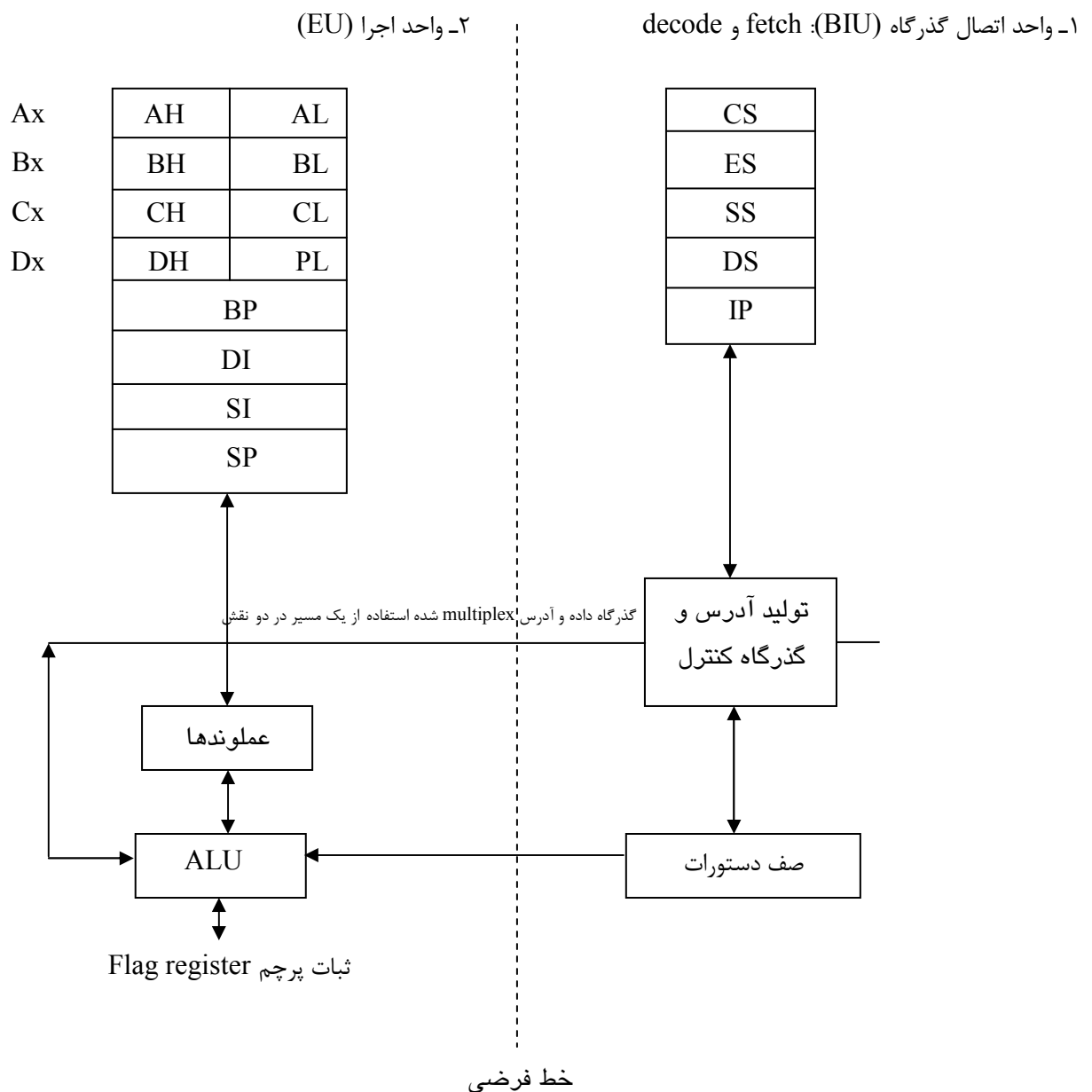
گذرگاه کنترل (Control Bus)

می خواهیم اطلاعاتی را از RAM خوانده یا داخل آن بنویسیم ، گذرگاه کنترل معین می کند که در حال حاضر چه عملی را می خواهیم انجام دهیم. به عبارت دیگر گذرگاهی است که از طریق آن سیگنال های کنترلی (خواندن از حافظه، نوشتن در حافظه) رد و بدل می شود.

سیگنال های کنترلی: خواندن از حافظه ، نوشتن در حافظه و سیگنال های بین سایر تجهیزات.

۲-۵- شمای کلی یک پردازنده 8086

ساختار داخلی یک پردازنده 8086



۲-۶- مفهوم خط لوله گی (pipelining)

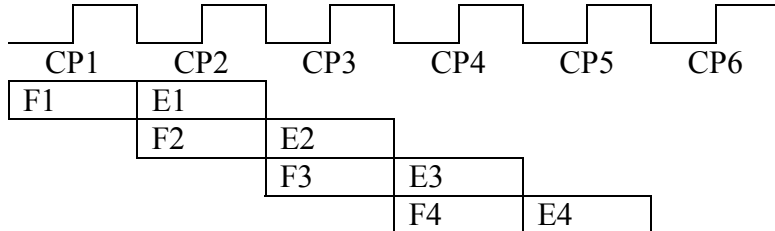
همانگونه که در شکل دیده می شود یک پردازنده ۸۰۸۶ به دو قسمت تقسیم شده است:

BIU: Bus interface unit

EU : Execution unit

قسمت اول وظیفه واکشی (fetch) کردن و رمزگشایی دستورات و عملوندها و قسمت دوم وظیفه اجرای دستورات را برعهده دارد.

با این تقسیم بندی، همزمان با اجرای یک دستور، دستور بعدی می تواند واکشی گردد. این تکنیک، خط لوله گی نامیده می شود.



۲-۷- مفهوم گذرگاه های multiplex شده

چون گذرگاه داده و آدرس در یک لحظه کوتاه هر دو مورد نیاز نیستند می توانیم از یک گذرگاه استفاده کنیم که برخی اوقات نقش گذرگاه

داده و برخی اوقات نقش گذرگاه آدرس را ایفا نماید این امر موجب کم شدن هزینه ساخت CPU می گردد. (اگر یکی از گذرگاه ها بزرگتر

باشد، گذرگاه multiplex شده به اندازه گذرگاهی که بزرگتر است ساخته می شود)

ثبات : حافظه های 1 تا 8 بایتی که برای ذخیره اطلاعات در داخل پردازنده استفاده می شوند. اندازه ثبات ها به گذرگاه داده داخلی بستگی

دارند. (زیرا از طریق گذرگاه داده داخلی اطلاعات بین ثبات ها رد و بدل می شود)

۲-۸- مشخصات تکنیکی پردازنده های شرکت intel

| نام پردازنده | 8085 | 8086 | 8088 | 80286 | 80386 | 80486 | pentium | Pentium pro | Pentium II | Pentium III | Pentium 4 |
|------------------|--------------|-------|-------|--------|--------|------------|------------|-------------|------------|-------------|-----------|
| سال تولید | 1976 | 1978 | 1979 | 1982 | 1985 | 1989 | 1992 | 1995 | ۱۹۹۷ | ۱۹۹۹ | ۲۰۰۰ |
| سرعت (MHZ) | 3-2 | 5-10 | 5-8 | 6-16 | 16-33 | 25-50 | 60-66 | 150 | ۲۳۳ | ۴۵۰ | 1500-3200 |
| تعداد ترانزیستور | 6500 | 29000 | 29000 | 130000 | 275000 | 1/2 میلیون | 3/1 میلیون | 5/5 میلیون | ۷/۵ میلیون | ۹/۵ میلیون | ۴۲ میلیون |
| حافظه فیزیکی | $2^{16}=64k$ | 1M | 1M | 16M | 4G | 4G | 4G | 64G | 64G | 64G | 64G |
| D.B داخلی | 8 | 16 | 16 | 16 | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
| D.B بیرونی | 8 | 16 | 8 | 16 | 32 | 32 | 64 | 64 | 64 | 64 | 64 |
| A.B | 16 | 20 | 20 | 24 | 32 | 32 | 32 | 36 | 36 | 36 | 36 |

فصل سوم: مبنای عددی

نمایش اعداد در مبنای مختلف: (مبنای ۲ (binary) ، مبنای ۸ (octal) ، مبنای ۱۰ (decimal) ، مبنای ۱۶ (Hexadecimal))

اعداد: صحیح، اعشاری

اعداد صحیح: علامتدار (signed)، بدون علامت (unsigned)

اعداد اعشاری: ممیز ثابت (Pixel point)، ممیز شناور (Floating Point)

تبدیل اعداد از مبنای ۱۰ به مبنای ۲، ۸، ۱۶: تقسیمات متوالی به مبنای مورد نظر

| | | |
|---|---|--|
| $250 / 2 = 125, 0$ $125 / 2 = 62, 1$ $62 / 2 = 31, 0$ $31 / 2 = 15, 1$ $15 / 2 = 7, 1$ $7 / 2 = 3, 1$ $3 / 2 = 1, 1$ $1 / 2 = 0, 1$ $(250)_{10} = (11111000)_2$ | $250 / 8 = 31, 2$ $31 / 8 = 3, 7$ $3 / 8 = 0, 3$ $(250)_{10} = (372)_8$ | $250 / 16 = 15, 10$ $15 / 16 = 0, 15$ $(250)_{10} = (FA)_{16}$ |
| | $1500 / 8 = 187, 4$ $187 / 8 = 23, 3$ $23 / 8 = 2, 7$ $2 / 8 = 0, 2$ $(1500)_{10} = (2734)_8$ | $1500 / 16 = 93, 12$ $93 / 16 = 5, 13$ $5 / 16 = 0, 5$ $(1500)_{10} = (5DC)_{16}$ |

| مبنا | رقمهای مورد استفاده | علامت مشخصه |
|------|---------------------|-------------|
| ۲ | 0,1 | b,B |
| ۸ | 0-7 | o,O |
| ۱۰ | 0-9 | d,D |
| ۱۶ | ۰-۹, A-F | h,H |

در مبنای ۱۶ با توجه به اینکه ۱۰ تا ۱۵ هم هر کدام یک رقم محسوب می شوند برای جلوگیری از اشتباه با اعداد ۲ رقمی از حروف زیر استفاده می شود.

| عدد مبنای | حرف |
|-----------|-----|
| 10 | A |
| 11 | B |
| 12 | C |
| 13 | D |
| 14 | E |
| 15 | F |
| 16 | G |

تبدیل اعداد از مبنای ۲ به سایر مبنای:

• مبنای ۲ به ۱۰: ضرب در توان های ۲

$$(11100110)_2 = (?)_{10}$$

$$1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 128 + 64 + 32 + 0 + 0 + 4 + 2 + 0 = 230$$

- مبنای 2 به 8: از سمت راست، 3 رقمی، 3 رقمی جدا می کنیم و معادل دهدهی هر یک را می نویسیم.

$$(11011001)_2 = (?)_8$$

$$011 \ 011 \ 001 = (331)_8$$

- مبنای 2 به 16: از سمت راست 4 رقمی 4 رقم جدا کرده و معادل دهدهی هر یک را می نویسیم.

$$(11011100)_2 = (?)_{16}$$

$$1101 \ 1100 = (DC)_{16}$$

- مبنای 8 و 16 به مبنای 10: ضرب در توانهای 8 یا 16

$$(275)_8 = (?)_{10} = 5 \times 8^0 + 7 \times 8^1 + 2 \times 8^2 = 5 + 56 + 128 = 189$$

$$(A2D5)_{16} = (?)_{10} = 5 \times 16^0 + 13 \times 16^1 + 2 \times 16^2 + 10 \times 16^3 = 80 + 208 + 40960 = 41248$$

- مبنای 8 به مبنای 2: هر رقم را جداگانه به مبنای 2 برده و بصورت 3 رقم صفر و یک می نویسیم در صورت لزوم صفر اضافه می کنیم تا 3 رقمی شود.

$$(275)_8 = (?)_2 = (010 \ 111 \ 0101)_2$$

- مبنای 16 به مبنای 2: هر رقم را جداگانه به مبنای 2 برده و بصورت 4 رقم می نویسیم در صورت لزوم صفر اضافه می کنیم تا 4 رقمی شود.

$$(A2D5)_{16} = (?)_2 = (1010 \ 0010 \ 1101 \ 0101)_2$$

۳-۱- اعداد صحیح

۳-۱-۱- اعداد صحیح بدون علامت

نحوه نمایش اعداد صحیح بدون علامت در داخل سیستمهای کامپیوتری: این نوع اعداد به روش گفته شده بالا به مبنای 2 برده شده و ذخیره می گردند

۳-۱-۲- اعداد صحیح علامتدار

نحوه نمایش اعداد صحیح علامت دار در داخل سیستمهای کامپیوتری:

۱. روش بیت علامت (sign bit): سمت چپ ترین بیت را به علامت عدد اختصاص می دهیم. صفر: مثبت، یک: منفی

۲. روش مکمل 1 (1's complement):

اعداد مثبت: مشابه روش بیت علامت

اعداد منفی: حالت مثبت عدد را می نویسیم و تمامی بیتها را مکمل می کنیم.

۳. روش مکمل 2 (2's complement):

اعداد مثبت: مشابه روش بیت علامت

اعداد منفی: حالت مثبت عدد را می نویسیم و از سمت راست تا رسیدن به اولین یک بدون تغییر و

بقیه بیتها را مکمل می کنیم.

مثال: عدد ۱۲- یک بایتی و دو بایتی در روشهای مختلف ذخیره سازی اعداد منفی

حالت یک بایتی:

بدون علامت:

$$(12)_{10}=(0000\ 1100)_2=0CH$$

بیت علامت:

$$\begin{aligned} (+12)_{10} &= (0000\ 1100)_2 = 0CH \\ (-12)_{10} &= (1000\ 1100)_2 = 8CH \end{aligned}$$

مکمل ۱:

$$\begin{aligned} (+12)_{10} &= (0000\ 1100)_2 = 0CH \\ (-12)_{10} &= (1111\ 0011)_2 = F3H \end{aligned}$$

مکمل ۲:

$$\begin{aligned} (+12)_{10} &= (0000\ 1100)_2 = 0CH \\ (-12)_{10} &= (1111\ 0100)_2 = F4H \end{aligned}$$

حالت دو بایتی:

بیت علامت:

$$\begin{aligned} (+12)_{10} &= (0000\ 0000\ 0000\ 1100)_2 = 000CH \\ (-12)_{10} &= (1000\ 0000\ 0000\ 1100)_2 = 800CH \end{aligned}$$

مکمل ۱:

$$\begin{aligned} (+12)_{10} &= (0000\ 0000\ 0000\ 1100)_2 = 000CH \\ (-12)_{10} &= (1111\ 1111\ 1111\ 0011)_2 = FFF3H \end{aligned}$$

مکمل ۲:

$$\begin{aligned} (+12)_{10} &= (0000\ 0000\ 0000\ 1100)_2 = 000CH \\ (-12)_{10} &= (1111\ 1111\ 1111\ 0100)_2 = FFF4H \end{aligned}$$

مثال : عدد 1720 - دوبایتی و چهار بایتی در روشهای مختلف ذخیره سازی اعداد منفی

حالت دو بایتی:

بدون علامت:

$$(1720)_{10}=(0000\ 0110\ 1011\ 1000)_2=06B8H$$

بیت علامت:

$$\begin{aligned} (+1720)_{10} &= (0000\ 0110\ 1011\ 1000)_2 = 06B8H \\ (-1720)_{10} &= (1000\ 0110\ 1011\ 1000)_2 = 86B8H \end{aligned}$$

مکمل ۱:

$$\begin{aligned} (+1720)_{10} &= (0000\ 0110\ 1011\ 1000)_2 = 06B8H \\ (-1720)_{10} &= (1111\ 1001\ 0100\ 0111)_2 = F947H \end{aligned}$$

مکمل ۲:

$$\begin{aligned} (+1720)_{10} &= (0000\ 0110\ 1011\ 1000)_2 = 06B8H \\ (-1720)_{10} &= (1111\ 1001\ 0100\ 1000)_2 = F948H \end{aligned}$$

حالت چهار بایتی:

بدون علامت:

$$(+1720)_{10}=(0000\ 0000\ 0000\ 0000\ 0110\ 1011\ 1000)_2=000006B8H$$

بیت علامت:

$$\begin{aligned} (+1720)_{10} &= (0000\ 0000\ 0000\ 0000\ 0110\ 1011\ 1000)_2 = 000006B8H \\ (-1720)_{10} &= (1000\ 0000\ 0000\ 0000\ 0110\ 1011\ 1000)_2 = 800006B8H \end{aligned}$$

مکمل ۱:

$$\begin{aligned} (+1720)_{10} &= (0000\ 0000\ 0000\ 0000\ 0110\ 1011\ 1000)_2 = 000006B8H \\ (-1720)_{10} &= (1111\ 1111\ 1111\ 1111\ 1001\ 0100\ 0111)_2 = FFFF947H \end{aligned}$$

مکمل ۲:

$$(+1720)_{10} = (0000\ 0000\ 0000\ 0000\ 0110\ 1011\ 1000)_2 = 000006B8H$$

$$(-1720)_{10} = (1111\ 1111\ 1111\ 1111\ 1111\ 1001\ 0100\ 1000)_2 = FFFFF948H$$

محدوده اعداد یک بایتی و دو بایتی در نمایشهای مختلف :

| دو بایتی | | یک بایتی | | روش نمایش | نوع عدد |
|--|--|---------------------------------|---------------------------------|------------|------------|
| Max | Min | Max | Min | | |
| 1111 1111 1111 1111 $2^{16}-1$ 65535 | 0000 0000 0000 0000 0 | 1111 1111 2^8-1 255 | 0000 0000 0 | بدون علامت | بدون علامت |
| 0111 1111 1111 1111 $+(2^{15}-1)$ +32767 | 1111 1111 1111 1111 $-(2^{15}-1)$ -32767 | 0111 1111 $+(2^7-1)$ +127 | 1111 1111 $-(2^7-1)$ -127 | بیت علامت | علامتدار |
| 0111 1111 1111 1111 $+(2^{15}-1)$ +32767 | 1000 0000 0000 0000 $-(2^{15}-1)$ -32767 | 0111 1111 $+(2^7-1)$ +127 | 1000 0000 $-(2^7-1)$ -127 | مکمل ۱ | |
| 0111 1111 1111 1111 $+(2^{15}-1)$ +32767 | 1000 0000 0000 0000 -2^{15} -32768 | 0111 1111 $+(2^7-1)$ +127 | 1000 0000 -2^7 -128 | مکمل ۲ | |

در اغلب سیستمهای سخت افزاری و نرم افزاری به دلیل مزایای زیر از روش مکمل ۲ برای اعداد علامتدار استفاده می شود:

۱- وجود نمایش منحصر بفرد برای عدد صفر:

| روش نمایش | +0 | -0 | توضیح |
|-----------|-----------|-----------|-----------|
| بیت علامت | 0000 0000 | 1000 0000 | $+0 < -0$ |
| مکمل ۱ | 0000 0000 | 1111 1111 | $+0 < -0$ |
| مکمل ۲ | 0000 0000 | 0000 0000 | $+0 = -0$ |

۲- امکان استفاده از عمل جمع بجای عمل تفریق :

| | | | |
|-------------|------|-----------------------------|-------------|
| 0001 1001 | 25 | به جای این تفریق از جمع | 0001 1001 |
| + 1111 0100 | - 12 | استفاده می کنیم ولی عدد دوم | - 0000 1100 |
| 0000 1101 | 13 | را مکمل ۲ می کنیم | 0000 1101 |

این مزیت باعث می شود نیازی به مدار سخت افزاری جداگانه تفریق کننده نباشد و از همان جمع کننده استفاده شود که باعث صرفه جویی در هزینه ساخت cpu و همچنین کاهش حجم cpu می شود.

۳-۲- اعداد اعشاری

۳-۲-۱- ممیز ثابت: اعداد اعشاری که در آنها تعداد ارقام اعشار ثابت است مانند اعداد نمایش دهنده دلار و سنت که تعداد ارقام اعشار همیشه ۲ است.

نحوه تبدیل به مبنای ۲:

روش اول: کل عدد را بدون در نظر گرفتن نقطه اعشار به عنوان یک عدد صحیح در نظر می گیریم و به مبنای ۲ تبدیل می کنیم. سپس تعداد ارقام اعشار را هم بصورت مبنای ۲ ذخیره می کنیم:

$$(17/345)_{10} = (?)_2$$

$$17345 = (0100\ 0011\ 1100\ 0001)_2$$

$$3 = (0000\ 0011)_2$$

0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1

روش دوم: سمت چپ نقطه اعشار به عنوان یک عدد صحیح و سمت راست آن هم به عنوان یک عدد صحیح ذخیره گردد.

$$17 = (0000\ 0000\ 0001\ 0001)_2$$

$$345 = (0000\ 0001\ 0101\ 1001)_2$$

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 1

۳-۲-۱- ممیز شناور

اعداد اعشاری که در آنها تعداد ارقام اعشار ثابت نیست و اعداد اعشاری خیلی بزرگ و خیلی کوچک (نزدیک به صفر) را هم شامل می شود.
- ابتدا مانند اعداد ممیز ثابت به مبنای 2 تبدیل می کنیم.

- عدد به دست آمده را به صورت نماد علمی با مانتیس نرمال می نویسیم ($0.1 \leq m \leq 1$)

- سپس مانتیس و نما را به صورت جداگانه ذخیره می کنیم.

- برای اعداد منفی از روش بیت علامت استفاده می کنیم و روشهای مکمل ۱ و ۲ استفاده نمی شود.

استاندارد ۳۲ بیتی اعداد ممیز شناور (float)

| مانتیس | نما (توان) | بیت علامت |
|--------|------------|-----------|
| 0 | 22 | 31 |
| | 23 | 30 |

استاندارد ۶۴ بیتی اعداد ممیز شناور (double)

| مانتیس | نما (توان) | بیت علامت |
|--------|------------|-----------|
| 0 | 51 | 63 |
| | 52 | 62 |

مثال : نحوه نمایش عدد ممیز ثابت 23.6- را در هر دو استاندارد ۳۲ و ۶۴ بیتی اعداد اعشاری ممیز شناور مشخص کنید.

$$\begin{aligned} (23.6)_{10} &= (10111.10011001)_2 \\ &= 1.011110011001 \times 2^4 \quad \text{غیر نرمال (M>1)} \\ &= 1011110011.001 \times 2^{-5} \quad \text{غیر نرمال (M>1)} \\ &= 0.001011110011001 \times 2^7 \quad \text{غیر نرمال (M<0.1)} \\ &= 10.11110011001 \times 2^3 \quad \text{غیر نرمال (M>1)} \\ &= 0.1011110011001 \times 2^5 \quad \text{نرمال (0.1<M<1)} \end{aligned}$$

$$(23.6)_{10} = 0.1011110011001 \times 2^5 \quad \text{نمایش بصورت نماد علمی با مانتیس نرمال:}$$

نکته: نما یک عدد علامتدار است. در این دو استاندارد به دلیل جلوگیری از درگیر شدن با توانهای منفی از توان ظاهری به جای توان واقعی

به صورت زیر استفاده می شود:

| تعداد بیتهای توان | محدوده توان واقعی | فرمول توان ظاهری | محدوده توان ظاهری |
|-------------------|---------------------------------|-------------------|---------------------------|
| 8 | $-2^7..2^7 = -128..127$ | توان واقعی + ۱۲۸ | $0..2^8 - 1 = 0..255$ |
| 11 | $-2^{10}..2^{10} = -1024..1023$ | توان واقعی + ۱۰۲۴ | $0..2^{11} - 1 = 0..2047$ |

نکته: در یک مانتیس نرمال همیشه سمت چپ ترین بیت برابر ۱ است لذا نیازی به ذخیره سازی آن نیست. در نتیجه دقت ذخیره مانتیس یک رقم افزایش خواهد یافت.

با توجه به نکات فوق عدد 23.6- بصورت زیر ذخیره می شود:

در استاندارد ۳۲ بیتی:

توان ظاهری $5 + 128 = 133 = 10000101$

| | | |
|----|----------|---------------------|
| 1 | 10000101 | 011110011001000...0 |
| 31 | 30 | 23 22 0 |

معادل این ۳۲ بیت در مبنای ۱۶: C2 BC C8 00H

در استاندارد ۶۴ بیتی:

توان ظاهری $5 + 1024 = 1029 = 10000000101$

| | | |
|----|-------------|---------------------|
| 1 | 10000000101 | 011110011001000...0 |
| 63 | 62 | 52 51 0 |

معادل این ۶۴ بیت در مبنای ۱۶: C0 57 99 00 00 00 00 00H

۳-۳- نمایش اعداد بدون علامت بصورت BCD(Binary Coded Decimal):

اگر یک عدد بدون علامت مبنای ۱۰ را مانند یک عدد مبنای ۱۶ در نظر بگیریم و به مبنای ۲ تبدیل کنیم، کد حاصل را BCD می گوئیم.

یعنی هر رقم عدد مبنای ۱۰ را به ۴ رقم باینری می نویسیم.

چند مثال:

| BCD | binary | عدد بدون علامت مبنای ۱۰ |
|---------------------|----------------|-------------------------|
| 0010 0101 | 0001 1001 | 25 |
| 0010 0101 0000 | 1111 1010 | 250 |
| 0001 0111 0010 0000 | 0110 1011 1000 | 1720 |

فصل چهارم: زبان اسمبلی پردازنده‌های ۸۰۸۶ و سازگار با آن

۴-۱- مفهوم ثبات و آشنایی با ثباتهای پردازنده های 8086

ثباتها حافظه‌های کوچک ۱، ۲، ۴ یا ۸ بیتی هستند که در داخل CPU قرار دارند و برای نگهداری کد دستور در حال اجرا، عملوندها و نتایج بدست آمده بکار می‌روند.

معرفی ثبات های پردازنده‌های ۱۶ بیتی (از ۸۰۸۶ تا ۸۰۲۸۶): در این پردازنده‌ها تمامی ثباتها ۱۶ بیتی هستند.

۱. ثبات های عمومی : پرکاربردترین ثباتها هستند که جهت ذخیره عملوندها بکار می‌روند. در اغلب دستورات زبان ماشین حداقل یکی از عملوندها باید در این ثباتها قرار گیرند.

| | 15 | 8 | 7 | 0 |
|----|----|---|---|----|
| AX | AH | | | AL |
| BX | BH | | | BL |
| CX | CH | | | CL |
| DX | DH | | | DL |

۲. ثبات های قطعه (Segment): هنگام اجرای دستورات نقشی ندارند ولی هنگام واکشی دستورات یا داده‌ها مورد استفاده قرار می‌گیرند. هر کدام از این ثباتها آدرس یکی از قطعات اصلی برنامه را در خود ذخیره می‌کنند.

DS: Data Segment قطعه داده

CS: Code Segment قطعه کد

SS: Stack Segment قطعه پشته

ES: Extra Segment قطعه اضافی

۳. ثبات‌های اشاره‌گر: برای دسترسی به داده‌ها از طریق آدرس آنها مورد استفاده قرار می‌گیرند.

IP : Instruction Pointer

SP : Stack Pointer

BP : Base Pointer

۴. ثبات های اندیس: مشابه ثباتهای اشاره‌گر

DI : Destination Index

SI : Source Index

۵. ثبات وضعیت یا پرچم (Flag register): جهت مشخص کردن وضعیت نتیجه محاسبه شده

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| R | R | R | R | O | D | I | T | S | Z | U | A | U | P | U | C |
| | | | | F | F | F | F | F | F | | F | | F | | F |

R: رزرو شده

Undefined: تعریف نشده

OF (Overflow Flag): اگر پس از انجام یک دستور بر روی اعداد علامت دار سر ریز رخ دهد این بیت برابر 1 می‌شود. (دو عدد یک بیتی با هم جمع شوند نتیجه در یک بایت جا نگیرد).

DF (Direction Flag): اگر 1 باشد عملیات مربوط به رشته‌ها از راست به چپ و اگر صفر باشد از چپ به راست انجام می‌شود. (مثلاً برای کپی کردن).

IF (Interrupt Flag): اگر 1 باشد «وقفه‌ها» فعال و اگر 0 باشد غیرفعال می‌شود.

TF (Trap Flag): اگر 1 باشد برنامه دستور به دستور اجرا می شود. (برنامه کامل اجرا می شود).

SF (Sign Flag): علامت نتیجه محاسبه شده در دستور قبل را نشان می دهد. صفر: مثبت یک: منفی

ZF (Zero Flag) :- صفر بودن یا نبودن نتیجه محاسبه شده در دستور قبل را نشان می دهد (هرگاه 1 بشود یعنی نتیجه صفر بوده).

AF (Auxiliary Flag) : اگر 1 باشد از بیت 3 به 4 رقم ثقلی داشته ایم .

PF (Parity Flag) : برای تست خطای انتقال داده ها به کار می رود اگر عدد حاصل تعداد زوج 1 داشته باشد PF نیز 1 می شود.

CF (Carry Flag) : اگر پس از انجام یک عمل بر روی اعداد بدون علامت سر زیر رخ دهد این بیت 1 می شود.

مثال: بررسی می کنیم که دستورات زیر چه تأثیری بر روی ثبات پرچم می گذارند.

MOV BH, 38 H

ADD BH, 2FH

ابتدا اعداد را به صورت مبنای 2 می نویسیم و بررسی می کنیم که مقادیر CF ، OF ، IF ، SF ، AF پس از این دستور چه عددی خواهد

بود؟

| | | | |
|-----------|-----------------|-------------------|-----------------|
| | | <u>بدون علامت</u> | <u>علامتدار</u> |
| 38H | 0011 1000 | 56 | +56 |
| + 2FH | + 0010 1111 | + 47 | + +47 |
| <hr/> 67H | <hr/> 0110 0111 | <hr/> 103 | <hr/> +103 |

CF : 0 OF : 0 ZF : 0 SF : 0 AF : 1 PF : 0

روش محاسبه OF :

OF=CF xor (رقمی نقلی رسیده به سمت علامت)

مثال :

MOV AL, 9CH

ADD AL, 64H

| | | | | |
|-----------|-----------------|--|-------------------|-----------------|
| | | | <u>بدون علامت</u> | <u>علامتدار</u> |
| 9CH | 1001 1100 | | 156 | -100 |
| + 64H | + 0110 0100 | | + 100 | + +100 |
| <hr/> 00H | <hr/> 0000 0000 | | <hr/> 256 | <hr/> 0 |

CF :1 OF :0 ZF :1 SF:0 AF :1 PF :1

اگر اعداد را علامت دار فرض کنیم حاصل عملیات اگر سر ریز داشته باشد در آن صورت مقدار OF برابر 1 خواهد بود.

مفهوم : حاصل جمع واقعا آنچه به دست آمده هست یا نه ؟

OF برای اعداد علامت دار بحث می شود. CF برای اعداد معمولی کافی است.

در مثال فوق اگر اعداد را علامت دار فرض کنیم، مجموع به دست آمده معتبر است به همین دلیل $OF = 0$ ولی اگر اعداد را بدون علامت فرض کنیم، مجموع به دست آمده معتبر نیست به همین دلیل $OF = 1$

مثال:

| | | بدون علامت | | علامتدار |
|-------|-------------|------------|-------|-----------|
| 1AH | 0001 1010 | 26 | 26 | |
| - 81H | - 1000 0001 | - 129 | - 127 | |
| | | | | 153 |
| | | | | |
| | | | | 1001 1001 |

CF :1 OF :1 ZF :0 SF:1 AF :0 PF :1

۴-۲- مفهوم پردازنده های n بیتی و آشنایی با ثباتهای پردازنده های ۳۲ بیتی

پردازنده 16 بیتی : یعنی پردازنده ای که حداکثر می تواند عملیات را بر روی داده های 16 بیتی انجام دهد مثلا دو عدد ۱۶ بیتی را با هم جمع یا در هم ضرب کند. مانند : 8086 ، 8088 ، 80286 (که خیلی قدیمی هستند).

پردازنده ۳۲ بیتی : یعنی پردازنده ای که حداکثر می تواند عملیات را بر روی داده های ۳۲ بیتی انجام دهد مثلا دو عدد ۳۲ بیتی را با هم جمع یا در هم ضرب کند. مانند : 80386 تا pentium 4

پردازنده ۶۴ بیتی : یعنی پردازنده ای که حداکثر می تواند عملیات را بر روی داده های ۶۴ بیتی انجام دهد مثلا دو عدد ۶۴ بیتی را با هم جمع یا در هم ضرب کند. مانند : itanium و xeon از محصولات اینتل

- سیستم های عاملی که بر روی یک سخت افزار n بیتی قرار می گیرند نیز n بیتی هستند. مثلا dos یک سیستم عامل ۱۶ بیتی، ویندوز ۹۵،۹۸،۲۰۰۰ و اکثر نسخه های xp و vista ۳۲ بیتی هستند. برخی نسخه های xp و vista، ۶۴ بیتی هستند.
 - کامپایلرها یا مفسرها، درایورها و در کل تمامی برنامه هایی که بر روی یک سیستم عامل n بیتی اجرا می شوند نیز n بیتی هستند.
 - نرم افزارهایی که با استفاده از یک کامپایلر n بیتی تولید و یا بر روی یک مفسر n بیتی اجرا می شوند نیز n بیتی محسوب می گردند.
- برنامه نویسی پردازنده های ۳۲ بیتی به دو حالت امکان پذیر است:

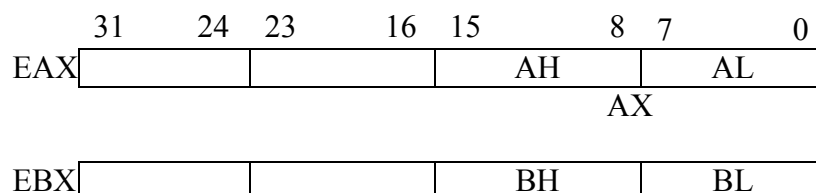
۱- حالت حفاظت شده (Protected mode): فرض می کنیم پردازنده 16 بیتی است.

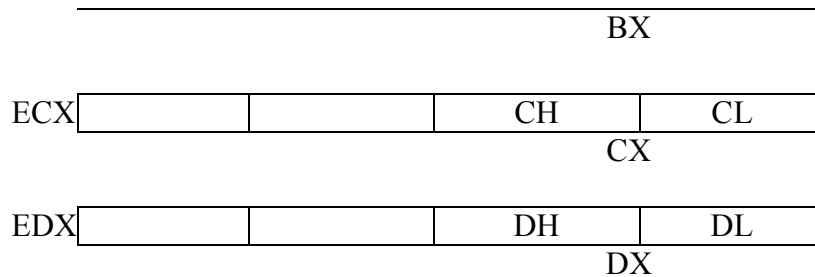
۲- حالت واقعی (Real mode): فرض می کنیم پردازنده 32 بیتی است.

اگر برنامه 32 بیتی را بخواهیم بر روی پردازنده 16 بیتی اجرا کنیم امکان ندارد.

معرفی ثبات های پردازنده های ۳۲ بیتی (از ۸۰۲۸۶ به بعد): در این پردازنده ها تمامی ثباتها به جز ثباتهای سگمنت ۳۲ بیتی هستند.

۱. ثبات های عمومی (۳۲ بیتی) :

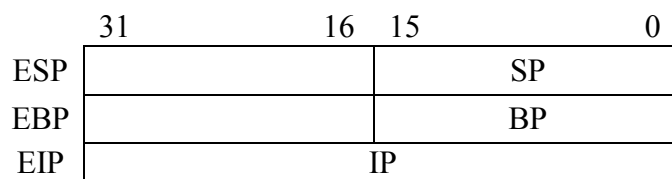




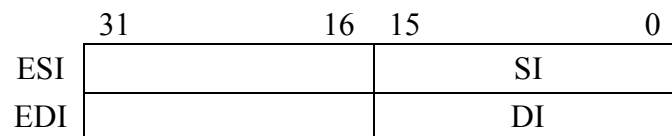
۲. ثبات های قطعه (۱۶ بیتی) :

CS,DS,SS,ES,FS,GS

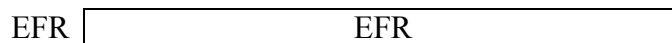
۳. ثبات های اشاره گر (۳۲ بیتی): برای دسترسی به داده ها از طریق آدرس آنها مورد استفاده قرار می گیرند.



۴. ثبات های اندیس: مشابه ثبات های اشاره گر



۵. ثبات وضعیت یا پرچم (۳۲ بیتی):



۶. ثبات های کنترل (Control register): CR0 , CR1 , CR2 , CR3

۴-۳- ساختار کلی برنامه ها در زبان اسمبلی:

هر برنامه اسمبلی از تعدادی قطعه (segment) تشکیل می گردد.

تعداد قطعه ها محدودیتی ندارد ولی اندازه هر قطعه حداکثر می تواند 64K بایت باشد.

هر قطعه در یکی از نقش های زیر بکار گرفته می شود: قطعه کد (code segment)، قطعه داده (data segment)، قطعه پشته (stack segment)

ثبات های cs,ds,ss به ترتیب باید به ابتدای قطعه های کد، داده و پشته اشاره نمایند ولی ES در اختیار برنامه نویس است و به هر سگمنتی می تواند اشاره کند.

نحوه تعریف قطعه های کد و داده :

| | |
|---------|----------|
| segment | نام قطعه |
| ends | نام قطعه |

در مورد قطعه پشته :

| | |
|---------------|----------|
| segment stack | نام قطعه |
| ends | نام قطعه |

۴-۴- یک برنامه نمونه در اسمبلی و شرح دستورات آن:

```

seg1 segment
    x DW 20
    y DW 50
    i DW ?
seg1 ends
myseg segment stack
    dw 100 dup(0)
myseg ends
seg2 segment
    L: Assume CS: Seg2 , DS: Seg1
        .
        .
        .
        mov ah,4CH
        int 21H
Seg2 ends
End L

```

نحوه تعریف متغیرها :

| نام متغیر | نوع متغیر | مقدار اولیه |
|-----------|-----------------|-------------|
| X | DB | 10 |
| | از نوع یک بایتی | مقدار |

در اسمبلی نوع داده های زیر را داریم:

| | |
|-------------------------|---------|
| DB: Define Byte | 1 بایت |
| DW: Define Word | 2 بایت |
| DD : Define Double Word | 4 بایت |
| DQ : Define Quad Word | 8 بایت |
| DT : Define Tetra Word | 10 بایت |

A DW ? بدون مقدار اولیه

A DW 20 Dup(0) تعریف آرایه به طول 20 و مقدار اولیه 0

x DB 5,-3,?,10

DW 5 داده بدون نام هم می توان تعریف کرد:

نحوه تعریف ثابت ها :

MAX EQU 1000

- داخل برنامه هر جا Max استفاده شود خود اسمبلی (نرم افزار مترجم) به جای آن 1000 قرار می دهد.

- ثابت ها خلاف متغیرها نیاز به حافظه ندارند زیرا قرار نیست مقدار آنها ذخیره شود و اسمبلی هنگام ترجمه مقدار ثابت را بجای متغیر استفاده شده قرار می دهد. برای همین تعداد بایت ندارند.

نکته ۱: زبان اسمبلی Case sensitive نمی باشد یعنی حروف کوچک و بزرگ تفاوتی ندارد.

دستورات و شبه دستورات:

شبه دستورات جهت راهنمایی اسمبلر استفاده می شود و در زمان اجرا اصلاً وجود ندارد ولی دستورات به زبان ماشین ترجمه شده و در زمان اجرا وجود دارند.

چند نمونه از شبه دستورات به صورت زیر است : DB , DW , DD , DQ , DT , EQU , Segment , ends , end

شبه دستور Assume (فرض کردن، تصور کردن) :

در اسمبلی سه نوع قطعه داریم (سگمنت کُد، سگمنت داده، سگمنت پشته) به کمک شبه دستور Assume نوع هر قطعه را تعیین می کنیم.
یک مثال ساده:

آشنایی با دستور Mov :

MOV dest , source

کپی کردن source در dest با شرایط و قوانین زیر:

۱- هر دو عملوند همزمان نباید مکانهای حافظه باشند.

۲- اندازه عملوندها باید یکسان باشد.

۳- مقصد نمی تواند ثبات IP یا CS یا ثابت باشد

۴- اگر مقصد DS,SS,ES باشد، مبدا باید یکی از ثباتهای عمومی باشد.

مثال :

درستی یا نادرستی کدهای زیر را تعیین کنید.

MOV x,10 ✓
MOV y, x ×
MOV 25 , z ×
MOV CS , 0 ×
MOV IP , Ax ×
MOV DS .0 ×
MOV DS , Ax ✓
MOV SS , x ×

نکته ۲: این محدودیتها در مورد تمامی دستوراتی که دو عملوند دارند وجود دارد. مانند: SUB، ADD و ...

نکته ۳: اعداد مبنای ۱۶ اگر با یکی از حروف A تا Z شروع می شوند حتما باید یک صفر به ابتدای آنها اضافه نماییم.

mov ax,0FF13H صحیح:

mov ax,FF13H غلط:

علاوه بر شبه دستور Assume باید در ابتدای برنامه در صورت لزوم آدرس شروع قطعه های داده و پشته به ترتیب در DS و SS قرار داده شوند. (در مورد قطعه کد این کار صورت نمی گیرد).

در برنامه قبلی بعد از دستور Assume باید نوشت :

MOV DS, Seg1 →

برای اتمام برنامه و برگشت به سیستم عامل باید دستورات زیر را به انتهای در برنامه (هر کجا که بخواهیم برنامه تمام شود) اضافه می کنیم.

MOV AH , 4CH

INT 21 H

• دستورات ADD و SUB :

ADD Dest , Source → Dest= Dest+ Source

SUB Dest و Source → Dect = Dest - Source

• دستورات ADC و SBB :

ADC Dest, Source → Dest= Source+carest + carry flag

SBB Dest و Source → Dect = Dest - Source - carry flag

این دو دستور موقعی مفیدند که می خواهیم رقم نقلی یا قرضی حاصل از جمع یا تفریق قبلی در جمع یا تفریق فعلی لحاظ گردد.

مثال : جمع کردن دو عدد FF253EH و 26E45H

Mov ax,253EH

Add ax,6E45H

Mov dx,0FFH

Adc dx,2

Dx:ax: حاصل جمع

مثال : تفریق کردن دو عدد FF253EH و 26E45H

mov ax,253EH

sub ax,6E45H

mov dx,0FFH

sbb dx,2

Dx:ax: حاصل تفریق

۴-۵- روش نوشتن ، ترجمه و اجرای یک برنامه در زبان اسمبلی :

روش اول: استفاده از نرم افزار EMU8086

این نرم افزار یک ویرایشگر زبان اسمبلی ۱۶ بیتی است و مراحل اشکالزدایی، ترجمه و اجرای برنامه های اسمبلی را برای برنامه نویس آسان

می سازد ولی با توجه به اینکه موقع ترجمه برنامه برخی دستکاریها در برنامه کاربر انجام می دهد، ممکن است این کار با مذاق برخی برنامه نویسان سیستم جور در نیاید.

```

edit: P:\asm\pp1.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help About
0337 inc ah
0338 putch al,ah,1,'*'
0339 inc ah
0340 putch al,ah,1,'*'
0341 inc ah
0342 putch al,ah,4,'*'
0343 popall
0344 endm
0345 draw2 macro color
0346 pushall
0347 mov bl,color
0348 mov al,x2
0349 mov ah,y2
0350 putch al,ah,1,'*'
0351 inc ah
0352 putch al,ah,1,'*'
0353 inc ah
0354 putch al,ah,1,'*'
0355 inc ah
0356 sub al,2
0357 putch al,ah,5,'*'
0358 popall
0359 endm
0360 dseg segment
0361 x1 db 10
0362 y1 db 10
0363 x2 db 20
0364 y2 db 10
0365 dseg ends
0366 sseg segment stack
0367 du 50 dup(?)
0368 sseg ends
0369 seg1 segment
0370 assume cs:seg1,ds:dseg,ss:sseg
0371 l:mov ax,dseg
0372 mov ds,ax
0373 mov ax,sseg
0374 mov ss,ax
0375 start:draw1 0FH
0376 draw2 0FH
0377 l0:getch
0378 cmp ah,1
0379 jne l1
0380 jmp exit
0381 l1:cmp ah,4bh ; left key
0382 je l2
0383 jmp R
0384 l2:draw1 0
0385 dec x1
0386 cmp x1,-1
0387 jne l3
0388 mov x1,76
0389 l3:jmp start
0390 R:cmp ah,4dh ;right key

```

روش دوم: ترجمه و اجرای دستی برنامه ها

۱- کد برنامه را در یک ویرایش گر نوشته (مثلاً notepad) با پسوند asm آن را ذخیره می کنیم(مثلا test.asm). نوع کدگذاری را ansi تعیین کنید نه unicode

۳- ترجمه برنامه به زبان ماشین با استفاده از یک اسمبلر(assembler) مانند masm(macro assembler) یا tasm(turbo

assembler). برای این منظور وارد پنجره command ویندوز شوید(از طریق قسمت start منوی run و وارد کردن دستور

cmd) سپس به مسیری که این برنامه ها در آنجا قرار دارند رفته و دستورات زیر را تایپ نمایید:

masm x.asm; یا tasm x.exe;

- اگر از ; استفاده نشود، یک سری سوال از کاربر پرسیده خواهد شد.

- ذکر پسوند asm. اجباری نیست.

۴- تولید فایل اجرایی با استفاده از یک پیوند دهنده(linker) مانند link.exe یا tlink.exe با تایپ دستورات زیر:

link x.obj; یا tasm x.obj;

- ذکر پسوند obj اجباری نیست.

۵- در صورت لزوم می‌توانید از یک نرم‌افزار شبیه‌ساز مانند td.exe(turbo debugger) یا cv(code viewer) و یا 8086 emulator جهت اشکالزدایی برنامه و مشاهده محیط زمان اجرا و trace کردن برنامه بهره بگیرید.

```
C:\WINDOWS.0\system32\cmd.exe
P:\asm>masm pp1.asm;
Microsoft (R) Macro Assembler Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985. All rights reserved.

48906 Bytes symbol space free
      0 Warning Errors
      0 Severe Errors
P:\asm>link pp1.obj;
Microsoft (R) 8086 Object Linker Version 3.05
Copyright (C) Microsoft Corp 1983, 1984, 1985. All rights reserved.
P:\asm>pp1_
```

مثال: مراحل فوق را بر روی برنامه زیر انجام دهید.

Seg1 Segment

```
X dw 15
Y dw 25
Z dw ?
Seg1 ends
Seg2 segment
L: assume cs: seg2, ds: seg1
MOV ax, seg1
MOV ds, ax
MOV ax, x
MUL y
MOV z, ax
MOV ah, 4CH
int 21H
Seg2 ends
End L
```

نرم افزارهای شبیه‌ساز مانند 8086 emulator و Turbo Debugger را اگر اجرا کنیم گویی که داخل CPU رفته باشیم، در این محیط لیست ثبات ها با مقدارشان نوشته شده است. (در حقیقت یک نرم افزار شبیه ساز، محیط CPU را شبیه سازی می کند) از داخل آن می توان فایل exe را باز کرد و دستورات زبان ماشین داخل آن را به زبان اسمبلی مشاهده کرد.

در یک برنامه ترجمه شده به زبان ماشین به جای نام متغیرها، آدرس آنها قرار می گیرد. کلیه نام های دیگر مانند برجسب دستورات، نام ثابتها

و زیربرنامه ها نیز به آدرس ها تبدیل می شوند. همچنین اعدادی که اینجا می بینیم در مبنای 16 هستند که برای مقایسه با عدد مبنای 10

نیاز به تغییر مبنا داریم.

مثلا x که اولین متغیر ما بود آدرس [0000] را گرفته است . (دو بایت حافظه اشغال می کند).

با کلید F7 برنامه Turbo Debugger خط به خط اجرا می شود.

۴-۶- دستورات mul , div (ضرب و تقسیم)

فرمت کلی دستور mul :

mul Source

اگر Source یک بایتی باشد:

$Ax = AL * Source$

اگر Source دوبایتی باشد:

$Dx : Ax = Ax * Source$

Source می تواند ثبات یا مکان حافظه باشد ولی نمی تواند یک ثابت باشد.

در مورد مکانهای حافظه با شبه دستورات byte ptr , word ptr یک بایتی بودن یا دوبایتی بودن آن محل مشخص می شود.

div word ptr[si] محل اشاره شده را دوبایتی در نظر بگیرید. (از جایی که Si اشاره می کند، 2 بایت در نظر بگیر).

div byte ptr[si] محل اشاره شده را یک بایتی در نظر بگیرید. (از جایی که Si به آن اشاره می کند 1 بایت در نظر بگیر)

مثال :

```
mov Ax,75
mov cx,2
mul cx
```

$Dx:ax=ax*cx=75*2=150=0096H \rightarrow dx=0,ax=96H=150$

مثال :

```
mov Al,5
mov bl,20
mul bl
```

$ax=al*bl=5*20=100=64H$

مثال :

```
mov Ax,500
mov cx,3
mul cx
```

$Dx:ax=ax*cx=500*3=1500=05DCH \rightarrow dx=0,ax=05DCH=220$

فرمت کلی دستور div :

div Source

اگر Source یک بایتی باشد: خارج قسمت $AL = \frac{Ax}{Source}$ و باقیمانده $AH = \frac{Ax}{Source}$.

اگر Source دوبایتی باشد: خارج قسمت $Ax = \frac{Dx : Ax}{Source}$ ، باقیمانده $Dx = \frac{Dx : Ax}{Source}$.

مثال: می‌خواهیم عدد ۲۵۶۰ را بر عدد ۱۲ تقسیم نماییم.

2560= 0A00H
MOV Dx,0AH

MOV Ax,0

MOV Bx,12

Div Bx → عملوند 2 بایتی است

31 Dx 16 15 Ax 0

Dx :Ax

| | |
|------|------|
| 0000 | 0A00 |
|------|------|

AX=213=D5H,Dx=4

نتیجه:

Dx

Ax

باقی مانده

| |
|----|
| 04 |
|----|

D5

| |
|----|
| D5 |
|----|

خارج قسمت

امکان دارد تقسیم به گونه ای باشد که سر ریز رخ دهد.

مثال : دستورات زیر را در نظر بگیرید.

MOV Ax , 50000

MOV BL,2

div BL → یک بایتی

$$\frac{Ax}{BL} = \frac{50000}{2} = 25000$$

$$\begin{array}{r|l} 50000 & 2 \\ \hline 50000 & 25000 \\ \hline 0 & \end{array}$$

AH ← 0 AL ← 25000

- خطای سرریز رخ می دهد

۴-۷- دستورات مقایسه و پرش

دستور cmp :

cmp op1,op2

این دستور دو عملوند را مقایسه می‌کند و ثبات پرچم را تحت تاثیر قرار می‌دهد. این دستور معمولاً به تنهایی بکار نمی‌رود بلکه بلافاصله بعد از آن از دستورات پرش شرطی استفاده می‌کنیم تا نتیجه مقایسه را تشخیص دهیم.

دستورات jmp و jcc :

دستور jmp دستور پرش غیرشرطی نامیده می‌شود و اجرای برنامه را به یک دستور منتقل می‌کند..

Jmp L

L برچسب دستور مورد نظر است. به جای برچسب می‌توان از آدرس دستور هم استفاده کرد.

مجموعه دستورات jcc پرش شرطی نامیده می‌شوند و بلافاصله بعد از دستور cmp بکار می‌روند و بر اساس نتیجه مقایسه رفتار می‌نمایند:

Jcc L

نکته مهم: فاصله بین این دستور پرش تا دستور L باید بین 128- تا 127 باشد

| دستورات پرش شرطی که بیت‌های ثبات پرچم را تست می‌کنند | | | | |
|--|---------------|-----------|----------|-------|
| دستور | توضیح | شرط | معادل | متضاد |
| JC | Jump if carry | Carry = 1 | JB, JNAE | JNC |

| | | | | |
|-----|----------|------------|---------------------|-----|
| JC | JNB, JAE | Carry = 0 | Jump if no carry | JNC |
| JNZ | JE | Zero = 1 | Jump if zero | JZ |
| JZ | JNE | Zero = 0 | Jump if not zero | JNZ |
| JNS | - | Sign = 1 | Jump if sign | JS |
| JS | - | Sign = 0 | Jump if no sign | JNS |
| JNO | - | Ovrflw=1 | Jump if overflow | JO |
| JO | - | Ovrflw=0 | Jump if no Ovrflw | JNO |
| JNP | JPE | Parity = 1 | Jump if parity | JP |
| JPO | JP | Parity = 1 | Jump if parity even | JPE |
| JP | JPO | Parity = 0 | Jump if no parity | JNP |
| JPE | JNP | Parity = 0 | Jump if parity odd | JPO |

| دستورات پرش شرطی برای مقایسه اعداد بدون علامت | | | | |
|---|-------------------------------------|-----------------------|----------|-------|
| دستور | توضیح | شرط | معادل | متضاد |
| JA | Jump if above (>) | Carry=0, Zero=0 | JNBE | JNA |
| JNBE | Jump if not below or equal (not <=) | Carry=0, Zero=0 | JA | JBE |
| JAE | Jump if above or equal (>=) | Carry = 0 | JNC, JNB | JNAE |
| JNB | Jump if not below (not <) | Carry = 0 | JNC, JAE | JB |
| JB | Jump if below (<) | Carry = 1 | JC, JNAE | JNB |
| JNAE | Jump if not above or equal (not >=) | Carry = 1 | JC, JB | JAE |
| JBE | Jump if below or equal (<=) | Carry = 1 or Zero = 1 | JNA | JNBE |
| JNA | Jump if not above (not >) | Carry = 1 or Zero = 1 | JBE | JA |
| JE | Jump if equal (=) | Zero = 1 | JZ | JNE |
| JNE | Jump if not equal () | Zero = 0 | JNZ | JE |

| دستورات پرش شرطی برای مقایسه اعداد علامتدار | | | | |
|---|---|-------------------------|-------|-------|
| دستور | توضیح | شرط | معادل | متضاد |
| JG | Jump if greater (>) | Sign = Ovrflw or Zero=0 | JNLE | JNG |
| JNLE | Jump if not less than or equal (not <=) | Sign = Ovrflw or Zero=0 | JG | JLE |
| JGE | Jump if greater than or equal (>=) | Sign = Ovrflw | JNL | JGE |
| JNL | Jump if not less than (not <) | Sign = Ovrflw | JGE | JL |
| JL | Jump if less than (<) | Sign Ovrflw | JNGE | JNL |
| JNGE | Jump if not greater or equal (not >=) | Sign Ovrflw | JL | JGE |
| JLE | Jump if less than or equal (<=) | Sign Ovrflw or Zero = 1 | JNG | JNLE |
| JNG | Jump if not greater than (not >) | Sign Ovrflw or Zero = 1 | JLE | JG |
| JE | Jump if equal (=) | Zero = 1 | JZ | JNE |
| JNE | Jump if not equal () | Zero = 0 | JNZ | JE |

مثال: اگر AX=85AEH و BX=53FFH باشد بعد از دستور CMP AX,BX کدامیک از پرشهای شرطی زیر انجام می شود؟

JA L
JG L

۴-۸- حل چند مثال:

۱- برنامه ای بنویسید که تعداد ارقام عدد n را محاسبه نموده و در X قرار دهد:

seg1 segment

```

n dw 65535
x dw?
seg1 ends
seg2 segment
    Assume CS:seg2, DS:seg1
    L: MOV ax,seg1
        Mov ds,ax
        MOV CX,0
        MOV BX,10
        mov ax,n
    Q: Mov dx,0
        DIV BX
        inc CX
        CMP ax,0
        JNE Q
        Mov x,cx
        mov ah,4ch
        int 21H
seg2 ends
end L

```

مثال: جمع اعداد ۱ تا n

```

Seg1 Segment
    n dw 1000
    s dw ?
Seg1 ends
Seg2 Segment
    Assume CS : Seg2 , DS:Seg1
    L:MOV ax, Seg1
        MOV DS, ax
        MOV ax,0
        MOV bx , n
    b:Cmp bx,0
        Je L1
        add ax,bx
        dec bx(کاهش یک واحد)
        jmp b
    L1: mov s,ax
    MOV AH, 4 CH
    Int 21H
Seg2 ends
End L

```

فصل پنجم: مفهوم آدرسها در زبان اسمبلی

۵-۱- آدرسهای نسبی، مطلق و واقعی

برای مشخص کردن آدرسها در داخل دستورات برنامه از علامت [] استفاده می شود.

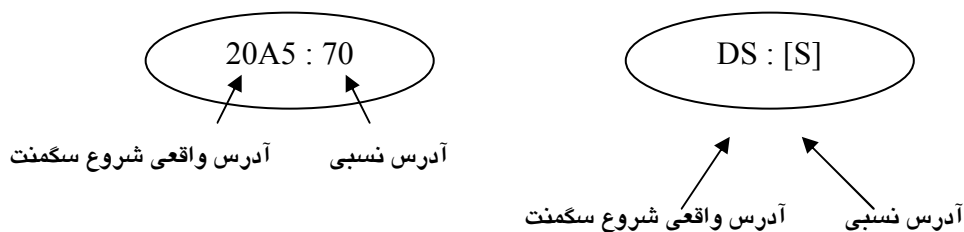
قطعات برنامه به همان ترتیبی که نوشته شده اند، توسط سیستم عامل در ram بارگذاری می شوند.

• آدرس مطلق، فیزیکی یا واقعی: یعنی آدرس دقیق یک داده یا دستور در RAM که در محدوده ۰ تا n-1 می باشد. (n ظرفیت

RAM). این آدرس تنها در زمان اجرا مشخص می شود.

گذرگاه آدرس در 8086 ۲۰ بیتی است ولی ثابتها ۱۶ بیتی هستند. راه حل این است که آدرس هر قطعه ضریب ۱۶ باشد یعنی ۴ بیت سمت راست آدرس شروع قطعات صفر باشد و ۱۶ بیت باقیمانده در ثابتهای قطعه (cs,ds,ss,es) ذخیره شود. مثلاً اگر cs=FFE2H آدرس واقعی شروع قطعه کد، FFE20H است.

- آدرس نسبی ، offset یا آدرس مؤثر (effective address): آدرس یک داده یا دستور نسبت به ابتدای قطعه مربوط مثلاً اولین متغیر تعریف شده در قطعه داده و اولین دستور تعریف شده در قطعه کد، دارای آدرس نسبی 0 است.
- آدرس منطقی: ترکیبی از آدرس فیزیکی شروع هر قطعه و آدرسی نسبی یک داده یا دستور است.



نحوه محاسبه آدرس واقعی از منطقی: یک صفر به سمت راست محتوای ثابت قطعه اضافه و با آدرس نسبی جمع می کنیم.

$$\begin{array}{r} 20A50 \\ + \quad 70 \\ \hline 20AC0 \end{array}$$

مثال: اگر CS=FA25H و DS=2EF6H و آدرس نسبی دستور L و متغیر x هر دو برابر 0AH باشد، آدرس واقعی این دو را محاسبه نمایید.

آدرس واقعی دستور FA250H+0AH=FA25AFH

آدرس واقعی متغیر 2EF60H+0AH=2EF6AH

سوال: آیا می توان از آدرس واقعی یک داده، آدرس منطقی آن را بدست آورد؟

مثال: فرض کنید برنامه زیر در آدرس 0A2E50H بار شده است (حتماً باید ضریب ۱۶ باشد). آدرس نسبی و واقعی کلیه متغیرها و دستورات را محاسبه نمایید.

```

sts segment stack
    dw 50 dup(?)
sts ends

seg1 segment
    max equ 100
    x dw 5 dup(?)
    y dd 200
    z db 120,13,15,45,56
    k dt 'alireza'
    
```


seg1 ends

seg2 segment

assume cs:seg2,ds:seg1,ss:s

L1:mov ax,max

L 2:add ax,ax

L 3:mov ah,4CH

L 4:int 21H

seg2 ends

end l1

| توضیح | آدرس واقعی | آدرس نسبی | تعداد بایتهای لازم | دستور یا متغیر یا سگمنت |
|-------------------|------------|-----------|--------------------|-------------------------|
| باید ضریب ۱۶ باشد | 0A2E50H | - | - | sts |
| | 0A2E50H | 0 | 64H | dw 50 dup(?) |
| باید ضریب ۱۶ باشد | 0A2EC0H | - | - | Seg1 |
| | - | - | - | max |
| | 0A2EC0H | 0 | 0AH | x |
| | 0A2ECAH | 0AH | 4 | Y |
| | 0A2ECEH | 0EH | 5 | Z |
| | 0A2ED3H | 13H | 0AH | K |
| باید ضریب ۱۶ باشد | 0A2EE0H | - | - | Seg2 |
| | 0A2EE0H | 0 | 3 | L1 |
| | 0A2EE3H | 3 | 2 | L2 |
| | 0A2EE5H | 5 | 2 | L3 |
| | 0A2EE7H | 7 | 2 | L4 |

ترجمه برنامه فوق به زبان ماشین بصورت زیر خواهد بود:

۵-۲- روشهای آدرس دهی در زبان اسمبلی: منظور، روشهای مشخص کردن عملوندهای داخل دستورات می باشد.

۱- روش آدرس دهی فوری (immediate Addressing Mode)

۲- روش آدرس دهی ثابتی

۳- روش آدرس دهی مستقیم

۴- روش آدرس دهی غیرمستقیم ثابتی

۵- روش آدرس دهی نسبی پایه

۶- روش آدرس دهی نسبی اندیس دار

۷- روش آدرس دهی نسبی اندیس دار پایه

۱- روش آدرس دهی فوری: مقدار عملوند در دستور مربوطه نوشته می شود: `mov ax,25`

۲- روش آدرس دهی ثباتی: منظور عملوند داخل ثبات قرار دارد و نام ثبات را می نویسیم. `Mov ax,25`

۳- روش آدرس دهی مستقیم: داده مورد نظر در حافظه است. از آدرس آن در داخل علامت [] استفاده می کنیم. `MOV DS:[12], AL`
یعنی خانه شماره 12 حافظه را در نظر بگیرد و AL را در آن محل کپی کند

۴- روش آدرس دهی غیرمستقیم ثباتی: در این روش عملوند در حافظه است و آدرس آن در یک ثبات قرار دارد. تنها از ثبات های BX ، SI ، DI و BP می توان به عنوان آدرس استفاده کرد.

بطور پیش فرض وقتی از ثبات BP استفاده می کنیم، آدرس نسبت به قطعه پشته stack seg و وقتی از BX و SI و DI استفاده می کنیم، آدرس نسبت به قطعه داده محاسبه می شود.

البته می توان با ذکر نام ثبات قطعه مربوطه، این پیش فرضها را تغییر داد.

`MOV AL, [BX]
ADD [BP], AH
MOV [SP], DL
MOV SS:[SI], CH`

`MOV [SI], CL
MOV AL, [DI]
MOV AL, CS:[BX]`

۵- روش آدرس دهی نسبی پایه: در این روش آدرس پایه در یکی از ثبات های BX یا BP قرار دارد و عملوند مورد نظر ما نسبت به این

آدرس پایه آدرس دهی می شود. در حافظه است و آدرس آن در یکی از ثبات های BX ، یا BP و یا چند بایت مجاور آنها قرار دارد.

مثال: `MOV [BX] +5, AL`

آدرس متغیر دقیقاً در داخل ثبات نیست (مثل آرایه) داده داخل BX را نمی خواهیم چند تا جلوتر از آن را می خواهیم. خود ثابت BX را در

قرار بده. `MOV BX, 1` `MOV [BP] +10, BL`

`MOV [BX]-2, AH` `MOV[BX]+2, 25`

`MOV 10[BX], 2S`

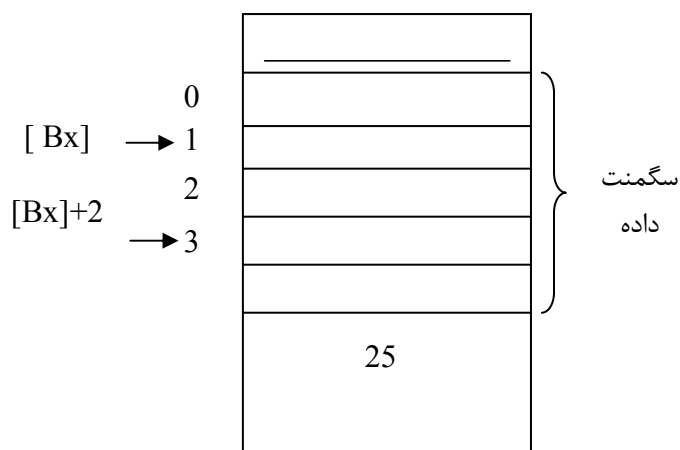
معادل

`MOV [BX]+10,25
MOV BX,1`

`MOV [BX], 1`

> خود ثبات Bx را در یک قرار بده <
> در محلی که Bx به آن اشاره می کند 1 قرار بده <

RAM



نکته : ثبات BX آدرس ها را نسبت به قطعه داده و ثبات BP داده ها را نسبت به قطعه پشته در نظر می گیرد. می توان هنگام آدرس دهی نام قطعه را هم مشخص کرد.

MOV SS: [BX] +5 , 0

MOV DS: [BX] +5,0 معادل MOV [BX]+5, 0

۶- روش آدرس دهی نسبی اندیس دار:

مانند روش پنجم است ولی از ثبات های SI و DI استفاده می شود. این دو ثبات آدرس ها را نسبت به قطعه داده در نظر می گیرند.

مثال : تفاوت دو دستور زیر در این است که :

MOV [Si]+10 , AL

نسبت به data segment

MOV SS: [Si]+10, AL

نسبت به stack segment

MOV 5 [Di], 0

۷- روش آدرس دهی نسبی اندیس دار پایه :

ترکیب روش های 5 و 6 (برای آرایه های دو بعدی مناسب است) .

آدرس نسبت به stack sag

از آدرس BP به اندازه Si جلو رفته و از آنجا 5 تا جلو می رویم.

MOV [Bp][Si]+5, 0

معادل

MOV [Bp+Si+5], 0

هر دو عبارت فوق معنای یکسانی را دارند.

MOV [BX] [Si+10] , DL

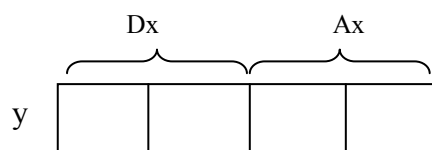
MOV 10[BX][SI] , DL

در این روش هم از ثبات های پایه و هم از ثبات های اندیس می توان استفاده کرد.

ثبات های پایه : (BX , BP) و ثبات های اندیس (SI , DI)

مثال : برنامه ای بنویسید که فاکتوریل عدد X را محاسبه کند و در Y بریزد.

X عدد یک بایتی است، ولی Y می تواند 4 بایتی باشد.



Seg1 Segment

X DB 7

Y DD ?

Seg1 ends

```

Seg2 Segment
Assume CS : Seg2 , DS:Seg1
L:MOV Ax, Seg1
MOV DS, Ax
MOV Ax,1
MOV Bx , 1    شمارنده
L2:Cmp BL,x
Jg L1
mul Bx
inc Bx
jmp L2
L1: MOV Si, offset y
MOV word ptr [Si], Ax
MOV word ptr [Si+2], Dx
MOV AH, 4CH
Int 21H
Seg2 ends
End L

```

شبه دستور offset آدرس نسبی یک داده یا دستور را مشخص می کند.
ترجمه برنامه فوق به زبان ماشین بصورت زیر خواهد بود:

The screenshot shows a DOS command prompt window with the following assembly code and register values:

```

C:\WINDOWS.0\system32\cmd.exe
File Edit View Run Breakpoints Data Options Window Help
cs:0000 B88A5B mov ax,5B8A ax 5B8A c=0
cs:0003 8ED8 mov ds,ax bx 0000 z=0
cs:0005 B80100 mov ax,0001 cx 0000 s=0
cs:0008 BB0100 mov bx,0001 dx 0000 o=0
cs:000B 3A1E0000 cmp bl,[0000] si 0000 p=0
cs:000F 7F05 jg 0016 di 0000 a=0
cs:0011 F7E3 jg 0016 bx 0000 i=1
cs:0013 43 inc bx sp 0000 d=0
cs:0014 EBF5 jmp 000B ds 5B8A
cs:0016 BE0100 mov si,0001 es 5B7A
cs:0019 8904 mov [si],ax ss 5B8A
cs:001B 895402 mov [si+02],dx cs 5B8B
cs:001E B44C mov ah,4C ip 0005
cs:0020 CD21 int 21
cs:0022 3EA23EA2 mov ds:[A23E],al
ds:0000 07 00 00 00 00 00 00 00
ds:0008 00 00 00 00 00 00 00 00
ds:0010 B8 8A 5B 8E D8 B8 01 00
ds:0018 BB 01 00 3A 1E 00 00 7F
ds:0020 05 F7 E3 43 EB F5 BE 01
ss:0002 0000
ss:0000 0007
ss:FFFE 3302
ss:FFFC 5B8B
ss:FFFA 0005
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```

نکته: همواره بایت کم ارزش در آدرس کم ارزش و بایت با ارزش در آدرس با ارزش ذخیره می شود. مثلا اگر جواب نهایی 2F51E502H باشد بصورت زیر در حافظه قرار می گیرد.

Dx : 2F51

Ax : E502

↓ ارزش کمتر ↓ ارزش بیشتر

y

| |
|----|
| . |
| . |
| . |
| 02 |
| E5 |
| 51 |
| 2F |
| |
| |

← Si

← Si + 2

مثال: برنامه‌ای بنویسید که مقسوم‌علیه‌های عدد n را در آرایه x و تعداد آنها را در k قرار دهد:

```

Seg1 segment
n DW 24
k DW?
x dw 100 dup(0)
Seg1 ends
seg2 segment
assume cs:seg2,ds:seg1
L:mov ax,seg1
mov DS,ax
mov bx,offset x
mov cx,1
L2:cmp cx,n
JA L3
mov ax,n
mov dx,0
div cx
cmp dx,0
JNE L1
mov word ptr[bx],cx
add bx,2
inc K
L1: inc cx
JMP L2
L3: mov ah,4ch
int 21H
seg2 ends
end L

```

مثال : از طریق دسترسی مستقیم به بافر کارت گرافیکی برنامه ای بنویسید که کلیه کاراکتر های اسکی را با یک رنگ دلخواه بر روی صفحه نمایش دهد.

آدرس فیزیکی B8000H آدرس بافر کارت گرافیکی است که در حالت متنی تمامی کاراکترهایی که در صفحه خروجی می‌بینیم، در این آدرس قرار دارند. برای هر کاراکتر صفحه خروجی، ۲ بایت ذخیره می‌شود که اولی کد اسکی کاراکتر و دومی کد رنگ آن می‌باشد. کد رنگ نیز دو قسمتی است . ۴ بیت کم‌ارزش رنگ متن و ۴ بیت با ارزش رنگ زمینه را نشان می‌دهد.

```

Seg1 Segment
Assume CS: Seg1
L:MOV AL, 0
MOV BX , 0B800H
MOV DS, BX
MOV SI , 0
L1: MOV [SI], AL
MOV byte ptr[SI]+1, 34H
Inc AL
add SI,2
Cmp AL, 255
JB L1          چرا JBE نوشته نشده تا خود عدد ۲۵۵ را نیز شامل گردد؟
MOV AH , 4CH

```

Int 21H
Seg1 ends
End L

فصل ششم: وقفه ها در زبان اسمبلی

مفهوم وقفه (interrupt):

– وقفه های سخت افزاری: پردازنده جهت ارتباط با سخت افزارهای دیگر دو راه پیش رو دارد: interrupt و pooling

در روش پولینگ پردازنده بطور متناوب به آن سخت افزار سرکشی می کند: کارایی پایین

در روش وقفه هر سخت افزاری جهت ارتباط با cpu سیگنالی را به یکی از پایه های cpu می فرستد و cpu کار جاری خود را موقتاً رها کرده و

متوجه آن سخت افزار می شود. پردازنده پس از تشخیص نوع وقفه زیرروال مربوط به آن را اجرا می کند و در نهایت به ادامه کار قبلی خود

برمی گردد.

در زندگی روزمره صدای زنگ تلفن یا خانه نوعی وقفه سخت افزاری محسوب می شود.

– وقفه های نرم افزاری: این وقفه ها از داخل برنامه ها و با دستور int فراخوانی می شوند. زیرروالهای مربوط به برخی از وقفه ها جزئی از نرم افزار

BIOS و برخی جزئی از سیستم عامل هستند.

چند نمونه از وقفه های نرم افزاری:

هر وقفه شامل چندین تابع می باشد که شماره تابع در ثبات AH قرار می گیرد.

شکل کلی فراخوانی وقفه:

شماره وقفه int

خواندن یک کلید از کاربر:

این دستور منتظر فشردن یک کلید می ماند، وقتی کلید فشرده شود کد اسکی آن در AL و کد پویش آن در AH ریخته می شود.

کد اسکی کلید خوانده شده در AL و کد پویش آن در AH قرار می گیرد.

شماره تابع MOV AH, 0

شماره وقفه Int 16H

کد پویش (scan code): کدگذاری کلیدهای روی صفحه کلید می باشد. کاراکترهایی مانند ۱ و ! که با یک کلید تایپ می شوند کدهای پویش

متفاوتی دارند. کاراکترهایی مانند ù که بر روی صفحه کلید وجود ندارند کد پویش ندارند در حالیکه کد اسکی دارند. در انتهای جزوه در

قسمت پیوسته جدول کامل کدهای اسکی و کدهای پویش آورده شده است.

انتقال مکان نما به یک محل خاص :

شماره صفحه MOV BH ,

شماره سطر MOV DH ,

شماره ستون MOV DL ,

شماره تابع MOV AH , 2

شماره وقفه Int 10H

۱- برنامه ما می تواند تا سقف 8 صفحه خروجی داشته باشد که ابتدا اطلاعات را در داخل آنها می نویسیم، سپس آنها را فعال می کنیم.

۲- سطر را داخل DH می ریزیم.

۳- محل مکان نما را به ما می دهد.

۴- مکان نما را به سطر و ستون مورد نظر می برد.

۵- وقفه 10H ، یک مجموعه وقفه ها است، 2 یکی از آنها برای انتقال مکان نما است.

نحوه تشخیص محل فعلی مکان نما:

MOV BH,0 شماره صفحه
MOV AH,3
Int 10H

پس از انجام این قطعه کد ، شماره سطر و ستون به ترتیب در دو ثبات DH و DL قرار می گیرد.

نوشتن یک کاراکتر در محل فعلی مکان نما بدون جابه جایی مکان نما

MOV AL , کد اسکی کاراکتر
MOV BH,0 شماره صفحه
MOV BL, شماره رنگ
MOV CX , تعداد نوشتن ها
MOV AH , 0AH
int 10H

نوشتن یک کاراکتر در محل فعلی مکان نما و جلو بردن مکان نما

MOV AL , کد اسکی کاراکتر
MOV BH,0 شماره صفحه
MOV BL, شماره رنگ
MOV AH , 0EH
int 10H

مثال: برنامه ای بنویسید که کل صفحه خروجی را با * پر نماید:

| روش اول: دسترسی مستقیم به بافر صفحه خروجی | روش دوم: استفاده از وقفه 10H تابع 0AH | روش سوم: استفاده از وقفه 10H تابع 0EH |
|---|---|--|
| <pre> myseg segment Assume CS: Seg1 b:MOV AL, '*' MOV BX , 0B800H MOV DS, BX MOV SI, 0 L1: MOV [SI], AL add SI,2 Cmp SI,4000 JB L1 MOV AH , 4CH Int 21H myseg ends End b </pre> | <pre> myseg segment Assume CS: Seg1 b:MOV AL, '*' MOV BH,0 MOV BL,25H MOV CX,2000 MOV AH,0AH int 10H MOV AH , 4CH Int 21H myseg ends </pre> | <pre> myseg segment Assume CS: Seg1 b:MOV AL, '*' MOV BH,0 MOV BL,25H MOV AH,0EH MOV CX,2000 L:int 10H DEC CX JNZ L MOV AH , 4CH Int 21H myseg ends End b </pre> |

مثال : برنامه ای بنویسید که کارکتر * را در وسط صفحه بنویسید و با کلیدهای ← آن را جابجا کند. کد پویش این کلیدها ۷۵ و ۷۲ و ۷۷ و ۸۰ می باشد. صفحه خروجی در حالت متنی استاندارد دارای ۸۰ ستون (0-79) و ۲۵ سطر (0-24) می باشد.

الگوریتم برنامه :

↑: 48H → 72

↓: 50H → 80

←: 4BH → 75

→: 4DH → 77

(ESC)Scape : 1

-۰ شروع

-۱ DH=12, DL=40

-۲ START: مکان نما را به مختصات (DH,DL) منتقل کن

-۳ کاراکتر * را بنویس (بدون جابجایی مکان نما)

-۴ یک کلید از کاربر را بخوان و scan code آن را در AH قرار

-۵ اگر AH=1 برو به exit

-۶ اگر AH<>75 برو به R

-۷ کاراکتر " " را بنویس (بدون جابجایی مکان نما) (پاک کردن * قبلی)

-۸ DL=DL-1

-۹ اگر DL>=0 برو به START

-۱۰ DL=79 (اگر * به ستون اول رسیده بود دوباره به ستون آخر برود) و برو به START

-۱۱ R: اگر AH<>77 برو به U

-۱۲ کاراکتر " " را بنویس (بدون جابجایی مکان نما) (پاک کردن * قبلی)

-۱۳ DL=DL+1

-۱۴ اگر DL<=79 برو به START

-۱۵ DL=0 (اگر * به ستون آخر رسیده بود دوباره به ستون اول برود) و برو به START

-۱۶ U: اگر AH<>72 برو به D

-۱۷ کاراکتر " " را بنویس (بدون جابجایی مکان نما) (پاک کردن * قبلی)

-۱۸ DH=DH-1

-۱۹ اگر DH>=0 برو به START

-۲۰ DH=24 (اگر * به سطر اول رسیده بود دوباره به سطر آخر برود) و برو به START

-۲۱ D: اگر AH<>80 برو به start

-۲۲ کاراکتر " " را بنویس (بدون جابجایی مکان نما) (پاک کردن * قبلی)

-۲۳ DH=DH+1

-۲۴ اگر DH<=24 برو به START

-۲۵ DH=0 (اگر * به سطر آخر رسیده بود دوباره به سطر اول برود) و برو به START

-۲۶ exit: پایان


```

seg1 segment
assume cs:seg1
l:  mov bh,0
    mov dh,12
    mov dl,40
start: mov ah,2
      int 10h
      mov al,'*'
      mov bh,0
      mov bl,0FH
      mov cx,1
      mov ah,0ah
      int 10h
      mov ah,0
      int 16h
      cmp ah,1
      je exit
      cmp ah,4bh
      jne R
      mov al,''
      mov bh,0
      mov cx,1
      mov ah,0ah
      int 10h
      dec dl
      cmp dl,0
      jge start

      mov dl,79
      jmp start
R:   cmp ah,4dh
      jne U
      mov al,''
      mov bh,0
      mov cx,1
      mov ah,0ah
      int 10h
      inc dl
      cmp dl,79
      jle start
      mov dl,0
      jmp start
U:   cmp ah,48h
      jne D
      mov al,''
      mov bh,0
      mov cx,1
      mov ah,0ah
      int 10h

      dec dh
      cmp dh,0
      jge start
      mov dh,24
      jmp start
D:   cmp ah,50h
      jne start
      mov al,''
      mov bh,0
      mov cx,1
      mov ah,0ah
      int 10h
      inc dh
      cmp dh,24
      jle start
      mov dh,0
      jmp start
exit: mov ah,4ch
      int 21h
seg1 ends
end l

```

فصل هفتم : مفهوم پشته، زیربرنامه ها و ماکروها در زبان اسمبلی

نحوه تعریف و استفاده از پشته (stack) در برنامه ها :

- در یک برنامه اسمبلی تعدادی سگمنت تعریف می کردیم که یکی از آنها Stack segment است.
- با تعریف یک قطعه پشته می توان از حافظه مربوط به آن برای ذخیره و بازیابی اطلاعات استفاده کرد و با دستور POP و Push اطلاعات در پشته قرار داده می شوند و برداشته می شوند (هدف از تعریف پشته ذخیره و دریافت اطلاعات از آن است).
- واحد خواندن و نوشتن از پشته word می باشد، یعنی هر بار باید دو بایت را push یا pop نماییم.
- ثبات SP (Stack pointer) همواره به بالاترین کلمه وارد شده در پشته اشاره می کند و اول کار مقدار آن 0 است.
- با هر دستور push، ابتدا دو واحد از Sp کم می شود و کلمه جدید در محلی که Sp اشاره می کند قرار می گیرد.
- با هر دستور pop، ابتدا کلمه محل اشاره Sp در ثبات مربوطه قرار می گیرد. سپس دو واحد به Sp اضافه می شود.

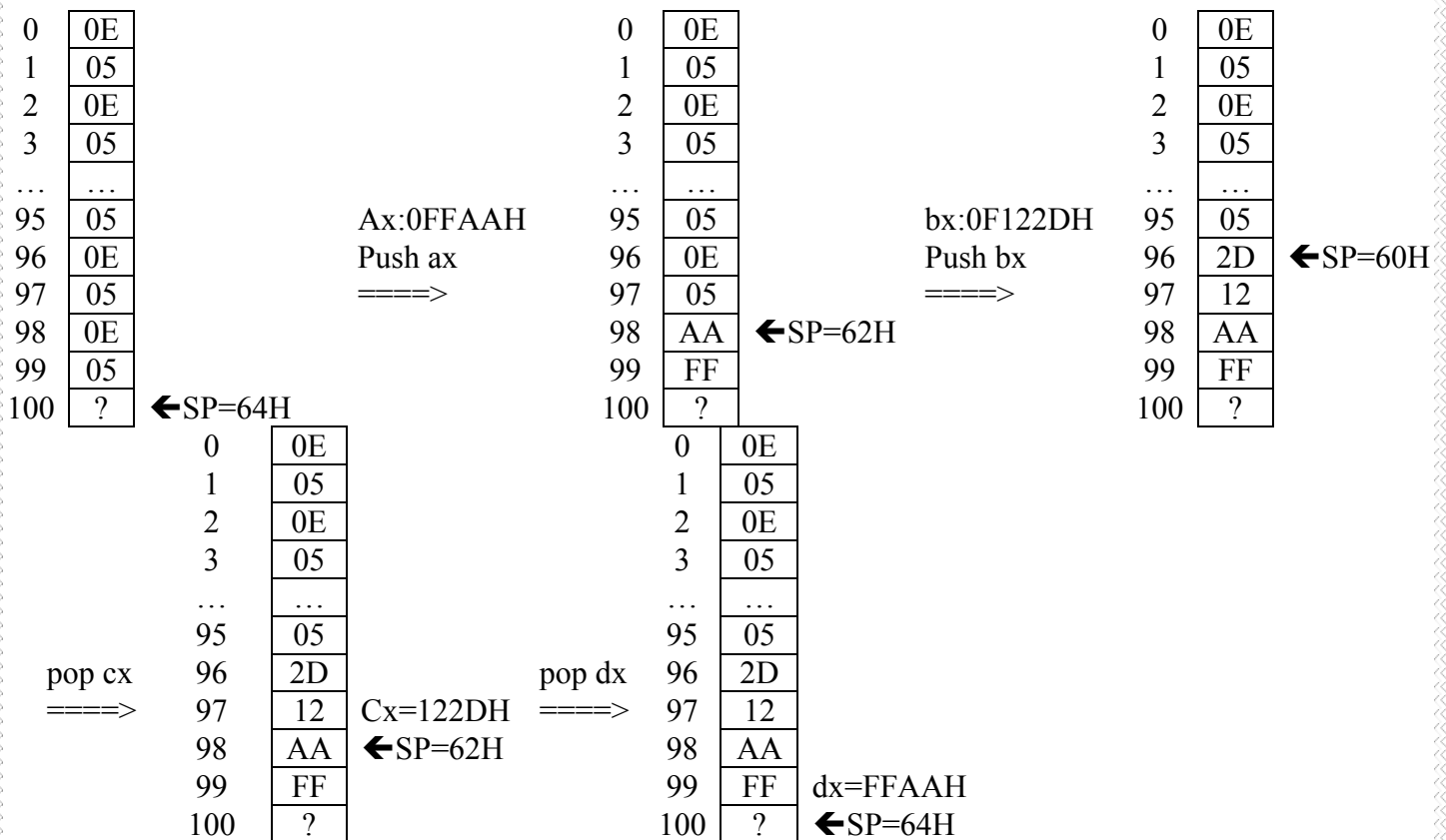
نکته: واحد push و pop

$$pop\ Cx \leftrightarrow \begin{cases} MOV\ Cx, word\ ptr[SP] \\ add\ Sp, 2 \end{cases} \qquad Push\ Ax \leftrightarrow \begin{cases} SUB\ SP, 2 \\ MOV\ word\ ptr \end{cases}$$

مثال: تعریف پشته و دستورات زیر را در نظر بگیرید.

```
seg2 segment stack
dw 50 dup(050EH)
seg2 ends
...
Mov ax,0FFAAH
Mov bx,122DH
Push ax
Push bx
Pop cx
Pop dx
```

وضعیت پشته در مراحل مختلف دستورات فوق:



اگر پشته به اندازه n بایت داشته باشیم، Sp=0 نشان دهنده پر بودن و sp=n نشان دهنده خالی بودن پشته است.

نحوه تعریف و استفاده از زیر برنامه ها و ماکروها در اسبلی 8086 : برنامه نویسی ماژولار: در یک برنامه می توان عملیات مختلفی را که باید انجام گیرد در بخش های مختلفی تقسیم بندی کرد که هر بخش کار خاصی را انجام دهد. به هر کدام از این بخشها یک ماژول می گوییم. مزایا: خوانایی بالا، جلوگیری از تکرار دستورات، کاهش حجم برنامه، استفاده مجدد از ماژولها در برنامه های مختلف و...

نحوه تعریف زیر برنامه :

نام Proc
دستورات
Ret
نام End p

نحوه فراخوانی زیر برنامه

نام زیر برنامه Call

مثال: با استفاده از زیر برنامه ها، برنامه ای بنویسید که صفحه خروجی را به چاپگر بفرستد.

```
sseg segment stack
    dw 10 dup(?)
sseg ends
Seg1 Segment
    assume cs:seg1,ss:sseg
    printscr proc
        int 5
        ret
    printscr endp
```

```
L: call printscr
MOV AH , 4CH
Int 21H
Seg1 ends
End L
```

دستورات ret , call , معادل دستورات زیر هستند:

```
call p ==> push ip
mov ip,offset(p)
ret ==> pop ip
```

مثال: برنامه‌ای بنویسید که یک عدد را بخواند و مقسوم علیه‌های آن را چاپ کند:

getnumber زیربرنامه ای است که یک عدد صحیح علامتدار را می‌خواند و در AX قرار می‌دهد.

shownumber زیربرنامه ای است که یک عدد صحیح علامتدار موجود در AX را بر روی صفحه خروجی نمایش می‌دهد.

| | | | |
|-----------------|-------------------|--------------------|---------------------|
| seg1 segment | push bx | NEG cx | change_cursor endp |
| DW 50 dup(?) | call getch | mov ax,cx | b:call get_number |
| X DW 0 | JMP start | JMP r2 | call change_cursor |
| seg1 ends | L3:mov bx,0 | r1 : mov cx,ax | mov di,0 |
| | push bx | r2:mov ax,cx | mov si,ax |
| seg2 segment | JMP L2 | mov dx,0 | mov bp,offset x |
| assume cs:seg2, | start: call putch | mov bx,10 | mov cx,1 |
| SS:seg1 | L2:sub al,30H | div bx | p2:cmp cx,si |
| getch proc | mov bl,al | INC si | JA p3 |
| mov ah,0 | mov bh,0 | mov cx,ax | mov ax,si |
| int 16H | L:call getch | add dx,30H | mov dx,0 |
| ret | call putch | push dx | div cx |
| getch endp | cmp al,13 | cmp cx,0 | cmp dx,0 |
| | JE Q | JE exit1 | JNE p1 |
| putch proc | sub al,30h | JMP r2 | mov word ptr[bp],cx |
| push BX | mov cl,al | exit1: pop ax | add bp,2 |
| mov bh,0 | mov ch,0 | call putch | inc di |
| mov bl,0FH | mov ax,10 | dec si | p1: inc cx |
| mov ah,0EH | mul bx | JNZ exit1 | JMP p2 |
| int 10H | add ax,cx | ret | p3:mov bp,offset x |
| pop BX | mov bx,ax | show_number endp | p4:cmp di,0 |
| ret | JMP L | | JNA p5 |
| putch endp | Q:mov ax,bx | change_cursor proc | mov ax,word ptr[bp] |
| | pop bx | push ax | add bp,2 |
| get_number proc | cmp bx,0FFFFH | mov bh,0 | dec di |
| call getch | JNE exit | mov dh,2 | call show_number |
| call putch | NEG ax | mov dl,0 | mov ax,20H |
| cmp AL'+', | exit: ret | mov ah,2 | call putch |
| JNE L1 | get_number endp | int 10H | JMP p4 |
| mov bx,0 | show_number proc | pop ax | p5:mov ah,4ch |
| push bx | mov si,0 | ret | int 21h |
| call getch | cmp ax,0 | | seg2 ends |
| JMP start | JG r1 | | end b |
| L1:cmp AL'-', | mov cx,ax | | |
| JNE L3 | mov al,2DH | | |
| mov bx,0FFFFH | call putch | | |

ماکروها (macro): با استفاده از ماکرو، می توان تعدادی دستور را با یک نام فراخوانی کرد (مشابه زیر برنامه)

پارامترها macro نام

دستورات

Endm

عملا موقع ترجمه، ماکرویی وجود ندارد و کلیه دستورات ماکرو جایگزین نام آن داخل برنامه می شود. بنابراین برای تعریف ماکرو، نیازی به تعریف پشته نیست در حالیکه در مورد زیربرنامه، وجود پشته اجباری است.

محل تعریف ماکرو خارج تمامی سگمنت ها است ولی زیربرنامه باید داخل سگمنت کد تعریف گردد

سرعت اجرای ماکرو بیشتر از زیربرنامه است ولی حجم برنامه ترجمه شده نیز بیشتر است.

مثال : می خواهیم ماکرو و زیربرنامه ای بنویسیم که یک کاراکتر x را بدون جابجایی مکان نما چاپ کند(در مورد ماکرو x به عنوان پارامتر در نظر گرفته می شود و در مورد زیربرنامه فرض می کنیم کد اسکی x در al قرار دارد)

```
writtech macro x
    mov al,x
    mov bh,0
    mov cx,1
    mov ah,0ah
    int 10h
```

Endm

```
writtech proc
    mov bh,0
    mov cx,1
    mov ah,0ah
    int 10h
    ret
```

Writtech Endp

مثال : می خواهیم ماکرو و زیربرنامه ای بنویسیم که یک کلید را از کاربر دریافت و کد اسکی آن را در al و کد پویش آن را در ah قرار دهد.

```
getch macro
    mov ah,0
    int 16h
```

Endm

```
getch proc
    mov ah,0
    int 16h
    ret
```

getch Endp

مثال : می خواهیم ماکرو و زیربرنامه ای بنویسیم که اجرای برنامه را خاتمه دهد.

```
theend macro
    mov ah,4CH
    int 21h
```

Endm

```
theend proc
    mov ah,4CH
    int 21h
    ret
```

theend endp

مثال: برنامه جابجایی ستاره را یکبار با استفاده از ماکرو و یکبار با زیربرنامه بازنویسی کنید.
با استفاده از ماکرو:

| | | |
|--|--|--|
| <pre> writech macro x mov al,x mov bh,0 mov cx,1 mov ah,0ah int 10h Endm getch macro mov ah,0 int 16h Endm theend macro mov ah,4CH int 21h Endm seg1 segment assume cs:seg1 l: mov bh,0 mov dh,12 mov dl,40 start:mov ah,2 </pre> | <pre> int 10h writech '*' getch cmp ah,1 je exit cmp ah,4bh jne R writech ' ' dec dl cmp dl,-1 jne start mov dl,79 jmp start R: cmp ah,4dh jne U writech ' ' inc dl cmp dl,80 jne start mov dl,0 </pre> | <pre> jmp start U: cmp ah,48h jne D writech ' ' dec dh cmp dh,-1 jne start mov dh,24 jmp start D: cmp ah,50h jne start writech ' ' inc dh cmp dh,25 jne start mov dh,0 jmp start exit: theend seg1 ends end l </pre> |
|--|--|--|

با استفاده از زیربرنامه:

| | | |
|---|---|---|
| <pre> writech proc mov bh,0 mov cx,1 mov ah,0ah int 10h ret Writech Endp getch proc mov ah,0 int 16h ret getch Endp theend proc mov ah,4CH int 21h ret theend endp seg1 segment assume cs:seg1 l: mov bh,0 mov dh,12 mov dl,40 start:mov ah,2 </pre> | <pre> int 10h Mov al,'*' Call writech Call getch cmp ah,1 je exit cmp ah,4bh jne R writech ' ' dec dl cmp dl,-1 jne start mov dl,79 jmp start R: cmp ah,4dh jne U writech ' ' inc dl cmp dl,80 jne start mov dl,0 </pre> | <pre> jmp start U: cmp ah,48h jne D writech ' ' dec dh cmp dh,-1 jne start mov dh,24 jmp start D: cmp ah,50h jne start writech ' ' inc dh cmp dh,25 jne start mov dh,0 jmp start exit: call theend seg1 ends end l </pre> |
|---|---|---|

نکته مهم: اگر در داخل ماکرو یا زیربرنامه هایی که ثباتها را تغییر می دهند بخواهیم مقدار این ثباتها پس از بازگشت به برنامه اصلی به مقدار اولیه برگردند، باید در ابتدای زیربرنامه یا ماکرو آن ثباتها را در پشته push و در پایان کار pop نماییم. مگر اینکه آن ثبات در حکم پارامتر ورودی یا خروجی زیربرنامه باشد.

مثال: برنامه جابجایی ستاره را با در نظر گرفتن نکته فوق بازنویسی کنید.

با استفاده از ماکرو:

| | | | |
|------------------|----------------|-----------|--------------|
| Pushall macro | popall | R: | cmp ah,4dh |
| Push ax | endm | | jne U |
| Push bx | theend macro | | writtech ' ' |
| Push cx | pushall | | inc dl |
| Push dx | mov ah,4CH | | cmp dl,80 |
| Push si | int 21h | | jne start |
| Push di | popall | | mov dl,0 |
| endm | Endm | | jmp start |
| Popall macro | seg1 segment | U: | cmp ah,48h |
| Pop di | assume cs:seg1 | | jne D |
| Pop si | l: mov bh,0 | | writtech ' ' |
| Pop dx | mov dh,12 | | dec dh |
| Pop cx | mov dl,40 | | cmp dh,-1 |
| Pop bx | start:mov ah,2 | | jne start |
| Pop ax | int 10h | | mov dh,24 |
| endm | writtech ' *' | | jmp start |
| writtech macro x | getch | D: | cmp ah,50h |
| pushall | cmp ah,1 | | jne start |
| mov al,x | je exit | | writtech ' ' |
| mov bh,0 | cmp ah,4bh | | inc dh |
| mov cx,1 | jne R | | cmp dh,25 |
| mov ah,0ah | writtech ' ' | | jne start |
| int 10h | dec dl | | mov dh,0 |
| popall | cmp dl,-1 | | jmp start |
| Endm | jne start | exit: | theend |
| getch macro | mov dl,79 | seg1 ends | |
| pushall | jmp start | end l | |
| mov ah,0 | | | |
| int 16h | | | |

با استفاده از زیربرنامه:

| | | | |
|----------------|---------------|-----------|--------------|
| writtech proc | int 10h | | jmp start |
| mov bh,0 | Mov al,'*' | U: | cmp ah,48h |
| mov cx,1 | Call writtech | | jne D |
| mov ah,0ah | Call getch | | writtech ' ' |
| int 10h | cmp ah,1 | | dec dh |
| ret | je exit | | cmp dh,-1 |
| Writtech Endp | cmp ah,4bh | | jne start |
| getch proc | jne R | | mov dh,24 |
| mov ah,0 | writtech ' ' | | jmp start |
| int 16h | dec dl | D: | cmp ah,50h |
| endp | cmp dl,-1 | | jne start |
| getch endp | jne start | | writtech ' ' |
| theend proc | mov dl,79 | | inc dh |
| mov ah,4CH | jmp start | | cmp dh,25 |
| int 21h | cmp ah,4dh | | jne start |
| ret | jne U | | mov dh,0 |
| theend endp | writtech ' ' | | jmp start |
| seg1 segment | inc dl | exit: | call theend |
| assume cs:seg1 | cmp dl,80 | seg1 ends | |
| l: mov bh,0 | jne start | end l | |
| mov dh,12 | mov dl,0 | | |
| mov dl,40 | | | |
| start:mov ah,2 | | | |

شبه دستور local:

اگر در یک ماکرو دستوری دارای label باشد، و بیش از یکبار فراخوانی گردد، بعد از ترجمه برنامه یعنی جایگزینی دستورات ماکرو در محل فراخوانی ماکرو، label یاد شده دو بار تکرار خواهد شد که غیر مجاز است. در چنین مواردی label یاد شده را با شبهه دستور local مشخص می کنیم:

```
fact macro x,y
local L
mov ax,1
mov cx,x
L:mul cx
dec cx
jnz L
mov y,ax
endm
seg1 segment
fact 5,bx
fact 7,dx
seg1 ends
```

اگر سطر دوم را حذف نماییم برای سطر ۱۰ پیغام خطای (۱۰) L duplicate declaration of: صادر می شود.

فصل نهم: بررسی چند دستور و چند مثال

ضرب و تقسیم اعداد علامتدار

imul

idiv

مشابه دستورات mul و div هستند و می توانند بصورت یک بایتی یا دو بایتی استفاده شوند. اگر در بین دو عدد ضرب یا تقسیم شده عدد منفی وجود داشته باشد، نتیجه این دو دستور با mul و div متفاوت خواهد بود. مثالهای زیر تفاوتها را نشان می دهد.

| | | | |
|--|---|---|---|
| <pre>mov al,FFH mov bl,2 imul bl</pre> <p>نتیجه:</p> <p>ax=-1*2=-2=FFFEH</p> | <pre>mov al,FFH mov bl,2 mul bl</pre> <p>نتیجه:</p> <p>ax=255*2=510=01FEH</p> | <pre>mov ax,8000H mov dx,0 mov bx,2 div bx</pre> <p>نتیجه:</p> <p>ax= $2^{15}/2=2^{14}=4000H$ dx=0</p> | <pre>mov ax,8000H mov dx,0 mov bx,2 idiv bx</pre> <p>نتیجه:</p> <p>ax= -215/2=-214=C000H dx=0</p> |
|--|---|---|---|

دستورات پردازش بیتی:

and : دو داده را بیت به بیت and می کند و نتیجه را در dest می ریزد.

and dest,source
1 and 1=1 0 and x =0

or : دو داده را بیت به بیت or می کند و نتیجه را در dest می ریزد.

or dest,source
0 or 0=0 1 or x =1

xor : دو داده را بیت به بیت xor می کند و نتیجه را در dest می ریزد.

xor dest,source
0 xor 0=1 1 xor 1 =0 0 xor 1=1 1 xor 0=1

not : dest را مکمل ۱ می نماید.

mov ax,0FE0H

not ax

نتیجه: ax=F01FH

neg : dest را مکمل ۲ می نماید.

mov ax,0FE0H

neg ax

نتیجه: ax=F020H

test : مانند and عمل می نماید ولی نتیجه را در dest ذخیره نمی کند بلکه فقط بر ثباتهای پرچم تاثیر می گذارد.

test dest,source

دستورات شیفت و چرخش

dest,n دستور

dest: مکان حافظه یا ثبات

n: عدد ۱ یا ثبات cl

dest را به تعداد n بیت شیفت می دهد.

فرض کنید داریم:

ax:1010 1001 1101 0001 CF=0

دستور shl ax,1 ثبات ax را یک بیت به سمت چپ شیفت می دهد. از سمت راست 0 وارد می شود و سمت چپ ترین بیت وارد CF می گردد:

ax:0101 0011 1010 0010 CF=1

دستور shr ax,1 ثبات ax را یک بیت به سمت راست شیفت می دهد. از سمت چپ 0 وارد می شود و سمت راست ترین بیت وارد CF می گردد:

ax:0101 0100 1110 1000 CF=1

دستور sal ax,1 ثبات ax را یک بیت به سمت چپ شیفت می دهد. از سمت راست 0 وارد می شود و سمت چپ ترین بیت وارد CF می گردد:

این دستور دقیقا مانند shl عمل می کند.

ax:0101 0011 1010 0010 CF=1

دستور sar ax,1 ثبات ax را یک بیت به سمت راست شیفت می دهد. از سمت چپ همان بیت علامت عدد (در این مثال ۱) وارد می شود و

سمت راست ترین بیت وارد CF می گردد:

ax:1101 0100 1110 1000 CF=1

دستور rol ax,1 ثبات ax را یک بیت به سمت چپ چرخش می دهد. بدین معنی که تمامی بیتها به سمت چپ شیفت پیدا کرده و سمت

چپ ترین بیت هم دوباره از سمت راست وارد می شود و هم وارد CF می گردد.

ax:0101 0011 1010 0011 CF=1

دستور ror ax,1 ثبات ax را یک بیت به سمت راست چرخش می دهد. بدین معنی که تمامی بیتها به سمت راست شیفت پیدا کرده و سمت

راست ترین بیت هم دوباره از سمت چپ وارد می شود و هم وارد CF می گردد.

ax:1101 0100 1110 1000 CF=1

دستور rcl ax,1 مانند rol است ولی محتویات فعلی CF نیز در چرخه وارد می شود.

ax:0101 0011 1010 0010 CF=1

دستور rcr ax, 1 مانند ror است ولی محتویات فعلی CF نیز در چرخه وارد می شود.

ax:0101 0100 1110 1000 CF=1

دستورات تنظیم فلگها:

stc CF =1
clc CF =1
cmc CF = not CF

دستورات حلقه:

loop L: dec cx, if cx > 0 jmp L
loopz L, loope L: dec cx, if cx > 0 and ZF=1 jmp L
loopnz L, loopne L: dec cx, if cx > 0 and ZF=0 jmp L

پیوستها:

| زمان اجرای برخی از دستورات در پردازنده 8086 بر حسب پالس ساعت | | | | | |
|--|------------------|-------------------------|-------|-----|-----|
| دستور ↓ روشن آدرس دهی | mov (both forms) | add, sub, cmp, and, or, | not | jmp | jxx |
| reg, reg | 5 | 7 | | | |
| reg, xxxx | 6-7 | 8-9 | | | |
| reg, [bx] | 7-8 | 9-10 | | | |
| reg, [xxxx] | 8-10 | 10-12 | | | |
| reg, [xxxx+bx] | 10-12 | 12-14 | | | |
| [bx], reg | 7-8 | | | | |
| [xxxx], reg | 8-10 | | | | |
| [xxxx+bx], reg | 10-12 | | | | |
| reg | | | 6 | | |
| [bx] | | | 9-11 | | |
| [xxxx] | | | 10-13 | | |
| [xxxx+bx] | | | 12-15 | | |
| xxxx | | | | 6-7 | 6-8 |

اندازه گذرگاه داده در پردازنده های 80x86

| اندازه گذرگاه داده | پردازنده |
|--------------------|----------------------------|
| 8 | 8088 |
| 8 | 80188 |
| 16 | 8086 |
| 16 | 80186 |
| 16 | 80286 |
| 16 | 80386sx |
| 32 | 80386dx |
| 32 | 80486 |
| 64 | 80586 class/ Pentium (Pro) |

اندازه گذرگاه آدرس در پردازنده های 80x86

| پردازنده | اندازه گذرگاه آدرس | حداکثر حافظه قابل آدرس دهی | معادل |
|----------|--------------------|----------------------------|-------------|
| 8088 | 20 | 1,048,576 | یک مگا بایت |
| 8086 | 20 | 1,048,576 | یک مگا بایت |

| | | | |
|-------------|---------------|----|-----------------------|
| یک مگا بایت | 1,048,576 | 20 | 80188 |
| یک مگا بایت | 1,048,576 | 20 | 80186 |
| ۱۶ مگا بایت | 16,777,216 | 24 | 80286 |
| ۱۶ مگا بایت | 16,777,216 | 24 | 80386sx |
| ۴ گیگا بایت | 4,294,976,296 | 32 | 80386dx |
| ۴ گیگا بایت | 4,294,976,296 | 32 | 80486 |
| ۴ گیگا بایت | 4,294,976,296 | 32 | 80586 / Pentium (Pro) |

| زمان اجرای برخی از دستورات در پردازنده 80286 بر حسب پالس ساعت | | | | | |
|---|------------------|-------------------------|-----|-------|---------|
| دستور روشن آدرس دهی | mov (both forms) | add, sub, cmp, and, or, | not | jmp | jxx |
| reg, reg | 2 | 4 | | | |
| reg, xxxx | 1 | 3 | | | |
| reg, [bx] | 3-4 | 5-6 | | | |
| reg, [xxxx] | 3-4 | 5-6 | | | |
| reg, [xxxx+bx] | 4-5 | 6-7 | | | |
| [bx], reg | 3-4 | 5-6 | | | |
| [xxxx], reg | 3-4 | 5-6 | | | |
| [xxxx+bx], reg | 4-5 | 6-7 | | | |
| reg | | | 3 | | |
| [bx] | | | 5-7 | | |
| [xxxx] | | | 5-7 | | |
| [xxxx+bx] | | | 6-8 | | |
| xxxx | | | | 1+pdf | 2 2+pdf |

ساختار کلی کدگذاری دستورات اسمبلی:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|--|
| i | i | i | r | r | m | m | m | |
|---|---|---|---|---|---|---|---|--|

iii

rr

mmm

000 = special

001 = or

010 = and

011 = cmp

100 = sub

101 = add

110 = mov reg, mem/reg/const

111 = mov mem, reg

00 = AX

01 = BX

10 = CX

11 = DX

000 = AX

001 = BX

010 = CX

011 = DX

100 = [BX]

101 = [xxxx+BX]

110 = [xxxx]

111 = constant

This 16-bit field is present only if the instruction is a jump instruction or an operand is a memory addressing mode of the form [bx+xxxx], [xxxxx], or a constant.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|--|
| 0 | 0 | 0 | i | i | m | m | m | |
|---|---|---|---|---|---|---|---|--|

ii

mmm (if ii = 10)

This 16-bit field is present only if the instruction is a jump instruction or an operand is a memory addressing mode of the form [bx+xxxx], [xxxxx], or a constant.

00 = zero operand instructions
01 = jump instructions
10 = not
11 = illegal (reserved)

000 = AX
001 = BX
010 = CX
011 = DX
100 = [BX]
101 = [xxxx+BX]
110 = [xxxx]
111 = constant

| | | | | | | | | |
|---|---|---|---|---|---|---|---|--|
| 0 | 0 | 0 | 0 | 1 | i | i | i | |
|---|---|---|---|---|---|---|---|--|

mmm (if ii = 10)

This 16-bit field is always present and contains the target address to jump move into the instruction pointer register if the jump is taken.

000 = je
001 = jne
010 = jb
011 = jbe
100 = ja
101 = jae
110 = jmp
111 = illegal

| کدهای پویش (Scan Codes) برای صفحه کلید کامپیوترهای شخصی (بصورت مبنای ۱۶) | | | | | | | | | | | |
|--|------|----|---------|------|----|---------|------|----|--------|-------------------------|----|
| Key | Down | Up | Key | Down | Up | Key | Down | Up | Key | Down | Up |
| Esc | 1 | 81 | [{ | 1A | 9A | , < | 33 | B3 | center | 4C | CC |
| 1 ! | 2 | 82 |] } | 1B | 9B | . > | 34 | B4 | right | 4D | CD |
| 2 @ | 3 | 83 | Enter | 1C | 9C | / ? | 35 | B5 | + | 4E | CE |
| 3 # | 4 | 84 | Ctrl | 1D | 9D | R shift | 36 | B6 | end | 4F | CF |
| 4 \$ | 5 | 85 | A | 1E | 9E | * PrtSc | 37 | B7 | down | 50 | D0 |
| 5 % | 6 | 86 | S | 1F | 9F | alt | 38 | B8 | pgdn | 51 | D1 |
| 6 ^ | 7 | 87 | D | 20 | A0 | space | 39 | B9 | ins | 52 | D2 |
| 7 & | 8 | 88 | F | 21 | A1 | CAPS | 3A | BA | del | 53 | D3 |
| 8 * | 9 | 89 | G | 22 | A2 | F1 | 3B | BB | / | E0 35 | B5 |
| 9 (| 0A | 8A | H | 23 | A3 | F2 | 3C | BC | enter | E0 1C | 9C |
| 0) | 0B | 8B | J | 24 | A4 | F3 | 3D | BD | F11 | 57 | D7 |
| - _ | 0C | 8C | K | 25 | A5 | F4 | 3E | BE | F12 | 58 | D8 |
| = + | 0D | 8D | L | 26 | A6 | F5 | 3F | BF | ins | E0 52 | D2 |
| Bksp | 0E | 8E | ; : | 27 | A7 | F6 | 40 | C0 | del | E0 53 | D3 |
| Tab | 0F | 8F | ' " | 28 | A8 | F7 | 41 | C1 | home | E0 47 | C7 |
| Q | 10 | 90 | ` ~ | 29 | A9 | F8 | 42 | C2 | end | E0 4F | CF |
| W | 11 | 91 | L shift | 2A | AA | F9 | 43 | C3 | pgup | E0 49 | C9 |
| E | 12 | 92 | \ | 2B | AB | F10 | 44 | C4 | pgdn | E0 51 | D1 |
| R | 13 | 93 | Z | 2C | AC | NUM | 45 | C5 | left | E0 4B | CB |
| T | 14 | 94 | X | 2D | AD | SCRL | 46 | C6 | right | E0 4D | CD |
| Y | 15 | 95 | C | 2E | AE | home | 47 | C7 | up | E0 48 | C8 |
| U | 16 | 96 | V | 2F | AF | up | 48 | C8 | down | E0 50 | D0 |
| I | 17 | 97 | B | 30 | B0 | pgup | 49 | C9 | R alt | E0 38 | B8 |
| O | 18 | 98 | N | 31 | B1 | - | 4A | CA | R ctrl | E0 1D | 9D |
| P | 19 | 99 | M | 32 | B2 | left | 4B | CB | Pause | E1 1D 45 E1 9D C5 | - |

لیست کامل کدهای اسکی و پویش برای صفحه کلید کامپیوترهای شخصی (بصورت مبنای ۱۶)

| Key | Scan Code | ASCII | Shift | Ctrl | Alt | Num | Caps | Shift Caps | Shift Num |
|-----|-----------|-------|-------|------|------|-----|------|------------|-----------|
| Esc | 01 | 1B | 1B | 1B | - | 1B | 1B | 1B | 1B |
| 1 ! | 02 | 31 | 21 | | 7800 | 31 | 31 | 31 | 31 |

| | | | | | | | | | |
|---------|----|----|-------|------|------|----|----|-------|-------|
| 2 @ | 03 | 32 | 40 | 0300 | 7900 | 32 | 32 | 32 | 32 |
| 3 # | 04 | 33 | 23 | - | 7A00 | 33 | 33 | 33 | 33 |
| 4 \$ | 05 | 34 | 24 | - | 7B00 | 34 | 34 | 34 | 34 |
| 5 % | 06 | 35 | 25 | - | 7C00 | 35 | 35 | 35 | 35 |
| 6 ^ | 07 | 36 | 5E | 1E | 7D00 | 36 | 36 | 36 | 36 |
| 7 & | 08 | 37 | 26 | - | 7E00 | 37 | 37 | 37 | 37 |
| 8 * | 09 | 38 | 2A | - | 7F00 | 38 | 38 | 38 | 38 |
| 9 (| 0A | 39 | 28 | - | 8000 | 39 | 39 | 39 | 39 |
| 0) | 0B | 30 | 29 | - | 8100 | 30 | 30 | 30 | 30 |
| - _ | 0C | 2D | 5F | 1F | 8200 | 2D | 2D | 5F | 5F |
| = + | 0D | 3D | 2B | - | 8300 | 3D | 3D | 2B | 2B |
| Bksp | 0E | 08 | 08 | 7F | - | 08 | 08 | 08 | 08 |
| Tab | 0F | 09 | 0F00 | | - | 09 | 09 | 0F00 | 0F00 |
| Q | 10 | 71 | 51 | 11 | 1000 | 71 | 51 | 71 | 51 |
| W | 11 | 77 | 57 | 17 | 1100 | 77 | 57 | 77 | 57 |
| E | 12 | 65 | 45 | 05 | 1200 | 65 | 45 | 65 | 45 |
| R | 13 | 72 | 52 | 12 | 1300 | 72 | 52 | 72 | 52 |
| T | 14 | 74 | 54 | 14 | 1400 | 74 | 54 | 74 | 54 |
| Y | 15 | 79 | 59 | 19 | 1500 | 79 | 59 | 79 | 59 |
| U | 16 | 75 | 55 | 15 | 1600 | 75 | 55 | 75 | 55 |
| I | 17 | 69 | 49 | 09 | 1700 | 69 | 49 | 69 | 49 |
| O | 18 | 6F | 4F | 0F | 1800 | 6F | 4F | 6F | 4F |
| P | 19 | 70 | 50 | 10 | 1900 | 70 | 50 | 70 | 50 |
| [{ | 1A | 5B | 7B | 1B | - | 5B | 5B | 7B | 7B |
|] } | 1B | 5D | 7D | 1D | - | 5D | 5D | 7D | 7D |
| enter | 1C | 0D | 0D | 0A | - | 0D | 0D | 0A | 0A |
| ctrl | 1D | - | - | - | - | - | - | - | - |
| A | 1E | 61 | 41 | 01 | 1E00 | 61 | 41 | 61 | 41 |
| S | 1F | 73 | 53 | 13 | 1F00 | 73 | 53 | 73 | 53 |
| D | 20 | 64 | 44 | 04 | 2000 | 64 | 44 | 64 | 44 |
| F | 21 | 66 | 46 | 06 | 2100 | 66 | 46 | 66 | 46 |
| G | 22 | 67 | 47 | 07 | 2200 | 67 | 47 | 67 | 47 |
| H | 23 | 68 | 48 | 08 | 2300 | 68 | 48 | 68 | 48 |
| J | 24 | 6A | 4A | 0A | 2400 | 6A | 4A | 6A | 4A |
| K | 25 | 6B | 4B | 0B | 2500 | 6B | 4B | 6B | 4B |
| L | 26 | 6C | 4C | 0C | 2600 | 6C | 4C | 6C | 4C |
| :: | 27 | 3B | 3A | - | - | 3B | 3B | 3A | 3A |
| ' " | 28 | 27 | 22 | - | - | 27 | 27 | 22 | 22 |
| ` ~ | 29 | 60 | 7E | - | - | 60 | 60 | 7E | 7E |
| Lshift | 2A | - | - | - | - | - | - | - | - |
| \ | 2B | 5C | 7C | 1C | - | 5C | 5C | 7C | 7C |
| Z | 2C | 7A | 5A | 1A | 2C00 | 7A | 5A | 7A | 5A |
| X | 2D | 78 | 58 | 18 | 2D00 | 78 | 58 | 78 | 58 |
| C | 2E | 63 | 43 | 03 | 2E00 | 63 | 43 | 63 | 43 |
| V | 2F | 76 | 56 | 16 | 2F00 | 76 | 56 | 76 | 56 |
| B | 30 | 62 | 42 | 02 | 3000 | 62 | 42 | 62 | 42 |
| N | 31 | 6E | 4E | 0E | 3100 | 6E | 4E | 6E | 4E |
| M | 32 | 6D | 4D | 0D | 3200 | 6D | 4D | 6D | 4D |
| , < | 33 | 2C | 3C | - | - | 2C | 2C | 3C | 3C |
| . > | 34 | 2E | 3E | - | - | 2E | 2E | 3E | 3E |
| / ? | 35 | 2F | 3F | - | - | 2F | 2F | 3F | 3F |
| Rshift | 36 | - | - | - | - | - | - | - | - |
| * PrtSc | 37 | 2A | INT 5 | 10 | - | 2A | 2A | INT 5 | INT 5 |
| alt | 38 | - | - | - | - | - | - | - | - |
| space | 39 | 20 | 20 | 20 | - | 20 | 20 | 20 | 20 |

| | | | | | | | | | |
|---------|----|------|------|------|------|------|------|------|------|
| caps | 3A | - | - | - | - | - | - | - | - |
| F1 | 3B | 3B00 | 5400 | 5E00 | 6800 | 3B00 | 3B00 | 5400 | 5400 |
| F2 | 3C | 3C00 | 5500 | 5F00 | 6900 | 3C00 | 3C00 | 5500 | 5500 |
| F3 | 3D | 3D00 | 5600 | 6000 | 6A00 | 3D00 | 3D00 | 5600 | 5600 |
| F4 | 3E | 3E00 | 5700 | 6100 | 6B00 | 3E00 | 3E00 | 5700 | 5700 |
| F5 | 3F | 3F00 | 5800 | 6200 | 6C00 | 3F00 | 3F00 | 5800 | 5800 |
| F6 | 40 | 4000 | 5900 | 6300 | 6D00 | 4000 | 4000 | 5900 | 5900 |
| F7 | 41 | 4100 | 5A00 | 6400 | 6E00 | 4100 | 4100 | 5A00 | 5A00 |
| F8 | 42 | 4200 | 5B00 | 6500 | 6F00 | 4200 | 4200 | 5B00 | 5B00 |
| F9 | 43 | 4300 | 5C00 | 6600 | 7000 | 4300 | 4300 | 5C00 | 5C00 |
| F10 | 44 | 4400 | 5D00 | 6700 | 7100 | 4400 | 4400 | 5D00 | 5D00 |
| num | 45 | - | - | - | - | - | - | - | - |
| scr1 | 46 | - | - | - | - | - | - | - | - |
| home | 47 | 4700 | 37 | 7700 | - | 37 | 4700 | 37 | 4700 |
| up | 48 | 4800 | 38 | - | - | 38 | 4800 | 38 | 4800 |
| pgup | 49 | 4900 | 39 | 8400 | - | 39 | 4900 | 39 | 4900 |
| - (kpd) | 4A | 2D | 2D | - | - | 2D | 2D | 2D | 2D |
| left | 4B | 4B00 | 34 | 7300 | - | 34 | 4B00 | 34 | 4B00 |
| center | 4C | 4C00 | 35 | - | - | 35 | 4C00 | 35 | 4C00 |
| right | 4D | 4D00 | 36 | 7400 | - | 36 | 4D00 | 36 | 4D00 |
| + (kpd) | 4E | 2B | 2B | - | - | 2B | 2B | 2B | 2B |
| end | 4F | 4F00 | 31 | 7500 | - | 31 | 4F00 | 31 | 4F00 |
| down | 50 | 5000 | 32 | - | - | 32 | 5000 | 32 | 5000 |
| pgdn | 51 | 5100 | 33 | 7600 | - | 33 | 5100 | 33 | 5100 |
| ins | 52 | 5200 | 30 | - | - | 30 | 5200 | 30 | 5200 |
| del | 53 | 5300 | 2E | - | - | 2E | 5300 | 2E | 5300 |

متغیرهای bios مرتبط با صفحه کلید

| نام | آدرس | اندازه | توضیح |
|------------------------------|-------|--------|--|
| KbdFlags1 (modifier flags) | 40:17 | Byte | این بایت وضعیت فعلی کلیدهای اصلاح کننده (modifier) را نگهداری می نماید. |
| | | | bit 7: Insert mode toggle |
| | | | bit 6: Capslock toggle (1=capslock on) |
| | | | bit 5: Numlock toggle (1=numlock on) |
| | | | bit 4: Scroll lock toggle (1=scroll lock on) |
| | | | bit 3: Alt key (1=alt is down) |
| | | | bit 2: Ctrl key (1=ctrl is down) |
| | | | bit 1: Left shift key (1=left shift is down) |
| | | | bit 0: Right shift key (1=right shift is down) |
| KbdFlags2 (Toggle keys down) | 40:18 | Byte | این بایت مشخص می کند کدامیک از کلیدهای کنترلی (Toggle keys) در حال حاضر فشرده شده اند. |
| | | | bit 7: Insert key (currently down if 1) |
| | | | bit 6: Capslock key (currently down if 1) |
| | | | bit 5: Numlock key (currently down if 1) |
| | | | bit 4: Scroll lock key (currently down if 1) |
| | | | bit 3: Pause state locked (ctrl-Numlock) if one |

| | | | |
|---|------|-------|-----------|
| bit 2: SysReq key (currently down if 1) | | | |
| bit 1: Left alt key (currently down if 1) | | | |
| bit 0: Left ctrl key (currently down if 1) | | | |
| BIOS uses this to compute the ASCII code for an alt-Keypad sequence. | Byte | 40:19 | AltKpd |
| Offset of start of keyboard buffer (1Eh). Note: this variable is not supported on many systems, be careful if you use it. | Word | 40:80 | BufStart |
| Offset of end of keyboard buffer (3Eh). See the note above. | Word | 40:82 | BufEnd |
| Miscellaneous keyboard flags. | | | |
| bit 7: Read of keyboard ID in progress | | | |
| bit 6: Last char is first kbd ID character | | | |
| bit 5: Force numlock on reset | | | |
| bit 4: 1 if 101-key kbd, 0 if 83/84 key kbd. | Byte | 40:96 | KbdFlags3 |
| bit 3: Right alt key pressed if 1 | | | |
| bit 2: Right ctrl key pressed if 1 | | | |
| bit 1: Last scan code was E0h | | | |
| bit 0: Last scan code was E1h | | | |
| More miscellaneous keyboard flags. | | | |
| bit 7: Keyboard transmit error | | | |
| bit 6: Mode indicator update | | | |
| bit 5: Resend receive flag | | | |
| bit 4: Acknowledge received | Byte | 40:97 | KbdFlags4 |
| bit 3: Must always be zero | | | |
| bit 2: Capslock LED (1=on) | | | |
| bit 1: Numlock LED (1=on) | | | |
| bit 0: Scroll lock LED (1=on) | | | |