

میکرопروسور ۸۰۸۶

استاد قرائی

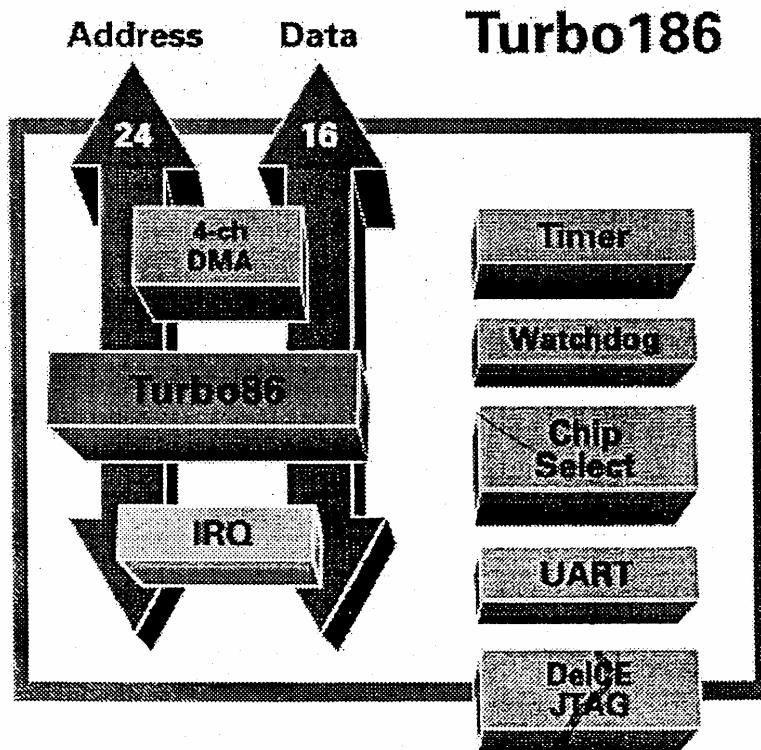
تهریه کننده : حامد مظاہری

دانشگاه آزاد اسلامی - واحد تهران جنوب
دانشکده فنی

برگرفته از سایت
ir-micro.com

كتاب ميكروپرفسسور

8086



تهيه و تنظيم : مهندس محمد حسن قراني

مقدمه

پیشرفت تکنولوژی الکترونیک در طی سه دهه گذشته چهره کامپیوترها را ابتدا سال به سال و سپس ماه به ماه، اخیراً روز به روز تغییر داده است.

اولین کامپیوتر بنام ENIAC معروف بوده و در سال ۱۹۴۶ میلادی بوسیله اکرت (EKKERT) و ماچلی (MAUCHLY) از مدرسه مهندسی برق دانشگاه پنسیلوانیا ساخته شد، که بیشتر شبیه یک ماشین حساب بود حتی یک ماشین حساب با کاربرد وسیع هم نبود بلکه بیشتر برای محاسبه جداول مربوط به پرتاب توپ از آن استفاده می‌کردند. در این ماشین از ۱۸۰۰ لامپ خلاء استفاده شده بود که ۴۰ قفسه بزرگ را پر می‌کرد. و این قفسه‌ها در یک اتاق 10×13 متر قرار گرفته بودند. همزمان با IBM شرکت ENIAC نیز اولین کامپیوتر خود را بنام IBM 603 به بازار عرضه کرد، که البته کامپیوتر الکترونیکی کوچکی بود. پس از آن کامپیوترهای بزرگتر IBM مدل 604 در سال ۱۹۴۸ (همان سال اختراع ترانزیستور) و IBM 605 در سال ۱۹۵۴ به بازار عرضه شد. عنصر اصلی در ساخت همگی این کامپیوترها همان لامپ خلاء بود و بنام کامپیوترهای نسل اول (FIRST GENERATION) معروف می‌باشد.

دومین نسل از کامپیوترها آنهایی بودند که از ترانزیستور بعنوان عنصر اصلی الکترونیکی استفاده کردند. مثل CDC6600، IBM7094، IBM7090 نیز از این دسته است. پیدایش تکنولوژی مدارهای مجتمع در اوایل سالهای ۶۰ و اوایل دهه ۶۰ دوره جدیدی را در صنعت کامپیوتر بوجود آورد که باعث تولید سومین نسل کامپیوتر شد. در نسل سوم از مدارهای مجتمع در مقیاس کوچک (SSI) (یک تا ۱۰ گیت در یک تراشه) و بعضی مدارهای در مقیاس متوسط (MSI) ده تا صد گیت در یک تراشه استفاده می‌شد. و از سالهای ۱۹۶۶ به بعد کامپیوترهای نسل سوم به بازار عرضه شدند (IBM360-370) از جمله کامپیوترهای معروف این نسل هستند. تا قبل از پیدایش مدارهای مجتمع گرانی‌قیمت کامپیوترها باعث می‌شد که کامپیوترهای بزرگ برای چند محل ساخته شوند (Main Frame) و ترمینالهای آنها در اختیار استفاده کننده بصورت اجاره‌ای قرار می‌گرفت. با پیدایش و پیشرفت مدارهای مجتمع کامپیوترهای کوچکتری بنام (Mini Computer) ساخته شد که شرکتها و مؤسسات نسبتاً بزرگ مثل دانشگاهها و شرکت بیمه بتوانند خریداری کنند. از مینی کامپیوترهای معروف PDP-11 را می‌توان نام برد که در دهه ۱۹۷۰ به بازار عرضه شد. پیشرفت روزافزون تکنولوژی و پیدایش تکنولوژی MOS امکان ساخت مدارهای مجتمع با مقیاس بزرگ (LSI بیش از هزار دروازه در IC) را فراهم نمود و این تکنولوژی باعث

شد که بتوانند میکرопرنسور طراحی نمایند. یعنی یک مدار مجتمع قابل برنامه زیزی که با گرفتن دستور می تواند عملیات جمع تفریق و مقایسه و یا عملیات منطقی را انجام دهد. اولین ریزپردازنده که به بازار ارائه شد تراشه 4004 بود که در سال ۱۹۷۱ شرکت اینتل (INTEL) البته نه به عنوان ریزپردازنده بلکه بعنوان تراشه قابل برنامه ریزی برای ماشین حساب آن را ارائه نمود. این مسئله که تولد ریزپردازنده نیز به آن گفته اند در صنعت الکترونیک باعث انقلابی شد. چون از یک تراشه تها با تغییر برنامه آن، می توان در بخش‌های مختلف صنعت استفاده کرد. عمدۀ هزینه تراشه طراحی آن است، هزینه ساخت برای یک تراشه بسیار ناچیز است، بکارگیری یک تراشه در جاهای مختلف باعث ارزان شدن قیمت سیستمهای ساخته شده می گردد. لذا ریزپردازنده جای خود را در صنعت (مثل کنترل ماشینهای صنعتی) در مصارف خانگی (مثل فرهای میکروویو) در مصارف علمی (مثل ماشینهای حساب و وسائل کمک آموزشی) باز کردند.

پیدایش ریزپردازنده ها طراحی سیستمهای دیجیتالی را بطور چشمگیری تغییر داد. طراحان امروزه بیشتر سراغ میکرопرنسور می روند و از نرم افزارهای مختلف بگونه ای استفاده می کنند که یک مدار طراحی شده در چند جای مختلف بکار گرفته شود. با استفاده از ریزکامپیوتر (Micro Computer) گفتهند. ریز کامپیوتر امروزه با قیمتی کمتر از ۳۰۰ دلار از نظر قدرت محاسباتی بالاتر از اولین کامپیوتر ساخته شده (ENIAC) است. این کامپیوتر ۳۰ برابر سریعتر از آن عمل می کند، خیلی بیشتر از آن حافظه دارد، هزاران بار بیشتر قابل اطمینان است. توان مصرفی آن در حد یک لامپ روشنائی است. کامپیوترهای جدیدی که با استفاده از تکنولوژی LSI ساخته شدند را نسل چهارم کامپیوتر نامیدند. از ریزپردازنده های این نسل Intel8080، PDP11 CRAY را برای میکروکامپیوتر و ۸۰۸۶ شرکت اینتل (ریزکامپیوتر) VX-11/780 (برای مینی کامپیوترها و کامپیوترهای بزرگ) می توان نام برد. در نسل پنجم کامپیوترها از مدارات مجتمع VLSI استفاده شد مثال هایی از این نوع مورد بررسی این کتاب مابه هیچ وجه قصد پرداختن به کامپیوترهای مختلف را نداریم. موضوع این بحث تنها به بررسی ریزپردازنده ها و کاربرد آنها و بصورت یک مثال ۸۰۸۶ مورد بررسی جزئی تر و دقیق تر قرار داده می شود.

فصل اول

۱-۱- تعاریف متداول

۱-۱-۱- بیت (BIT)

بیت (BIT)، منظور از یک بیت یک رقم از اعداد مبنای ۲ می باشد کلمه بیت خلاصه ای از ترکیب دو واژه Digital و Binary بدست آمده است. واضح است تنها مقادیر ۰,۱ را بخود اختصاص می دهد. این دو رقم را باید با ارقام ۰ تا ۹ پایه ده مقایسه نمود (درس مدار منطقی) تا به چگونگی انجام محاسبات در کامپیوتر واقع شوید. علت انتخاب پایه دو در کامپیوتر سادگی ایجاد (ساخت) این اعداد است. به ۸ بیت یک بایت می گویند و به ۴ بیت یک نیبل (Nibble) گفته می شود. طول کلمه یک ریزپردازنده عبارت است از تعداد بیتی که ریزپردازنده می تواند بطور همزمان پردازش نماید (Word Length). طول کلمه ریزپردازنده ۴۰۰۴ اینتل ۴ بیتی بود. طول کلمه ریزپردازنه های ۸۰۸۰ ۸۰۸۶ M68000 Z80 موتورولا و Z-80 شرکت زایلوگ ۸ بیتی است و طول کلمه ۸۰۸۶ ۸۰۸۰ M68000 Z8000 ۱۶ بیت دارای ۳۲ می باشند. نکته ای که باید یادآور شد اینکه محاسباتی که میکرو انجام می دهد. بطول کلمه آن محدود نمی شود. یعنی میکرورسسور Z-80 می تواند دو عدد ۱۶ بیتی را نیز جمع کند (البته تحت شرایطی).

۱-۲- واحد محاسبات عددی و منطقی (ALU = Arithmetic Logic Unit)

ALU یا واحد محاسبات عددی و منطقی یک ریزپردازنده، عبارت است از بخشی از سیستم که در آن کلیه محاسبات عددی و منطقی صورت می گیرد. طول لغتی که در این واحد پردازش می شود برابر طول کلمه ریزپردازنده می باشد.

۱-۳- انواع حافظه (MEMORY)

تشکیل شده از تعدادی سلولهای دیجیتالی که در آنها می توان اطلاعات باینری را ثبت کرد و در موقع لزوم مورد استفاده قرار داد. حافظه ها از نظر کاربرد به دو دسته تقسیم می شوند. حافظه با قابلیت خواندن و نوشتن و حافظه فقط با قابلیت خواندن.

۱-۳-۱- ثبات (REGISTER)

عبارت است از یک واحد حافظه‌ای که می‌توان یک کلمه ریزپردازندۀ را در آن وارد نمود و نگهداری کرد. هر ریزپردازندۀ دارای چند ثبات می‌باشد: ثبات‌ها به دو دسته تقسیم می‌شوند.

(الف) ثبات‌های عمومی، که در کارهای مختلف از آنها می‌توان استفاده کرد.

(ب) ثبات‌ها برای کاربرد خاص، همانطور که از نام آنها پیداست برای کارهای خاصی استفاده می‌شوند.

۱-۳-۲- حافظه‌های فقط خواندنی (ROM=Read Only Memory)

این حافظه امروزه، در حد وسیعی و در انواع مختلف با استفاده از تکنولوژی LSI, VLSI ساخته می‌شوند. اطلاعات آنها غیر قابل تغییر است. و در بیرون کامپیوتر برنامه ریزی می‌شوند. به آنها حافظه پاک نشدنی (NON VOLATILE) می‌گویند. گاهاً هنگام ساخت توسط کارخانه سازنده پر می‌شوند (ROM) انواع دیگر دارند (E²PROM, EPROM, PROM) که توسط استفاده کننده پر می‌شوند.

۱-۳-۳- حافظه‌ها با قابلیت خواندن و نوشتن (RW MEM)

اصطلاحاً به این حافظه‌های RAM یعنی (Random Access Memory) حافظه با دسترسی تصادفی می‌گویند. یعنی زمان دسترسی به کلیه اطلاعات یا عبارت دیگر به کلیه مکانهای این حافظه یکسان است و مستقل از محل قرار گرفتن آن محل مورد نظر در ابتدا یا آخر حافظه است. در این حافظه‌ها هم می‌توان نوشت و هم می‌توان هر محلی که مورد نظر است خواند، یعنی اطلاعات آنرا برداشت.

۱-۴-۱- آدرس بس AddressBus

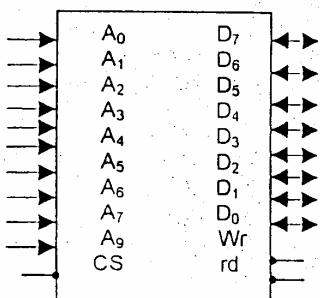
در تراشه‌های حافظه تعدادی پایه وجود دارد که توسط آنها مشخص می‌شود با کدام محل حافظه کار دارید این پایه‌ها را، پایه آدرس (ADDRESS BUS) می‌گویند اگر ۳ پایه آدرس داشته باشد، حافظه شما $= 8^3 = 8 \times 8 \times 8 = 512$ محل دارد اگر بیش از ۸ محل نیاز داشته باشد باید $= 16^4 = 16 \times 16 \times 16 \times 16 = 65536$ محلی باشد بهمین ترتیب ظرفیت حافظه زیاد می‌شود اگر حافظه ۱۰ پایه آدرس دارد $= 1024^2 = 1024 \times 1024 = 1048576$ محل که نزدیکترین عدد به ۱۰۰۰ است به آن یک کیلو حافظه گفته می‌شود.

۱-۴-۲- دیتا بس (DATA BUS)

پایه های دیگری هم روی تراشه حافظه ملاحظه می کنید بنام مسیر داده یا (DATA BUS) که آنچه را که می خواهید بنویسید یا از حافظه بخوانید روی این پایه ها عبور می کند. تعداد این پایه برآبر با تعداد بیت های کلمه سیستم (Word Length) می باشد اگر سیستمی ۸ بیتی باشد حافظه های آن هم ۸ بیتی و تعداد خطوط داده آن نیز ۸ عدد می باشد. شکل صفحه بعد یک حافظه با قابلیت خواندن و نوشتن بظرفیت $8 \times 1K$ را نشان می دهد.

خطوط آدرس فقط ورودی به حافظه و خطوط دیتا دو طرفه است.

۱۱- خط آدرس $= 2^9 = 512$ محل و **۸ خط دیتا** $= 8 \times 8 = 64$ یعنی هر محل ۸ بیت اطلاعات را ذخیره می کند.



شکل ۱-۱: نمایش یک IC حافظه

۱-۵- باب های ورودی و خروجی (INPUT/OUTPUT PORTS)

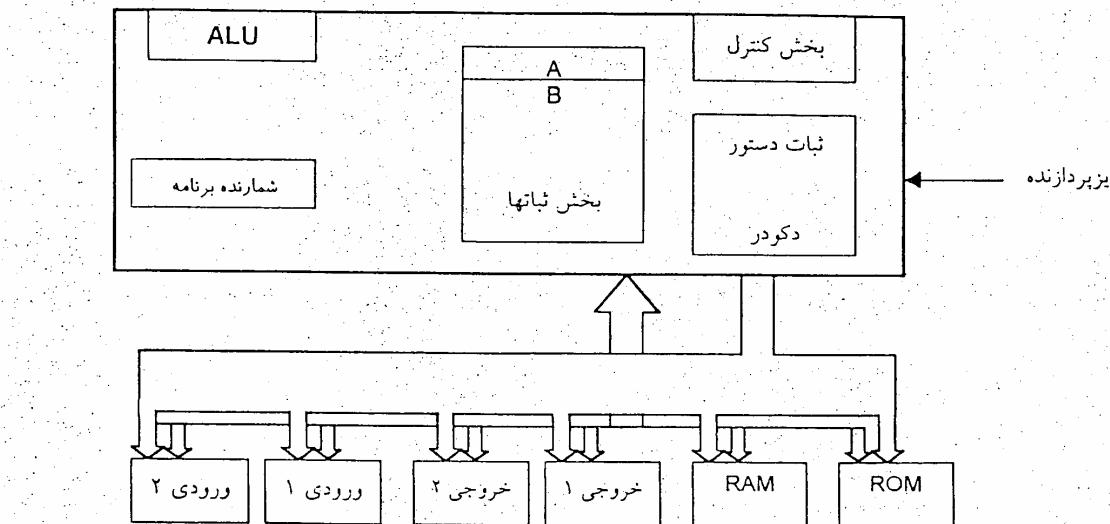
برای اینکه ریزپردازنده ها مفید واقع شوند باید بتوان از خارج به آنها برنامه و داده های مختلفی داد تا کارهای متعددی انجام دهند و نتایج کارها را از آنها بازستانیم این ارتباط از محل هائی صورت می گیرد بنام دروازه یا بابهای ورودی و خروجی

۱-۶- ساختار کلی ریزپردازنده

ریزپردازنده بعنوان بخش کلی و فعالی از ریز کامپیوتر تمام امور مربوط به پردازش اطلاعات، کنترل بخش های مختلف و تولید پالس ساعت سیستم را بعده دارد. شکل ۱-۲ نمای داخلی و ارتباط ریزپردازنده، با دستگاههای دیگر سیستم را نشان می دهد.

برای اتصال حافظه RAM یا ROM، باب های ورودی و خروجی به ریزپردازنده از یک سری مسیرهای عمومی (BUS) استفاده می شود که عیناً این پایه ها در ریزپردازنده هم وجود دارد. برای درک چگونگی استفاده از BUS ها و برقراری ارتباط بین ریزپردازنده با المانهای جانبی به ذکر مثال زیر می تردیم: فرض کنید میکرورسسور بخواهد از محل ۱۲۱ RAM اطلاعاتی را بخواند. ابتدا آدرس (عنی عدد ۱۲۱) را روی مسیر آدرس قرار می دهد سیگنال RD=Read را فعال کنترلی را فعال می کند که چیپ RAM به آدرس جواب دهد. بعد سیگنال RD=Read را فعال

می کند که چیز اطلاعات محل ۱۲۱ خود را به بیرون بفرستد، پس از آن دیتای موجود بوسیله مسیر دینا درون ثبات مورد نظر در میکروپرسسور قرار می گیرد. ممکن است آنچه که خوانده شد یک دستور باشد، در اینصورت آدرس این محل باید توسط PC=Program Counter داده شود بعد از آنکه وارد میکروپرسسور شد حال باید ترجمه شود که چه دستوری است پس از مشخص شدن باید اجرا گردد.



شکل ۱-۲ یک شمای کلی از ساختمان ریز کامپیوترا

۷-۱- کاربرد ریزپردازنده

استفاده روزافزون از ریزپردازنده در زمینه های مختلف در سه دهه گذشته نظر افراد بسیاری را بخود جلب کرده است. بطوریکه در عرض کمتر از چهار سال پس از عرضه اولین ریزپردازنده 4004 (در سال ۱۹۷۱) جای خود را به عنوان عناصر اصلی سیستم های مختلف باز کرده اند. ریزپردازنده های مختلف با ویژگی های خاص به زودی به بازار عرضه شد و در زمینه های مختلف مثل دستگاه های کنترل بسیار دقیق و پیچیده، ترمینال های باهوش (INTELEGENT)، ماشین حساب های فروشگاهها، انتقال اطلاعات با سرعت های کم و متوسط و پردازش سیگنالها بکار گرفته شدند. با وجود استفاده وسیعی که ریزپردازنده ها در قسمت های مختلف پیدا کرده اند، ولی همیشه یک عامل باعث محدودیت استفاده از ریزپردازنده ها شده است. این محدودیت سرعت ریزپردازنده است، یک ریزپردازنده معمولاً در یک ثانیه یک میلیون عمل (جمع، تفريق، عمل منطقی یا وارد و خارج کردن اطلاعات) انجام می دهد، فرض کنید در یک سیستم عمل پردازش سیگنال توسط ریزپردازنده انجام بگیرد. ریزپردازنده سیگنال را دریافت می کند روی هر نمونه فرض کنید ۲۰ دستور پردازش نماید و

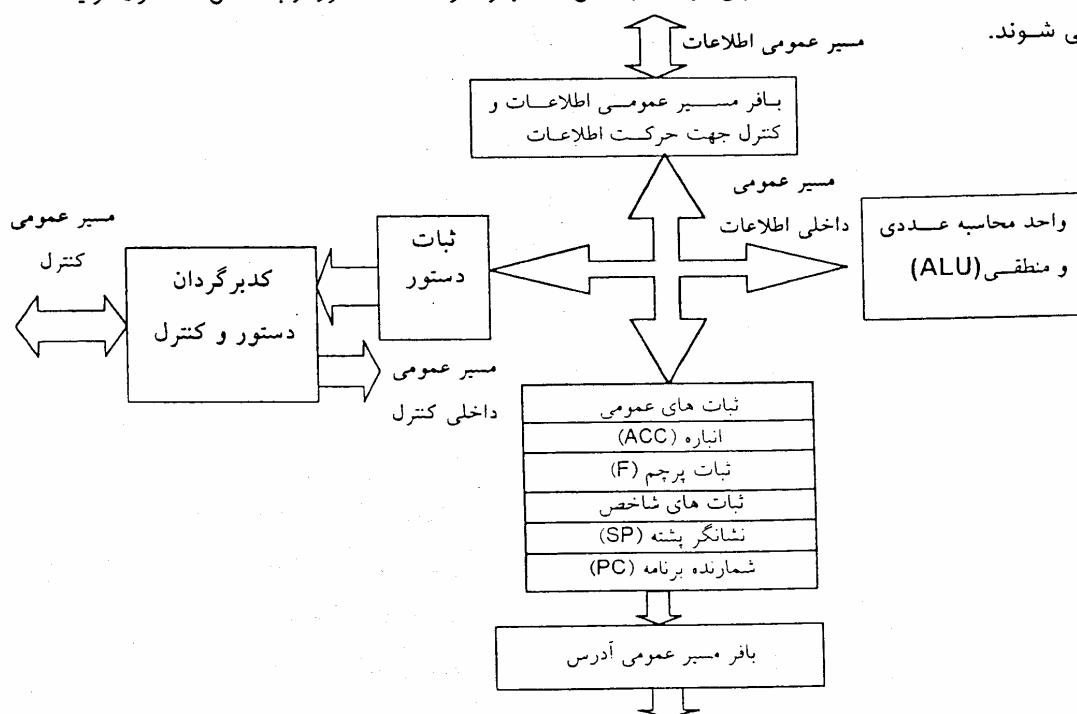
اطلاعات پردازش شده را به خروجی ارسال نماید. در اینصورت در هر ثانیه بتوانید ۵۰۰۰۰ نمونه از سیگنال ورودی دریافت کرده و بدون اشکال پردازش نماید. حال اگر سرعت نمونه برداری بیش از این مقدار باشد، ریزپردازنده عقب خواهد ماند و کار سیستم مختلف می‌شود.

۱-۸- ساختمان ریزپردازنده

در بخش قبل با شمای کلی ساختمان ریزپردازنده آشنا شدیم. توضیح دقیقتری بصورت کلی درباره ریزپردازنده در این قسمت ارائه می‌شود و انشاء‌الله چگونگی برنامه ریزی ریزپردازنده‌ها و تکنیک مختلف برنامه ریزی به زبان ماشین (با زبان اس‌بی‌لی) موضوع فصول بعدی خواهد بود.

۱-۸-۱- ریزپردازنده یا واحد پردازش مرکزی ریزکامپیوتر

قبل‌آنیز اشاره شد که ریزپردازنده واحد پردازش مرکزی کامپیوتر است و لذا به آن (Central Processing Unit) CPU گفته می‌شود. در شکل ۱-۳ یک شمای کلی نشان داده شده است. در داخل CPU نیز مسیرهای عمومی برای مبادله اطلاعات وجود دارد (BUS). کنترل بخش‌ها نیز توسط سیگنال‌هایی که از طریق همین مسیرها ارسال می‌شود اجرا می‌گردد. این سیگنال‌های کنترلی توسط بخش کد برگردان دستور، و بخش کنترل تولید می‌شوند.



شکل ۱-۳- شمای کلی از ساختمان داخلی یک ریزپردازنده

در دنباله این بحث به توضیح بخش های مختلف شکل ۱-۳ می پردازیم.

۱-۸-۲ - ثبات های عمومی (General Purpose Register)

قبل ذکر شد که درون یک ریزپردازنده تعدادی ثبات داریم که برای نگهداری بخشی از اطلاعات موجود در سیستم می توانند مورد استفاده قرار بگیرند. ظرفیت هر ثبات به اندازه طول کلمه سیستم است. دسترسی به محتوای ثبات ها بمراتب ساده تر و سریعتر از دسترسی به محتوای حافظه کامپیوتر است. لذا ریزپردازنده در ثبات های عمومی اطلاعاتی را نگهداری می کند. که بیش از هر اطلاعات دیگری به آن نیاز دارد.

ریزپردازنده ثبات های دیگری دارد که هر کدام کار خاصی را انجام می دهند و به آنها ثبات با اهداف خاص (Special Purpose Reg) گویند. که در ادامه به بحث تک آنها پرداخته خواهد شد.

۱-۸-۳ - ابیاره (Accumulator)

در اکثر ریزپردازنده ها یکی از ثبات های عمومی بیش از ثبات های دیگر مورد استفاده قرار می گیرد. این ثبات را معمولاً ابیاره یا آکومولاتور می نامند. همیشه یکی از داده های مورد برداش درون آکومولاتور است و حاصل هم به آن ریخته می شود. البته ریزپردازنده هایی هم هستند که تعداد ابیاره های آنها بیش از یکی است. بعنوان مثال ریزپردازنده M6800 دو ابیاره A,B می باشد.

۱-۸-۴ - ثبات پرچم (Flag Register)

این ثبات با سایر ثباتها فرق دارد، هر یک از بیتها این ثبات میان اطلاعات خاصی است و ممکن است تمام بیتها آن قابل استفاده نباشد. اطلاعاتی که این ثبات در اختیار ریزپردازنده قرار می دهد به شرح زیر است :

۱-۸-۴-۱ - پرچم صفر (ZeroFlag)

در صورتیکه نتیجه حاصل از عملی که ریزپردازنده انجام می دهد صفر شود پرچم صفر (Z) مساوی یک می شود و در غیر اینصورت برابر صفر است.

۱-۸-۴-۲ - پرچم نقلی (CarryFlag)

در صورتیکه آخرین عمل انجام شده دارای یک بیت نقلی باشد پرچم C مساوی یک می شود و الا صفر خواهد بود.

۱-۸-۴-۳ - پرچم سرریز (OverflowFlag)

در صورتیکه حاصل آخرین عمل انجام شده از تعداد بیتهای سیستم بیشتر شود بعارت دیگر سرریز شود این بیت (V) برابر یک و در غیر اینصورت صفر خواهد بود.

۱-۸-۴-۴ - پرچم علامت (SignFlag)

آخرین بیت ظرف سیستم (آکومولاتور) را بیت علامت می نامند اگر عدد مورد نظر مثبت باشد بیت علامت صفر است و اگر عدد مورد نظر منفی باشد بیت علامت یک است و عدد بصورت متمم ۲ خواهد بود، حال اگر حاصل آخرین عمل انجام شده توسط ریزپردازنه منفی شود بیت علامت (S) یک می شود و اگر مثبت باشد این بیت صفر است.

۱-۸-۴-۵ - پرچم توازن (ParityFlag)

در صورتیکه تعداد بیت های ۱ موجود در نتیجه حاصل آخرین عمل ریزپردازنه عدد زوجی باشد (توازن زوج) و یا عدد فردی باشد (توازن فرد) پرچم توازن (P) مساوی یک و در غیر اینصورت برابر صفر می شود. معمولاً در سیستمها از توازن زوج استفاده می شود. علاوه بر بیتهای پرچم مذکور، دو بیت دیگر وجود دارد که کاربرد آنها در محاسبات BCD است و آنها عبارتند از :

۱-۸-۴-۶ - پرچم نقلی میانی (HalfCarryFlag)

پرچم رقم نقلی میان (HC) که در ریزپردازنه های ۸ بیتی کاربرد دارد، در صورتی مساوی ۱ خواهد شد که آخرین عمل انجام شده دارای یک رقم نقلی بین دو رقم میانی خود باشد.

۱-۸-۴-۷ - پرچم تفربیق (SubtractFlag)

اگر عمل انجام شده یک عمل تفربیق باشد این پرچم (N) مساوی ۱ خواهد شد، در غیر اینصورت صفر است. کاربرد عمدۀ این بیت ها، بغیر از اینکه اعلام وضعیت سیستم را بعده دارند، در پرسهای شرطی در برنامه نویسی نیز کاربرد دارند، که به تفصیل صحبت خواهد شد.

۱-۸-۵ - شمارنده برنامه PC (ProgramCounter)

شمارنده برنامه همیشه حاوی آدرس کلمه ای از حافظه است که در اولین فرصت باید به داخل ریزپردازنه خوانده شود، کلمه خوانده شده می تواند که متناظر با یک دستور باشد و یا اطلاعاتی که ریزپردازنه برای اجرای دستور موجود به آن نیاز دارد. در بعضی از ریزپردازنه ها به این ثبات اشاره کننده به دستور العمل می گویند (InstructionPointer)

۶-۱-۸-۱- ثبات شاخص (IndexRegister)

در بعضی از ریزپردازنده ها، از ثبات شاخص بمنظور تگهداری اطلاعات لازم جهت تعیین آدرس استفاده می شود. محتوی این ثبات با اطلاعاتی که در دستورات مربوطه وجود دارد جمع می شود تا آدرس مورد نظر بدست آید.

۶-۱-۸-۲- ثبات نشانه گر پشته (SP=StackPointer)

پشته چیست؟ پشته (Stack) عبارت است از حافظه ای که در هر لحظه تنها به آخرین اطلاعات وارد شده به آن می توان دست یافت، بهمین جهت می گویند دارای سازمان LIFO (Last In First Out) است. یعنی اولین اطلاعاتی که می توانید بردارید، آخرین اطلاعاتی است که در آن قرار داده اید. برای قرار دادن اطلاعات در پشته از دستور PUSH و برای برداشتن اطلاعات از دستور POP استفاده می شود. پشته (Stack) معمولاً بخشی در حافظه RAM سیستم است که برای این کار اختصاص داده می شود. برای قرار دادن اطلاعات در پشته یا برداشتن اطلاعات از پشته نیاز به آدرس محل مورد نظر داریم که نشانگر پشته (SP) آدرس مورد نظر را در اختیار ریزپردازنده قرار می دهد.

۶-۱-۸-۳- ثبات دستور (InstructionRegister)

کدهای مربوط به دستورات متوالی که باید توسط ریزپردازنده اجرا شود، در ابتدا به این بخش از ریزپردازنده وارد می شوند، و از آنجا به عنوان ورودی بخش کدبرگردان دستور بکار برده می شوند.

۶-۱-۹- کدبرگردان دستور (InstructionDecoder)

این بخش از ریزپردازنده که دستوری که باید اجرا شود را از ثبات دستور می گیرد و با توجه به آن سیگنالهای لازم جهت کنترل بخش های مختلف ریزپردازنده را تولید می کند. علاوه بر این، آن سیگنالهایی که به منظور کنترل بخش های داخلی ریزپردازنده بکار می روند، سیگنالهای جهت کنترل حافظه ها و باب های ورودی و خروجی، که در اطراف ریزپردازنده قرار دارند، نیز تولید می شوند. این سیگنالها توسط مسیر عمومی کنترل به قسمتهای مختلف ریز کامپیوتر انتقال می یابند. همچنین سیگنالهای کنترلی از خارج ریزپردازنده به بخش کنترلی آن وارد می شوند. این سیگنالها به منظورهای خاص که بعداً گفته خواهد شد بکار برده می شود.

۱-۱- واحد محاسبات عددی و منطقی (ALU=Arithmetic Logic Unit)

کلیه محاسبات عددی (از قبیل جمع، تفریق و احیاناً ضرب و تقسیم) و منطقی از قبیل (AND-OR-...) در این بخش انجام می‌گیرند. همچنین عملیاتی از قبیل انتقال چرخش به راست و به چپ، مقایسه دو کلمه در این بخش از ریزپردازنده صورت می‌گیرد، بطور خلاصه می‌توان گفت که کلیه عملیات پردازش در ALU انجام می‌شود و بخش‌های دیگر ریزپردازنده تنها به منظور حفظ اطلاعات یا کنترل مسیر حرکت اطلاعات بکار می‌روند. نتایج حاصل از عملیات ALU در حالت کلی، به انباره می‌ریزد.

سؤالات دوره ای فصل اول

۱- پایه‌های یک IC حافظه یا یک IC میکروپر سسور معمولاً به سه دسته تقسیم می‌شوند آنها را نام ببرید.

۲- اگر تعداد پایه‌های آدرس یک IC حافظه ۱۲ عدد باشد، چند محل قابل برنامه ریزی دارد؟

۳- اندازه لفت یک IC حافظه چگونه بیان می‌شود، و به کدام دسته از پایه‌های آن مربوط می‌شود؟

۴- قسمت‌های مختلف داخلی یک ریزپردازنده کدامند؟

۵- انواع حافظه‌ها کدامند؟

۶- واحد پردازش مرکزی (CPU) به چه قسمت‌هایی از یک سیستم کامپیوتری گفته می‌شود؟

۷- میکروپر سسور شامل چه قسمت‌هایی از یک سیستم کامپیوتری می‌باشد.

۸- ثبات‌های معروف یک کامپیوتر کدامند؟

۹- پرچم‌های (Flags) معروف یک سیستم کامپیوتری کدامند؟

۱۰- راجع به ثبات شمارنده برنامه (PC) و یا اشاره گر به دستور العمل (IP) چه میدانید؟

۱۱- پشته (Stack) چیست؟ و وظیفه اشاره گر به پشته (SP) کدام است؟

فصل دوم

«میکروپرسسور ۸۰۸۶»

یکی از مهمترین ریزپردازنده های ۱۶ بیتی که امروزه کاربرد بسیاری پیدا کرده است بخصوص در صنعت استفاده زیادی دارد، ریزپردازنده ۸۰۸۶ می باشد. ۸۰۸۶ اولین ریزپردازنده ۱۶ بیتی است که ساخت کارخانه اینتل است با ۲۹۰۰ ترانزیستور و با تکنولوژی H-MOS در سال ۱۹۷۸ ساخته شد. این ریزپردازنده فرم توسعه یافته ۸۰۸۵ است که کلیه دستورالعمل های آنها را اجرا می کند. این ریزپردازنده دارای ۲۰ خط آدرس بوده و قادر است $M=1^{20}$ بایت حافظه را آدرس دهی کند بعلاوه دارای ۱۶ خط دیتا است این تعداد خطوط آدرس و دیتا، تسهیلاتی جهت اجرای چند برنامه (Multi Programming) در آن واحد را فراهم می سازد. همچین با بکارگیری ریزپردازنده کمکی اش (8087) سیستم دارای قدرت چند پردازش نیز می شود (Multiy Proccesing).

شرکت اینتل در سال ۱۹۸۲ با متمرکز کردن بعضی از IC های جانی، ریزپردازنده های ۸۰۱۸۶، ۸۰۱۸۸ را با تعدادی دستورالعمل اضافی نسبت به ۸۰۸۶ به بازار عرضه کرد، یک سال بعد ۸۰۲۸۶ را با کارآئی بیشتر ساخت که پرداشگر کامپیوترهای AT شد. بعد از آن ۸۰۳۸۶ را برای آنکه تمام ریزپردازنده های بعدی بر پایه ۸۰۸۶ ساخته شده اند، لذا مطالعه و بررسی ۸۰۸۶ نسبت به سایر ریزپردازنده ها ضروری خواهد بود.

۱-۲- معماری (8086-Architecture)

شکل ۱-۲ تعداد و نام پایه های ۸۰۸۶ را نشان می دهد. ۲۰ خط آدرس و ۱۶ خط دیتا دارد. که ۱۶ خط آدرس با ۱۶ خط دیتا مشترک و بنام AD₁₅ تا AD₀ نشان داده شده است. جهت صرفه جویی از پایه های مشترک استفاده شده در واقع عمل مولتی پلکس روی آدرس و دیتا انجام شده است. به نظر می رسد امکان کاهش سرعت ریزپردازنده وجود دارد. ولی پس از مطالعه تایمینگ ریزپردازنده متوجه خواهد شد که سرعت تغییر نخواهد کرد. ریزپردازنده دارای ۱۶ خط کنترلی است که کلیه سیگنالهای مورد نیاز CPU را جهت انتقال دیتا از باس و ارتباط با دنیای بیرون CPU فراهم می کنند.

GND	1	PVcc
AD14	2	PAD ₁₆
AD13	3	PA ₁₆ /S ₃
AD12	4	PA ₁₇ /S ₄
AD11	5	PA ₁₈ /S ₅
AD10	6	PA ₁₉ /S ₆
AD9	7	PBHE/S ₇
AD8	8	PMN/MX
AD7	9	PRD
AD6	10	PHODL(RQ/GT0)
AD5	11	PHLDA(RQ/GT1)
AD4	12	PWR(LOCK)
AD3	13	PM/I/O(S ₂)
AD2	14	PDT/R(S ₁)
AD1	15	PDEN(S ₀)
AD0	16	PALE(QS ₀)
NMI	17	INTA(QS ₁)
INTR	18	TEST
CLK	19	READY
GND	20	RESET

شکل ۱-۲-نمای ظاهری IC ریزپردازندۀ ۸۰۸۶

۲-۲-شرح پایه های (Pin Allocation)

پایه ۴۰ ولتاژ تغذیه (Vcc) سیستم و پایه ۱ و ۲۰ (GND) زمین منبع تغذیه است که این ریزپردازندۀ با $Vcc=5V$ کار می کند هر دو GND باید وصل شود.

پایه های ۲ تا ۱۶ و ۳۹ برای آدرس و دیتا استفاده شده است (AD₀~AD₁₅). پایه های ۳۵، ۳۶، ۳۷ و ۳۸ برای A₁₆ تا A₁₉ که این پایه ها با S₁، S₅، S₄، S₆ مالتی پلکس شده اند.

پایه ۱۷ NMI (non-Maskable Interrupt) : پایه درخواست کننده وقفه است که بالبه (از صفر به یک) فعال می شود، نمی توان جلوی آن را گرفت (نمی توان آنرا پوشاند).

راجع به چگونه سرویس دادن به آن بعداً صحبت خواهیم کرد. این پایه ورودی به سیستم است.

پایه ۱۸ INTR (Interrupt Request) ورودی به سیستم است، درخواست کننده وقفه ای است که اگر پرچم اینترابت یک باشد (IF=1) می تواند سرویس بگیرد.

پایه ۱۹ CLK ورودی به سیستم است که تنظیم کننده زمان بندی سیستم نیز همین CLK ورودی است.

پایه ۲۱ RESET ورودی به سیستم است. اگر لول صفر منطقی برای مدت چهار پالس ساعت فعال باشد ریزپردازندۀ را به شرح زیر به اول برمی گرداند. چهار ثبات در سیستم وجود دارد : ES=0000H, SS=0000H, DS=0000H, CS=FFFFH و ثبات اشاره گر IP=0000H نیز خواهد شد. وضع سایر ثبات ها تعریف نشده است.

راجع به شرح وظایف ثباتها بعداً صحبت می‌شود. چون آدرس مؤثر هر دستور از جمع CS شیفت داده شده با IP بذست می‌آید ریزپردازنده به محل FFFF0H رفته و از آنجا شروع به اجرای برنامه می‌کند. یعنی با RESET کردن سیستم از محل FFFF0H شروع به خواندن دستورالعمل و اجرای آن می‌کند.

پایه ۲۲ (READY) ورودی به سیستم است، برای تطبیق سرعت ریزپردازنده با حافظه یا ورودی و خروجی که کنترل از ریزپردازنده هستند بکار می‌رود قرار گرفتن ولتاژ صفر منطقی روی این پایه بمنزله عدم آمادگی حافظه یا ورودی و خروجی مربوطه است و ریزپردازنده در انتظار می‌ماند.

پایه ۲۳ (TEST) ورودی به سیستم است. زمانی که ریزپردازنده یک دستور Wait دریافت می‌کند (دستور نرم افزاری در برنامه بنام WAIT). مادامیکه روی این پایه صفر باشد سیستم در حالت WAIT می‌ماند یعنی دستور دیگری اجرا نمی‌شود، بمحض HIGH شدن این پایه از WAIT خارج شده و به اجرای دستورات بعدی می‌پردازد.

پایه ۲۴ (INTA) پایه خروجی از سیستم است. پاسخ به درخواست وقفه (INTR) است. هدف ریزپردازنده از این پاسخ بدست آوردن آدرس برنامه ای است که قرار است به اجرای آن بپردازد.

پایه ۲۵ (Address Latch Enable) ALE خروجی از سیستم است برای بافر کردن آدرس Latch ها و جدا کردن آن از دیتا بکار می‌رود در ابتدا هر ماشین سیکل، سیستم آدرس را روی خط آدرس بس قرار می‌دهد و توسط ALE در مدارات که برای این منظور فراهم شده، بافر می‌شود. در پالس های بعدی دیتا روی بس وجود خواهد داشت که مسیر آنها جدا خواهد شد.

پایه ۲۶ (DEN) خروجی است. Active-low می‌باشد و 3-State است. (Data Enable) این پایه وقتی که ریزپردازنده در مد مینیمم اگر از درایور دیتا استفاده شود خروجی آنرا فعال می‌کند.

پایه ۲۷ (DataTransmit/Resive) DT/R خروجی از سیستم و 3-State است. وقتی که یک منطقی باشد ریزپردازنده دیتا ارسال می‌کند و زمانیکه صفر است ریزپردازنده دیتا دریافت می‌کند. لذا برای آماده کردن درایور دیتا بکار می‌رود تا جهت ارسال و دریافت را برایش مشخص نماید.

پایه ۲۸ M/I/O خروجی و 3-State است. بوسیله آن مشخص می‌کند که ریزپردازنده با حافظه سروکار دارد یا با ورودی و خروجی، اگر $M/I/O=0$ باشد با ورودی و خروجی و اگر $M/I/O=1$ باشد با حافظه سروکار دارد.

پایه ۲۹ Wr خروجی از سیستم و 3-State است. برای نوشتن در حافظه یا خروجی استفاده می‌شود که توسط پایه ۲۸ (M/I/O) حافظه یا خروجی مشخص می‌شود.

پایه ۳۱ (HOLD) ورودی به سیستم و پایه ۳۰ (HLDA) خروجی از سیستم و پاسخ به HOLD است. وقتی که واحدی از بیرون بخواهد به حافظه سیستم دسترسی پیدا کند سیگنال HOLD را می فرستد. زمانی که ریزپردازنده HDLA را فعال کند به او اجازه استفاده از باس را می دهد و پایه های خروجی خودش به ۳-State DMA می رود به اینگونه دسترسی به حافظه تکنیک می گویند (Direct Memory Access).

پایه ۳۲ (RD) این پایه برای خواندن از حافظه یا ورودی فعال می شود که باز از روی M/I/O می توان حافظه یا ورودی را انتخاب کرد.

پایه ۳۳ (MN/MX=1) ریزپردازنده ۸۰۸۶ دارای دو مد کاری است اگر باشد ریزپردازنده در مد ماکریم است و کار بعضی از پایه ها عوض می شود مثل پایه های ۲۴ تا ۳۱ که کارهای دیگری در داخل پرانتز ذکر شده است. شرح کار این پایه ها بعداً ذکر خواهد شد.

پایه ۳۴ (BHE₅₇) پایه خروجی و دارای وضعیت ۳-State است. با صفر فعال می شود و بایت بالای دیتای را فعال می کند. عبارتی انتخاب کننده آدرس های فرد حافظه است. بهمراه پایه A₀ آدرس بس زمینه دسترسی به بخش های فرد و زوج حافظه را فراهم می آوردند که بعداً شرح داده خواهد شد. این پایه نیز با S₇ مالتی پلکس شده که شرح پایه های وضعیت سیستم تشریح خواهد شد.

پایه های ۳۵، ۳۶، ۳۷، و ۳۸ که S₆, S₅, S₄, S₃ با خطوط A₁₆, A₁₇, A₁₈, A₁₉ آدرس بس مالتی پلکس شده اند. پس از آنکه ALE غیر فعال شد بیان گر وضعیت سیستم هستند که به شرح زیر است :

S₆ همیشه مقدار صفر خواهد داشت.

S₅ نشان دهنده وضعیت پرچم اینترپریت است.

S₄ و S₃ بیانگر این مطلب هستند که کدام سگمنت در این سیکل بس مورد دسترسی قرار گرفته اند. جدول ۲-۱ این مطلب را نشان می دهد.

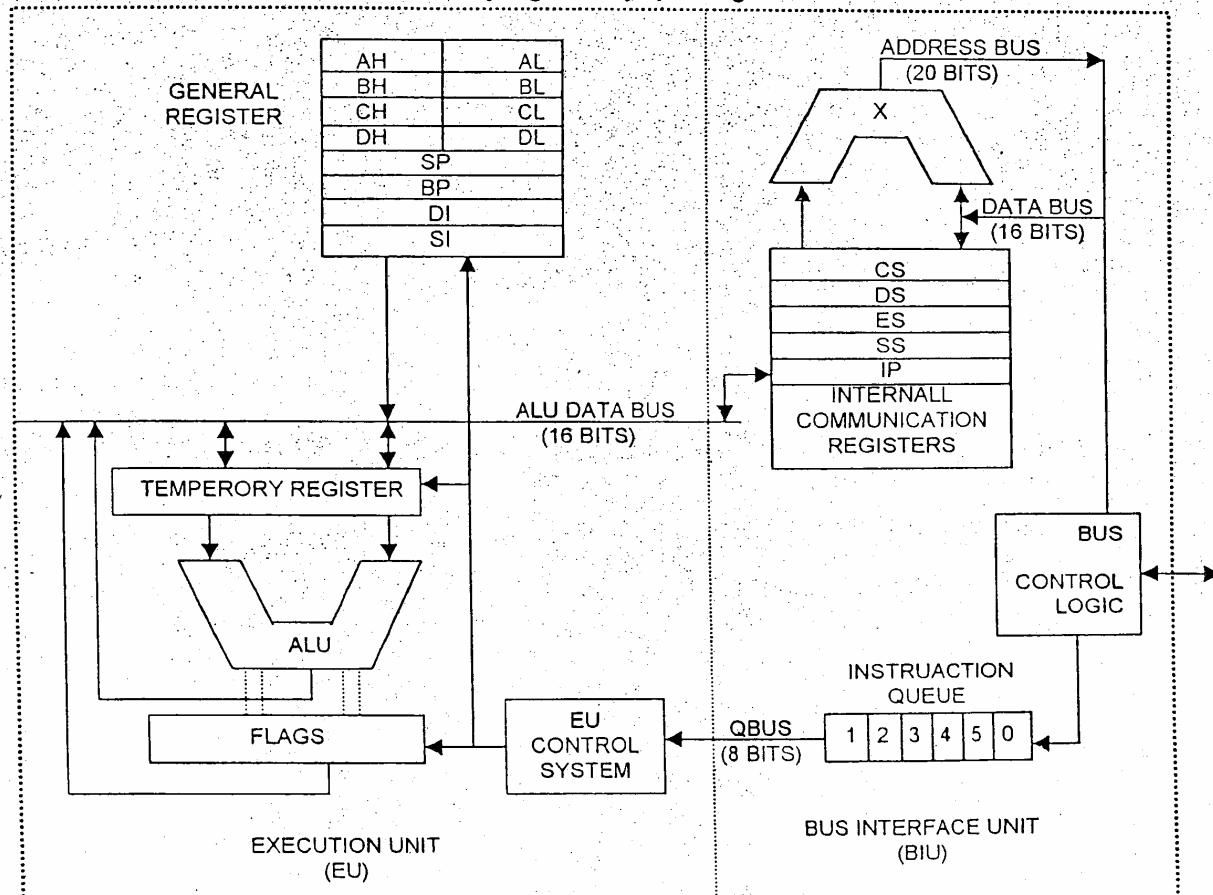
۲-۱-جدول حالات بیت های وضعیت S₃ و S₄

S ₄	S ₃	FUNCTION
0	0	Extra Segment
0	1	Stack Segment
1	0	Code Segment
1	1	Data Segment

لازم به ذکر است که با استفاده از این دو بیت می توان 4MB حافظه به میکروپروسسور ۸۰۸۶ وصل کرد که هر یک مگابایت برای یک سگمنت باشد. سه بیت S₀, S₁, S₂, در مد ماکریم ۸۰۸۶ مورد استفاده قرار می گیرد، که بعداً توضیح داده خواهد شد.

۲-۳- ساختار داخلی CPU :

داخل CPU بصورت شکل زیر است :



۸۰۸۶ را به دو قسمت مستقل از هم به نام های BIU=BusInterfaceUnit و EU=ExecutionUnit تقسیم می کنند. BIU شامل ثبات های سگمنت و IP و صفحه ۶ بايتی دستورالعمل است. EU نیز شامل بخش کنترل، ALU و پرچم های (FLAGS) و ثبات های چهارگانه AX,BX,CX,DX و ثبات های اشاره گر مثل DI,SI,BP,SP می شود.

این تقسیم بندهی صرفاً بمنظور بالا رفتن سرعت میکروپرسسور صورت گرفته است. وظیفه BIU عبارت است از ارسال آدرس روی آدرس بس و خواندن دستورالعمل از حافظه و یا دیتا از حافظه و ورودی و نوشتمن اطلاعات در حافظه و خروجی. عبارت دیگر کلیه ارسال و دریافت اطلاعات به حافظه یا I/O وظیفه BIU است. صفحه دستورالعمل که در این ریزبردازنده پیش بینی شده برای افزایش سرعت است. زیرا همواره ۶ بایت دستورالعمل از پیش Fetch شده و آماده ترجمه و اجرا است. این بخش بصورت FIFO=First In First Out کار می کند. زمانی که بخش EU مشغول ترجمه و تفسیر و اجرای یک دستورالعمل فج شده است بخش BIU به خواندن

فصل دوم

دستورالعمل بعدی و قرار دادن آنها در صفتی پردازد، لذا به غیر از اولین دستورالعمل مابقی زمان خواندن ندارند یعنی صرفه جویی در زمان شده فقط کافی است بخش EU از صفت دستورالعمل بعدی را بردارد. بدیهی است اگر دستوری که ترجمه و تفسیر شد از نوع JUMP باشد باید صفت باک شود و مجدداً دستوری را که دستور جاری تعیین می‌کند CALL Fetch نماید، یعنی این بار زمان خواندن دستور خواهد داشت. اما باز دستورات بعد از اولین دستور در صفت آماده خواهند شد. به این عمل صفت نوعی پردازش خط لوله Pipe Line Processing می‌گویند.

۱-۳-۲- ثبات های ۸۰۸۶

ثبتات های ۸۰۸۶ به سه گروه تقسیم می‌شوند:

- الف) ثبات های گروه دیتا، که یک مجموعه ثبات محاسباتی هستند.
- ب) ثبات های اشاره گر: که شامل ثبات های پایه و شاخص (INDEX) می‌باشند، همچنین شمارنده برنامه و اشاره گر به پشته در این گروه جای دارند.
- ج) ثبات های سگمنت: که یک مجموعه ثبات پایه برای اهداف خاصی هستند، تمامی این ثبات ها ۱۶ بیتی هستند.

ثبتات های گروه دیتا شامل چهار ثبات است بنام های DX, AX, BX, CX این ثبات ها قادرند هم اپرند عملیات را باشند هم می‌توانند حاصل عملیات را ذخیره کنند. و هر کدام از آنها می‌توانند بصورت ۱۶ بیتی یا بصورت دو ثبات ۸ بیتی مورد استفاده قرار بگیرند. بعنوان مثال ثبات AX هم می‌تواند ۱۶ بیت دیتا را در خود ذخیره کند و هم اینکه می‌توان آن را بصورت دو ثبات ۸ بیتی کاملاً مجزا و مستقل از هم مثل AH, AL مورد استفاده قرار داد. زیرا گفته شد این میکرورسسور می‌تواند جانشین ۸۰۸۵ شود و با سیستم های ۸ بیتی بکار رود. بصورت زیر می‌تواند جانشین ۸۰۸۰ یا ۸۰۸۵ باشد.

8086	8080
AL	A
BH	H
BL	L
CH	B
CL	C
DH	D
DL	E

مضافاً اینکه سه ثبات BX, CX, DX علاوه بر سرویسی که در محاسبات می‌دهند می‌توانند کاربرد خاصی مثل آدرس دهنی - شماره نده ای و نقش آدرس دهنی ورودی و خروجی به شرح زیر را بعهده بگیرند:

- ۱ - AX در بعضی موارد بعنوان آکومولاتور مورد استفاده قرار می گیرد، دریافت و ارسال دیتای I/O از طریق این ثبات است.
- ۲ - BX می تواند بعنوان ثبات پایه (BASE REGISTER) در محاسبات آدرس مورد استفاده قرار بگیرد.
- ۳ - CX را می توان بعنوان شمارنده (COUNTER) در دستوراتی که نیاز به شمارش است ب استفاده کرد (مثل عملیات رشته ای).
- ۴ - DX را می توان بعنوان نگهدارنده آدرس I/O در عملیات مورد استفاده قرار داد.
- ثبات های اشاره گر: شامل DI, SI, BP, SP, IP همان شمارنده برنامه و اشاره گر به پشته می باشند. البته برای بدست آوردن آدرس تکمیلی ۲۰ بیتی باید محتوای آنها تحت شرایطی با SS, CS جمع شود که در زیر شرح داده خواهد شد.
- بنحوان ثبات پایه برای دسترسی به پشته و ممکن است با سایر ثبات ها و یا با محتوای میدان BP آدرس (Displacement) که بخشی از دستورالعمل محسوب می شود، بکار برده شود.
- دو ثبات DI, SI دو ثبات شاخص هستند. اگر چه ممکن است آنها به تنهایی مورد استفاده قرار بگیرند، ولی معمولاً آنها با ثبات های BX, BP و محتوای میدان آدرس (Displacement) بکار می روند. بجز IP که استثناست، یک ثبات اشاره گر می تواند حاوی اپرند نیز باشد. در اینصورت باید به تنهایی در دستور قرار بگیرد.
- در شیوه های آدرس دهی خواهید دید که برای داشتن مانور در آدرس دهی، آدرس یک دیتا را ممکن است با جمع کردن ترکیبی از BX یا SI, DI, BP و محتوای میدان آدرس دستورالعمل بدست آورد. حاصل چنین محاسبه ای را آدرس مؤثر می گویند (EA = Effective Address) یا Offset می گویند. در کتب Intel Manuals وقتی درباره زبان ماشین صحبت می کند واژه آدرس مؤثر (Effective Address) بکار برده شده است. وقتی درباره زبان اسمنبلی صحبت می کند واژه Offset بکار می برد. واژه Displacement برای بیان مقداری که به محتوای ثبات ها اضافه می شود تا آدرس مؤثر بدست آید بکار می رود. البته توجه دارید که بحث آدرس دهی مخصوص دسترسی به دیتا است و برای بدست آوردن آدرس فیزیکی (۲۰ بیتی) باید آدرس مؤثر (EA) با یکی از ثبات های سگمنت جمع شود.
- ثبات های سگمنت: عبارتند از ES, DS, SS, CS همانگونه که تاکنون بحث شد سیستم ۱۶ بیتی است، آدرس مؤثر نیز ۱۶ بیتی است ثبات های سگمنت هم ۱۶ بیتی هستند، اما آدرس فیزیکی یا آن مقداری که قرار است روی آدرس باس قرار داده شود ۲۰ بیتی است، ۴ بیتی اضافه از جمع آدرس مؤثر با یکی از ثبات های سگمنت بصورت شکل ۲-۳ بدست می آید. بدین ترتیب که ثبات سگمنت مربوطه چهار بیت به سمت چپ شیفت داده می شود بعد چهار

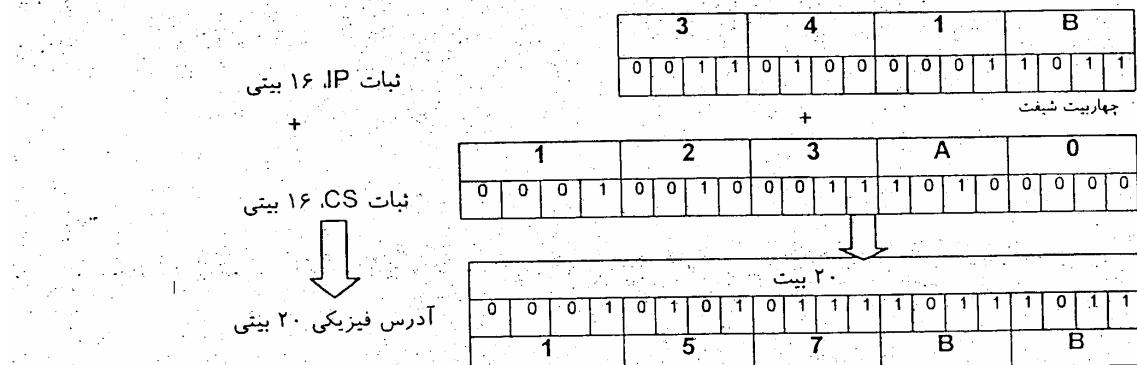
فصل دوم

صفز به آن وارد می شود و با آدرس مؤثر جمع می شود. فرض کنید $(CS=123A)$ باشد و $(IP=341B)$ آدرس فیزیکی برای فیج کردن دستور العمل مورد نظر بصورت زیر است.

341B	EA
123A0	CS

آدرس فیزیکی است

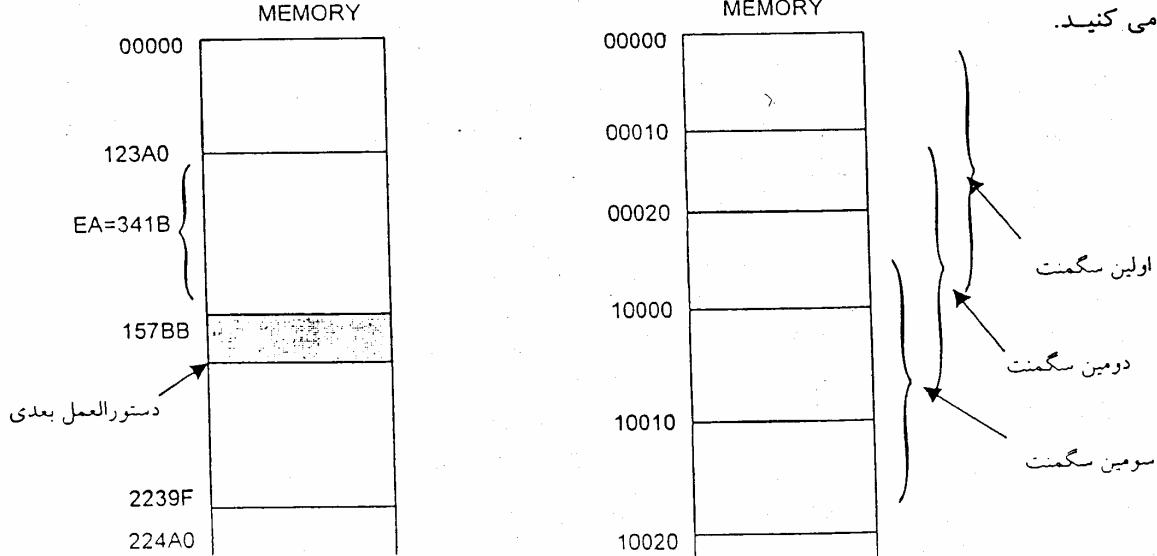
برانتری که دور IP و CS قرار داده شده علامت استانداری است و معنی آن محتوای IP و CS مورد نظر است.



شکل ۲-۳- نحوه شکل دهی آدرس فیزیکی

اساساً بکار گرفتن ثبات های سگمنت باعث می شود که فضای حافظه به قسمت هایی تقسیم شوند که با هم تداخل دارند. هر سگمنت دارای 64 KB طول خواهد داشت که شروع آن از محلی است که محتوای ثبات سگمنت ضریب ۱۶ نشان می دهد. برای مثال نمایش بخش کد حافظه مثال قبلی بصورت زیر می باشد. (شکل ۲-۴-a)

در شکل ۲-۴-b طرز قرار گرفتن سگمنت های مختلف با شرحی که گذشت را مشاهده می کنید.



شکل ۲-۴-a- نمایش یک سگمنت دستور العمل

شکل ۲-۴-b- سگمنت هایی که بخشن منطبق دارند

فصل دوم

سؤالی که اینجا قابل طرح است این است که آیا سگمنت کردن حافظه مزینی هم دارد یا نه؟ پاسخ مثبت است مزایای تقسیم حافظه به بخش های مختلف به شرح زیر است:

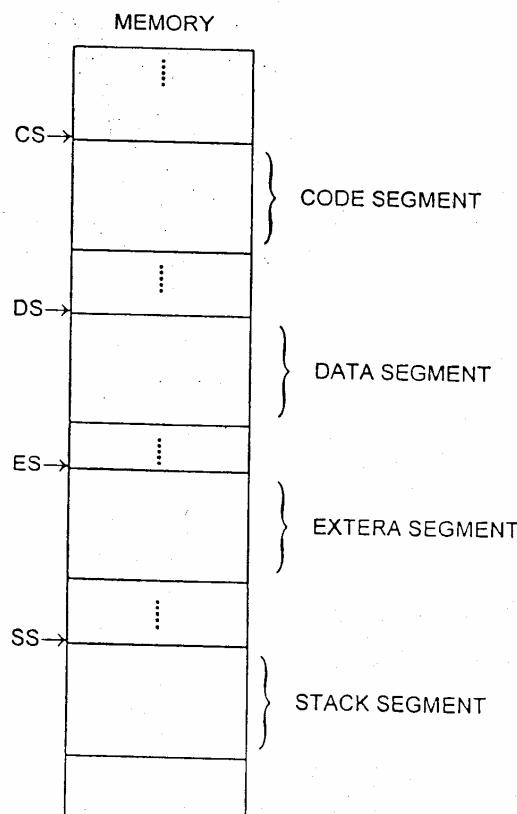
- ۱- اجازه می دهد که یک ریزبردازنده ۱۶ بیتی قادر شود تا یک مگا محل حافظه را آدرس دهدی کند.

۲- اجازه می دهد که بخش های دستورالعمل، دیتا و پشته در یک برنامه بیش از 64KB باشد.

- ۳- تسهیلات بکارگیری نواحی جداگانه ای را برای دستورات و دیتا و پشته یک برنامه فراهم می کند.

- ۴- اجازه می دهد برای هر بار اجرای یک برنامه محل جدیدی را برای ذخیره دیتای آن در نظر بگیرید.

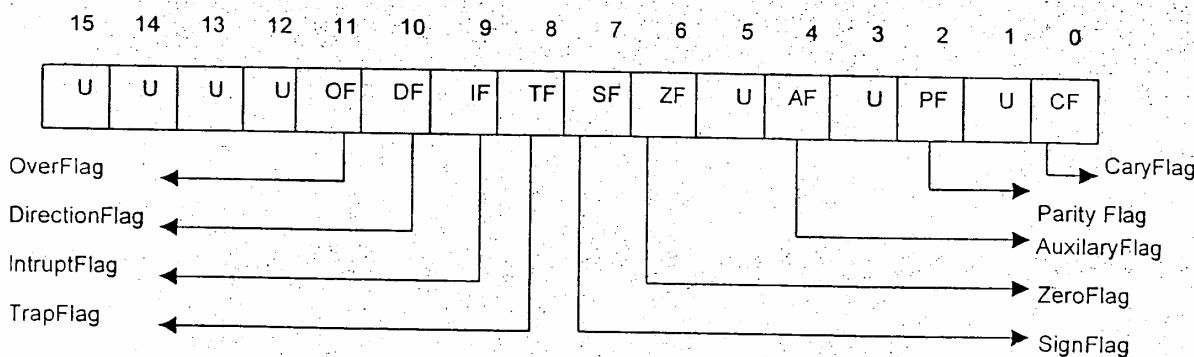
حتی شما می توانید برای یک برنامه ناحیه دستورالعمل و دیتا و پشته را بدون داشتن تداخلی با یکدیگر بصورت کاملاً مجزا انتخاب نمایید. (شکل ۲-۵) این اختصاص خیلی مناسب و ساده است حتی می توانید ناحیه پشته را همیشه در یک محل ثابت انتخاب نمایید، مثل شروع پشته همیشه آدرس 08000 می باشد.



شکل ۲-۵- جدا بودن کد سگمنت، دیتا سگمنت، اکسترا سگمنت و استک سگمنت

۲-۳-۲- ثبات پرچم (FLAGS REGISTER)

ثبات پرچم ها Flag Register، یک پرچم یک فلایپ فلاپ است که نشان دهنده بعضی شرایط ایجاد شده بوسیله اجرای یک دستور یا کنترل کننده یک عمل مطمئن در واحد اجرائی است. ثبات پرچم ۸۰۸۶ یک ثبات ۱۶ بیتی است که ۹ بیت آن فعال است. شکل زیر بیت های فعال را نشان می دهد و بیت های غیر فعال با لا معرفی شده بمعنی Undefined است.



شکل ۲-۶ ثبات پرچم ها

در ۸۰۸۶ پرچم ها به دو دسته تقسیم می شوند، اول پرچم های شرایط (UnditionalFlags) که بیانگر نتیجه عمل قبلی است که در ALU انجام شده است. دومین گروه پرچم های کنترلی (ControlFlags) هستند که کنترل کننده عملیات خاصی هستند.

شش تا از آنها برای نمایش وضع ایجاد شده از اجرای یک دستور است مثل CF بیت اضافه شده به تعداد ۱۶ بیت سیستم در عمل جمع اگر یگ بیت اضافه شود $CF=1$ خواهد شد والا ۰ است. با یک شدن CF سیستم اعلام می کند که بیتی بعنوان با ارزشترین بیت به مقدار اضافه شده است. PF بیت دیگری است که در اثر عملیات فعال می شود. یک می شود، اگر ۸ بیت کم ارزشتر عملیات دارای تعداد زوجی یک باشد در غیر این صورت صفر است. اگر در عملیات جمع در چهار بیت اول Cary ایجاد شود یا در عملیات تفریق Borrow $AF=1$ نیاز شود والا صفر خواهد بود.

اگر حاصل اجرای عمل قبلی صفر شود، در غیر اینصورت صفر خواهد بود. SF برابر است با MSB نتیجه عملیات یعنی اگر یک شود پس $MSB=1$ است و عدد حاصله منفی است، یعنی بصورت متمم ۲ خواهد بود.

اگر حاصل عملیات دجار سرریزی گردد. یعنی نتیجه از رنج مجاز سیستم بیشتر شود. سه پرچم دیگر بیت های کنترلی ریزپردازنده هستند:

جهت حرکت عملیات را در پردازش های رشته ای تعین می کند اگر $DF=0$ باشد اجرای رشته دیتا از آدرس ابتدای المانها شروع و تا آخر ادامه می یابد و اگر $DF=1$ باشد رشته دیتا از آدرس بالائی شروع و اجرا و به اولین آدرس ختم می شود.

IF پرجم فعال ساز اینترابت است. اگر $IF=1$ باشد اجازه ورود اینترابت قابل پوشش (MaskableInterrupt) داده شده است. والا باید از اینترابت چشم پوشی گردد.

TF پرجمی است که اگر $TF=1$ شود بعد از اجرای هر دستور العمل سیستم متوقف می شود یعنی برنامه قدم به قدم اجرا می گردد، لازم به ذکر است که ۸ بیت اول این پرجم ها شبیه ۸ بیت پرجم ۸۰۸۵ و ۸۰۸۰ است.

۴- عملکرد داخل میکروپروسسور

عملکرد عمومی یک کامپیوتر به شرح زیر است:

- ۱- خواندن دستور العمل (Fetch) از حافظه که آدرس آن با توجه به محتوای کد سگمنت و IP محاسبه می شود

$$\text{آدرس فیزیکی } 20 \text{ بیتی} = CS*16+IP$$

- ۲- قراردادن دستور العمل خوانده شده در ثبات دستور العمل و ذکر شدن آن در زمانیکه به محتوای IP یکی اضافه می شود تا آدرس دستور العمل بعدی را تولید کند.
- ۳- اجرای دستور العمل مذکور و در صورتیکه دستور فوق یک انشعاب باشد پر کردن IP با آدرس محلی که قرار است به آنجا منشعب شویم.
- ۴- تکرار قدم یک تاسه

علاوه بر عملیات مذکور ممکن است کارهای دیگری در هنگام اجرای کارهای فوق اجرا شود مثل پرشدن صف ۶ بایتی دستور العمل هر وقت که سیستم باس برای کار دیگری مورد استفاده قرار نگیرد. بدیهی است که زمان لازم برای خواندن دستور العمل بعدی ذخیره می شود. اگر یک دستور انشعاب خوانده شده باشد دستوراتی که در صف قرار داده اید مفید نخواهد بود باید دستور جدیدی که در این انشعاب تعریف شده خوانده شود، بدون اینکه در زمان صرفه جویی شود البته اینگونه دستورات خیلی کم است.

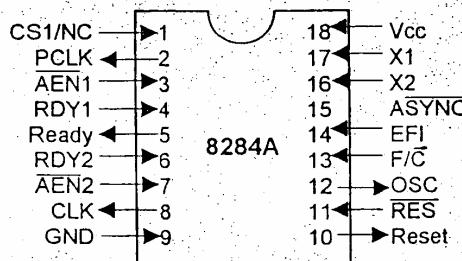
۵- مولد سیگنال کلاک برای ۸۰۸۶/۸۰۸۸

در این قسمت به معروفی IC مولد کلاک (8284A)، سیگنال RESET و تأمین کننده سیگنال Ready پرداخته می شود.

IC شماره 8284 یک المان مناسبی برای میکروپروسسورهای 8086/8088 می باشد، بدون این مولد کلاک، به مدارات اضافی زیادی نیاز خواهد داشت، لذا مناسب ترین المان IC مذکور است که سیگنالهای زیر را تولید می کند:

تولید کلاک، ایجاد همزمانی برای سیگنال RESET و ایجاد همزمانی برای ورود سیگنال Ready

بعده این IC است. این IC دارای ۱۸ پایه بشکل و شرح زیر می باشد:



شکل ۲-۲-۱C-تولیدکننده پالس ساعت

دو پایه ۹ و ۱۸ که مشخص است، Vcc و GND خواهد بود و با ولتاژ +5V با تolerانس $\pm 10\%$ کار می کند.

دو پایه ۱۷، ۱۶ بنام های X1، X2 به یک کریستالی که تأمین کننده کلاک سیستم است وصل می شود، در اینصورت باید پایه شماره ۱۳ که (Frequency/Crystal) نام دارد به زمین وصل شود.

نحوه دیگر که می توان کلاک سیستم را تأمین کرد، استفاده از یک کلاک خارجی آماده است که باید به ۱۴ وصل شود (External Frequency Input) در اینصورت پایه ۱۳ باید به Vcc وصل شود. اگر از کریستال داخلی استفاده شود باید EFI نیز گراند شود.

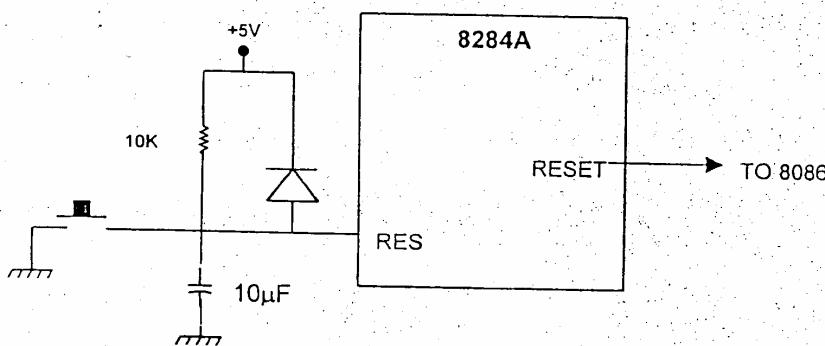
پایه شماره ۸ کلاک ساخته شده توسط 8284A برای میکروپروسسور سایر IC های سیستم است، این CLK یک خروجی دارد که ۱/۳ فرکانس کریستال یا فرکانس ورودی EFI می باشد. دارای زمان دوام پالس (Duty cycle) 33% می باشد.

پایه شماره ۲، PCLOCK: این پایه کلاک جانبی (Peripheral Clock) نام دارد، یک سیگنال ارائه می دهد که فرکانس آن ۱/۶ فرکانس کریستال یا فرکانس ورودی از پایه EFI می باشد. و زمان دوام آن (Duty Cycle) 50% است. این سیگنال بعنوان کلاک برای دستگاههای جانبی سیستم مورد استفاده قرار می گیرد.

پایه شماره ۱۲، OSC نام دارد (Oscillator Output). این پایه سیگنالی ارائه می دهد که دارای سطح ولتاژ TTL و فرکانس برابر فرکانس کریستال یا برابر فرکانس ورودی EFI می باشد. هدف از این سیگنال تهیه سیگنالی برای ورودی به EFI، IC های 8284A دیگری که برای استفاده از چند پردازشگر مورد نیاز می باشد.

پایه شماره یک CSYNC نام دارد. همزمان کننده کلک است (Clock Synchronization) در سیستمی که چند پردازشگر مورد نیاز است و فرکانس CLK از EFI استفاده می شود، این پایه برای همزمانی کلکهای سیستمها بکار می رود. اما اگر از کریستال استفاده می کنید باید به GND وصل شود.

پایه شماره ۱۱ (RESET)، ورودی است که از دکمه فشاری سیستم یک زمین دریافت می کند، معمولاً از یک مدار RC مطابق شکل ذیر استفاده می شود:

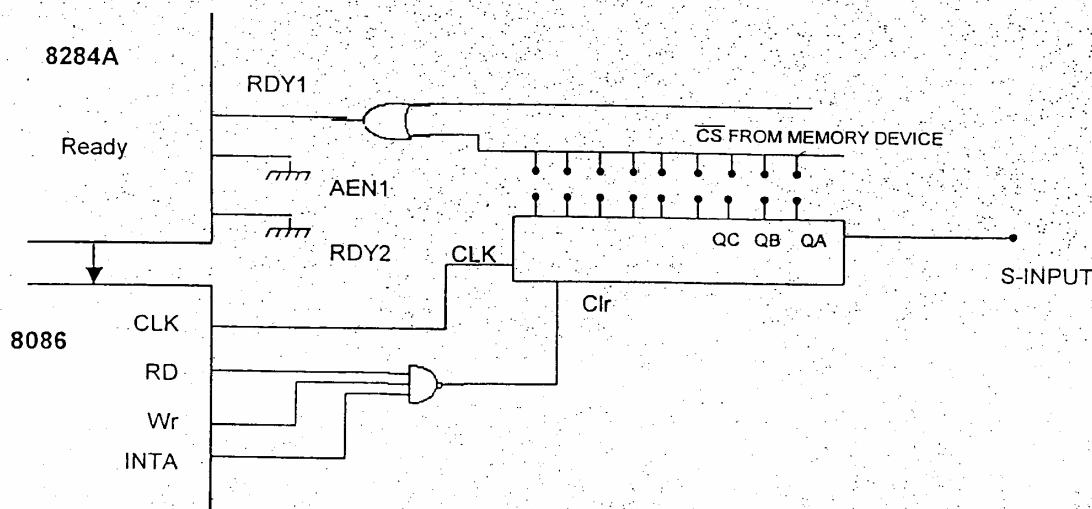


شکل ۲-۸- طریقه وصل کردن دکمه RESET به IC شماره 8284

۲-۵-۱- سیگنال Ready و Wait-State

بطوریکه قبلًا گفته شد پایه Ready در میکرورسسور سبب می شود که سرعت میکرو بخارت حافظه ها یا I/O ها کنتر شود. برای تحقق این امر یک یا چند پالس ساعت TW (Wait-State) بین T4, T3 اعمال می شود. اگر یک TW اعمال شود زمان دسترسی به حافظه که بطور معمول 460nSec است با فرکانس کاری میکرورسسور که MHz ۵ فرض می شود به 660nSec افزایش یافته است. پایه ورودی Ready در لبه پایین رونده T2 هر سیکل ماشین نمونه برداری می شود. اگر صفر منطقی بود یک TW ساخته می شود و تولید T4 به تأخیر می افتد و یک TW جایگزین آن می شود، مجدداً در اواسط TW عمل نمونه برداری تکرار می شود و هر وقت Ready دارای سطوح یک منطقی شد تولید TW متوقف و T4 تولید می شود. توسط 8284A IC دو دستگاه (مثل حافظه و I/O) می توانند تقاضای Wait کنند. پایه های 3,7 (AddressEnable) AEN2, AEN1 و 6,4 (AEN1, AEN2) این IC یک سیگنال Ready مناسب برای ۸۰۸۶ تهیه می کنند. آمدن RDY مربوطه در پایه ۵ این IC یک سیگنال Ready با دامنه ۵ ولت برای میکرورسسور ارسال می نمایند که به پایه Ready میکرورسسور وصل می شود، RDY1 و RDY2 از طرف دستگاه جانبی ای که نیاز به TW دارد ارسال می شوند (شکل ۲-۹).

مدار زیر مولد مناسبی برای Ready می باشد.

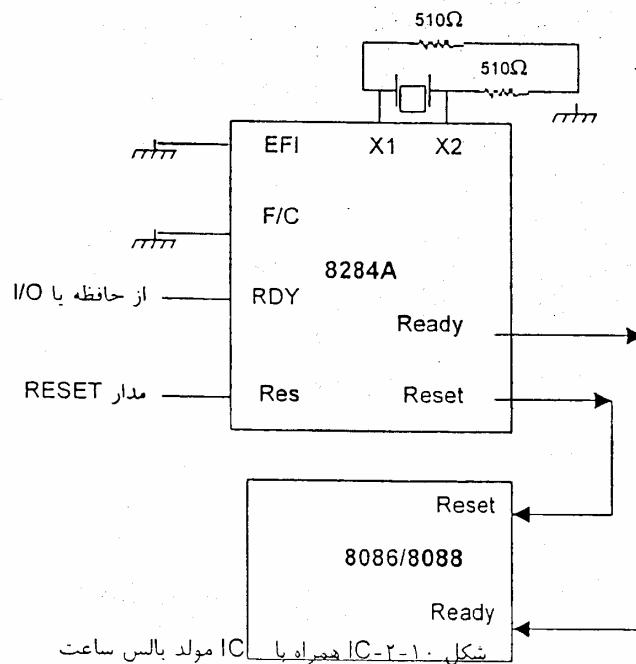


شکل ۲-۹- تولید سیگنال Wait

در مدار بالا در سه موقع نیاز به TW داریم، خواندن - نوشتن و پاسخ به اینترپت لذا ابتدا شیفت رجیستر مربوطه (IC) شماره 74LS164 می شود با CLK پاک (Clr) بعدی یک بداخل آن شیفت پیدامی کند و در QA قرار می گیرد پس از یک کلاک به QB میرسد و یک به ورودی 8284A RDY1 می فرستد. یعنی یک TW بین T₃ و T₄ اعمال می کند.

لازم به ذکر است که میکرورپرسسور 8086 دارای انواع مختلفی است که رنج CLK آنها از 5MHz تا 10MHz می تواند تغییر کند. شکل ۲-۱۰ میکرورپرسور را همراه با IC مولد

نشان می دهد.

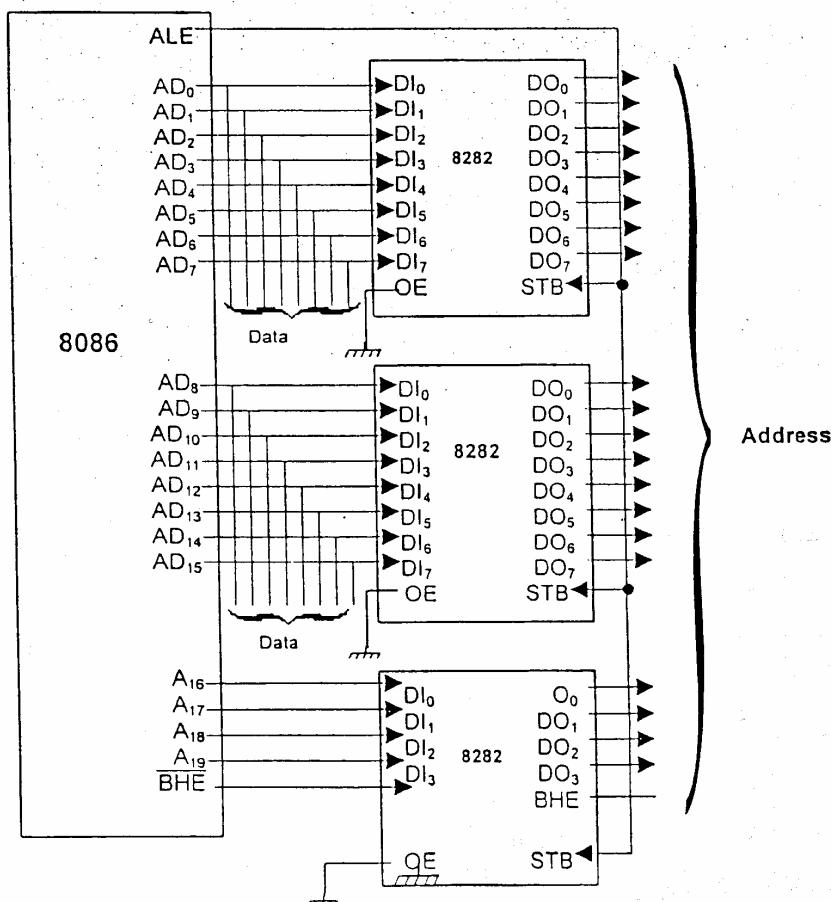


شکل ۲-۱۰- همراه با IC مولد بالس ساعت

۶-۲- جداسازی آدرس بآس از دیتا بآس

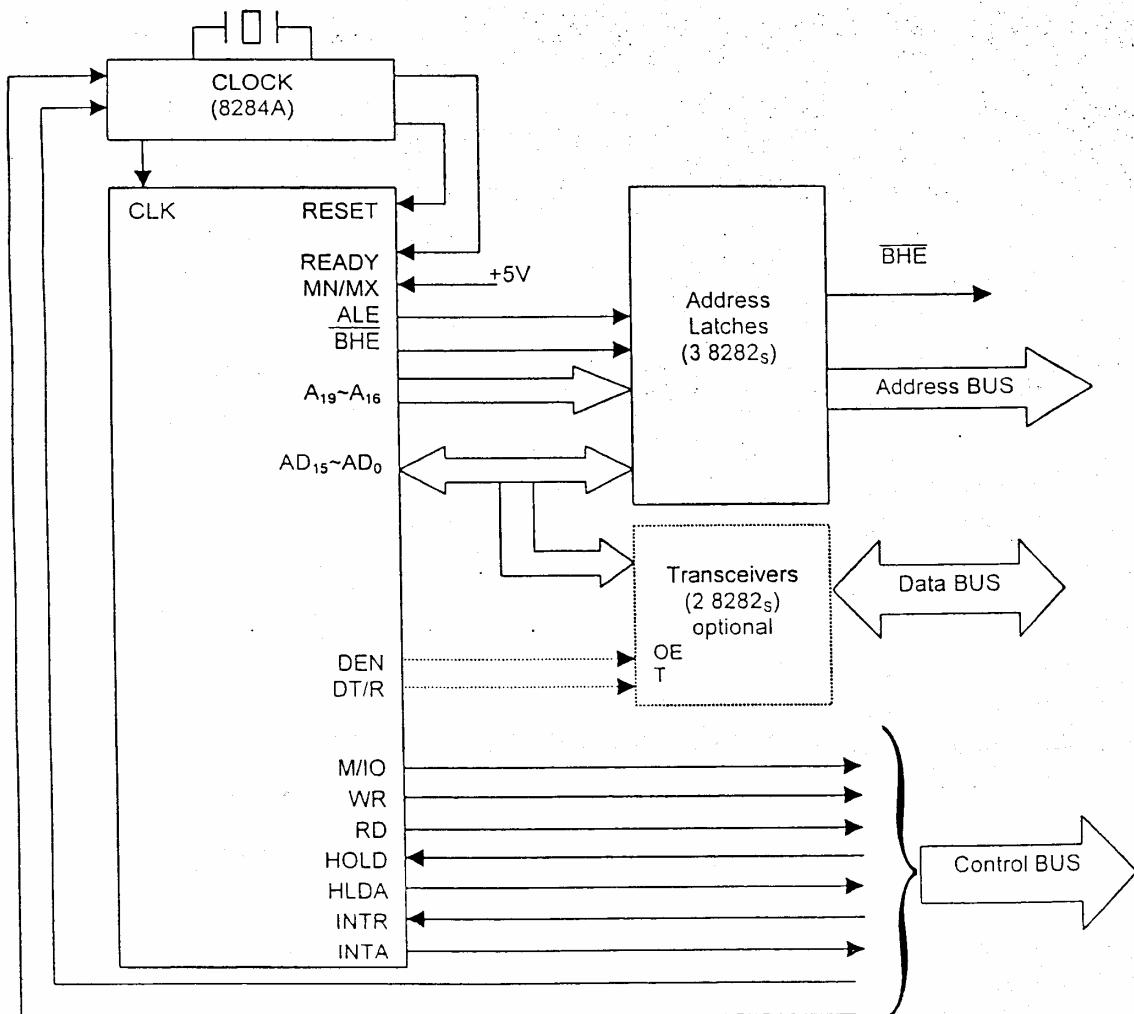
عملت جداسازی آدرس از دیتا اینست که هر دو روی بآس مشترک ارسال می شوند، برای اینکار از IC شمار 8282 بنام AddressLatch استفاده می شود. البته می توانید از IC شماره 74LS373 نیز استفاده نمایید هر کدام دارای ۸ عدد Latch می باشند. در واقع می توان گفت ۸ ورودی ۸ خروجی و یک پایه STB که بازکننده ورودی Latch برای دریافت اطلاعات هستند و یک پایه هم OutputEnable که نامیده می شود بعلاوه GND, Vcc از پایه های دیگر این IC است. خطوط AD₀ و AD₇ به IC اول و AD₈ تا AD₁₅ به IC دوم و AD₁₆ تا AD₁₉ به IC سوم داده می شود.

میکرورپرسور در اولین پالس هر عملیاتی آدرس را روی AD₀~AD₁₉ قرار می دهد بعد از آن ALE (پایه ۲۵) را فعال می کند. پایه ۲۵ را به STB هر سه IC لج مذکور وصل می کنیم تا AD₀~AD₁₉ در لج ها ذخیره شود خروجی آنها A₀~A₁₉ است. مطابق شکل ۲-۱۱ :



شکل ۲-۱۱- میکرورپرسور به همراه IC های Latch

OE مربوط به آهای 8282 را همواره فعال می‌کنیم تا آدرس همیشه در خروجی آنها یعنی روی آدرس بآس باشد. البته اگر قرار باشد ریزپردازنده با ریزپردازنده‌های دیگر کار کند و بخواهیم از تکنیک DMA استفاده کنیم نباید همیشه OE را به زمین وصل کرد. اگر به زمین وصل نباشد یعنی Vcc داشته باشد خروجی بافرها به امپدانس بالا می‌رود (Hz). در مسیر دیتا بآس نیز اگر تعداد دستگاه زیاد نباشد نیاز به بافر کردن دیتا بآس نیست اما اگر قرار باشد سیستم تعدادی زیادی IC حافظه یا ورودی و خروجی داشته باشد یا نیاز به تکنیک DMA باشد (با چند پردازنده کار کند) باید سر راه دیتا بآس نیز بافر گذاشته شود. اینتل IC 8286 را معرفی می‌کند که دو عدد IC مورد نیاز است. AD₀-AD₁₅ به ورودی آنها وصل می‌شود. این IC نیز دو پایه کنترلی دارد، OE و پایه T که OE را به پایه ۲۶ یعنی DEN وصل می‌شود و پایه T به پایه ۲۷ یعنی DT/R وصل می‌شود مطابق شکل ۲-۱۲:

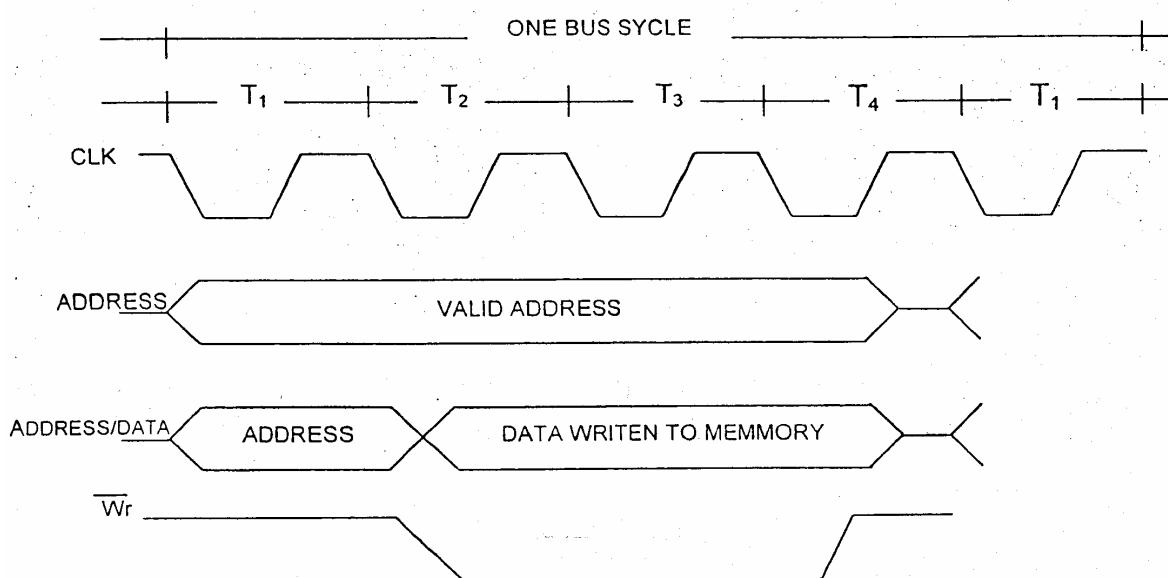


شکل ۲-۱۲-بلوک دیاگرام مینیمم سیستم ۸۰۸۶ در مد مینیمم

۲-۷- زمان بندی باس های (BUS Timing) ۸۰۸۶

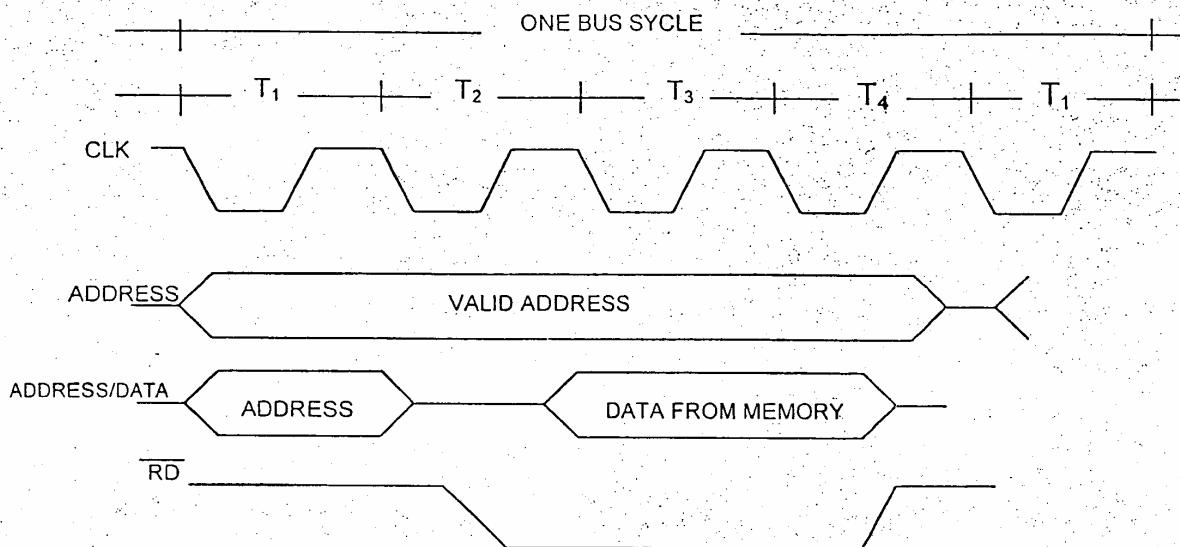
قبل از اینکه حافظه ای یا ورودی و خروجی برای ۸۰۸۸ یا ۸۰۸۶ انتخاب کنید باید از وضعیت زمان بندی باس آن مطلع باشید. در این قسمت نگاهی به زمان بندی باس هنگام دو عمل خواندن و نوشتن خواهیم داشت.

سه باس دیتا، آدرس و کنترل ۸۰۸۶ دقیقاً مثل باس های سایر میکروپرسسورها عمل می کند. برای نوشتن دیتا در حافظه به فرم ساده زمان بندی شکل ۲-۱۳ نگاه کنید. میکروپرسسور آدرس را روی آدرس باس قرار می دهد، سپس دیتائی که قرار است در حافظه نوشته شود روی دیتا باس قرار می دهد. سیگنال کنترلی \overline{Wr} را صادر می کند. (در ۸۰۸۶ خط $M/\overline{IO}=1$ در ۸۰۸۸ $\overline{IO/M}=0$ می شود)



شکل ۲-۱۳ سیکل باس برای نوشتن در حافظه بطور خلاصه

اگر قرار است دیتا از حافظه خوانده شود شکل ۲-۱۴ را ملاحظه فرمائید. میکروپرسسور آدرس را روی خطوط آدرس قرار می دهد، یک سیگنال RD صادر می کند، سپس دیتای موجود در دیتا باس که از حافظه خارج شده است را قبول می کند.



شکل ۱۴-۲-سیکل باین برای خواندن از حافظه بطور خلاصه

۱-۷-۲- زمان بندی عمومی

۸۰۸۶ حافظه و I/O را بصورت پریودیک بکار می گیرد که این زمان سیکل باس (BUS CYCLE) نامیده می شود، که برابر ۴ پالس ساعت است. اگر فرکانس پالس ساعت ۵ مگاهرتز باشد، زمان تکمیل عملیات خواندن و نوشتن این میکرورپرسور ۸۰۰ نانوثانیه می شود. یعنی میکرورپرسور در هر ثانیه ۱/۵ میلیون دفعه بین I/O و یا حافظه می خواند و می نویسد. صفحه داخل میکرورپرسور می تواند ۲/۵ میلیون دستور العمل را در ثانیه اجرا نماید. (با این واحد سرعت میکرورپرسورها مقایسه می شود MIPS) البته انواع دیگر این میکرورپرسور با سرعتی بالاتر از این مقدار عمل می کنند. در اولین پالس ساعت هر سیکل که T_1 نام دارد آدرس روی خطوط آدرس یا عبارت دیگر روی خطوط دیتا و آدرس مالتی پلکس شده ارسال می گردد.

در پایان حالت T_1 سیگنالهای کنترلی ALE، M/I/O، DT/R، \overline{RD} صادر می شوند. در زمان دومین پالس (T_2) میکرورپرسورها کنترلی \overline{RD} یا \overline{DEN} یا \overline{Wr} را صادر می کند. اگر سیکل نوشتن است دیتائی که قرار است در حافظه یا I/O نوشته شود روی خطوط دیتا باس قرار داده می شود. سیگنال کنترلی \overline{DEN} بافر دیتا را باز می کند و حافظه یا I/O دیتای ارسال شده را دریافت می کند.

در خلال T_2 حافظه و I/O به اندازه کافی وقت دارند که دیتای روی دیتا باس را دریافت کنند. یک اتفاق مهمی که در خلال T_2 رخ می دهد سیگنال Ready نمونه برداری می شود اگر

فصل دوم

دارای ولتاژ یک منطقی باشد بعد از T_3 ، سیگنال T_4 صادر می شود اما اگر Low باشد پالس ساعت بعدی TW خواهد بود.

در T_4 تمام سیگنالها غیر فعال می شوند و آماده تدارک سیکل بعدی می شوند، همچنین زمان نمونه برداری از اتصال دیتا باس توسط میکرورسسور است که دیتای خوانده شده از حافظه یا I/O را بردارد. بعلاوه در این لحظه سیگنال Wr که دارای یک لبه بالا رونده است (زیرا به یک منطقی برمی گردد) دیتای ارسالی میکرو را به حافظه یا I/O منتقل می کند.

با توجه به این زمانها شما باید برای میکرورسسور حافظه ای انتخاب کنید که از نظر زمان دسترسی با میکرورسسور مشکلی نداشته باشد. در واقع بعد از T_1 تا پایان T_4 زمان مناسبی برای دسترسی به حافظه می تواند باشد تقریباً 600 نانوثانیه البته 110 نانوثانیه بعد از T_1 زمان لازم است تا آدرس ظاهر شود که باید از 600 نانوثانیه کم شود. مضافاً اینکه 30 نانوثانیه قبل از اتمام T_4 باید دیتا در خروجی باشد، یعنی زمان دسترسی به حافظه $= 460 - 30 = 430$ نانوثانیه می شود.

سؤالات دوره ای فصل دوم

- ۱- میکرورسسور ۸۰۸۶ دارای چند خط آدرس و چند خط دیتا است؟
- ۲- میکرورسسور کمکی ۸۰۸۶ کدام است؟
- ۳- میکرورسسور ۸۰۸۶ چه مقدار حافظه را آدرس دهی می کند؟
- ۴- میکرورسسور ۸۰۸۶ دارای چند خط کنترلی است؟
- ۵- سگمنت رجیسترهاي ۸۰۸۶ چه نام دارند؟
- ۶- وقتی که میکرورسسور ON POWER و یا RESET می شود از چه آدرسی شروع به اجرای برنامه می کند؟ چرا؟
- ۷- چیپ ۸۰۸۶ چند پایه دارد؟
- ۸- پایه READY به چه منظوری تدارک دیده شده است؟
- ۹- ALE چه عملی انجام می دهد؟
- ۱۰- سیگنالهای وضعیت S_4, S_5 چه اطلاعاتی در اختیار می گذارند؟
- ۱۱- با استفاده از سیگنالهای وضعیت S_4, S_5 چگونه می توان 4 مگابایت حافظه به میکرورسسور وصل کرد؟
- ۱۲- BIU چه وظایفی بعده دارد؟
- ۱۳- پیش واکشی (Prefetch) در میکرورسسور ۸۰۸۶ یعنی چه؟
- ۱۴- سه گروه ثبات های ۸۰۸۶ را، شرح دهید؟

- ۱۵- مزایای سگمنت بندی حافظه چیست؟
- ۱۶- آیا می توان ۴ سگمنت حافظه در فضای KB 64 باشد؟
- ۱۷- چند پرچم برای بیان وضعیت سیستم در میکرورپرسسور ۸۰۸۶ وجود دارد نام ببرید؟
- ۱۸- سه پرچمی که بیت های کنترلی ریزپردازنده هستند جه نام دارند؟ نام ببرید.
- ۱۹- سه وظیفه IC شماره 8284 که مولد پالس ساعت سیستم است کدامند؟
- ۲۰- تکنیک DMA یعنی چه؟ و چه پایه هایی از ۸۰۸۶ در این تکنیک نقش دارند؟

فصل سوم

«واسط حافظه و ورودی/خروجی

هر سیستم کامپیوترا اعم از اینکه ساده یا پیچیده باشد نیاز به حافظه دارد. ۸۰۸۶ نیز از این نیاز متنشی نیست. برای طراحی یک مینیمم سیستم معمولاً نوع حافظه بعنوان حافظه اصلی مورد استفاده قرار می‌گیرند، حافظه فقط با قابلیت خواندن (ROM) و حافظه با قابلیت خواندن و نوشتن (R/W MEM) که اصطلاحاً به آن حافظه با دسترسی تصادفی (RAM) گفته می‌شود. در این فصل چگونگی متصل کردن حافظه‌های مذکور به ریزپردازنده مورد بررسی قرار می‌گیرد.

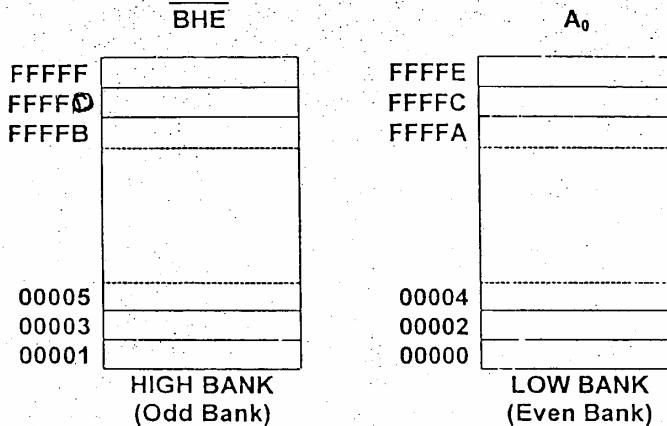
البته در اینجا فرض مابرا این است که خواننده محترم (دانشجویان عزیز) با انواع حافظه‌های ROM و DRAM و SRAM آشنائی کامل دارند.

۱-۳-۴- کودکردن آدرس :

برای دسترسی به یک محل معین یک حافظه توسط ریزپردازنده لازم است که آدرس میکرو دکود شود تا برای یک محل یک عدد معینی اختصاص داده شود؛ اگر از روش دکود کردن آدرس استفاده نشود، فقط یک IC حافظه می‌توان به ریزپردازنده وصل کرد. که استفاده از تمام محل‌های قابل آدرس دهی توسط ریزپردازنده غیر ممکن می‌شود.

قبل از اینکه به اتصال حافظه به ریزپردازنده پرداخته شود لازم است که ذکر گردد اتصال حافظه به ریزپردازنده ۸۰۸۶ متفاوت از وصل کردن حافظه به ریزپردازنده ۸۰۸۸ است زیرا این دو ریزپردازنده در سه مطلب با هم اختلاف دارند، اول اینکه دیتا باس، ۸۰۸۶ شانزده بیتی است در حالیکه دیتا باس ۸۰۸۸ هشت بیتی است، دوم اینکه پایه $\overline{M/I/O}$ که در ۸۰۸۶ بحث شد در ۸۰۸۸ بصورت $\overline{I/O/M}$ می‌باشد و سوم اینکه پایه BHE که در ۸۰۸۶ وجود ندارد در ۸۰۸۸ چنین پایه وجود ندارد.

چون ریزپردازنده ۸۰۸۶ دارای ۱۶ خط دیتاست و اینکه هم قادر است بصورت ۸ بیتی کار کند و هم بصورت ۱۶ بیتی لذا باید حافظه آن به دو بانک فرد و زوج و یا بانک بالا (HIGH) و بانک پایین (LOW) تقسیم شود. مطابق شکل ۳-۱



شکل ۳-۱- نمایش بانک بالا(فرد) و بانک پایین (زوج) در ریزپردازنده ۸۰۸۶

ریزپردازنده با استفاده از پایه A_0 و BHE مطابق جدول ۳-۱ قادر است به هر یک از بانکها به تهایی (وقتی که ۸ بیتی کار می کند) و یا به هر دو بانک (وقتی که بصورت ۱۶ بیتی کار می کند) دسترسی پیدا کند.

جدول ۳-۱ : جدول صحت A_0 و BHE

BHE	A_0	حالت دسترسی
0	0	هر دو بانک فعال است
0	1	بانک بالا(فرد) فعال است
1	0	بانک پایین (زوج) فعال است
1	1	هیچکدام فعال نیستند

ملاحظه می فرمائید که این ریزپردازنده $1M \times 2^20$ بایت محل حافظه را آدرس دهی می کند بعبارت دیگر $512K \times 16$ مقدار حافظه را آدرس دهی می کند.

به دو روش می توان بانکها را انتخاب کرد :

۱- یک سیگنال Wr جداگانه برای هر بانک اختصاص داد.

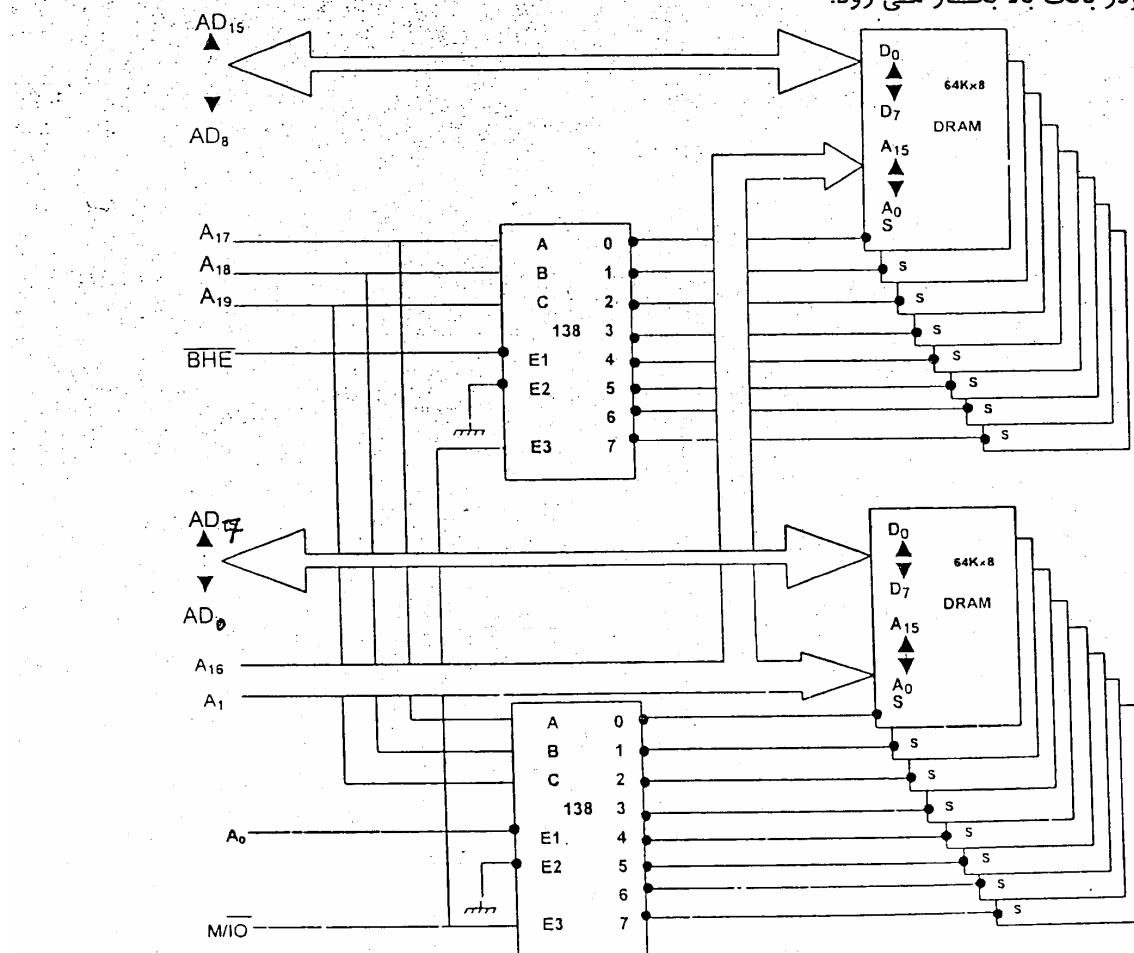
۲- یک دکودر جداگانه برای هر بانک اختصاص داد.

روش دوم توصیه می شود. زیرا مطمئن ترین راه برای آدرس دهی است و قابل فهم تر نیز می باشد. لذا با استفاده از دو دکودر 74LS138 دو بانک به ظرفیت $1M \times 8$ به ریزپردازنده مطابق

شکل ۳-۲ متعلق می کنیم.

فصل سوم

هر بانک ۸ ۶۴Kx8 دارد که هر IC نیاز به ۱۶ خط آدرس و ۸ خط دیتا دارد. زیرا $2^{16}=64K$ از A_{16} تا A_{19} شانزده خط آدرس است که وارد هر IC می‌شود هم در بانک بالا و هم در بانک پایین و سه خط وارد دکودرهای بانکها شده که هر کدام از دکودرهای خروجی ۸ دارند که ۸ IC را انتخاب می‌کنند. A_0 برای انتخاب دکودر بانک پایین و \overline{BHE} برای انتخاب دکودر بانک بالا بکار می‌رود.

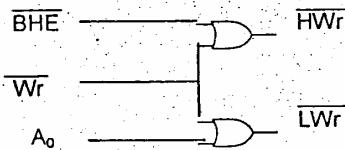


شکل ۳-۲- متصل کردن یک مگابایت حافظه به ریزپردازنده ۸۰۸۶

توجه داشته باشید که A_0 ریزپردازنده داخل IC های حافظه نمی‌رود بلکه به E دکودر وصل می‌شود، پس باید A_1 ریزپردازنده، به A_0 IC های حافظه و A_2 به A_1 و الی آخر وصل نمود. روش اول مؤثرتر از روش قبلی است که تشریح شد. این تکنیک فقط به یک دکودر نیاز دارد و یک محل ۱۶ بیتی را انتخاب می‌کند، روشی است که از نظر اقتصادی بصرفه و تعداد المان کمتری بکار می‌گیرد.

فصل سوم

شکل ۳-۳ چگونگی تولید دو سیگنال کنترلی خواندن را نشان می‌دهد که با استفاده از دو گیت منطقی OR با استفاده از IC 74LS32 طرح شده است. که یکی برای انتخاب بانک بالا و دیگری برای انتخاب بانک پایین کاربرد دارد.



شکل ۳-۳ تولید در سیگنال کنترلی خواندن برای بانک بالا و پایین

حال با ذکر یک مثال بحث را دنبال می‌کنیم. فرض کنید بخواهیم به یک ریزپردازنده ۸۰۸۶ حافظه با قابلیت فقط خواندن (ROM) به میزان $32K \times 16$ از آدرس F0000 تا FFFFF که از IC های 2764 که هر کدام $8K \times 8$ حافظه EPROM هستند وصل نمائیم، بعلاوه $8K \times 16$ حافظه از نوع RAM که از آدرس 00000H تا 03FFFFH را اشغال نماید با استفاده از IC های 4016 که هر کدام $2K \times 8$ هستند وصل نمائیم مطابقت رسم بلوك دیاگرام مدار در صورتیکه از تکنیک یک سیگنال کنترلی خواندن برای هر بانک استفاده کنیم.

شکل ۳-۴ بلوك دیاگرام رسم شده مدار را نشان می‌دهد. بانک بالای RAM تشکیل شده از ۴ عدد IC 4016 که $2K \times 8$ هستند مجموعاً $8K \times 8$ می‌شود عیناً به همین ترتیب $8K \times 8$ هم بانک پایین حافظه RAM می‌باشد. IC های $2K \times 8$ یازده خط آدرس دارد از A₀ تا A₁₀ که به A₁ تا A₁₁ خطوط ریزپردازنده وصل می‌شوند همزمان بانک بالا و پایین را با دکودر 74LS139 IC، 2×4 فعال می‌کنیم خطوط A₁₂ و A₁₃ میکرو وارد دکودر می‌شود چهار خروجی دکودر برای حافظه مذکور زمانی فعال می‌شود که بقیه خطوط آدرس (A₁₉, A₁₈, A₁₇, A₁₆, A₁₅, A₁₄) که همگی در صفر هستند با NOT کردن آنها و NAND کردن آنها با سیگنال M/IO تولید یک صفر منطقی می‌کند تا دکودر مذکور فعال شود، لذا از آدرس 00000H~03FFFFH (یعنی تمامی خطوط A₀ تا A₁₃ یک می‌شوند) حافظه RAM مورد نظر تکمیل می‌شود.

حال به بخش ROM نظر افکنید ۴ عدد 2764 برای بانک بالا (هر کدام $8K \times 8$) و ۴ عدد از همان IC برای بانک پایین انتخاب شده است، یعنی $32K \times 8$ بانک بالا و $32K \times 8$ بانک پایین که با هم تشکیل حافظه $32K \times 16$ را می‌دهند، پایه های آدرس هر IC از A₀ تا A₁₂ می‌باشد که به پایه های A₁ تا A₁₃ ریزپردازنده وصل می‌شوند، A₁₄ و A₁₅ به یک دکودر دیگر 74LS139 وصل می‌شوند تا IC های $8K \times 8$ را فعال کند. چون قرار است این حافظه ها از آدرس F0000 شروع شوند پس چهار خط باقیمانده آدرس یعنی A₁₆, A₁₇, A₁₈, A₁₉ باید یک باشند تا با M/IO بتوانند دکودر مربوط به ROM را فعال کنند.

به این ترتیب حافظه ها همانگونه که صورت مسئله خواسته بود در آدرس‌های مورد نظر قرار گرفته اند ولی هر وقت آدرس دهی شوند یک محل ۱۶ بیتی فعال می‌شود. ممکن است نیاز باشد ما حافظه را بصورت ۱۶ بیتی مورد استفاده قرار دهیم که در این حالت باید هم A₀ و هم BHE فعال شوند یعنی LWr، HWr هر دو فعال شوند، ممکن هم است که حافظه بصورت ۸ بیتی مورد استفاده قرار بگیرد که این مشکل با تهیه HWr و LWr حل می‌شود. هر کدام فعال شوند در آن بانک مربوطه نوشته می‌شود.

سوالی که ممکن است به ذهن برسد اینکه هنگام خواندن خافظه چه باید کرد؟ هنگام خواندن مشکل ندارید زیرا ریزپردازنده در هر سیکل باس فقط بایت مورد نظر را از روی دیتا باس برمی‌دارد. حتی اگر ۱۶ بیت دیتا روی دیتا باس باشد بایت مورد نظر خوانده می‌شود. مگر اینکه هدف ریزپردازنده خواندن دیتای ۱۶ بیتی باشد.

نکته دیگری که باید توجه کرد سیگنال فعال کننده دکودر EPROM به منزله سیگنال RDY به ریزپردازنده ارسال می‌شود. زیرا معمولاً EPROM به یک TW نیاز دارد تا اطلاعات در خروجی اش ظاهر شود شکل ۳-۴ کلیه موارد صحبت شده را بوضوح نشان می‌دهد.

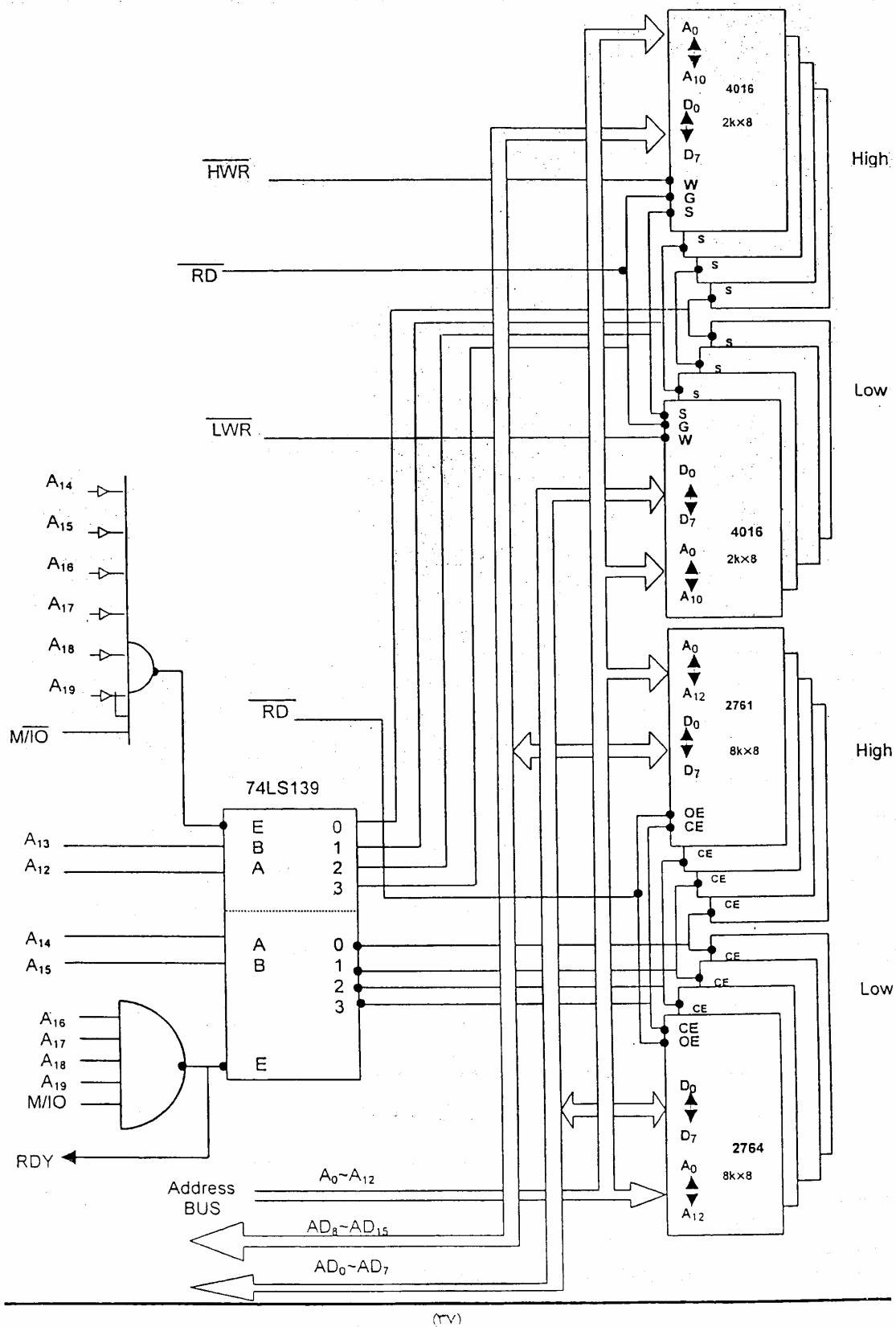
۳-۲- ارتباط یک ریزپردازنده با دنیای بیرون

یک ریزپردازنده ساخته شده تا بوسیله اطلاعاتی که دریافت می‌کند مسائلی را حل نماید بدیهی است باید قادر باشد با بیرون ارتباط برقرار کند. در این بخش روش‌های اصلی ارتباط موازی و سری بین انسان و ریزپردازنده معرفی می‌شوند. ابتدا واسطه‌های اصلی I/O معرفی می‌شوند و درباره دکود کردن آدرس‌های ورودی و خروجی بحث می‌شود سپس راجع به جزئیات ارسال و دریافت بصورت موازی و سری صحبت خواهد شد.

در ابتدا کلیه دستورات ورودی و خروجی ریزپردازنده ۸۰۸۶ معرفی می‌شوند. مفهوم I/O مستقیم یا ایزووله شده بیان می‌گردد و اتصال I/O به سیستم در قالب حافظه (I/O MEMORY-MAPPED) را مطرح خواهیم کرد.

۳-۳- دستورات ورودی و خروجی ۸۰۸۶

ریزپردازنده ۸۰۸۶ فقط یک دستور برای وارد کردن اطلاعات دارد بنام IN و یک دستور برای خارج کردن اطلاعات از ریزپردازنده دارد بنام OUT، هر کدام از دستورات IN، OUT به چهار صورت بکار گرفته می‌شوند. دو صورت از آن آدرس ۸ بیتی است و دیتا ۸ بیتی یا ۱۶ بیتی منتقل می‌شود و دو صورت دیگر آدرس ۱۶ بیتی و دیتا ۸ یا ۱۶ بیتی منتقل می‌شود. جدول شماره ۳-۲ هشت دستور این ریزپردازنده لیست شده است.



جدول ۳-۲ انواع مختلف کاربرد دستورات ورودی و خروجی

INSTRUCTION	عرض دیتا	شرح دستور
IN AL, D8	۸ بیت	یک بایت از یک I/O که آدرس ۸ بیتی دارد می خواند
IN AL, DX	۸ بیت	یک بایت از یک I/O که آدرس آنرا می دهد می خواند
IN AX, D8	۱۶ بیت	۱۶ بیت از یک I/O که آدرس ۸ بیتی دارد می خواند
IN AX, DX	۱۶ بیت	۱۶ بیت از یک I/O که آدرس DX آنرا می دهد می خواند
OUT D8, AL	۸ بیت	یک بایت را در I/O بی که آدرس ۸ بیتی دارد می نویسد
OUT DX, AL	۸ بیت	یک بایت را در I/O بی که آدرس آنرا می دهد می نویسد
OUT D8, AX	۱۶ بیت	۱۶ بیت را در I/O بی که آدرس ۸ بیتی دارد می نویسد
OUT DX, AX	۱۶ بیت	۱۶ بیت را در I/O بی که DX آنرا می دهد می نویسد

هر دو دستور فوق دیتا را بین ریزپردازنده و دستگاه ورودی و خروجی مبادله می نمایند. آدرس پورت یا بصورت فوری (Immediate Addressing) یا در داخل رजیستر ۱۶ بیتی DX قرار دارد. شرکت اینتل آدرس ۸ بیتی را آدرس ثابت (Fixed) شده می گویند چون هنگام نوشتن برنامه تعیین شده ولی آدرس ۱۶ بیتی که درون رجیستر DX است هر لحظه امکان تغییر دارد. وقتی که دیتا توسط دستورات OUT, IN منتقل می شود عدد آدرسی که ریزپردازنده روی خط آدرس می گذارد، شماره پورت نیز گفته می شود. دستگاه خارجی عدد روی آدرس باس را دکود می کند تا آن پورت مورد نظر فعال شود. اگر عدد آدرس ۸ بیتی باشد روی A₀ تا A₇ قرار می گیرد، اگر ۱۶ بیت باشد روی A₀ تا A₁₅ قرار می گیرد.

۴-۳-۴ I/O ایزوله شده و I/O در قالب حافظه (MEMORY-MAPPED I/O)

دو روش کاملاً مستقلی برای اتصال I/O به ریزپردازنده ۸۰۸۶ وجود دارد I/O ایزوله شده و I/O در قالب حافظه. اگر ورودی و خروجی در قالب I/O ایزوله شده معرفی شوند ما می توانیم از تمام ظرفیت ریزپردازنده برای حافظه استفاده کنیم. سیگنال M/I/O معنی خودش را دارد. مطالبی که در بالا برای I/O گفته شد صادق خواهد بود از دستورات OUT, IN استفاده می کنیم و عدد آدرس شماره پورت خواهد بود. اما اگر I/O در قالب حافظه معرفی شود. دیگر مجاز نیستید از دستورات OUT, IN استفاده کنید، باید تصور کنید که اطلاعات بین ریزپردازنده و حافظه مبادله می شود یعنی از دستورات Write, Read باید استفاده کرد. شمانی توانید از تمام ظرفیت حافظه استفاده کنید باید آن محلهایی که آدرس آنها را به پورت اختصاص داده اید بلاستفاده بمانند، زیرا I/O شما بجای آن محل ها معرفی شده یعنی ریزپردازنده I/O نمی شناسد، بلکه تصور می کند با حافظه سروکار دارد.

فصل سوم

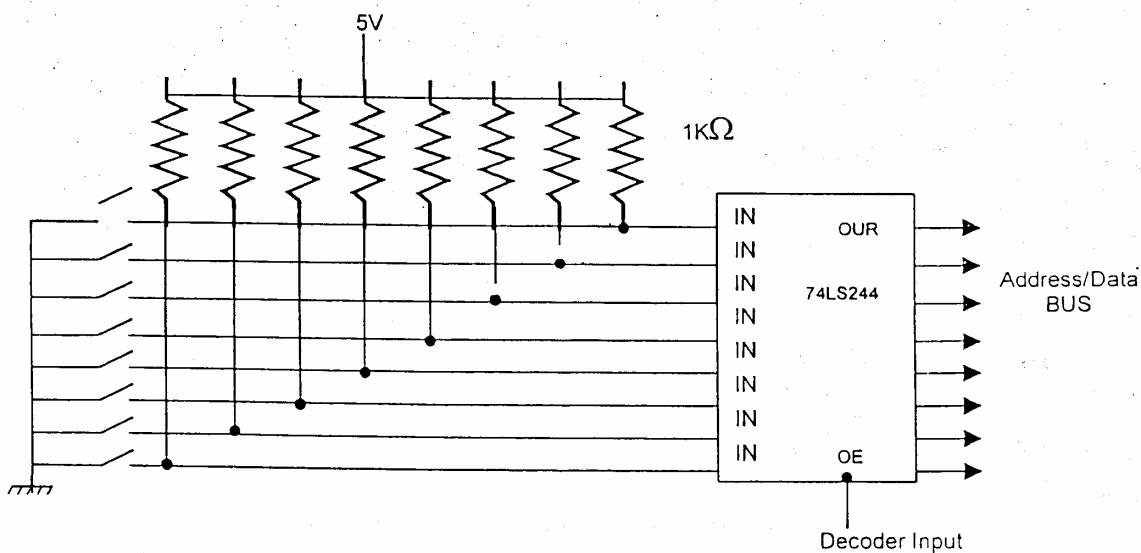
یک ورودی که در قالب I/O ایزوله شده به سیستم معرفی می‌شود، می‌تواند یک Latch با خروجی Tri-State باشد (۸ بیتی یا ۱۶ بیتی) مطابق شکل ۳-۵.

توسط ۸ سوئیچ می‌توانید هر گونه دیتائی که بخواهید تنظیم کنید با ارسال آدرس پورت خروجی بافرهای Tri-State باز شده و دیتائی تنظیم شده روی خط دیتا باس قرار می‌گیرد که ریزپردازنده دیتا را دریافت می‌کند.

دروازه خروجی دیتا از ریزپردازنده دریافت می‌کند. معمولاً باید آنرا توسط بافر برای دستگاهی تکه‌داری کند.

شکل ۳-۶ یک پورت خروجی را نشان می‌دهد. فرض کنید دیتا در بافر 74LS374 ارسال شود. خروجی آن برای نمایش به ۸ لامپ LED وصل شده باشد.

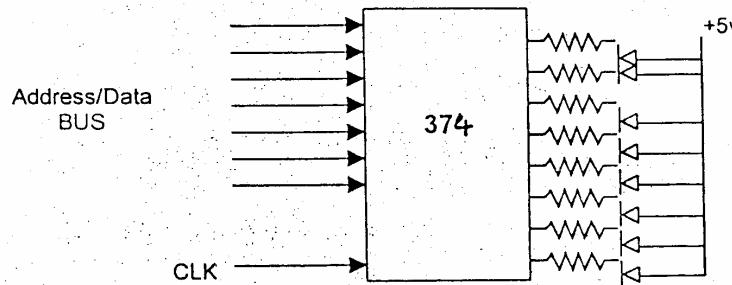
زمانی که دستور OUT اجرای می‌شود محتوای AL یا AX روی دیتا باس ارسال می‌شود که ما توسط IC 74LS374 که ۸ عدد Latch است دریافت می‌کنیم که در اینجا فرض شده دکودر کلاک لازم را ارسال می‌کند و دیتا وارد Latch می‌شود. البته در این مدار هر دیتائی که صفر منطقی باشد LED مربوطه آن روشن می‌شود.



شکل ۳-۵- یک ورودی ۸ بیتی را استفاده از ۸ سوئیچ و ۸ بیت Tri-State با خروجی

فصل سوم

همانطور که مشاهده می کنید دکود کردن آدرس برای ورودی و خروجی شبیه دکود کردن آدرس برای حافظه است علی الخصوص زمانی که I/O در قالب حافظه معرفی شده است اختلاف اصلی بین دکود کردن آدرس I/O و آدرس حافظه در تعداد خطوط آدرس است. زمانی که ریزپردازنده آدرس حافظه را می دهد از ۲۰ خط آدرس استفاده می کند (A_0 تا A_{19}) ولی وقتیکه ریزپردازنده I/O را آدرس دهی می کند از ۱۶ خط آدرس (A_0 تا A_{15}) و یا از ۸ خط آدرس (A_7 تا A_0) استفاده می کند. اختلاف دیگر در پایه MIO می باشد.



شکل ۳-۶- یک دروازه خروجی

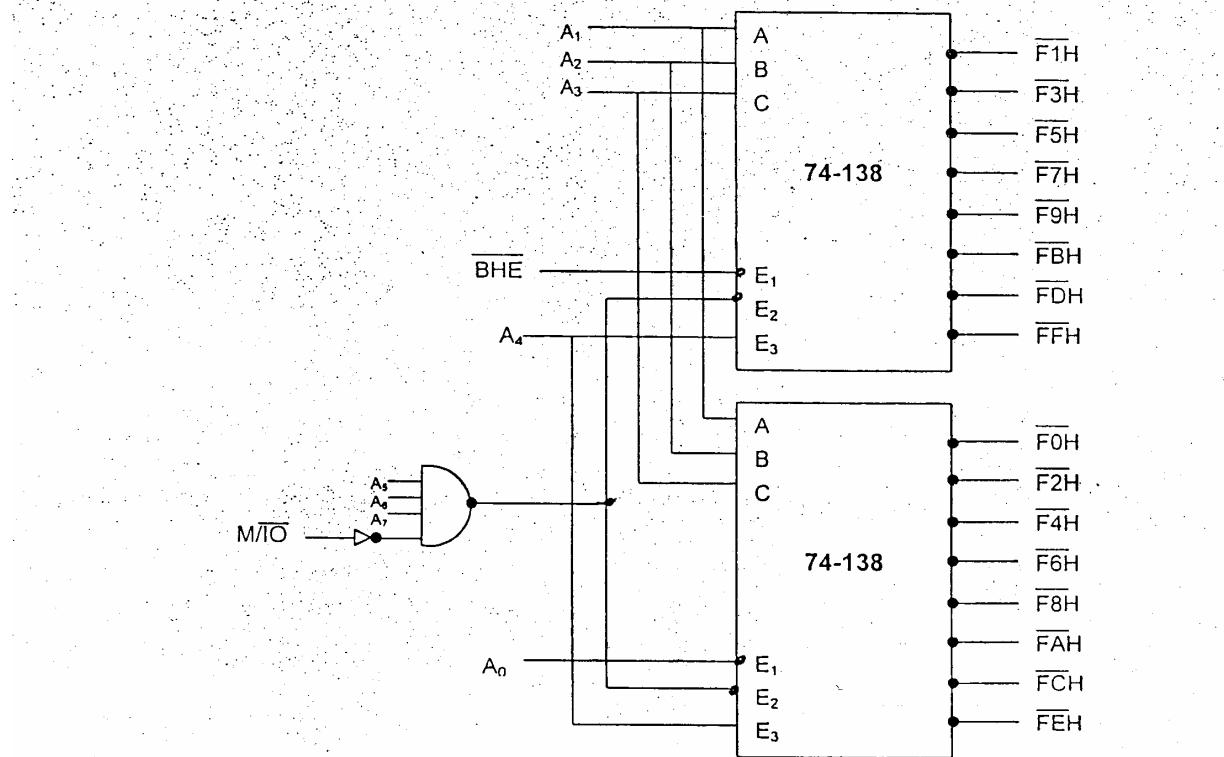
۳-۴-۱ ۸ بیت I/O با دو بانک

I/O هم می تواند مثل حافظه دارای دو بانک بالا و پایین باشد شکل ۳-۷ اتصال یک I/O را بصورت دو بانک بالا و پایین نشان می دهد.

زمانی که A_6 , A_5 , A_4 , A_3 , A_2 , A_1 , A_0 همگی یک باشند و $MIO=0$ باشد، پایه $E2$ دکودرهای 74138 را فعال می کند خط $E4$ نیز $E1$ را فعال می کند، پایه $E1$ بانک بالا توسط BHE و بانک پایین توسط A_0 فعال می شود. با تغییرات A_1 , A_3 , A_5 , A_7 شانزده آدرس $F0$ تا FF فعالی می شوند.

۳-۴-۲ اتصال ۱۶ بیتی به 8086

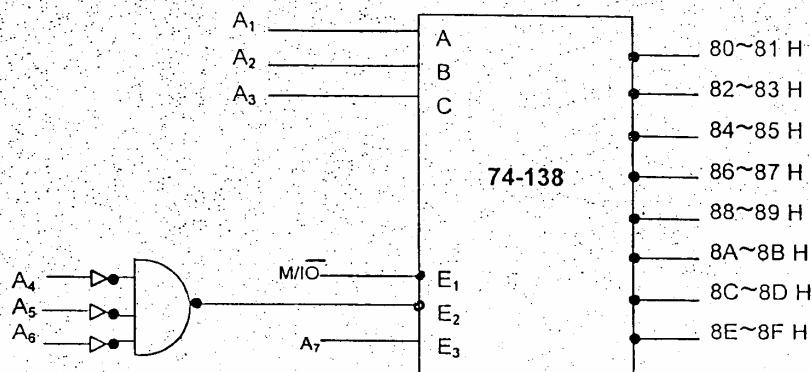
اگر قرار باشد ۱۶ I/O ۸ بیتی به 8086 متصل شود باید از هر دو BHE و A_0 چشم پوشی کرد زیرا آنها هر کدام برای ۸ بیت فعال می شوند. ورودی و خروجی ۱۶ بیتی هم زمانی مورد نیاز است که بخواهیم اطلاعات یک دستگاه مبدل آنالوگ به دیجیتال (ADC) و یا اطلاعات یک دستگاه مبدل دیجیتال به آنالوگ (DAC) را بخوانیم. زیرا این سیستم ها معمولاً ۱۰ تا ۱۲ بیت عرض دیتا دارند. شکل ۳-۸ دکود کردن ۸ پورت ۱۶ بیتی را نشان می دهد.



شکل ۷-۲-۳- مداری که در بانک بالا و پایین را برای ریزپردازنده با ۸ خط آدرس برای آدرس دهنده I/O طراحی شده است

با اندکی دقت به فعال ساز دکودر 3×8 ، IC شماره 74LS138 متوجه می شوید که A₆ تا A₄ باید دارای سطح صفر منطقی و A₇ دارای سطح یک منطقی باشد از A₆ هم که قرار شد چشم پوشی شود هشت ترکیب A₅A₂A₁ آدرسهای مورد نظر را دکود می کند.

تا این حدآشنائی با نحوه اتصال ورودی و خروجی به سیستم کافی است انشاء الله بعداً راجع به ICهای مناسب برای ۸۰۸۶ که بعنوان پورت ورودی و خروجی کاربرد دارند صحبت خواهد شد. لازم به ذکر است که گاهی I/O بصورت اینترپیت از ریزپردازنده سرویس می گیرند. که پس از بحث درباره اینترپیت و چگونگی سرویس دهی آن در این مورد هم صحبت خواهد شد.



شکل ۸-۳-۸- مداری که قادر است ۱۶ بیتی را آدرس دهی کند

سوالات و مسائل دوره ای :

- ۱- هدف از پایه های CE و یا CS که در آهای حافظه تعییه شده است چیست؟
- ۲- هدف از پایه های OE که در آهای حافظه تعییه شده است چیست؟
- ۳- اختلاف در ریزپردازنده ۸۰۸۶ و ۸۰۸۸ را بیان کنید.
- ۴- میزان حافظه ای که ۸۰۸۶ آدرس دهی می کند چقدر است؟
- ۵- با استفاده از دو دکودر جداگانه برای هر بانک یک می نیم سیستم طراحی کنید که به اندازه کافی حافظه داشته باشد.
- ۶- با استفاده از تهیه دو سیگنال Wr برای هر بانک بالا و پایین حداقل حافظه مورد نیاز یک مینیم سیستم را به ریزپردازنده ۸۰۸۶ وصل نمایند.
- ۷- روش های اتصال I/O به ریزپردازنده را شرح دهید.
- ۸- برای یک ریزپردازنده ۸۰۸۶ یک پورت ورودی و یک پورت خروجی وصل نمایند.
- ۹- دستورات وارد و خارج کردن اطلاعات در ۸۰۸۶ کدامند.
- ۱۰- به چند صورت دستور وارد کردن اطلاعات بکار می رود.
- ۱۱- به چند صورت دستور خارج کردن اطلاعات مورد استفاده قرار می گیرد.
- ۱۲- زمانیکه ریزپردازنده از آدرس ۱۶ بیتی برای یک دروازه ورودی یا خروجی استفاده می کند کدام رجیستر آدرس را دارد.
- ۱۳- توضیح دهید چرا شرکت اینتل آدرس های ۸ بیتی آدرس ثابت شده می گوید.
- ۱۴- چرا خروجی حافظه ها باید دارای وضعیت سوم (Hi-Z) باشند.
- ۱۵- آیا ورودی و خروجی را در ۸۰۸۶ نیز بصورت بانک بالا و پایین به سیستم معرفی می کنند؟
- ۱۶- معمولاً در چه موقعی از ورودی و خروجی ۱۶ بیتی استفاده می شود.

فصل چهارم

«(8086 ADDRESSING MODES) ۸۰۸۶ آدرس دهی»

کار کردن موثر و مفید با یک سیستم و یا نوشتن برنامه خوب برای یک سیستم نیازمند آشنا بودن با مدهای آدرس دهی آن ریزپردازنده برای هر دستور می باشد، در این فصل بحث را با بکار گرفتن دستور MOV که یکی از ساده ترین دستورات ۸۰۸۶ می باشد شروع می کنیم؛ دستور MOV یک بایت یا یک کلمه ۱۶ بیتی را بین ثبات ها و یا یک ثبات و حافظه انتقال می دهد.

۱-۴-۱- مدهای آدرس دهی دیتا

با استفاده از دستور MOV مدهای رجیستری-فوری-مستقیم-رجیستری غیرمستقیم-پایه بعلاوه شاخص-رجیستری نسبی و پایه نسبی بعلاوه شاخص را مورد بررسی قرار می دهیم. در شکل ۱-۴ دستور MOV جهت انتقال دیتا از منبع به مقصد برای تمامی مدها مشخص شده است. چون جهت انتقال دیتا در ریزپردازنده های مختلف متفاوت است، توضیحاً یادآور می شویم که

در دستور $MOV AX, BX$ ثبات BX منبع است و AX مقصد یعنی :

$$AX \leftarrow BX$$

ضمناً بخطاب داشته باشدی که در این گونه عملهای انتقال دیتا، منبع دیتا هرگز تغییر نخواهد کرد. و محتوای مقصد همواره تغییر می کند.

۱-۴-۱- آدرس دهی مد رجیستری

در این مد یک بایت یا یک کلمه ۱۶ بیتی اطلاعات را از ثبات منبع به ثبات مقصد انتقال می دهد. عنوان مثال دستور $MOV CX, AX$ محتوای ثبات CX را به ثبات AX کپی می کند.

نوع مد	دستور	منبع	مقصد
۱- رجیستری	MOV AX, BX	رجیستر	AX
۲- فوری	MOV BL, 3AH	دیتا	BL
۳- مستقیم	MOV 1434H, AX	رجیستر	1434H حافظه
۴- رجیستری غیر مستقیم	MOV [BX], AX	رجیستر	[BX] حافظه
۵- پایه بعلاوه شاخص	MOV [BX+SI], AX	رجیستر	[BX+SI] حافظه
۶- نسبی رجیستری	MOV [BX+4], AX	رجیستر	[BX+4] حافظه
۷- رجیستری نسبی بعلاوه شاخص	MOV Array [BX+SI], AX	رجیستر	Array[BX+4] حافظه

شکل ۴-۱-۴- مدهای آدرس دهی دیتا

۴-۱-۲- آدرس دهی سریع (فوری):

یک بایت یا یک کلمه ۱۶ بیتی دیتا را به رجیستر مقصد منتقل می کند. عنوان مثال MOV AX, 1234H دیتائی که مقدار آن ۱۲۳۴ در مبنای ۱۶ را به ثبات AX منتقل می کند. توجه دارید که حرف H بدنال هر عددی که آمد بیانگر آن است که دیتا در مبنای ۱۶ است.

۴-۱-۳- آدرس دهی مستقیم:

یک بایت یا یک کلمه ۱۶ بیتی را بین حافظه و یک ثبات انتقال می دهد که آدرس مورد نظر حافظه با دستور العمل ذخیره می شود، عنوان مثال دستور MOV AX, List محلی از حافظه که ۱۶ بیتی است و آدرس آن در List است به ثبات AX منتقل می کند.

۴-۱-۴-آدرس دهی رجیستری غیر مستقیم :

یک بایت یا یک کلمه ۱۶ بیتی را بین حافظه و یک ثبات انتقال می‌دهد که آدرس مورد نظر حافظه در ثبات دیگری است مثل دستور $MOV AX, [BX]$ ، ۱۶ بیت اطلاعات را از محلی که آدرس آن در ثبات BX است به ثبات AX منتقل می‌کند.

۴-۱-۵-آدرس دهی پایه بعلاوه شاخص :

یک بایت یا یک کلمه ۱۶ بیتی بین حافظه و یک ثبات منتقل می‌شود که آدرس محل حافظه از جمع دو ثبات پایه و شاخص بدست می‌آید. مثل دستور $MOV AX, [BX+SI]$ که ۱۶ بیت اطلاعات از محلی از حافظه که آدرس آن بصورت زیر محاسبه می‌شود به ثبات AX منتقل می‌نماید.

$$AX \leftarrow M [DS \times 10H + BX + SI]$$

۴-۱-۶-آدرس دهی نسبی رجیستری :

در این مد نیز یک بایت یا یک کلمه بین رجیستر و حافظه منتقل می‌شود که آدرس محل مورد نظر حافظه از جمع یک رجیستر و عدد دیگری که از Displacement است بدست می‌آید.

مثل $MOV AX, [BX+4]$ که محلی از حافظه به آدرس

یا $MOV AX, Array[SI]$ یعنی در AX بریزد محلی از حافظه که از رابطه زیر بدست می‌آید :

$$AX \leftarrow M [DS \times 10H + Array + SI]$$

۴-۱-۷-آدرس دهی نسبی پایه بعلاوه شاخص :

یک بایت یا یک کلمه بین رجیستر و حافظه انتقال داده می‌شود که آدرس حافظه از جمع محتوای رجیستر پایه و رجیستر ایندکس و عدد داده شده بدست می‌آید، مثل دستور :

$MOV AX, Array[BX+DI]$

$$AX \leftarrow M [DS \times 10H + Array + BX + DI]$$

۴-۲-آدرس دهی رجیستری (Register Addressing)

آدرس دهی رجیستری یکی از ساده‌ترین مدهای آدرس دهی در ۸۰۸۶ یا ۸۰۸۸ می‌باشد، این ریزپردازنده دارای ۸ ثبات ۸ بیتی است بنام های AH, AL, BH, BL, CH, CL, DH, DL و

ثبات های ۱۶ بیتی به نام های AX, BX, CX, DX, SP, BP, DI, SI, CS, DS, SS, ES می‌باشد.

به غیر از ثبات های سگمنت که فقط در چند دستور خاص مثل MOV , $PUSH$, POP کاربرد دارند، مابقی ثبات ها می‌توانند هر کدام منبع یا مقصد باشند. نکته دیگر اینکه ظرفیت منبع و مقصد باید برابر باشد والا اسembler اعلام خطای کند. در جدول ۴-۱ بعضی از دستورات از انواع

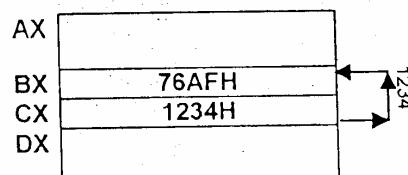
مختلف MOV لیست شده است. بدینه ایست لیست کردن تمامی دستورات MOV که ۲۵۶ دستور می شود کار ساده نیست.

جدول ۴-۱- دستورات آدرس دهنده رجیستری

Assembly Language	Operation
MOV AL,BL	BL → AL
MOV CH,CL	CL → CH
MOV AX,CX	CX → AX
MOV SP,BP	BP → SP
MOV DS,AX	AX → DS
MOV SI,DI	DI → SI
MOV DI,SI	SI → DI
MOV BX,ES	ES → BX
MOV CS,DS	Not allowed (segment-to-segment)
MOV BL,BX	Not allowed (Mixed Size)

در جدول ۴-۱- دقت کنید دو دستور آخر اجازه داده نشده است.

شکل ۴-۲ نحوه مبادله اطلاعات توسط دستور CX MOV BX را نشان می دهد. توجه دارید که اطلاعات منبع تغییر نمی کند، اما اطلاعات قبلی مقصد از بین می رود و عدد هگز ۱۲۳۴h از محتوای CX بداخل BX کپی می شود.



شکل ۴-۲- انتقال دیتا از نبات منبع به نبات مقصد

۴-۳-آدرس دهنده فوری (Immediate Addressing)

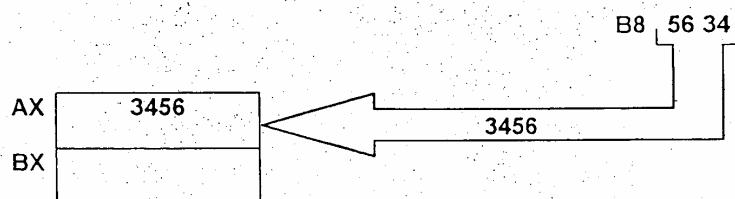
مد آدرس دهنده دیگری که نسبتاً ساده است، زیرا دیتا بصورت یک بایت یا یک کلمه ۱۶ بیتی بلافاصله بعد از Op-Code در داخل حافظه ذخیره می شود. در بعضی از زبان های اسembly معمولاً قبل از دیتا علامت # را قرار می دهند مثل دستور MOV AX,#1654H.

البته در ریزپردازنده ۸۰۸۶ اگر علامت # را نیز نگذاریم صحیح است و مترجم اسembly آنرا ترجمه می کند. دیتا را به هر دو صورت اعشار (پایه ۱۰) و هگز (پایه ۱۶) در این دستور معرفی می کند. اگر در پایه ۱۰ بود بعد از دیتا هیچ علامتی ندارد اما اگر در پایه ۱۶ باشد باید حرف H آورده شود. جدول ۴-۲ مثالهایی از مدد آدرس دهنده فوری را با دستور MOV نشان می دهد.

Assembly Language	Operation
MOV BL,44	BL ← 2CH
MOV AX,44H	AX ← 0044H
MOV SI,0	SI ← 0000H
MOV CH,100	CH ← 64H
MOV SP,3000H	SP ← 3000H

جدول ۴-۲ مثالهایی از مد آدرس دهی فوری با دستور MOV

بعنوان مثال اگر دستور $MOV AX,3456H$ را در نظر بگیرید، کد عملیاتی AX عدد 8B بعنوان دستور در حافظه به شکل ۴-۳ ذخیره می شود. یعنی دستور سه بایت جانیاز دارد.



شکل ۴-۳- ذخیره سازی دینتا در ثبات

با اجرای دستور عدد 3456 در AX کپی می شود.

۴-۴- مد آدرس دهی مستقیم (Direct Addressing)

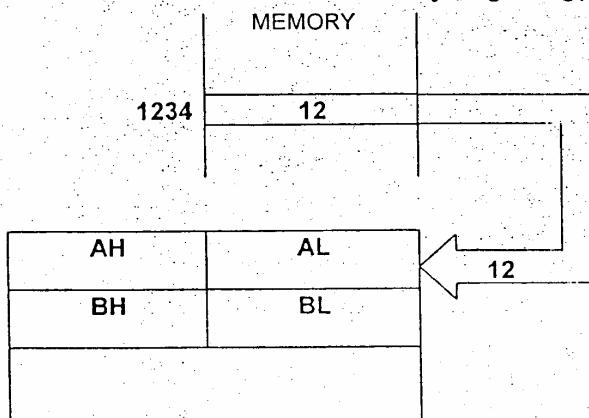
این مد آدرس دهی دو فرم کلی دارد. یکی آدرس دهی مستقیمی که فقط در دستور MOV بکار می رود و یا بطور اخص در دستور MOV بین حافظه و AX یا AL بکار می رود و فرم دیگر آنرا گویند و تقریباً با هر دستوری در ۸۰۸۶ یا ۸۰۸۸ بکار می رود. Displacement Addressing

الف : آدرس دهی مستقیم

وقتی که صحبت از آدرس دهی مستقیم می شود، دستورات STA، LDA که در ۸۰۸۵ معرفی شده اند و ۸۰۸۶ نیز آنها را براحتی اجرا می کند را در بر می گیرد. هر زمانی که دینتا بین حافظه و AL و یا حافظه AX مبادله شود این مد کاربرد دارد. دستور MOV که در ۸۰۸۶ کاربرد دارد یک بایت دینتا را از یک محل حافظه که Data آدرس آن محل می باشد بداخل AL کپی می کند این دستور سه بایت طول دارد بعضی وقت بشکل ۴-۴ نوشته می شود.

فصل چهارم

با اجرای دستور $MOV AL,[1234H]$ اگر فرض کنیم در آدرس ۱۲۳۴ حافظه عدد ۱۲ باشد این مقدار در AL کپی می شود. مطابق شکل زیر :



شکل ۴-۴- طریقه انتقال دینا از محل حافظه به AL

جدول ۴-۳ چهار صورت ممکنه این مد را نمایش می دهد. مجدداً یادآور می شود که دستور MOV فقط بین AL یا AX با حافظه کاربرد دارد.

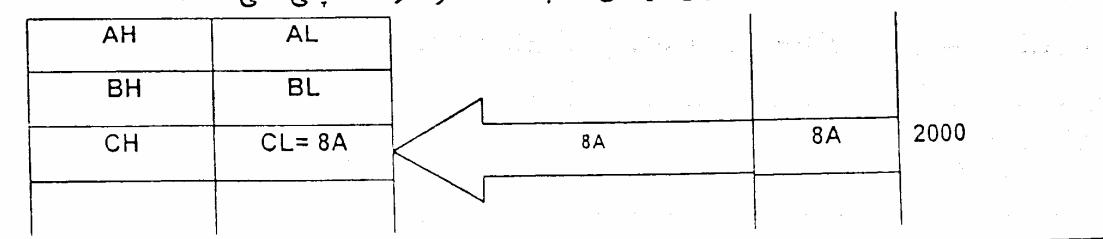
جدول ۴-۳- چهار صورت ممکنه دستور MOV برای انتقال اطلاعات بین حافظه و AL یا AX

Assemble Language	Operation
$MOV AL,NUMB$	یک بایت از حافظه به آدرس NUMB به AL کپی می شود.
$MOV AX,COW$	یک کلمه ۱۶ بیتی از حافظه به آدرس COW در AX کپی می شود.
$MOV NEW,AL$	محتوای AL در محلی از حافظه به آدرس NEW ذخیره می شود.
$MOV THERE,AX$	محتوای AX در محلی از حافظه به آدرس THERE ذخیره می شود.

ب : Displacement Addressing

در این مد دستور برخلاف آدرس دهی مستقیم که صحبت شد و ۳ بایت طول داشت دارای طول چهار بایتی است. و استفاده از آن خیلی ساده و اکثرآ ۸۰۸۸ یا ۸۰۸۶ از آن استفاده می کند. دستور $MOV CL,[2000H]$ را در نظر بگیرید خیلی شبیه دستور $MOV AL,[1234]$ است با این تفاوت که آن سه بایت بود و این ۴ بایت طول دارد.

دستور قبلی مخصوص AL بود در حالیکه این دستور برای تمام ثبات ها کاربرد دارد. اگر فرض کنیم محتوای محل ۲۰۰۰ حافظه دارای دیتای ۸A باشد. آنرا در CL کپی می کند.



فصل چهارم

مقایسه دو دستور فوق از نظر تعداد بایت ها و Opcode به شکل زیر می باشد.

سه بایت	A03412	MOV AL,[1230H]
چهار بایت	BA0E0020	MOV CL,[2000H]

جدول ۴-۴ تعدادی از دستورات MOV با فرم Displacement را نشان می دهد. البته تمام

اینگونه دستورات 512 عدد می شود که امکان لیست کردن تمامی آنها وجود ندارد.

جدول ۴-۴- لیست تعدادی از دستورات MOV Displacement

Assembly Language	Operation
MOV CH,DOG	محتوای محلی از حافظه که با DOG در دیتا سگمنت مشخص شده در CH کپی می شود (مقدار عملی DOG توسط اسمبلی تعین می شود)
MOV CH,[1000H]	محتوای محلی از حافظه که با آدرس 1000H مشخص شده در CH کپی می شود (البته سگمنت مربوطه DS خواهد بود)
MOV DATA,BP	محتوای ثبات BP کپی می شود در محلهای از حافظه به آدرس DATA, DATA+1 و DATA+2 (در بخش DS)
MOV NUMBER,SP	محتوای ثبات SP در محلهای به آدرس NUMBER, NUMBER+1, NUMBER+2 کپی می شود (ناحیه DS)

۴-۵- آدرس دهی رجیستری غیر مستقیم (Register Indirect Addressing)

در این مد یک محل از حافظه توسط یکی از ثبات های DI, SI, BP, BX آدرس دهی می شود.

بعنوان مثال اگر محتوای ثبات BX عدد 1000H باشد، دستور MOV AX,[BX] محتوای

محل 1000 حافظه در DS را در AX کپی می کند.

بعضی از دستوراتی که از مد آدرس دهی رجیستری غیر مستقیم استفاده می کند در جدول شماره ۴-۵ لیست شده اند.

جدول ۴-۵- لیست بعضی از دستورات آدرس دهی رجیستری غیر مستقیم

Assembly Language	Operation
MOV CX,[BX]	یک کلمه ۱۶ بیتی از محلی از حافظه در DS در CX کپی می کند
MOV [BP],DL	محتوای DL را در محلی از حافظه در SS که با BP آدرس دهی شده ذخیره می کند
MOV [DI],BH	محتوای BH را در محلی از حافظه در DS که با DI آدرس دهی شده کپی می کند
MOV [DI],[BX]	محتوای حافظه به حافظه اجازه داده نشده، مگر اینکه دستور رشته ای استفاده شود

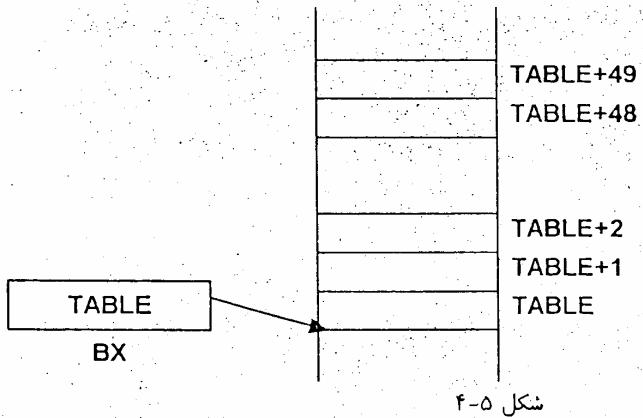
نکته ای که باید توجه کرد این است که وقتی از ثبات های DI, SI, BX استفاده می کنید آدرس

محلي از حافظه در ناحیه دیتا سگمنت DS را می دهید و هنگامیکه از ثبات BP استفاده می کنید

آدرس محلی از حافظه در ناحیه استک سگمنت SS را می دهید.

فصل چهارم

نکته دیگر اینکه معمولاً از مد غیر مستقیم رجیستری وقتی استفاده می شود که ارجاع به یک جدول از داده ها باشد. فرض کنید شما مجبورید یک جدول از دیتائی که از یک ولست متر دیجیتالی می خوانید تشکیل دهید. مثلاً ۵۰ نمونه خوانده شده و مطابق شکل ۴-۴ که هم ۵۰ دیتا را نشان می دهد و هم ثبات BX برای مشخص کردن هر محل جدول استفاده شده است.



شکل ۴-۵

برای تکمیل کردن مثال شما نیاز دارید با دستور MOV محل TABLE را در BX بروزیزد بعد با استفاده از مد آدرس دهی رجیستری غیر مستقیم مقادیر را به ترتیب در جدول وارد کنید، مثال زیر این مطلب را نشان می دهد.

;EXAMPLE 4-1

;Instructions That Read 50 Bytes of Data From DATA-PORT and Stores Them in a TABLE

```

BEHIN : MOV   BX,Offset TABLE ; Address TABLE
        MOV   CX,50      ; Load Loop Connt
AGAIN  : IN    AL,DATA-PORT ; Get DATA
        MOV   [BX],AL    ; Save DATA
        INC   BX        ; BX ← BX+1
        LOOP AGAIN     ; Repeat Until CX=0
        :
        :

```

در مثال فوق کلمه Offset Directive که به اسملر می گوید که BX را با افست آدرس TABLE پر کن و همچنین شماره CX هم با عدد ۵۰ پر می شود، از خصایص دستور LOOP است که به برچسب مورد نظر برود تا CX=0 شود.

۴-۶-آدرس دهی پایه بعلاوه شاخص (Base Plus Index Addressing)

این مد خیلی شبیه مد آدرس دهی رجیستری غیر مستقیم است. زیرا آدرس حافظه با جمع یکی از ثبات های پایه مثل BX یا BP با یک ثبات شاخص مثل DI یا SI بدست می آید. غالباً ثبات

فصل چهارم

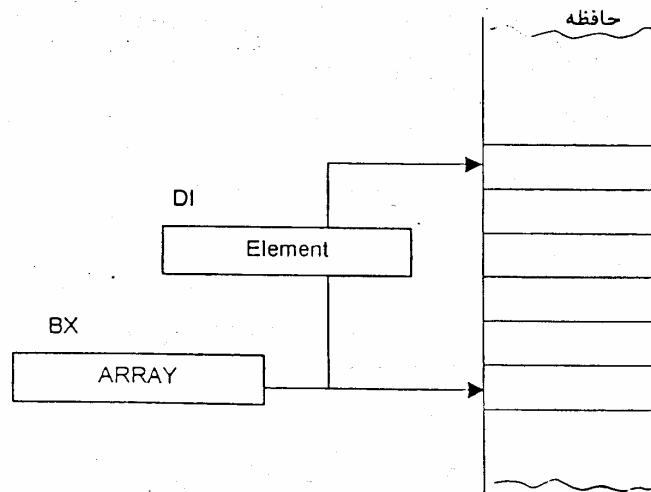
پایه آدرس شروع یک حافظه آرایه ای را دارد و ثبات شاخص آدرس نسیی را در آرایه پنهاداری می کند مثل دستور : $MOV DX, [BX+DI]$

فرض کنید که : DS=0100H , DI=10H , BX=1000H باشد آدرس فیزیکی تولید شده عبارت است از $02010H + 100+10 = 02010H \times 10 + 100+10 = 2010H$ لذا دیتای محل های 02010H و 02011H وارد ثبات DX می شوند.

جدول ۴-۶- مثال هایی از دستورات با مدل آدرس دهنده ثبات پایه بعلاوه ثبات شاخص

Assembly Language	Operation
MOV CX,[BX+DI]	ثبات CX از محلی از حافظه در DS با آدرس عملیاتی که از جمع BX+DI بدست می آید. پر می شود
MOV CH,[BP+SI]	ثبات CH از محلی از حافظه در SS با آدرس عملیاتی که از جمع BP+SI بدست می آید. پر می شود
MOV [BX+SI],SP	در محلی از حافظه در DS با آدرسی که از BX+SI که از SP بدست می آید ذخیره می شود
MOV [BP+DI],CS	در محلی از حافظه در SS با آدرسی که از BP+DI بدست می آید ذخیره می شود

مزیت عمده کاربرد این مدل در آدرس دهنی المان های یک آرایه ای از دیتاست. فرض کنید یک آرایه ای از دیتا در بخش DS در محل ARRAY واقع شده را بخواهیم آدرس دهنی کنیم. برای اینکار نیاز داریم BX را با آدرس ARRAY پر کنیم و DI ردیف المان های آرایه را داشته باشد. مطابق شکل ۴-۶ بخوبی مشخص است که چگونه BX ابتدای آرایه و DI شمارش آنرا بعده دارد.



شکل ۴-۶- مثالی که نشان دهنده آدرس دهنده ثبات پایه بعلاوه ثبات شاخص باشد.

در مثال ۴-۲ توسط قطعه برنامه ای چگونگی استفاده از این مد آدرس دهی نشان داده شده است، در این برنامه عنصر محل ۱۰ آرایه به محل ۲۰ آرایه منتقل می‌شود.

'Example 4-2

'Using The Base Plus Index Addressing Mode

```
MOV  BX,Offset Array    'Address Array
MOV  DI,10H             'Address Element 10H
MOV  AL,[BX+DI]         'Get Data in Element 10
MOV  DI,20H             'Address Element 20H
MOV  [BX+DI],AL          'Save Data at Element 20H
```

۴-۷-آدرس دهی نسبی رجیستری (Register Relative Addressing)

این مد نیز شبیه آدرس دهی ثبات پایه بعلاوه ثبات شاخص است و همچنین شبیه Displacement Addressing است. در این مد، دیتای موجود در یک سگمنت حافظه از روش جمع کردن عدد Displacement با محتوای یکی از ثبات‌های پایه یا شاخص (BX, BP, SI, DI) آدرس دهی می‌شود، دستور $MOV AX, [BX+1000]$ را در نظر بگیرید با فرض اینکه DS=200H و BX=100H و محتوای محل هایی از حافظه به آدرس 03100H و 03101H درون AX قرار می‌گیرد چونکه $200 \times 10 + 100 + 1000 = 3100H$ آدرس فیزیکی

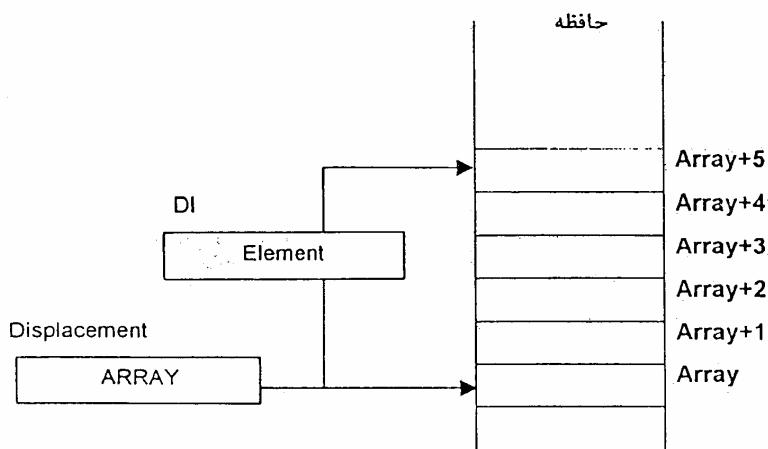
یادآور می‌شود که اگر از ثبات BP استفاده شود دیتا در بخش SS آدرس دهی خواهد شد.

جدول ۴-۷ لیست چند دستور در این مد را نشان می‌دهد.

جدول ۴-۷-مثالهایی از آدرس دهی رجیستری نسبی

Assembly Language	Operation
MOV AX,[DI+100H]	دیتا از بخش DS با آدرس DS+100+DI به AX منتقل می‌شود.
MOV Array[SI],BL	در بخش DS به آدرس DS+SI Array+SI ذخیره می‌شود.
MOV LIST[BP]+CL	در بخش SS به آدرس SS+BP LIST+BP ذخیره می‌شود.
MOV DI,SET[BX]	دیتا از بخش DS با آدرس DS+SET+BX به DI منتقل می‌شود

همانگونه که در مد آدرس دهی ثبات پایه بعلاوه ثبات شاخص گفته شد این مد نیز توانائی خوبی برای آدرس دهی آرایه‌ای دارد. شکل ۴-۶ همانند شکل ۴-۷ چگونگی این مدر را با این تفاوت که نقش ثبات پایه به Displacement داده شده است را نشان می‌دهد.



شکل ۴-۷

همچنین مثال ۳-۴ نیز عیناً قطعه برنامه ای را در رابطه با مدل آدرس دهی رجیستری نسبی نشان می دهد.

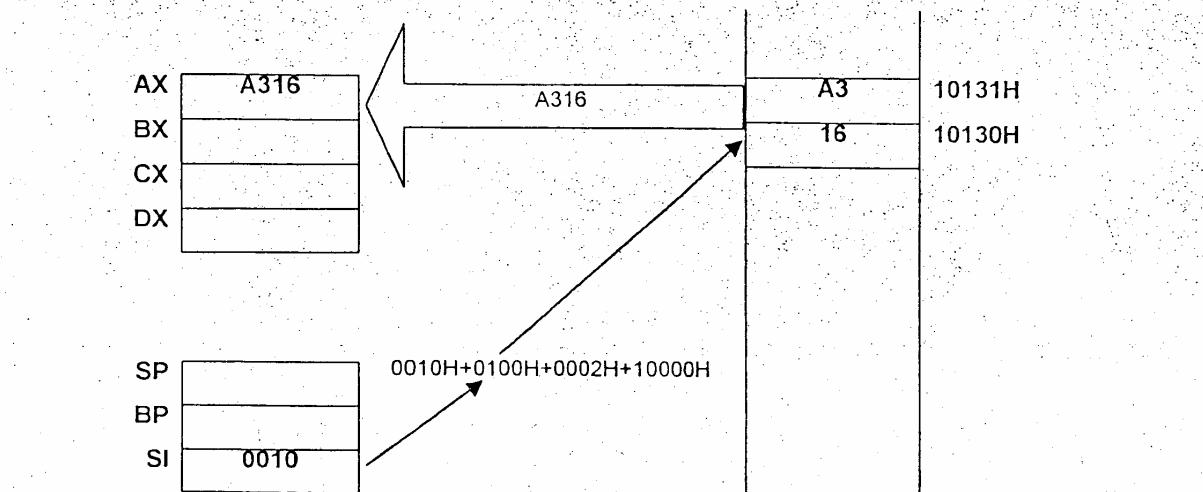
'Example 4-3

'Using Register Relative Addressing

MOV DI,10H	'Address Element 10H
MOV AL,Array[DI]	'Get Data From Element 10H
MOV DI,20H	'Address Array Element 20H
MOV Array[DI],AL	'Save Data at Element 20H

۴-۸- آدرس دهی نسبی پایه بعلاوه شاخص (Base Relative-Plus-Index Addressing) آخرین مدل کاربردی در ۸۰۸۶ این مدل است. این مدل نیز شبیه ثبات پایه بعلاوه ثبات شاخص است، با این تفاوت که به حاصل جمع ثبات پایه و شاخص باید محتوای Displacement را نیز افزود تا آدرس مورد نظر تولید شود. این نوع آدرس دهی معمولاً برای آدرس دهی آرایه های دو

بعدی بکار می رود دستور زیر را در نظر بگیرید [BX+SI+100H]
MOV AX, [BX+SI+100H]
 بفرض اینکه DS=1000, DIS=100H, BX=0020H, SI=0010H باشد
 $DS \times 10 + DIS + BX + SI = 0010H$
 و 1013H به درون AX می ریزد



شکل ۴-۸- نشان دهنده آدرس دهی نسبی پایه بعلاوه شاخص

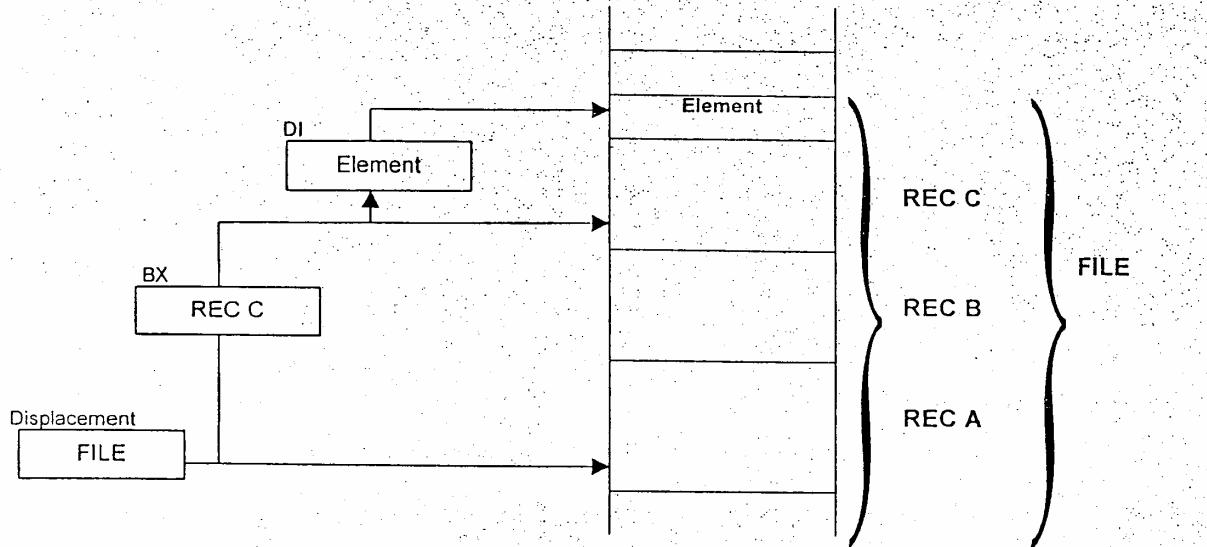
واضح است که محاسبه آدرس فیزیکی در این مد کاملاً پیجیده است. بعضی از دستوراتی که از این مد استفاده می کنند در جدول ۴-۸ لیست شده اند.

جدول ۴-۸- بعضی از دستوراتی که از مد آدرس ذهنی نسبی پایه بعلاوه شاخص استفاده می کنند.

Assembly Language	Operation
MOV DH,[BX+DI+20H]	محتوی محلی از حافظه در DS به آدرس BX+DI+2 در DH کپی می شود.
MOV AX,FILE[BX+DI]	محتوی محلی از حافظه در DS به آدرس BX+DI+FILE در AX کپی می شود.
MOV LIST[BP+DI],CL	محتوی CL در محلی از حافظه که آدرس آن LIST+BP+DI است ذخیره می شود.

فرض کنید فایلی داریم شامل تعدادی رکورد و هر رکورد شامل تعدادی عنصر (المان) باشد. آدرس فایل را بدهد و ثبات پایه آدرس یک رکورد در فایل را بدهد و ثبات شاخص آدرس هر عنصر در رکورد را بدهد. این مد آدرس دهی در شکل ۴-۸ نشان داده شده است.

مثال شماره ۴-۴ یک برنامه ایست که انتقال اطلاعات از المان صفر رکورد A به المان ۲ رکورد C را با استفاده از مد آدرس دهی نسبی بعلاوه شاخص نشان می دهد.



شکل ۴-۸- آدرس دهی نسبی پایه بعلاوه شاخص که آدرس یک المان از یک رکورد در یک فایل را نمایش می دهد

'Example 4-4

'Using The Base Relative Plus Index Addressing Mode

```

MOV  BX,Offset-REC A
MOV  DI,0
MOV  AL,FILE[BX+DI]
MOV  BX,Offset-REC C
MOV  DI,2
MOV  FILE[BX+DI],AL

```

۴-۹- مد های آدرس دهی برنامه

مد های آدرس دهی برنامه با دستورات CALL, JMP دارای سه شکل مستقیم، نسبی و غیر مستقیم می باشد، که در این بخش با بکارگیری دستور JMP به تشریح هر سه نوع پرداخته می شود.

۴-۹-۱- مد آدرس دهی برنامه بصورت مستقیم

مد مستقیم، همان صورتی است که ریزپردازنده ۸۰۸۵ هم برای تمام دستورات JMP و CALL مورد استفاده قرار می دهد. (البته تقریباً تمام ریزپردازنده های ۸ بیتی هم استفاده می کنند) ۸۰۸۶ نیز از این مد استفاده می کند البته نه مثل مد های نسبی و غیر مستقیم دستوراتی که از مد مستقیم استفاده می کنند آدرس مورد نظر را با Op-Code ذخیره می کنند بعنوان مثال اگر

فصل چهارم

در یک برنامه ای قرار باشد به محل 10000H پرش داشته باشیم عدد آدرس باید با ذخیره شود. شکل ۴-۹ نحوه آدرس دهی مستقیم را نشان می دهد. ملاحظه می فرمائید که ۴ بایت برای ذخیره کردن سگمنت بعدی و آفست بعدی مورد نیاز است. یعنی برای JMP to 10000 باید یک عددی مثل 1000 بعنوان سگمنت و عددی مثل 0000 بعنوان آفست ذخیره شده که اولی در CS و دومی در IP ریخته خواهد شد.

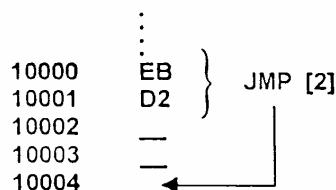
OP-CODE	OFFSET-LOW	OFFSET-HIGH	SEGMENT-LOW	SEGMENT-HIGH
EA	00	00	00	10

شکل ۴-۹- پنج بایت زبان ماشین برای دستور JMP به محل 10000H حافظه

تنها دستور دیگری که در مد آدرس دهی مستقیم برنامه کاربرد دارد دستور CALL داخل همان سگمنت است. و اغلب از یک اسمی بعنوان LABEL برای آدرس دهی استفاده می شود.

۴-۹-۲- مد آدرس دهی برنامه بصورت نسبی :

این مد در ۸۰۸۵ وجود ندارد، اما بعضی از ریزپردازنده های ۸ بیتی همانند ۸۰۸۶ دارند که به شرح آن می پردازیم. کلمه نسبی (Relative) عملأً به موقعیت ثبات IP برمی گردد یعنی نسبت به اشاره گر به دستور العمل آدرس دهی می کند. بعنوان مثال اگر دستور JMP اشاره به دو محل بعد دارد، یعنی دو بایت بعد از آنچه که IP اشاره دارد. برای بدست آوردن چنین آدرسی باید IP+2 شود. به قطعه برنامه زیر توجه کنید :



ملاحظه می کنید که Op-Code دستور JMP یک بایت است و یک بایت هم عدد مورد نظر جمعاً دستور دو بایت فضا اشغال می کند. بهمین دلیل است که کاربرد این مد بیش از آدرس دهی مستقیم برنامه است.

دستورات با مد نسبی دو دستور JMP یا CALL هستند و شامل یک محل ۸ بیتی Displacement یا ۱۶ بیتی هستند. اغلب اسما برها دستور JMP Far نیز دارند و بطور اتوماتیک فضای مناسب

برای Displacement خود انتخاب می کنند. اگر میزان پرش خیلی بزرگ باشد بعضی اسمبلرها JMP مستقیم را ترجیح می دهند. با یک بایت فضاشما می توانید تا $+127$ محل به بعد و یا -128 - محل به قبل برنامه برگردید. اگر از ۱۶ بیت فضا برای Displacement استفاده شود مقدار پرش به $\pm 32k$ می رسد.

۴-۹-۲- مد آدرس دهی برنامه بصورت غیر مستقیم :

ریزپردازنده ۸۰۸۶ چند فرم از پرش ها و یا صدا زدن زیربرتامه ها را بصورت غیرمستقیم اجازه می دهد در جدول شماره ۴-۹ چند دستور JMP بصورت غیر مستقیم لیست شده است که هر کدام می توانند از هر یک از ثبات های ۱۶ بیتی (AX, BX, CX, DX, SP, BP, SI, DI) و هر ثبات نسبی ای مثل [BX], [DI], [SI], [BP]، و هر ثبات نسبی با Displacement را بکار بگیرند.

جدول شماره ۴-۹- مثالهای از آدرس دهی غیرمستقیم برنامه

Assembly Language	Operation
JMP AX	به محلی از حافظه که آدرس آن در AX است در همان سگمنت پرش می کند.
JMP CX	به محلی از حافظه که آدرس آن در CX است در همان سگمنت پرش می کند.
JMP [BX]	به محلی از حافظه که آدرس آن در محلی از حافظه است و BX آدرس آن محل در DS را مشخص می کند در همان سگمنت پرش می کند.
JMP [DI]	به محلی از حافظه در همان سگمنت که آدرس آن محلی در حافظه در DS بوسیله DI مشخص شده پرش می کند.
JMP TABLE[BX]	به محلی از حافظه در همان سگمنت که آدرس آن محل در حافظه بوسیله BX+TABLE مشخص می شود پرش می کند.

برای تشریح دستور JMP به مثال زیر توجه کنید :

Example 4-5

Using Indirect Addressing for a JMP

```

MOV      BX,#4
JMP      TABLE[BX]
TABLE   DW    LOC0
        DW    LOC1
        DW    LOC2
        DW    LOC3
  
```

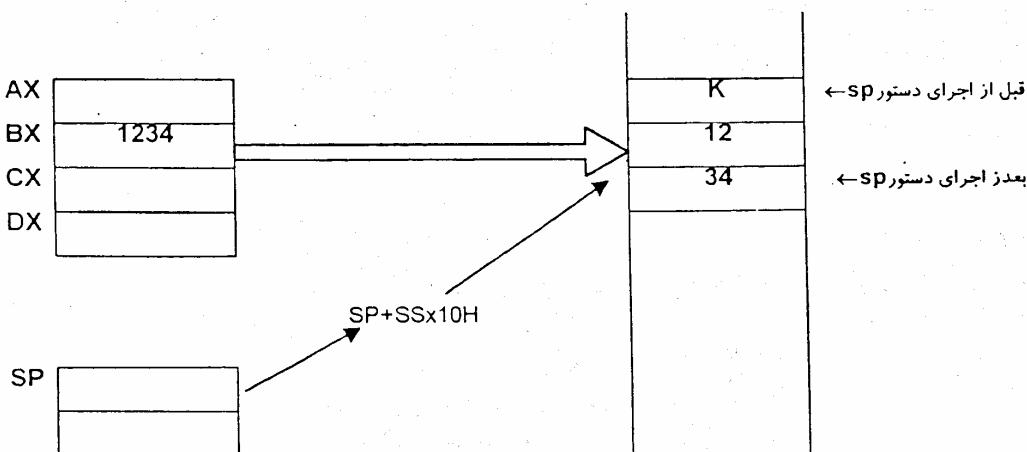
فرض کنید TABLE نیز بصورت زیر معرفی شده باشد. که استفاده از این TABLE امکان پرش به جاهای مختلف را می دهد.

در TABLE ما آدرس هایی برای چهار برنامه مختلف داریم که هر آدرس دارای ۲ بایت Offset می باشد. وقتی که شما از دستور $JMP \text{ TABLE}[BX]$ استفاده می کنید، وجود TABLE ما را به اول TABLE هدایت می کند به شرطی که $BX=0$ باشد. هر (DefineWord)=DW دو بایت فضای حافظه را اشغال می کند لذا وقتی که در BX عدد ۴ را قرار می دهید $\text{TABLE}+[BX]$ آدرس LOC2 را بوجود می آورد.

۴-۱- آدرس دهی ناحیه پشته (StackMemoryAddressing)

ناحیه پشته در تمام ریزپردازنده ها از اهمیت خاصی برخوردار است. در این حافظه اطلاعات موقتی و آدرس های مراجعه از زیربرنامه را نگهداری می کنند. پشته دارای سازمانی LIFO (Last in First out) می باشد. دیتا توسط دستور PUSH در پشته قرار داده می شود. البته اگر از دستور CALL استفاده کنید بطور اتوماتیک آدرس برگشت در پشته قرار داده می شود. برداشت دیتا از پشته توسط دستور POP یا اجرای دستور RET در آخر زیربرنامه صورت می گیرد.

اصطلاحاً می گویند SP همیشه به بالای پشته اشاره می کند یعنی به آخرین دیتائی که در پشته قرار داده اید. در ریزپردازنده ۸۰۸۶ حافظه پشته از بالا پر می شود شکل ۴-۱۰ را ببینید. فرض کنید K آخرین دیتائی است که در پشته ذخیره کرده اید. حال اگر قرار باشد دیتای جدیدی در پشته PUSH کنید ابتدا $SP \leftarrow SP-1$ می شود اولین بایت شما ذخیره می شود مجدداً این عمل تکرار می شود تا بایت دوم را ذخیره کنید.



شکل ۴-۱۰- جگونگی فراردادن ثبات BX را در ناحیه پشته نشان می دهد.

فصل چهارم

نحوه اجرای دستور POP هم بهمین ترتیب است. ابتدا بایت اول برداشته می شود پس از آن $SP \leftarrow SP+1$ می شود آنوقت بایت دوم را از پشته بر میدارد. جدول شماره ۴-۱۰ لیست بعضی از دستورات POP, PUSH قابل استفاده در ۸۰۸۶ را نشان می دهد.

توجه داشته باشید که دستورات POP, PUSH همیشه یک کلمه ۱۶ بیتی را به پشته وارد و یا از آن خارج می کنند یعنی هرگز بصورت بایت عمل نمی کنند.

جدول شماره ۴-۱۰- مثالهایی از دستورات POP و PUSH

Assembly Language	Operation
PUSHF	ثبات پرچمها (FLAGS) را در پشته ذخیره می کند.
POPF	ثبات پرچمها (FLAGS) را از پشته بر میدارد.
PUSH AX	ثبات AX را در پشته ذخیره می کند.
POP BX	ثبات BX را از پشته پر می کند.
PUSH DS	ثبات DS را در پشته ذخیره می کند.
POP CS	اجرای این دستور امکان پذیر نیست
PUSH [BX]	یک کلمه از حافظه که آدرس آن در BX درون پشته ذخیره می شود.

جدول شماره ۱۱-۴- تمام مودهای آدرس دهی مورد استفاده در ۸۰۸۶ لیست شده است

MOV AL,BL	آدرس دهی رجیستری
MOV AL,LIST	$DS \times 10 + LIST =$ آدرس فیزیکی
MOV AL,12	آدرس دهی فوری
MOV AL,[BX]	$DS \times 10 + BX =$ آدرس فیزیکی
MOV AL,[BP]	$SS \times 10 + BP =$ آدرس فیزیکی
MOV AL,[SI]	$DS \times 10 + SI =$ آدرس فیزیکی
MOV AL,[DI]	$DS \times 10 + DI =$ آدرس فیزیکی
MOV AL,LIST[BX]	$DS \times 10 + BX + LIST =$ آدرس فیزیکی
MOV AL,LIST[BP]	$SS \times 10 + BP + LIST =$ آدرس فیزیکی
MOV AL,LIST[SI]	$DS \times 10 + SI + LIST =$ آدرس فیزیکی
MOV AL,LIST[DI]	$DS \times 10 + DI + LIST =$ آدرس فیزیکی
MOV AL,[BX+SI]	$DS \times 10 + BX + SI =$ آدرس فیزیکی
MOV AL,[BX+DI]	$DS \times 10 + BX + DI =$ آدرس فیزیکی
MOV AL,[BP+SI]	$SS \times 10 + BP + SI =$ آدرس فیزیکی
MOV AL,[BP+DI]	$SS \times 10 + BP + DI =$ آدرس فیزیکی
MOV AL,LIST[BX+SI]	$SS \times 10 + BX + SI + LIST =$ آدرس فیزیکی
MOV AL,LIST[BX+DI]	$SS \times 10 + BX + DI + LIST =$ آدرس فیزیکی
MOV AL,LIST[BP+SI]	$DS \times 10 + BP + SI + LIST =$ آدرس فیزیکی
MOV AL,LIST[BP+DI]	$DS \times 10 + BP + DI + LIST =$ آدرس فیزیکی

فصل چهارم

سوالات دوره ای :

۱- ثبات های ۸ بیتی ۸۰۸۶ را نام ببرید.

۲- ثبات های ۱۶ بیتی ۸۰۸۶ کدامند؟

۳- کدامیک از دستورات زیر غلط است؟ چرا؟

MOV AX,BX MOV SP,DI MOV CS,AX MOV CS,SS

۴- علامت [] نشان دهنده چیست؟

۵- فرض کنید که DI=0300H, BX=0200H, DS=0200H باشد، آدرس دسترسی به ذیتا در حافظه توسط

هر کدام از دستورات زیر را تعیین کنید.

MOV AL,[2000H] MOV AL,[BX] MOV [DI],AL

۶- آیا دستور زیر صحیح است؟ چرا؟

MOV [DI],[BX]

۷- توضیح دهید فرق دو دستور زیر در چیست؟

MOV BX,Data
MOV BX,OffsetData

۸- اگر DI=0100H, BP=1000H, SS=2000H, DS=1000H را داشته باشیم آدرس ذیتا را در

هر یک از دستور زیر تعیین کنید؟

MOV AL,[BP+DI]
MOV DX,[BP]
MOV CX,[DI]

۹- سه مد آدرس دهی برنامه را نام ببرید.

۱۰- یک دستور JMP چند بایت طول دارد و محتوای هر بایت چیست؟

۱۱- شرح عملکرد PUSH [DI] را بیان کنید.

فصل پنجم

«دستورات انتقال داده ها در ۸۰۸۶»

مقدمه : دستورات هر ریزپردازنده به سه دسته تقسیم می شوند، این تقسیم بندی بمنظور سادگی در مطالعه و بررسی انواع دستورات می باشد :

الف - دستورات انتقال داده ها (Data Movement Instruction)

این دستورات فقط عمل انتقال داده از منبع به مقصد را بعهده دارند بدون اینکه هیچ تغییری در داده داشته باشد.

ب - دستورات محاسباتی ریاضی و منطقی (Arithmetic & Logic Instruction)

این دستورات محاسبات ریاضی و منطقی را بعهده دارند عبارت دیگر پردازش‌های مورد نظر توسط این دستورات انجام می گیرد.

ج - دستورات کنترل برنامه (Program Control Instruction)

دستوراتی که در این گروه مورد بررسی قرار می گیرند هیچ عملی و یا ربطی به دیتا ندارند بلکه فقط کنترل برنامه را بعهده دارند.

در این فصل سعی می شود تمام دستوراتی که انتقال و یا جابجایی دیتا در ریزپردازنده ۸۰۸۶ را

بعهده دارند معرفی کنیم. این دستورات عبارتند از IN, MOV, PUSH, POP, XCHG, XLAT،

MOVS OUT, LEA, LDS, LES, LAHF, SAHF

علت اینکه اول این دستورات معرفی می شوند این است که اولاً کاربری بیشتری دارند بعلاوه ساده تر از سایر دستورات از نظر درک دانشجو می باشد.

و در خاتمه ما غالب دستورالعمل ها (Format Instructions) مورد تجزیه و تحلیل قرار داده تا

توانید به زبان ماشین نیز برنامه بنویسید و یا برنامه ای را تحلیل نمایید.

۱-۵- مرور مجدد بر دستور MOV

در فصل چهارم راجع به دستورات MOV در بحث مدهای آدرس دهی صحبت شده، ذر اینجا ما این دستورات را بصورت زبان ماشین مورد بررسی قرار می‌دهیم؛ زیرا ما باید قادر به تفسیر یک برنامه نوشته شده به زبان اسembly باشیم. علی الخصوص زمانی که بخواهیم آنرا اشکال زدایی یا ^{آن}تغییری بدهیم.

۱-۱-۵- زبان ماشین

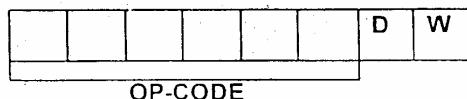
دستورات ۸۰۸۶ به زبان ماشین دارای طول متغیری از یک بایت تا ۶ بایت می‌باشند، اولین بایت هر دستور کد عملیاتی (OP-CODE) آن است که بیانگر نوع عملی است که باید انجام دهد. شکل ۱-۵-۱ فرم کلی یک کد عملیاتی ریزپردازنده ۸۰۸۶ برای تعدادی از دستورات را نشان می‌دهد، ۶ بیت اولیه بایت اول هر دستور OP-CODE آن است و دو بیت باقیمانده نشان دهنده جهت حرکت دیتا (D) و اینکه دیتا بصورت ۸ بیتی است یا ۱۶ بیتی می‌باشد(W).

اگر D=1 است یعنی دیتا به ثباتی که در بایت دوم معرفی می‌شود وارد خواهد شد.

اگر D=0 است یعنی دیتا از ثباتی که در بایت دوم معرفی می‌شود خارج خواهد شد.

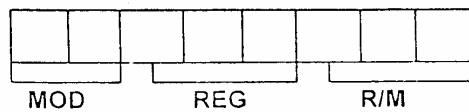
اگر W=0 است یعنی عمل انجام شده روی یک بایت دیتا خواهد بود.

اگر W=1 است عمل انجام شده روی دیتای ۱۶ بیتی خواهد بود.



شکل ۱-۵-۱- بایت اول اغلب دستورات زبان ماشین ریزپردازنده ۸۰۸۶

شکل ۱-۵-۲ بایت دوم دستور العمل را در صورتیکه داشته باشد نشان می‌دهد. این بایت بیانگر مد (MOD) آدرس دهی دیتا است. دو بیت برای MOD، سه بیت برای معرفی ثبات (REG) و سه بیت برای معرفی حافظه یا ثبات دیگر (R/M) که یکی برای منبع و دیگری برای مقصد مورد استفاده قرار می‌گیرد.



شکل ۱-۵-۲- بایت دوم یک دستور العمل زبان ماشین ریزپردازنده ۸۰۸۶

بخش مد یکی از انواع مختلف آدرس دهی و Displacement را طبق جدول ۵-۱ برای دستورات انتخاب می کند.

جدول ۵-۱- بیانگر مدهای مورد استفاده

MOD	عملیات
00	NODISPLACEMENT
01	یک ۸ بیتی دارد.
10	یک ۱۶ بیتی دارد.
11	یک ثبات است.

طبق جدول اگر MOD=11 باشد بخش R/M بایت دوم، یک ثبات را معرفی می کند اما در سه حالت دیگر MOD، موضوع R/M به سایر مدهای عملیاتی برمی گردد. اگر در مد آدرس دهی از 00 استفاده شده باشد در محاسبه آدرس Displacement وجود ندارد. و دو حالت 01 و 10 یک DISP هشت بیتی و یا ۱۶ بیتی دارد. وقتی که بایت استفاده می شود ۷F~00 مقدار DISP با احتساب بیت علامت خواهد بود، اگر این مقدار منفی باشد FF~80 خواهد شد. در صورتی که DISP شانزده بیتی استفاده شود (توسعه یافته) مقدار آن ۰۰۰۰~۰۰۷F و در حالتی که ۱۶ بیت باشد مقدار آن نیز منفی باشد FF80~FFFF خواهد شد.

۱-۲-۵- تخصیص کد به ثبات ها :

جدول شماره ۵-۲ کد اختصاص داده شده به ثبات ها (REG) و یا حافظه ثبات (R/M) را وقتی که MOD=11 می باشد نشان می دهد.

جدول شماره ۵-۲- تخصیص کد به ثبات (REG) و ثبات حافظه (R/M) که ۱۱ است.

CODE	W=0(BYTE)	W=1(WORD)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

توجه دارید که این جدول دارای دو لیست به کدها اختصاص داده شده است یکی به ازای W=0 که دیتا انتقالی بایت است و لیست دوم به ازای W=1 که دیتا انتقالی ۱۶ بیتی است.

فصل پنجم

فرض کنید که یک دستور دو بایتی که مقدار آن 8BECH است داشته باشیم. آنرا بصورت باینری بطور مجرزا بایت ۱ و بایت ۲ مشخص کنید شکل ۵-۳ بدست می آید. این دستور MOV BP, SP است

OP-CODE	D	W	MOD	REG	R/W
1 0 0 0 1 0	1	1	1 1	1 0 1 1	1 0 0

OP-CODE=100010	MOV
D=1	جهت به ثبات
W=1	دیتای ۱۶ بیتی
MOD=11	R/W یک ثبات است
REG=101	ثبات BP است
R/M=100	ثبات SP است

شکل ۵-۳- نمایش باینری دستور 8BECH و تفسیر بیت های آن

با مراجعه به ضمیمه یک این کتاب ملاحظه می فرمائید که OP-CODE مذکور متعلق به دستور MOV است توجه دارید که هر دو بیت D و W یک هستند، یعنی اینکه یک کلمه ۱۶ بیتی به ثبات انتقال داده می شود. و ثبات آن 101 می باشد که BP خواهد بود. در واقع دیتای ۱۶ بیتی به ثبات BP منتقل می کند. چون MOD=11 است پس R/M نیز یک ثبات است که طبق جدول ۵-۲ کد 100 ثبات SP می باشد. یعنی دیتا از BP به SP منتقل می شود فرم دستور آن MOV BP, SP است.

۳-۱-۵- آدرس دهی حافظه برای R/M

اگر بخش MOD دارای مقادیر 00 یا 01 یا 10 باشد، میدان R/M معنی جدیدی بخود می گیرد که همانا محلی از حافظه خواهد بود. در اینصورت به این بخش مدهای آدرس دهی مختلفی اختصاص داده می شود که توسط سه بیت مریبوطه مطابق جدول ۵-۳ تعیین می شود. البته دقت فرمائید تمامی این مدها در فصل چهار صحبت شده است. با توجه به جدول اگر MOD=0 باشد و R/M=101 باشد مدل آدرس دهی [DI] خواهد بود. اما اگر MOD=01 باشد آنوقت مدل عنوان مثال [DI] DATA[DI] خواهد شد.

شکل ۵-۴ فرمت دستور العمل و زبان ماشین دستور MOV DL,[DI] را نشان می دهد. این دستور دارای دو بایت طول است و کد عملیاتی آن 10 OP-CODE=100010 است. D=1 است (انتقال به

فصل پنجم

ثبات) و $W=0$ (بایت منتقل می شود) و $DISP=00$ (ندارد) و $REG=010$ (بیانگر ثبات DL) و $R/M=101$ (یعنی محلی از حافظه که آدرس آن در DI است)

جدول ۵-۳- آدرس دهی مدهای R/M

CODE	شرح عملیات
000	$[BX+SI]$
001	$[BX+DI]$
010	$[BP+SI]$
011	$[BP+DI]$
100	$[SI]$
101	$[DI]$
110	$[BP] *$
111	$[BX]$

* در مورد [BP] تحت عنوان مد آدرس دهی خاص دقیق مطالعه فرمائید.

OP-CODE	D	W	MOD	REG	R/M
1 0 0 0 1 0	1	0	0 0	0 1 0	1 0 1

Displacement بیانگر MOD ندارد

REG بیانگر ثبات DL

R/M بیانگر ثبات [DI]

MOV بیانگر OP-CODE

جهت انتقال به D

W یک بایت منتقل می شود

شکل ۴-۵- بیان و نمایش میدان های دستور [MOV DL,[DI]]

۵-۱-۴- مد آدرس دهی خاص

یک مد آدرس دهی خاص داریم که در جداول ۵-۱ و ۵-۲ و ۵-۳ ظاهر نشده است. این مد زمانی کاربرد دارد که آدرس دیتا فقط با Displacement مشخص شده باشد. عنوان مثال دستورات MOV [1000H],DL و MOV DATA,DL را در نظر بگیرید، دستور اول محتوای ثبات DL را در حافظه ای که آدرس آن $DS \times 10 + 1000$ است ذخیره می کند. هر وقت که یک دستور فقط یک Displacement داشته باشد، میدان مد آن همیشه MOD=00 خواهد بود و میدان R/M همیشه کد 110 را دارد. این ترکیب باعث می شود تصور کنید که Displacement ندارد و از مد آدرس دهی [BP] بدون Displacement استفاده می کنید اما شما عملانه نمی توانید مد [BP] را بدون Displacement استفاده کنید، زبان اسمنبلی یک بایت Displacement بطور

فصل پنجم

اتوماتیک برای آن قرار می دهد ولی مقدار آن را ۰۰ می گذارد و از مد MOD=01 استفاده می کند، لذا هنگام استفاده از [BP] عملاً از دستور [BP+00H] استفاده خواهد شد شکل ۵-۵ میدان و بیت های باینری دستور MOV [1000H],DL را تشریح می کند، اگرچه به تنها این MOV [BP],DL مد شناخته شده نیست این دستور بطور غیر صحیح فرض شده که بصورت DL معرفی شود.

شکل ۵-۵ فرم اسambilی و شرح بیت های دستور MOV [BP],DL را نشان می دهد ملاحظه می کنید که سه بایت است و بایت سوم ۰۰H است.

OP-CODE	D	W	MOD	REG	R/M
1 0 0 0 1 0	0	0	0 0	0 1 0	1 1 0
Byte 1					
Displacement Low					
0 0 0 0 0 0 0 0					
Byte 3					
0 0 0 1 0 0 0 0					
Byte 4					
Displacement High					

شکل ۵-۵- تعداد بایت ها و فرمت دستور MOV [1000H],DL

OP-CODE	D	W	MOD	REG	R/M
1 0 0 0 1 0	0	0	0 1	0 1 0	1 1 0
Byte 1					
Displacement Low					
0 0 0 0 0 0 0 0					
Byte 3					
0 0 0 1 0 0 0 0					
Byte 4					
Displacement High					

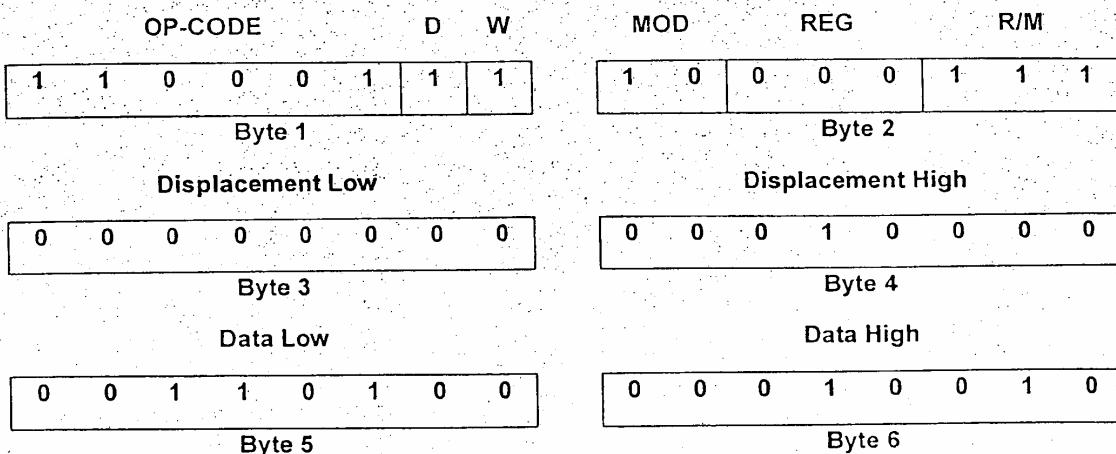
شکل ۵-۶- تعداد بایت ها و فرمت دستور MOV [BP],DL با سومین بایت که ۱۰۰H است

۵-۱-۵- دستور MOV با آدرس دهی فوری

دستور MOV [BX+1000],1234H را در نظر بگیرید. این دستور عدد 1234H را در محلی از حافظه که آدرس آن بصورت زیر محاسبه می شود ذخیره می کند :

$$\text{آدرس فیزیکی} = DS \times 10H + BX + 1000H$$

این دستور ۶ بایت طول دارد دو بایت برای کد عملیاتی REG, MOD, W, D و R/M مثل دستورات قبلی، دو بایت برای ذخیره کردن دیتا ۱۲۳۴ و دو بایت Displacement برای ذخیره کردن عدد 1000H شکل ۷-۵ بایت ها و فرمت دستور را نشان می دهد:



شکل ۵-۷ - شش بایت و فرمت دستور MOV [BX+1000],1234H

۵-۶-۵- دستور MOV بر انتقال ثبات های سگمنت

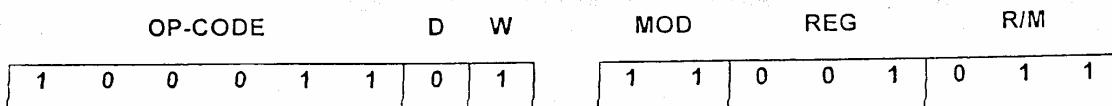
اگر بخواهیم محتوای یک ثبات سگمنت را بوسیله دستورات PUSH, POP, MOV انتقال دهیم در میدان REG بایت دوم کد عملیاتی کدهای مخصوصی باید در نظر گرفت که در جدول ۵-۴ ثبات های مختلف سگمنت را معرفی می کند.

جدول ۵-۴ - میدان REG برای ثبات های سگمنت

کد	ثبات سگمنت
000	ES
001	CS *
010	SS
011	DS

* توجه داشته باشید که POP CS, MOV CS,XX اجازه داده نشده است

شکل ۵-۸ دستور MOV BX,CS را نشان می دهد. دقیت داشته باشید که این OP-CODE با MOV های دیگر تفاوت دارد و REG براساس ثبات های سگمنت انتخاب می شود.



شکل ۵-۸ - دستور MOV BX,CS بصورت زبان ماشین

اگر چه در این مبحث ما بطور کامل راجع به زبان ماشین صحبت نمی کنیم، ولی این مقدار می تواند شروع خوبی برای علاقمندان به برنامه نویسی به زبان ماشین باشد. با خاطر داشته باشید که معمولاً برنامه باید بصورت اسembلی نوشته می شود و مترجم آن اسembلر نام دارد که آن را به زبان ماشین تبدیل می کند. ندرتاً ممکن است شما نیاز پیدا کنید که زبان ماشین را بخواهید بصورت مستقیم بنویسید.

۵-۲- دستورات PUSH/POP

دو دستور POP، PUSH از جمله دستورات مهمی در ریزپردازنده ۸۰۸۶ جهت قرار دادن و برداشتن دیتا در سازمان LIFO پشته (Stack) می باشند. در این ریزپردازنده چهار فرم دستور POP، PUSH به شرح زیر داردیم :

الف- ثبات (REGISTER)

ب- حافظه / ثبات (Register/Memory)

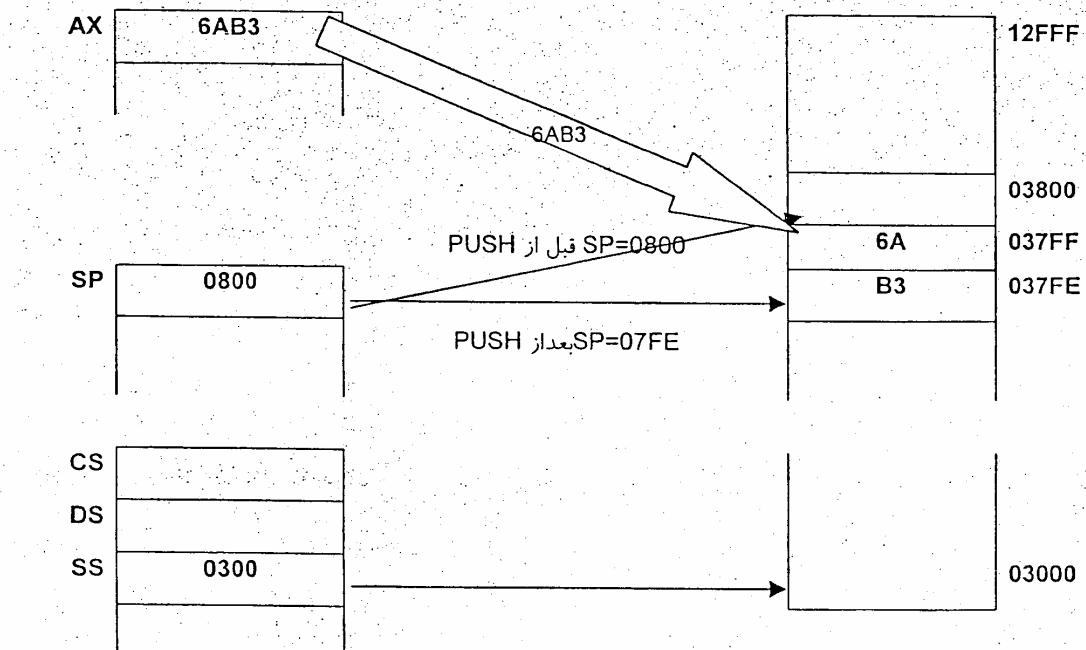
ج- ثبات سگمنت (SegmentRegister)

د- پرچمها (Flags)

در فرم ثبات اجازه دارید هر ثبات ۱۶ بیتی را به پشته PUSH و یا از آن POP نمایید. در فرم حافظه / ثبات مجازید محتوای یک ثبات ۱۶ بیتی یا یک محل حافظه را در پشته ذخیره کنید. در فرم ثبات های سگمنت مجازید هر ثبات سگمنت را در پشته PUSH نمایید. اما کد سگمنت را نمی توان از پشته POP کرد باید بصورت دیگری مقدار PUSH شده را بازیافتد. در مورد آخرین فرم، پرچمها (Flags) را می توان بین پشته و ثبات موقعیت سیستم مبادله نمود.

۵-۲-۱- دستور PUSH

دستور PUSH همیشه دو بایت دیتا بداخل پشته منتقل می کند. منبع این دیتا می تواند ثبات پرچم، هر ثبات ۱۶ بیتی داخلی ریزپردازنده و یا ۲ بایت دیتا از حافظه باشد. هر وقت قرار باشد دیتا وارد پشته شود ابتدا با ارزشترین بایت وارد حافظه پشته در محلی SP-1 قرار می گیرد پس از آن بایت دوم (کم ارزشتر) وارد پشته شده و در محل ۲ SP-2 ذخیره می گردد. یعنی در هر دستور PUSH باید ثبات SP دو عدد کم شود. شکل ۵-۹ عملیات اجرائی دستور PUSH AX را نشان می دهد. ملاحظه می فرمایید که محتوای ثبات AX بصورت زیر در پشته ذخیره می شود $[SP-1]=AH$ ، $[SP-2]=AL$



شکل ۵-۹- نمایش نحوه اجرای دستور PUSH AX

شکل ۵-۵ لیست چهار فرم دستور PUSH ممکن‌ه در ریزپردازنده ۸۰۸۶/۸۰۸۸ را نمایش می‌دهد. بایت یک که می‌باشد نیز بصورت باینری مشخص شده است.

جدول ۵-۵- چهار صورت دستورات PUSH

SYMBOLIC	BYTE1	BYTE2	مثال
PUSH reg	01010rrr		PUSH BX
PUSH mem	11111111		PUSH [BX]
PUSH seg	000ss110	mm110aaa	PUSH DS
PUSHF	10011100		PUSHF

توجه : در جدول بالا

aaa = یعنی هر مد آدرس دهنده حافظه

= همان کد میدان mod در بایت دوم است

= rrrr هر یک از ثبات های ۱۶ بیتی داخلی سیستم

= ss هر کدام از ثبات های سگمنت می تواند باشد

۲-۵-۲- دستور POP

دستور POP عکس دستور PUSH عمل می کند. این دستور دیتا را از پشته بر می دارد و آنرا در محلی که تعیین کرده اید مثل یک ثبات یا پرچمها یا یک محل ۱۶ بیتی حافظه که آدرس دهی کرده اید قرار می دهد. POP هم به چهار فرم ذکر شده برای PUSH عمل می کند (ثبات، حافظه / ثبات، ثبات سگمنت و پرچمها)

تصویر کنید که یک دستور مثل `POP BX` اگر اجرا شود ابتدا اولین بایت دیتا از پشته (از محلی SP اشاره به آن دارد) آورده می شود و در `BL` قرار داده خواهد شد بایت دوم دیتا از پشته از محلی که `SP+1` نشان می دهد آورده و در `BH` قرار داده می شود مجدداً `SP` افزوده می شود. در واقع پس از انتقال هر دو بایت `SP` دو بار افزوده شده است. جدول ۵-۶ کدهای عملیاتی و ترکیباتی باینتری و یک مثال برای هر نوع POP را لیست کرده است. توجه داشته باشید که دستور `POP CS` مجاز نیست.

جدول شماره ۵-۶- کدهای عملیاتی و ترکیباتی باینتری دستور `POP` را با ذکر مثال نشان می دهد.

شکل سمبولیک	BYTE1	BYTE2	یک مثال
<code>POP reg</code>	01011rrr		<code>POP DI</code>
<code>POP mem</code>	10001111	mm000aaa	<code>POP[DI+2]</code>
<code>POP seg</code>	000ss111		<code>POP ES</code>
<code>POPF</code>	10011101		<code>POPF</code>

توجه aaa = هر نوع در آدرس دهی می تواند باشد

mm = کد میدان mod در بایت دوم

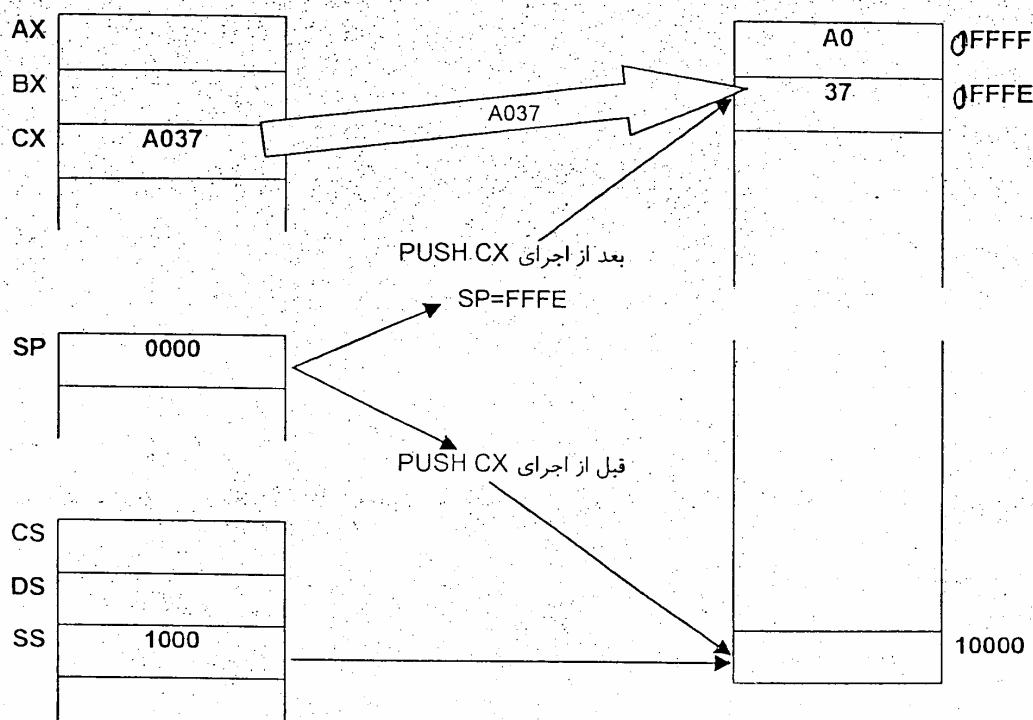
rr = هر ثبات ۱۶ بیتی می تواند باشد

ss=هر ثبات سگمنت می تواند باشد به غیر از CS

۳-۵-۲- برقراری پشته (Initializing the Stack)

وقتی که ناحیه پشته برقرار می شود. هر دو ثبات مربوطه یعنی ثبات سگمنت پشته (SS) و اشاره گر به پشته (SP) پر می شوند. در اغلب حالات معمول آن است که ناحیه ای از حافظه ای را به پشته اختصاص بدھیم و SS را با مقدار مربوطه پر کنیم. بعنوان مثال اگر بخواهیم پشته از ناحیه `H 10000` شروع شود و تا `FFFFFH` ادامه داشته باشد. آنوقت باید `SS=1000H` باشد که با `10` برابر شدن برابر آدرس شروع مورد نظر شود.

در اینصورت باید `SP=0000H` باشد. شکل ۵-۱۰ نشان می دهد که چگونه هنگام اجرای دستور PUSH دیتا در بالای (TOP) پشته قرار می گیرد. برای نشان دادن مطلب فوق از دستور PUSH CX استفاده شده است، بخاطر داشته باشید که تمام سگمنت ها طبیعتاً دوره ای (CYCLIC) هستند یعنی پس از برشدن پایین ترین محل دوباره بالاترین محل (TOP) پشته پر می شود.



شکل ۱۱-۵- نمایش اینکه ناحیه پشتۀ طبیعتاً دوره‌ای است

۵-۳- دستور (Load Effective Address) LEA

سه دستور داریم که برای پر کردن یک ثبات و یا یک ثبات معمولی و یک ثبات سگمنت از آدرس موثر استفاده می شود. دقت کنید یک آدرس در ثبات ریخته می شود نه یک دیتا جدول شماره ۵-۷ هر سه فرم این دستورات را که در ۸۰۸۶/۸۰۸۸ کاربرد دارند نشان می دهد.

جدول ۵-۷- دستورات ممکنۀ LEA ها در ۸۰۸۶

فرم سمبولیک	شرح عمل
LEA AX,DATA	ثبات AX با یک آدرس از محل DATA پر می شود.
LDS DI,LIST	DI از DS های موجود در LIST پر می شوند.
LES BX,CAT	ES از آدرس های موجود در CAT پر می شوند.

دستور LEA معمولاً یک ثبات را با یک آدرسی که بصورت اپرنند معرفی شده پر می کند مانند مثال اول جدول شماره ۵-۷ که اپرنند محتوای دیتا در ثبات AX ریخته می شود نه محتوای محلی که دیتا آدرس آن باشد.

فصل پنجم

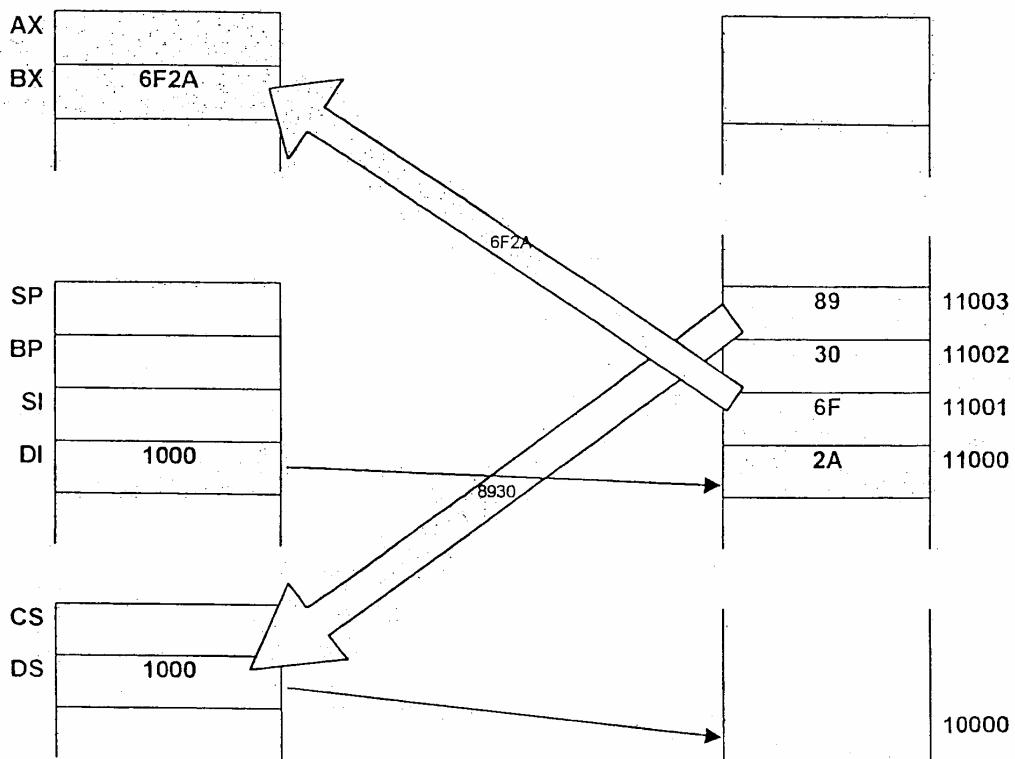
اگر LEA را با یک دستور MOV مقایسه کنید. قطعاً متوجه خواهید شد دو دستور داریم [MOV BX,[DI], LEA BX,[DI]] خیلی شبیه هم هستند در LEA محتوای DI درون BX کپی می شود اما در دستور MOV محتوای محلی از حافظه که DI دارای آدرس آن محل است درون BX کپی می شود، در واقع LEA BX,OFFSET LIST شیوه BX,LIST از دستور MOV BX,OFFSET LIST زا در BX می ریزند. چرا مامش به دستور OFFSET آدرس محل حافظه بنام LEA نیازی به آن نداریم. جواب این است که ما OFFSET را بکار می بردیم در صورتی که در LEA نیازی به آن نداریم. جواب این است آنرا برای اپرندهای ساده بکار می بردیم LIST, LIST و غیره و امکان ندارد را برای اپرندهای مثل [BX,SI], [DI] بکار برد. لذا برای اپرندهای ساده خیلی مناسب تر است از دستور LEA استفاده شود. در ریزپردازنده ۸۰۸۶ اجرای دستور BX,LIST LEA از نظر زمانی طولانی تر از دستور MOV BX,OFFSET LIST می باشد. اولی ۸ پالس ساعت زمان می برد در صورتیکه دستور MOV ۴ پالس ساعت طول می کشد. در واقع دستور MOV BX, LIST دو بار اجرا می شود تا دستور LEA باشد و همچنین دستور MOV BX,DI از دستور LEA BX,[DI] خیلی مناسب تر است.

مثال دیگری که در این مورد می تواند جالب باشد بصورت زیر است به دستور [LEA CX,[BX+DI]] توجه کنید باید محتوای BX با DI جمع شود بعنوان آدرس درون ثبات CX ریخته شود. در این مثال مشکلی که پیش می آید جمع مدول 64K خواهد بود، یعنی اگر بعنوان مثال BX=FFF00 باشد و DI=1000 باشد آنچه در CX ریخته می شود مقدار 0F00 خواهد بود. یعنی از CARRY حاصله از جمع صرف نظر شده است.

۱-۳-۵- دستورات LDS, LES

دستورات LDS, LES یک رجیستر ۱۶ بیتی را با آدرس Offset بعلوی DS, ES را نیز با آدرس جدید سگمنت پر می کنند. این دستورات نمی توانند دارای MOD=11 باشند. بعنوان مثال همانگونه که در شکل ۵-۱۲ مشخص شده است دستور LDS BX, [DI] یک آدرس ۳۲ بیتی را از محلی از حافظه که DI مشخص به ثبات DS, BX اشاره به یک دیتا سگمنت یا اکسترا سگمنت جدید دارد، منتقل می کند. دستورات LES, LDS در ادامه این فصل هنگامیکه ما راجع به دستورات رشته ای صحبت کنیم مجدداً این دستورات را نیز بررسی خواهیم کرد.

توجه کنید که آدرسی که در ثبات ها ریخته می شود باید قبل از حافظه ذخیره شده باشد.



شکل ۱۲-۵- نمایش دستور [BX,DI] LDS با ت BX از محل 11000 ، 11001 ، 11002 از محلی 11003 بر می شود و ثبات DS از محلی 11000 ، 11001 ، 11002 بر می شود.

۴-۵- انتقال دیتای رشته ای : (String Data Transfer)

سه دستور برای انتقال دیتا بصورت رشته ای وجود دارد. LODS, STOS, MOVS هر کدام اجازه می دهند که دیتا بصورت گروهی، بلوکی یا تک بایتی و یا تک کلمه ای انتقال یابد، قبل از اینکه وارد بحث این دستورات شویم، اجازه دهید تا درباره پرچم D (Direction Flag) و ثبات های سگمنتی که به ثبات های DI, SI اختصاص داده می شود تا بتوان عملیات رشته ای را انجام دهیم صحبت کنیم.

۴-۵-۱- پرچم D

پرچم تعیین کننده جهت (D)، نحوه خود افزایشی (به ازای $D=0$) و خود کاهشی (به ازای $D=1$) را برای ثبات های SI, DI در خلال عملیات رشته ای انتخاب می کند، این پرچم توسط دو

دستور CLD دارای مقدار صفر و یک می شود لذا دستور CLD برای ثبات های DI, SI دستور خود افزایشی و دستور STD حالت خود کاهشی انتخاب می کند.

اگر هدف انتقال اطلاعات بصورت باقیمانده میزان خود افزایشی و خود کاهشی یکی خواهد بود. اما اگر قرار باشد کلمه ۱۶ بیتی انتقال دهیم میزان افزودن و یا کاستن از ثبات های DI, SI باید بصورت دوتا، دوتا می باشد. اگر در عملیاتی فقط یک ثبات را مورد استفاده قرار دهید (عنوان مثال از DI استفاده شود) فقط همان ثبات افزوده و یا کاسته می شود.

۴-۵- ثبات های DI در عملیات رشته ای

در خلال اجرای دستور رشته ای ما می توانیم به ثبات های DI یا SI و یا هر دوی آنها دسترسی داشته باشیم. در DI آدرس شروع بلوکی را قرار می دهیم که در سگمنت اکسترا (ES) وجود دارد. و در SI آدرس شروع بلوکی را قرار می دهیم که در سگمنت دیتا (DS) وجود دارد. سگمنت اختصاص داده شده به ثبات SI را می توانید عوض کنید، که این کار توسط افزودن بخش سگمنت به دستور عملی می شود و در آینده راجع به آن صبحت خواهد شد. اما همواره در بخش اکسترا سگمنت (ES) قرار خواهد داشت و امکان عوض کردن سگمنت DI وجود ندارد.

۳-۵- دستور LODS

این دستور AL یا AX را از محلی از حافظه که توسط ثبات SI آدرس دهی می شود پر می کند. جدول ۵-۸ لیست دستورات LODS را به دو صورت LODSW یا LODSB که باعث می شود یک بایت یا یک کلمه ۱۶ بیتی از حافظه درون AL یا AX ریخته شود را نشان می دهد. عموماً بدنبال دستور LODS انتخاب کننده بایت یا کلمه ۱۶ بیتی خواهد بود. عموماً اپرندوها بمنزله بایت توسط (Define Byte) DB و به منزله یک کلمه ۱۶ بیتی توسط (Define Word) DW معرفی می شوند. عبارت دیگر DB یک شبیه دستور العمل است بمعنی تعریف بایت (Defie Byte) و DW یک شبیه دستور العمل است به معنی تعریف یک کلمه ۱۶ بیتی (Define Word).

جدول ۵-۸- فرم دستورهای LODS

فرم سمبلیک	شرح عمل
LODS B	محتوی حافظه ای که SI آدرس می دهد = AL
LODS W	محتوی حافظه ای که SI+1 آدرس می دهد = AX
LODS BYTE	به شرطی که BYTE به منزله یک بایت معرفی شده باشد $AL = M[SI]$
LODS WORD	به شرطی که WORD به منزله یک کلمه معرفی شده باشد $AX = M[SI]$

5-۴-۴- دستور STOS

این دستور AL یا AX را در محلی از حافظه که توسط DI در بخش اکسترا سگمنت (ES) مشخص شده است ذخیره می‌کند. جدول ۵-۹ لیست دستورات قابل اجرای STOS را نشان می‌دهد. مانند دستور LODS با حروف B و W مقدار ۸ بیتی یا ۱۶ بیتی بودن دیتا معین می‌شود.

جدول ۵-۹-نمایش دستورات STOS

فرم سمبلیک	شرح عمل
STOS B	M [DI] ← AL
STOS W	M [DI] ← AX
STOS BYTE	M [DI] ← AL به شرطی که BYTE به منزله ۸ بیت معرفی شده باشد
STOS WORD	M [DI] ← AX به شرطی که WORD به منزله ۱۶ بیت معرفی شده باشد

5-۴-۵- دستورات STOS با REP

کلمه REP بمنزله تکرار کردن (Repeat) ممکن است بعنوان پیشوند به هر دستور رشته‌ای اضافه شود. این اضافه شده باعث می‌شود که اجرای دستور مذکور ادامه یابد تا ثبات CX (بمنزله شمارنده) برابر صفر شود.

تصویر کنید تعداد ۱۰ بایت دیتا در یک منطقه‌ای از حافظه داشته باشیم و بخواهیم آنها را پاک کنیم یعنی آن ۱۰ محل را صفر کنیم. این عمل را می‌توانیم با یک سری از دستورات STOS (یعنی با ۱۰ تا از آن) تکمیل کنیم یا اینکه با یک دستور STOS که با پیشوند REP باید به مثال ۵-۱ دقت کنید:

Example 5-1

```

LES   DI,BUFFER    ;
MOV   CX,10        ;
CLD
MOV   AL,0          ;
REP   STOSB        ;

```

ملحوظه می‌کنید از دستور LES استفاده کرده تا از حافظه دو بایت بنام Offset آدرس در DI بریزد و دو بایت بنام سگمنت آدرس در ES بریزد. از CLD استفاده کرده تا D=0 شود و DI بطور اتوماتیک افزوده شود.

در بعضی از اسembلی‌ها باید برحسب آدرسی که در دستور LES استفاده می‌کند بصورت شبیه دستور DD (Define Double-Word) معرفی می‌شوند. کار این شبیه دستور این است که اعلام می‌کند حاوی آدرس ۳۲ بیتی می‌باشد.

دستور دیگر در AL عدد صفر را قرار می‌دهد و آنرا توسط STOSB در حافظه ذخیره می‌کند. با خاطر وجود REP هر بار به DI افزوده می‌شود و CX کاسته می‌شود و این عمل تکرار می‌شود تا $CX=0$ گردد.

راه سریعتر از آن این است که از دستور REP STOSW استفاده کنید. مانند مثال ۵-۲ :

Example 5-2 :

```
LES    DI,BUFFER      ;
MOV    CX,5           ;
CLD
MOV    AX,0           ;
REP
STOSW
```

۵-۴-۶ دستور MOVS

قوی‌ترین دستور رشته‌ای انتقال دیتا MOVS است زیرا این دستور یک بایت یا یک کلمه از محلی از حافظه را به محل دیگری از حافظه انتقال می‌دهد. دستور MOVS یک دیتا را از محلی که توسط DS و SI آدرس می‌دهد به محل دیگری که ES و DI تعیین می‌کند منتقل می‌نماید. جدول ۵-۱۰ لیست دستورات قابل اجرا توسط MOVS را نشان می‌دهد. لازم به ذکر است که DS را توسط دستوری که در آینده راجع به آن صحبت خواهیم کرد می‌توان به سکمنت دیگری تغییر داد

جدول ۵-۱ لیست دستورات MOVS

فرم سمبولیک	شرح عمل
MOVS	یک بایت از محلی از حافظه به محل دیگر می‌ریزد
MOVSW	یک کلمه از محلی از حافظه به محل دیگر می‌ریزد
MOVS BYTE1, BYTE2	اگر BYTE1, BYTE2 به منزله یک بایت باشد
MOVS WORD1, WORD2	اگر WORD1, WORD2 به منزله یک کلمه باشد

تصور کنید یک منطقه ای از حافظه میزان ۱۰۰ بایت را بخواهیم به منطقه دیگری منتقل کنیم استفاده از دستور MOVSB با پیشوند REP دستور مناسبی خواهد بود، به مثال ۵-۳ توجه کنید :

Example 5-3

```
LES    DI,List1      ;
LDS    SI,List2      ;
CLD
MOV    CX,100         ;
REP    MOVSB
```

فصل پنجم

دو دستور LES و LDS بکار برده شده اند تا سگمنت منبع اطلاعات و سگمنت مقصد اطلاعات و SI و DI را از آدرس مربوطه پر کنند. این آدرسها در محل های نام LIST1 و LIST2 قرار دارند اگر DS و ES مقدار مناسب را داشته باشند بجای آن دو دستور از MOV می توان استفاده کرد و فقط OFFSET آدرس را در DI و SI قرار داد. شمارنده را هم (CX) با تعداد دیتائی که باید منتقل شود پر می کنیم دستور REP MOVSB مادامیکه CX برابر صفر نشده است تکرار می شود.

۱-۵-۵- سایر دستورات انتقال دیتا

این دستورات عبارتند از OUT, IN, XLAT, SAHF, LAHF, XCHG که مهمترین دستورات انتقال هستند، کاربرد آنها کمتر از سایر دستورات است. و راجع به IN و OUT قبلًا مفصل صحبت شده است.

۱-۵-۵-۱- دستور XCHG

این دستور محتوای یک ثبات را با محتوای ثبات دیگر یا حافظه عوض می کند، این دستور نمی تواند محتوای ثبات های سگمنت را و یا محتوای دو محل از حافظه را عوض کند. اما برای تعویض اطلاعات یک ثبات با یک محل از حافظه می تواند از تمام مدهای آدرس دهی ۸۰۸۶ استفاده نماید. جدول شمار ۱۱-۵ لیست تمام صورتهای دستور XCHG را نشان می دهد. حتی ترکیب بیت های باینری هر بایت را نشان می دهد. آنطور که از جدول برمی آید بهترین و مناسب ترین فرم این دستور تعویض محتوای AX با هر ثبات ۱۶ بیتی دیگر است، زیرا این دستور فقط یک بایت طول دارد و برای ذخیره کردن این دستور در حافظه یک محل ۸ بیتی کافی است.

جدول ۱۱-۵- فرم های مختلف دستور XCHG

فرم سمبولیک	BYTE1	BYTE2
XCHG AX,reg	10010rrr	-----
XCHG reg,reg	1000011w	11rraaa
XCHG reg,mem	1000011w	mmrrraaa

(mod)=مد آدرس دهی است
mm=هر نبایی می تواند باشد بجز بیانگر بایت با Word بودن دیتای است
w=هر نوع آدرس دهی یا هر مدی می تواند باشد
aaa=aaa

۵-۵-۲- دستورات LAHF, SAHF

علت استفاده از این دو دستور برای ۸۰۸۶ از این جهت است که این ریزپردازنده باید قادر باشد برنامه های نوشته شده برای ۸۰۸۵ را نیز اجرا کند. لذا در نوشتن برنامه برای ۸۰۸۶ کاربردی ندارند. عملکرد دو دستور SAHF, LAHF بطور کلی این است که بایت کم ارزش Flag، که شبیه FLAG ۸۰۸۵ است درون AH می ریزد یا از AH به درون ثبات Flag می ریزد این دستورات در ارتباط با POP AX, PUSH AX دو دستور POP PSW و PUSH PSW را در ۸۰۸۵ معادل سازی می کنند. ترتیب اجرای دستورات مثال ۵-۴ چگونگی این معادل سازی را نشان می دهد.

Example 5-4

LAHF	;	Flag را در AH قرار می دهد
XCHG	AL,AH	؛ محتواهی AH و AL را عوض می کند
PUSH	AX	؛ A و F را در حافظه پشته ذخیره می کند

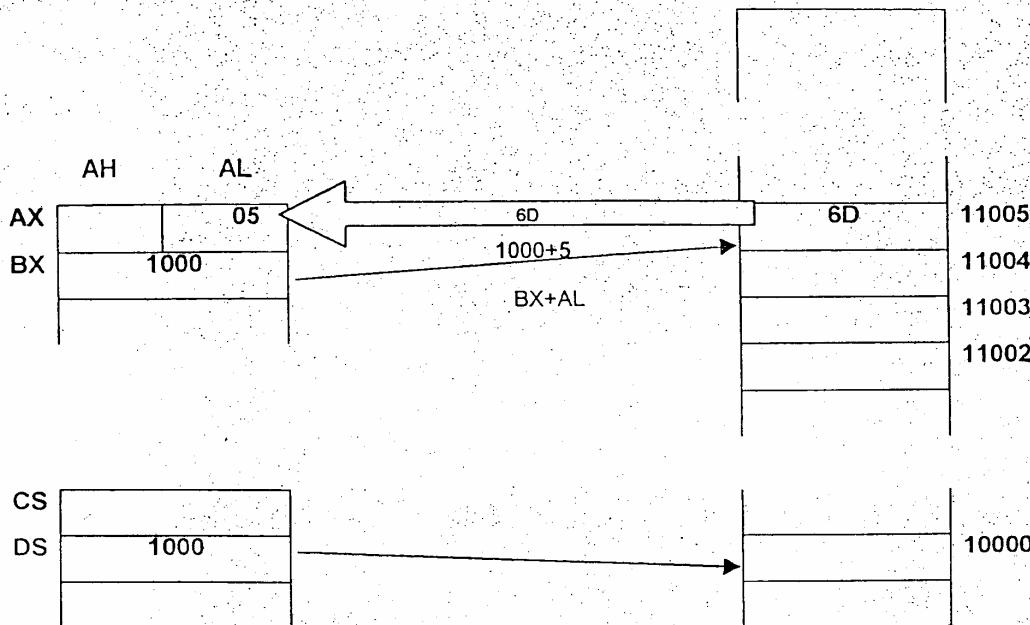
۵-۵-۳- دستور XLAT

این دستور محتواهی AL را به یک عددی که در جدولی ذخیره شده است تبدیل می کند. در واقع این دستور با تکنیک LOOKUP TABLE کدی را به کد دیگر تبدیل می کند. یک دستور XLAT ابتدا محتواهی AL را به محتواهی ثبات BX اضافه می کند تا آدرس یک محل حافظه در DS را بدست آورد. سپس دیتای ذخیره شده در این آدرس را به ثبات AL منتقل می کند. فرض کنید بخواهیم کد BCD را به 7-SEG تبدیل کنیم و کدهای 7-SEG بصورت LOOKUP-TABLE در جدولی که آدرس اول آن در محلی بنام TABLE ذخیره شده باشد. مثال ۵-۵ این عمل را انجام می دهد و شکل ۵-۱۳ نحوه عملکرد این دستورات را نشان می دهد. اگر TABLE=1000H و DS=4000H و مقدار اولیه AL=05 (یعنی عدد ۵ در BCD) طبق شکل جواب بدست آمده برابر ۶D می شود.

Example :

```
MOV     BX,OFFSET TABLE
XLAT
```

بکار گرفتن دستوراتی که ترجمه یا تبدیل کد BCD به 7-SEG باشد.



شکل ۵-۱۳ - تابیر اجرای دستور XLAT و نمایش چکونگی آدرس دهن آن

۶-۵- عوض کردن سگمنت در یک دستور (Segment Override Prefix)

اعوض کردن ثبات سگمنت تقریباً در تمامی دستور ۸۰۸۶ با هر مدل آدرس دهنی اجازه داده شده است. این عمل بوسیله اضافه کردن سگمنت بصورت پیشوند به آدرس امکان پذیر می باشد. و در زبان ماشین هم یک بایت به کد عملیاتی اضافه می شود. بعنوان مثال در دستور $MOV AX,[DI]$ معمولاً در بخش دیتا سگمنت (DS) آدرس دهنی می شود می توان این دستور را به شکل $MOV AX,ES:[DI]$ تغییر و عوض کرد. که در این صورت آدرس دهنی در بخش اکسترا سگمنت انجام می پذیرد یعنی :

$$\text{آدرس فیزیکی} = ES \times 10H + DI$$

جدول ۵-۱۲ لیست بعضی از دستوراتی که سگمنت را عوض می کنند آورده است. دقیق داشته باشید که در اغلب این دستورات دیتا را در هر سگمنتی می توان یافت.

جدول ۵-۱۲- دستوراتی که سگمنت را عوض می کنند

فرم سمبولیک	سگمنتی که در اختیار قرار می گیرد	سگمنتی که معمولاً باید در اختیار قرار می گرفت
$MOV AX, DS:[BP]$	DS دیتا سگمنت	SS سگمنت پشته بود
$MOV AX, ES:[BP]$	ES اکسترا سگمنت	SS سگمنت پشته بود
$MOV AX, SS:[DI]$	SS سگمنت پشته	DS دیتا سگمنت
$MOV AX, CS:[SI]$	CS کد سگمنت	DS دیتا سگمنت
$MOV AX, ES:LIST$	ES اکسترا سگمنت	DS دیتا سگمنت

۷-۵-۱- احکام اسambilر (Assembler Directive)

احکام اسambilر دستوراتی هستند که هدایت کننده اسambilی بمنظور جهت دهی و پیشبرد برنامه اسambilی هستند، در این قسمت، متن نشان می دهد که چگونه احکام اسambilر را بکار بگیرید و هم اینکه چگونه برنامه به زبان اسambilی بنویسید. این احکام زیاد هستند جدول ذیر را در نظر بگیرید.

جدول ۵-۱۳- راهنمای اسambilی

WORD	FUNCTION
ALIGN	از یک محل حافظه به آدرس زوج شروع می کند.
ASSUMME	نشان می دهد که سکمت کجا واقع شده است.
AT	سکمت را در حافظه ذخیره می کند.
BYTE	همانند یک اپرنده باقی برخورد می کند.
DB	تعریف یک بایت (۸ بیتی) است.
DD	تعریف یک دیتا دو کلمه ای (32bit) است.
DQ	تعریف چهار کلمه ای است (64bit).
DT	تعریف دیتا ۱۰ بایتی است (80bit).
DUP	کاراکترهای زیری دو برابر شده هستند.
DW	تعریف word کلمه (۱۶ بیتی) است.
END	نشان دهنده پایان لیست است.
ENDP	نشان دهنده پایان زیربرنامه است.
ENDS	نشان دهنده پایان سکمت است.
EQU	معادل یا مساوی است.
FAR	بیان عوض شدن سکمت است.
NEAR	وجود دینا در همین سکمت است.
ORG	مشخص کردن آدرس است.
PROC	پینزله شروع یا لول سکمت است.
PTR	تعریف شروع زیر برنامه است.
SEGMENT	پینزله اشاره گر به حافظه است.
STACK	طرح شروع سکمت است.
THIS	نشان دهنده قسمت پشته است (SS).
	بکار می رود برای بیان THIS WORD با THIS BYTE

توجه داشته باشید که اغلب این کلمات کلیدی (KEY WORD) توسط اغلب اسambilی های ۸۰۸۶/۸۸ پذیرفته شده اند.

۷-۵-۲- راهنمای :

اغلب اسambilی های ۸۰۸۶ اغلب کلمات جدول ۵-۱۳ را تشخیص می دهند و متوجه می شوند که با یک اپرنده چگونه برخورد نمایند یا یک برنامه را چگونه اجرا نمایند. در مثال ۵-۶ استفاده از DD, DW, DB بیان شده است.

Example 5-6

DATA-ONE	DB	1,2,3	سه بایت تعریف شده بمنزله ۱۰۲و۳
	DB	45H	یک بایت عدد ۴۵ را در پایه ۱۶ نشان می دهد
	DB	'A'	کد اسکی'A' را اعلام می دارد
	DB	11110000B	یک بایت را بصورت باینری معرفی شده
	DW	12,13	دو کلمه ۱۶ بیتی معرفی شده ۱۰۱۲و۱۳
DATA-TWO	DW	LIST	آفست آدرسی بنام LIST تعریف شده
	DW	3456H	یک کلمه ۱۶ بیتی با مقدار H3456 معرفی شده
	DD	0000FFFFH	دو کلمه ۱۶ بیتی معرفی شده

همانگونه که در مثال ۵-۶ نشان داده شده است مشخصه های مثل DB و DW یا DD قادرند یک بایت یا یک کلمه ۱۶ بیتی یا دو کلمه ۱۶ بیتی از دیتا را در حافظه ذخیره کنند. حافظه را می توان حتی برای دیتاهایی که در آتی بدست می آوریم توسط DD, DW, DB بوسیله DUP راهنمایی کرد.

DUP به اسملبر نشان می دهد که یک کارکتری که درون () گذاشته شده است باید دو برابر شود. عدد قبل از پرانتز تعداد دفعات دو برابر شدن را نشان می دهد. اگر درون پرانتز علامت ؟ قرار داده شود، حالا دیگر معنی دو برابر شدن کارکتر را نمی دهد بلکه می گوید حافظه بدون تغییر ترک شود به مثال ۵-۷ توجه کنید بعضی از مثال های راهنمای DUP را نشان می دهد.

Example 5-7

LIST-A	DB	?	یک بایت برای LIST-A ذخیره می کند
LIST-B	DB	10DUP(?)	۲۰ بایت ذخیره می کند
	ALIGN		یک آدرس زوج می سازد
LIST-C	DW	10DUP(?)	۲۰ کلمه ذخیره می کند
LIST-D	DD	22DUP(?)	۲۲ تا دو کلمه ذخیره می کند
ZEROS	DB	100DUP(0)	۱۰۰ عدد صفر در ذخیره می کند

۵-۷-۲-THIS و EQU راهنمای

عبارت EQU (Equavte) برای ارزیابی دو داده عددی که ASCII یا ارزیابی یک برچسب (Label) با برچسب دیگر بکار می رود. دستور EQU اسملبر برنامه را مجبور می کند تا پس از زمانی آنرا اشکال زدائی کند مثال ۵-۸ چند نوع مختلف عبارت EQU را نشان می دهد :

Example 5-8

TEN	EQU	10	;	برچسب های TEN را مساوی 10 می کند
LETTER-A	EQU	'A'	;	برچسب LETTER-A را مساوی 'A' می کند
COUNT	EQU	99	;	برچسب COUNT را مساوی 99 می کند
NUMBER-10	EQU	TEN	;	برچسب NUMBER-10 را مساوی 10 می کند

راهنمای THIS یک نام برچسب را بدون تعریف یک محل حافظه جدیدی نشان می دهد. عنوان مثال وقتی که یک پشته برقرار می شود ثبات اشاره گر به پشته (SP) دارای مقدار آدرس TOP پشته می شود این کار توسط راهنمای EQU, THIS انجام می شود مثال ۵-۹ را با دقت بینید، چگونه واژه WORD را با THIS و EQU نشانده TOP پشته می باشد.

Example 5-9

STACK-MEM	DW	50 DUP(?)	;	کلمه رزرو می کند
STACK-TOP	EQU	THIS WORD	;	است TOP OF STACK
	DB	10	;	این ۱۰ را اینجا ذخیره کن
DATA	EQU	THIS BYTE	;	است DATA=10

۳-۷-۵- سازمان حافظه

اسمبلر برای خبر دادن از وجود سگمنت های مختلف راهنمای خاص را بکار می گیرد. اسامی راهنمای سگمنت ها شروع هر سگمنت و کلمه های ENDS نشان دهنده پایان آنهاست. وقتی که یک SEGMENT و ENDS بیان گر شروع و پایان سگمنت باشد کلمه ASSUME به اسмبلر اطلاع می دهد که کدام یک از DS یا CS یا SS ساخته شده است.

مثال ۵-۱۰ چگونگی کاربرد این راهنمای را نشان می دهد. سیستم عامل معمولاً ثبات های کد سگمنت (DS) و اکسترا سگمنت (ES) را پر نمی کند. این ها باید با شروع هر برنامه پر شوند. که در این صورت ثبات AX با آدرس سگمنت دیتا پر می شود و سپس با دستور MOV به DS ریخته خواهد شد. توجه داشته باشید که راهنمای OFFSET در اینجا کاربرد ندارد. اسмبلر بطور آutomاتیک OFFSET را برای هر برچسب تعریف شده عنوان اسم سگمنت اعمال می کند. ES نیز به همین طریق که ذکر شد پر می شود. اگر مجبور باشیم که سگمنت را در هر بخش جزئی از حافظه قرار داده شود از راهنمای AT استفاده می شود.

مثال ۵-۱۱ نشان می دهد که چگونه عبارت AT سگمنت دیتا را بوسیله پر کردن ثبات سگمنت از 0100H در 0100H قرار می دهد. و عملآ محمل حافظه آدرس H0100 خواهد شد.

مثال ۵-۱۰ نشان دهنده چگونگی استفاده از ASSUME و ENDS-SEGMENT

```

Example 5-10 ;  

DATA SEGMENT ; اسم سگمنت  

; ;  

LIST-A DB 10 DUP(?) ; ۱۰ بایت رزرو می کند ;  

LIST-B DB 12 DUP(6) ; از ۶تا ۱۲ کلمه ذخیره می کند ;  

; ;  

DATA ENDS ; بیان سگمنت  

; ;  

EXTRA SEGMENT ;  

; ;  

STRING DB 'HELLO' ; HELLO را ذخیره می کند ;  

; ;  

EXTR ENDS ;  

; ;  

STACK SEGMENT ;  

; ;  

DW 50 DUP(?) ; ۵۰ کلمه رزرو می کند ;  

; ;  

STACK ENDS ;  

; ;  

CODE SEGMENT ;  

; ;  

ASSUME CS : CODE DS : DATA  

ASSUME EX : EXTRA SS : STACK  

; ;  

BEGIN :  

    MOV AX,DATA ; برای DATA را از DS برمی کند  

    MOV DS,AX ;  

    MOV AX,EXTRA ; برای EXTRA را از ES برمی کند  

    MOV ES,AX ;  

    --- ---  

    --- ---  

    --- ---  

CODE ENDS ;  

END BEGIN

```

یک نمونه برنامه برای نشان دادن راهنمایی ORG, AT

```

Example 5-11 ;  

; ;  

DATA SEGMENT AT 0100H ; اختصاص دادن آدرس سگمنت  

; ;  

ORG 100H ; اختصاص دادن آدرس به OFFSET  

; ;  

LIST-A DW 10 DUP(?) ; ۱۰ کلمه ۱۶ بیتی رزرو کردن ;  

; ;  

DATA ENDS

```

در مثال بالانه تنها دیتا در آدرس سگمنت ذخیره شد. بلکه LIST-A همچنین شروعش را در آدرس 0100H قرار می‌دهد. که با قرار دادن DS=100H محل فیزیکی حافظه 01100H خواهد شد. در اغلب برنامه‌ها واژه‌های AT، ORG بکار برده نمی‌شود.

۴-۵-۷-۴- زیربرنامه (SUBROUTIN/PROCEDURE)

زیربرنامه بواسیله راهنمای PROC و ENDP نشان داده می‌شود. نام شروع زیربرنامه همراه با نوع زیربرنامه (NEAR FAR) را نشان می‌دهد. ENDS نشان دهنده آخر زیربرنامه است. بعنوان مثال تصور کنید که محتوای ثبات‌های DX، CX، BX باید با هم جمع شود و حاصل جمع در AX ریخته شود. این کار توسط زیربرنامه ADDEM در مثال ۴-۵-۱۲ بطور کامل نشان داده شده است.

زیربرنامه ای که محتوای DX، CX، BX را جمع می‌کند و حاصل جمع را در AX ذخیره می‌کند.

```

;               معرفی زیربرنامه
ADDEM      PROC      FAR      ;
;               مراجعه از زیربرنامه به برنامه اصلی
    MOV      AX,BX      ;
    ADD      AX,CX      ;
    ADD      AX,DX      ;
    RET      ;           پایان زیربرنامه
;
ADDEM      ENDP      ;

```

همانگونه که ملاحظه می‌فرمائید زیربرنامه با نام ADDEM و از نوع FAR معرفی شده است. معنی آن این است که در هر محلی از حافظه می‌تواند قرار داده شود. یعنی هم باید آدرس OFFSET و هم آدرس سگمنت تعیین شود. اگر بصورت NEAR معرفی شود باید در همان سگمنت قرار داده شود و فقط آدرس OFFSET آن مشخص گردد. اگر دنبال LABEL دو نقطه : (COLON) ظاهر شود، زیربرنامه از نوع NEAR است ولی اگر : ظاهر نشود از نوع FAR است. طبیعی است که اجرای زیربرنامه NEAR راحت‌تر و سریع‌تر صورت می‌گیرد.

۴-۵-۷-۵- آدرس شروع و اشاره گر (OFFSET and PTR)

راهنمای OFFSET نشان دهنده یک آدرس شروع (OFFSET) به اسمبلر است. راهنمای PTR که مخفف POINTER است نوع دیتا (WORD یا BYTE) را به اسمبلر معرفی می‌کند. بعنوان مثال آدرس DATA را توسط دستور MOV SI, OFFSET DATA در SI ریخته، در صورتیکه دستور MOV SI, DATA محتوای محلی از حافظه که آدرسش DATA است درون SI می‌ریزد ولی OFFSET آدرس دیتا را درون SI می‌ریزد.

فصل پنجم

زمانیکه نوع (TYPE) دیتا برای اسمبیلر مشخص نباشد از دستور PTR استفاده می شود، بعنوان مثال تصور کنید که دستور زیر در یک برنامه ظاهر شود $MOV [BX],2$. این دستور یک بایت ۱۶ بیت دیتا که مقدرا آن ۲ است در حافظه ذخیره می کند. راهنمای PTR با کلمه WORD که $MOV BYTE PTR[BX],2$ باشد شد. لذا دستور خواهد شد $MOV WORD PTR[BX],2$. اما ۲ بکار می گیرد. یک محل ۱۶ بیتی از حافظه را که BX آدرس آنرا می دهد برای ذخیره کردن عدد ۲ بکار می گیرد.

۶-۵-۷-۶- یک نمونه برنامه

مثال ۵-۱۳ یک نمونه برنامه است که یک BLOCK اطلاعات را به BLOCK دیگری منتقل می کند. برای اینکار از دستور MOVSB استفاده می کند. این مثال بکارگیری خیلی از راهنمایی که در این بخش صحبت شد را نشان می دهد. توجه دارید که COUNT مساوی ۱۰۰ است که با استفاده از دستور MOV CX,COUNT در شمارنده قرار می گیرد. همچنین توجه دارید که آخرین دستور INT 20H است. این دستور نوعی وقفه است که کنترل کامپیوتر را به سیستم DOS برمی گرداند. در آینده راجع وقفه های مختلف در ۸۰۸۶ صحبت خواهیم کرد.

مثالی از زیربرنامه نویسی که ۱۰۰ بایت از بلوك ۱ را به بلوك ۲ منتقل می کند.

Example	5-13	
COUNT	EQU	100 ; تعریف شمارنده
DATA	SEGMENT	
BLOCK-1	DB	100 DUP(?) ; ۲۰۰ بایت دزرو می کند
DATA	ENDS	
EXTRA	SEGMENT	
BLOCK-2	DB	100 DUP(?) ; ۲۰۰ بایت دزرو می کند
EXTRA	EDNS	
CODE	SEGMENT ASSUME	CS:CODE, DS:DATA, ES:EXTRA
BEGIN	MOV AX, DATA	برگردان DS از آدرس مربوطه
	MOV DS, AX	برگردان ES از آدرس مربوطه
	MOV AX, EXTRA	
	MOV ES, AX	
	MOV SI, OFFSET BLOCK-1	آدرس دینا مینع
	MOV DI, OFFSET BLOCK-2	آدرس مقصد
	CLD	انتخاب خودآفرابشی آدرسها
	MOV CX, COUNT	شارنده، را برابر می کند
	REP	
	MOVSB	
	INT 20H	۱۰۰ بایت انتقال می دهد
CODE	ENDS	DOS برمی گردد به
	END	
	BEGIN	

سوالات و مسائل

- ۱- اولین بایت هر دستور ۸۰۸۶ چه نام دارد؟
 - ۲- هدف از بیت های D, W چیست؟ شرح دهد.
 - ۳- میدان MOD در یک دستور به زبان ماشین چه اطلاعاتی دارد؟
 - ۴- اگر میدان REG یک دستور ۰۱۰ و $W=0$ چه ثباتی توسط دستور انتخاب می شود؟
 - ۵- اگر مقدار R/M=001 و MOD=00 باشد چه مد آدرس دهی تعریف شده است؟
 - ۶- بیان کنید سگمنت معمولی (Default Segment) برای هر یک از ثبات های زیر کدام است؟
- | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|
| الف: | SI | هـ | DI | دـ | BP | جـ | BX | بـ | SP |
|------|----|----|----|----|----|----|----|----|----|
- ۷- دستور 8B07H را به زبان اسambilی تبدیل کنید.
 - ۸- دستور 8B1E9004CH را به زبان اسambilی تبدیل کنید.
 - ۹- دستور MOV SI,[BX+2] را به زبان ماشین تبدیل کنید.
 - ۱۰- آیا دستور MIOV CS,AX مجاز است پاسخ خود را شرح دهد.
 - ۱۱- چه نوع دستور MOV دارای ۶ بایت طول خواهد بود.
 - ۱۲- دستورات POP, PUSH همیشه چند بیت در حافظه قرار می دهند؟
 - ۱۳- چه ثبات سگمنتی را نمی توان از حافظه POP کرد؟
 - ۱۴- شرح دهید هنگام اجرای دستور PUSH BX چه عملی اتفاق می افتد، فرض کنید که SS=0200H, SP=100H باشد BL, BH در چه آدرسهای ذخیره می شود؟
 - ۱۵- عملکرد دستور LEA BX, DATA را با دستور MOV BX, DATA مقایسه کنید؟
 - ۱۶- شرح دهید دستور LDS BX, DATA چگونه عمل می کند؟
 - ۱۷- نقش فلیپ فلاپ D چیست؟
 - ۱۸- دستوراتی که باعث تغییر وضعیت D می شوند کدام است؟
 - ۱۹- برنامه ای بنویسید که ۱۲ بایت اطلاعات را از محلی از حافظه به آدرس SOURCE به محل دیگر از حافظه به آدرس DEST که هر دوی آنها دارای ۳۲ بیت آدرس دارند منتقل نماید؟
 - ۲۰- با استفاده از راهنمای اسambilی ۳۰ بایت از حافظه را برای LIST1 رزرو کنید.
 - ۲۱- توضیح دهید چرا PTR در دستوراتی مثل MOV WORD PTR[DI],3 ضروری است.

فصل ششم

«دستورات ریاضی و منطقی»

در این فصل دستوراتی را آزمایش و بررسی می کنیم که در مجموعه دستورات ۸۰۸۶/۸۸ وجود دارند و عملیات ریاضی و منطقی را انجام می دهند. دستورات ریاضی عبارتند از جمع، تفریق، ضرب، تقسیم، مقایسه، منفی کردن، افزودن و کاستن. دستورات منطقی در برگیرنده AND، OR، XOR، NOT، شیفت ها، چرخش ها و دستورات مقایسه منطقی بنام TEST می باشد.

ما همچنین سعی می کنیم دستورات مقایسه ای رشته ای را نیز معرفی کنیم. زیرا این دستورات برای نمونه برداری و مقایسه دیتاهای جدولی خیلی مفید هستند برای مقایسه دیتاهای دو ناحیه از حافظه برای تطبیق و یا عدم تطبیق شرط کارآئی خوبی دارند.

۱-۶- دستورات جمع، تفریق و مقایسه :

در بین دستورات ریاضی هر ریزپردازنده ای دستور جمع و تفریق و مقایسه بجشم می خورد. ریزپردازنده ۸۰۸۶ نیز از این قاعده مستثنی نیست. در این بخش می خواهیم این دستورات را معرفی کنیم و نحوه دستکاری دیتا در ثبات ها و حافظه ها را آزمایش دهیم.

۱-۶-۱- دستور جمع (ADD) :

دستور جمع در ریزپردازنده ۸۰۸۶ صورتهای مختلفی دارد. در این بخش ما جزئیات کاربرد دستور جمع ۸ بیتی و ۱۶ بیتی بعلاوه دستور افزودن به دیتا را بررسی می کنیم. و در بخش ۶-۳ صور دیگر دستور جمع مثل جمع BCD و ASCII بررسی می شود. جدول ۶-۱ مدهای آدرس دهی مختلفی که برای جمع استفاده می شوند را نشان می دهد و از آنجاییکه متجاوز از ۱۰۰۰ دستور جمع مختلف برای ریزپردازنده ۸۰۸۶ وجود دارد بدیهی است که نمایش تمام صور آن دستورات در این جدول امکان پذیر نیست.

مطلوبی که اینجا لازم است بیادآوری شود، اینکه محتوای ثبات‌های سگمنت را نمی‌توان در جمع شرکت داد. و یا یک محل از حافظه را با محل دیگر آن نمی‌توان جمع کرد. قبل‌اهم اعلام شد که محتوای ثبات‌های سگمنت را فقط ممکن است با دستور MOV و PUSH و POP انتقال داد.

جدول ۱-۶- دستورات جمع

دستورات	جدول ۱-۶ دستورات جمع شرح عملیات
ADD AL,BL	$AL \leftarrow AL + BL$
ADD CX,DI	$CX \leftarrow CX + DI$
ADD BL,44H	$BL \leftarrow BL + 44H$
ADD BX,345FH	$BX \leftarrow BX + 345FH$
ADD [BX],AL	$M[DS \times 10H + BX] \leftarrow M[DS \times 10H + BX] + AL$
ADD CL,[BP]	$CL \leftarrow M[SS \times 10H + BP] + CL$
ADD BX,[SI+2]	$BX \leftarrow BX + M[DS \times 10H + SI + 2]$
ADD CL,TEMP	$CL \leftarrow CL + M[DS \times 10H + TEMP]$
ADD BX,TEMP[DI]	$BX \leftarrow BX + M[DS \times 10H + TEMP + DI]$
ADD [BX+DI],DL	$M[DS \times 10H + BX + DI] \leftarrow M[DS \times 10H + BX + DI] + DL$

۱-۱-۶- جمع ثباتی :

مثال ۱-۶ یک نمونه برنامه کوچکی را ارائه می‌کند که دستورات جمع را برای بعضی از ثبات‌ها نشان می‌دهد. دقت کنید این قطعه برنامه محتوای DX, CX, BX را جمع می‌کند و حاصل آن را در AX می‌ریزد. همچنین دقت دارید که بعد از هر جمع ریزبردازنده محتوای ثبات پرچم را تازه می‌کند. این مطلب مهمی است که همواره بیاد داشته باشید که دستورات ریاضی و منطقی همیشه محتوای ثبات پرچم را تغییر می‌دهند.

دستور جمع از هر نوعی که باشد روی پرچمهای علامت (SIGN), نقلی (Carry), صفر (Zero)، نقلی کمکی (Auxiliary carry) و برابری (Parity) و سرفونگی (Overflow) تاثیری می‌گذارد.

Example 6-1
 ADD AX,BX
 ADD AX,CX
 ADD AX,DX

۱-۱-۶- جمع فوری :

در مثال ۲-۶ که یک جمع ۸ بیتی را نشان می‌دهد، وضعیت بیت‌های پرچم را نیز مشخص کرده است. در ابتدا عدد ۱۲H درون ثبات DL قرار داده می‌شود سپس با ۳۳H جمع می‌شود (از مدد

آدرس دهی فوری استفاده می کند) و حاصل جمع درون DL قرار می گیرد. مانند هر جمعی پرچمها تغییر می کنند و حاصل تغییر در این مثال به شرح زیر خواهد بود.

S=0, A=0, C=0, Z=0, O=0, P=0

Example 6-2

```
MOV  DL,12H
ADD  DL,33H
```

۳-۱-۱-۶- جمع ثبات با محتوای حافظه :

فرض کنید لازم باشد در یک کاربرد خاصی محتوای محلی از حافظه را با ثبات AL جمع کنیم، مثال ۳-۶ چنین مسئله ای را نشان می دهد :

Example 6-3

```
MOV  DI,OffsetData
MOV  AL,0
ADD  AL,[DI]
ADD  AL,[DI+1]
```

نحوه عمل به اینصورت است که ابتدا آدرس دیتا توسط شبه دستور OffsetData در DI قرار می گیرد چون DI با DS آدرس دهی می کند، پس دیتا در سگمنت دیتا قرار دارد. معمولاً همیشه دیتا در سگمنت دیتا قرار دارد مگر اینکه Offset آدرس را از BP یا SP بگیرید یا بوسیله دستور مربوط به فرم پیشوند سگمنت را عوض کنید.

۴-۱-۱-۶- جمع آرایه ای

فرض کنید رشته ای از دیتا داریم بنام ARRAY که ۱۰ بایت حافظه را اشغال کرده است با ۰ تا ۹ نام گذاری شده است.

توسط مثال ۴-۶ می خواهیم محتوای عناصر ۳ و ۵ و ۷ را با هم جمع کنیم، ابتدا درون AL صفر قرار می دهیم و عدد ۳ را در SI می ریزیم. در اولین جمع عنصر سوم با صفر جمع می شود و در دستورات بعدی، سه عنصر مذکور با هم جمع می شوند.

Example 6-4

```
MOV  AL,0
MOV  SI,3
ADD  AL,ARRAY[SI]
ADD  AL,ARRAY[SI+2]
ADD  AL,ARRAY[SI+4]
```

۴-۱-۱-۶-۵- جمع با یک (INCEREMENT ADDITION)

جمع کردن مقداری با یک (INC) با هر مر آدرس دهی ممکن است بکار رود بغیر از آدرس دهی ثبات های سگمنت. جدول ۶-۲ لیست تعدادی از دستورات بعلاوه یک را نشان

فصل ششم

می دهد. بدیهی است این تمام دستورات موجود در این زمینه نیست. در مثال ۶-۵ مشخص شده که مثال ۳-۶ را می توان با استفاده از دستور INC نیز نوشت.

(INCREMENT)

جدول ۶-۲ دستورات جمع با یک.

دستورات	توضیحات
INC BL	$BL \leftarrow BL + 1$
INC SP	$SP \leftarrow SP + 1$
INC BYTE PTR[BX]	یکی به محتوای حافظه $[DS \times 10H + BX]$ می افزاید
INC WORD PTR[SI]	یکی به محتوای حافظه ۱۶ بیتی $[DS \times 10H + BX]$ می افزاید

یعنی با INC آدرس بعدی را در DI تولید می کنیم. یک تفاوت ظرفی بین INC و سایر ADD ها وجود دارد و آن اینکه INC روی پرچم نقلی (CARRY) اثر ندارد. در صورتیکه بقیه پرچمها را متاثر می سازد.

Example 6-5

```

MOV DI,OffsetData
MOV AL,0
ADD AL,[DI]
INC DI
ADD AL,[DI]

```

۶-۱-۶-۲- تفریق (SUBTRACTION)

در این قسمت راجع به دستوراتی که عمل تفریق (SUB) روی دیتاهای ۸ بیتی و ۱۶ بیتی انجام می دهند به تفصیل بحث می شود، همچنین راجع به کاستن از یک دیتا بمیزان یک واحد (DECREMENT) نیز صحبت خواهیم کرد.

در بخش ۶-۴ نیز راجع به تفریق BCD و تفریق در کد ASCII بحث خواهد شد.

در جدول ۶-۳ لیستی از دستورات تفریق (SUB) با مدهای مختلف آدرس دهی ارائه شده است. همانند جمع حدود ۱۰۰۰ دستور در زمینه تفریق نیز وجود دارد که لیست تمامی آنها کار آسانی نیست. آنچه را که نمی شود روی آنها عمل تفریق انجام داد روی محتوای ثبات های سگمنت است، همچنین امکان تفریق دو محل حافظه نیز وجود ندارد.

تفریق نیز روی تمام بیت های پرچم اثر هی گذارد.

۶-۱-۶-۱- تفریق محتوای ثبات ها

مثال ۶-۶ یک برنامه ساده ای است که کاربرد دستوراتی که عمل تفریق روی ثبات انجام می دهد را نشان داده است. دقت کنید که محتوای CX, DX از ثبات BX کم می شود پس از هر

فصل ششم

عمل تفريقي ثبات پرچمها تغيير پيدا مي کند يعني مانند جمع تفريقي روی همه پرچمها اثر مي گذارد.

Example 6-6

SUB	BX,CX
SUB	BX,DX

جدول ۶-۳ دستورات تفريقي

دستورات	شرح عمليات
SUB CL,BL	CL \leftarrow CL-BL
SUB AX,SP	AX \leftarrow AX-SP
SUB DH,6FH	DH \leftarrow DH-6FH
SUB AX,0CCCH	AX \leftarrow AX-0CCCH
SUB [DI],CH	M[DI] \leftarrow M[DI]+CH
SUB CH,[BP]	CH \leftarrow CH-M[SS \times 10H+BP]
SUB AH,TEMP	AH \leftarrow AH-M[DS \times 10H+TEMP]
SUB DI,TEMP[BX]	DI \leftarrow DI-M[DS \times 10H+TEMP+BX]

۱-۲-۶-۱- تفريقي محتواي ثبات ها :

مثال ۶-۶ يك برنامه ساده اي است که كاربرد دستوراتي که عمل تفريقي روی ثبات انجام مي دهد را نشان داده است. دقت کنيد که محتواي CX و DX از ثبات BX کم مي شود پس از هر عمل تفريقي ثبات پرچمها تغيير پيدا مي کند يعني مانند جمع و تفريقي روی همه پرچمها اثر مي گذارد.

۱-۲-۶-۲- تفريقي با آدرس دهي فوري:

مثال ۶-۷ يك تفريقي ۸ بيتی با آدرس دهي فوري انجام مي دهد. همه پرچم ها را متاثر مي کند. ابتدا عدد 22H در CH وارد مي شود و در دستور بعدی با 44H وارد تفريقي مي شود و پس از اجرای تفريقي پرچمها بصورت زير داراي مقدار مي شوند :

Example 6-7

MOV	CH,22H
SUB	CH,44H

Z=0, C=1, A=1, S=1, P=1, O=0

توجه داريد که پرچمهاي C و A نگهدارنده دو بر يك قرض گرفته شده مي باشند همانند رقم نقلی پس از هر جمع بوجود مي آيند. مسلماً در عمل تفريقي سررفتگی بوجود نمي آيد چون اعداد از هم کم مي شوند.

۳-۱-۲-۶- کاستن مقدار یک واحد از یک دیتا (DECREMENT)

کاستن به میزان یک واحد از یک دیتا با دستور DEC انجام می شود جدول شماره ۴-۶ لیست تعدادی از دستورات DEC را نشان می دهد. مجدداً یادآور می شوم که لیست کلیه دستورات در این زمینه ممکن نیست. زیرا تعداد آنها زیاد است.

جدول ۴-۶ دستورات کاستن (DECEREMENT)

دستورات	توضیحات
DEC BH	$BH \leftarrow BH - 1$
DEC BP	$BP \leftarrow BP - 1$
DEC WORD PTR[DI]	از محلی از حافظه که کلمه ۱۶ بیتی است و در DS فرار دارد آدرس می دهد یک واحد کاسته می شود.
DEC WORD PTR[SI+2]	از محل ۱۶ بیتی از حافظه در DS که بوسیله SI+2 آدرس دهنده می شود یک واحد کاسته می شود.

۴-۱-۲-۶- جمع با رقم نقلی (ADD WITH CARRY)

جمع دو دیتا بعلاوه Carry زمانی کاربرد دارد که نیاز به جمع دو عدد بیش از ۱۶ بیت باشد. جدول شماره ۵-۶ تعدادی از دستورات جمع با رقم نقلی را لیست کرده است (ADC).

شرح هر دستور در مقابل آن آورده شده است. دستور ADC نیز مانند ADD روی تمام پرگمها تاثیر می گذارد.

جدول ۴-۵- جمع دو دیتا با رقم نقلی

دستورات	شرح
ADC AL,AH	$AL \leftarrow AL + AH + C$
ADC CX,BX	$CX \leftarrow CX + BX + C$
ADC [BX],AL	$M[DS \times 10H + BX] \leftarrow M[DS \times 10H + BX] + AL + C$
ADC BX,[BP+2]	$BX \leftarrow BX + M[SS \times 10H + BP + 2] + C$

فرض کنید عدد ۳۲ بیتی که در ثباتهای AX, BX, DX نگهداری می شود را بخواهیم با عدد ۳۲ بیتی دیگری که در ثباتهای CX, CX نگهداری می شود جمع کنیم. در اینگونه موارد ADC کارساز می شود. مثال ۶-۸ این عمل را انجام می دهد. دقت کنید ابتدا بخش کم ارزش دیتابا عی خ، BX با هم جمع می شوند سپس بخش با ارزش دیتابا با هم و با رقم نقلی احتمالی پیش آمده از جمع بخش کم ارزش جمع می گردد. و حاصل جمع ۳۲ بیتی در ثباتهای BX, AX, DX قرار می گیرد.

Example 6-8

ADD	BX,DX
ADC	AX,CX

۶-۱-۲-۵- تفريقي با رقم قرض گرفته شده :

دستور تفريقي با رقم قرض گرفته شده بيت موجود در فيليپ فلاپ Cary را از اپرند ديتا کم می کند اين بيت همان رقم قرض گرفته شده است، اين دستور کاربردش زمانی است که دو عدد بيش از ۱۶ بيت را بخواهيم از هم کم کنيم. جدول ۶-۶ ليستي از دستورات تفريقي با رقم قرض گرفته شده را (SBB) نشان می دهد. همانند دستور SUB اين دستور نيز روی تمامي پرجمها (FLAGS) اثر می گذارد. اگر بخواهيم ۳۲ بيت ديتائي که در DI, SI قرار دارند را از ۳۲ بيت ديگر که در AX, BX قرار دارند کم کنيم. پس از اينکه ۱۶ بيت اول را کم کردیم شاید رقم قرض گرفته شده داشته باشيم ۱۶ بيتی دوم را باید با دستور SBB کم می کنيم که تاثير رقم قرض گرفته شده برای ۱۶ بيت اوليه را هم دخالت دهد. مثال ۶-۹ را ملاحظه کنيد اين عمل را نشان می دهد، همانگونه که ملاحظه می فرمائيد ۱۶ بيت کم ارزش را نياز ندارد که با SBB از هم کم کند.

جدول ۶-۶- تفريقي با رقم قرض گرفته شده

دستورات	شرح
SBB AH,AL	AH \leftarrow AH-AL-C
SBB AX,BX	AX \leftarrow AX-BX-C
SBB CL,3	CL \leftarrow CL-3-C
SBB [DI],AL	M[DS \times 10H+DI] \leftarrow M[DS \times 10H+DI]-AL-C
SBB DI,[BP+2]	DI \leftarrow DI-M[SS \times 10H+BP+2]-C

Example 6-9
 SUB BX,SI
 SBB AX,DI

۶-۱-۳- مقاييسه (COMPARISON)

دستور مقاييسه (CMP)، عملاً يك تفريقي است که هيچکدام از ديتاها را تغيير نمی دهد، فقط پرجمها را متاثر می سازد، مهمترین کاربرد آن برای بررسی محتوای ثبات یا يک محل حافظه است که آیا دارای مقدار مشخصی هست یا خير، معمولاً بدنبال اين دستور يك JMP شرطي خواهيم داشت که با بررسی پرجمها تحقق پيدا خواهد کرد.

جدول ۶-۷ تعدادي از دستورات CMP را ليست کرده است که مثل دستورات جمع و تفريقي از مدهای مختلف آدرس دهی استفاده می کنند. تنها مد آدرس دهی که مجاز نیست مقاييسه دو محل حافظه با هم و یا مقاييسه محتوای ثبات سگمنت ها می باشد.

در مثال ۰-۶ نشان داده شده که کاربرد CMP چگونه است. اگر محتوای AL برابر ۱۰H باشد حاصل تفربیت انجام شده صفر می شود و پرجم Z=۱ خواهد شد. دستور شرطی سطر دوم (در آینده مطالعه خواهیم کرد) یک دستور شرطی روی پرجم Z است می گوید اگر Z=۱ شده به محلی که "CNTEN" مشخص می کند پرش کن والا ادامه بده.

جدول ۶-۲- دستورات مقایسه

دستورات	شرح
CMP CL,BL	BL را مقایسه می کند بدون اینکه محتوای هیجکدام تغییر کند.
CMP AX,SP	AX و SP را مقایسه می کند بدون اینکه محتوای هیجکدام تغییر کند.
CMP AX,0CCCCH	AX و ۰CCCCH را مقایسه می کند بدون اینکه محتوای AX تغییر کند.
CMP [DI],CH	M DS _x 10+DI] و CH را مقایسه می کند بدون اینکه محتوای CH با حافظه تغییر کند.
CMP CH,[BP]	CH و M SS _x 10+BP] را مقایسه می کند بدون اینکه محتوای CH با حافظه تغییر کند.
CMP AH,TEMP	AH و M DS _x 10+TEMP] را مقایسه می کند بدون اینکه محتوای AH با حافظه تغییر کند.
CMP DI,TEMP[BX]	DI و M DS _x 10+BX+TEMP] را مقایسه می کند بدون اینکه محتوای DI با حافظه تغییر کند.

Example 6-10

```
CMP AL,10H
JZ CNTEN
```

اگر محتوای AL دیتائی بغير از 10H باشد حاصل تفربیت صفر نمی شود و پرجم Z برابر صفر خواهد شد، شرط تحقق نمی یابد و برنامه را ادامه می دهد.

۶-۲- ضرب و تقسیم :

ریزپردازنده های ۸ بیتی قادر نیستند عملیات ضرب و تقسیم را مستقیماً انجام دهند. بلکه باید این عملیات را بصورت برنامه ای درآورد آنوقت اجرا نمود، اما از ریزپردازنده های ۱۶ بیتی مثل ۸۰۸۶ اجرای این دستورات ممکن گردید. زیرا مدارات ضرب و تقسیم در آنها پیش بینی شد و اجرای این دستورات میسر گردید. ریزپردازنده ۸۰۸۶/۸۰۸۸ قادر است هم ضرب و تقسیم اعداد ۸ بیتی و ۱۶ بیتی و بعلاوه هم می تواند عملیات را روی دیتاهای علامت دار و بدون علامت انجام دهد. بدیهی است برنامه هایی که برای اینگونه ریزپردازنده ها نوشته می شود کوتاهتر و سریعتر از برنامه های ریزپردازنده های قبلی اجرا می شود.

۶-۲-۱- دستور ضرب (MULTIPLICATION)

دستور ضرب می تواند روی دیتاهای ۸ بیتی یا ۱۶ بیتی و همچنین روی دیتاهای علامت دار با دستور IMUL و یا روی دیتای بدون علامت با دستور MUL اجرا شود. دقت داشته باشید که اگر

فصل ششم

دو عدد ۸ بیتی را در هم ضرب کنیم جواب ۲۶ بیت می شود یعنی دو عدد ۸ بیتی اگر در هم ضرب شوند جواب ۱۶ بیتی خواهد شد، و همچنین حاصلضرب دو عدد ۱۶ بیتی ۳۲ بیت خواهد شد. بعضی از پرچمها (مانند C و OV) متاثر می شوند و مابقی تغییر نمی یابند، ولی مقدار آنها نامشخص است. اگر ۸ بیت با ارزش نتیجه صفر باشد هر دو پرچم C و OV صفر خواهد بود در غیر اینصورت هر دو برابر یک (SET) می شوند، و در ضرب دو عدد ۱۶ بیتی اگر ۱۶ بیت با ارزش نتیجه صفر باشد هر دو پرچم C، OV صفر می شوند ولی اگر ۱۶ بیت با ارزش دارای مقدار باشد هر دوی آنها برابر یک (SET) می شوند. بغیر از مدل آدرس دهی فوری تمام مدهای آدرس دهی که برای جمع و تفریق ذکر شد برای دستور ضرب نیز قابل اجرا هستند.

۱-۱-۶- ضرب اعداد ۸ بیتی :

در ضرب اعداد ۸ بیتی اعم از اینکه علامت دار باشد یا بدون علامت، مضروب همواره در ثبات AL است. برنامه نویس فقط یک اپرنند می تواند در دستور تعریف کند و آنهم مضروب فيه است، مانند دستور BL MUL. این دستور AL را در BL ضرب می کند و حاصلضرب را در AX قرار می دهد. جدول ۶-۸ لیست بعضی از دستورات ضرب ۸ بیتی را نشان می دهد.

جدول ۶-۸- دستورات ضرب ۸ بیتی

دستورات	شرح
MUL CL	عدد بدون علامت موجود در AL در عدد بدون علامت CL ضرب و جواب در AX قرار می گیرد.
IMUL DH	عدد علامت دار موجود در AL در عدد DH ضرب و جواب در AX خواهد بود.
IMUL BYTE PTR[BX]	عدد علامت دار موجود در AL در یک بایت عدد موجود در DS که بتوسط BX آدرس دهی می شود ضرب می شود و جواب در AX است.
MUL TEMP	عدد بدون علامت موجود در AL در عدد موجود در DS که با TEMP آدرس دهی شده ضرب و جواب در AX است.

تصویر کنید که BL و CL هر کدام دارای یک عدد بدون علامت هستند و در هم ضرب شده اند و جواب در DX باشد. البته این کار با یک دستور امکان پذیر نیست. یعنی چندین دستور باید بکار گرفته شود تا مطلب بالا محقق گردد. مثال ۶-۱۱ طی قطعه برنامه این کار را انجام داده است. البته در مثال مقادیر اولیه را نیز در CL، BL قرار داده است.

Example 6-11

```

MOV BL,5
MOV CL,10
MOV AL,CL
MUL BL
MOV DX,AX

```

اگر ضرب علامت دار انجام دهد، حاصل ضرب ممکن است مثبت باشد ممکن است منفی شود اگر منفی شود باید بصورت متمم ۲ باشد. البته که هر عددی در ۱۸۰۶ اگر منفی باشد باید بصورت متمم ۲ در سیستم وجود داشته باشد. در مثال ۱۱-۱۶ اگر قرار بود دو عدد علامت دار را در هم ضرب کنیم فقط کد عملیاتی (OP-CODE) دستور ضرب عوض می شود، یعنی MUL باید با IMUL جایگزین می گردد.

۲-۱-۶- ضرب اعداد ۱۶ بیتی :

این ضرب خیلی شبیه ضرب اعداد ۸ بیتی است. ثبات AX همواره حامل مضروب ۱۶ بیتی خواهد بود و ثبات های DX، دارای حاصل ضرب ۳۲ بیتی خواهند بود. توجه داشته باشید که همواره ۱۶ بیت با ارزش باید در DX قرار بگیرد. جدول ۴-۹ لیست بعضی از دستورات ضرب ۱۶ بیتی را نشان می دهد.

جدول ۴-۹- دستورات ضرب ۱۶ بیتی

دستورات	شرح
MUL CX	عدد بدون علامت موجود در AX ضربدر عدد CX شده، وجواب در AX:DX قرار می گیرد.
IMUL DI	عدد علامت دور موجود در AX در محتوای DI ضرب می شود و حاصل در AX:DX قرار می گیرد.
MUL WORD PTR[SI]	عدد بدون علامت AX در محتوای حافظه ای در DS که بوسیله SI آدرس دهی می شود ضرب و جواب در AX:DX قرار می گیرد.

۲-۲-۶- دستور تقسیم (DIVISION) :

دستور تقسیم مانند دستور ضرب هم روی دیتاها ۸ بیتی و هم روی دیتاها ۱۶ بیتی کاربرد دارد. هم دیتاها می توانند علامت دار باشند (IDIV) و هم ممکن است بدون علامت باشند (DIV). اعداد همیشه به میزان دو برابر عریض می شوند، یعنوان مثال در یک تقسیم ۸ بیتی مقسوم در یک ظرف ۱۶ بیتی قرار می گیرد. دستور تقسیم نیز همانند دستور ضرب از مد آدرس دهی فوری ۳۲ بیتی قرار می گیرد. دستور تقسیم نیز همانند دستور ضرب از مد آدرس دهی فوری نمی تواند استفاده کند. وضعیت هیچکدام از پرچم ها تعریف نشده است. اگر در تقسیم خارج قسمت بیش از حد بزرگ شود سیستم اعلام خطا می کند. یا اگر عددی تقسیم بر صفر شود، یک اینترپریت در سیستم اتفاق می افتد بنام تقسیم بر صفر (DIVIDED-BY-ZERO). در آینده بیشتر راجع به اینترپریت صحبت خواهد شد.

۱-۲-۲-۶- دستور تقسیم ۸ بیتی :

همانطور که ذکر شد، مقسم ۸ بیتی باید در ظرف ۱۶ بیتی قرار بگیرد لذا مقسم در AX و مقسوم علیه عددی است که با یکی از مدهای آدرس دهی برای دستور انتخاب می شود. حاصل تقسیم دو عدد ۸ بیتی دو مقدار بنام خارج قسمت و باقیمانده است که خارج قسمت در AL و باقیمانده در AH قرار می گیرد. در تقسیم اعداد علامت دار همواره علامت باقیمانده متن علامت خارج قسمت می باشد. جدول شماره ۱۰-۶ لیست بعضی از دستورات ۸ بیتی را نشان می دهد.

جدول ۱۰-۶- دستورات تقسیم ۸ بیتی

دستورات	شرح
DIV CL	عدد بدون علامت موجود در AX بر CL تقسیم می شود و خارج قسمت در AL و باقیمانده در AH قرار می گیرد.
IDIV BL	عدد علامت دار درون AX بر BL تقسیم و خارج قسمت در AL و باقیمانده در AH قرار می گیرد.
DIV BYTE PTR[BP]	عدد بدون علامت ۸ بیتی موجود در AX بر یک بایت موجود در SS که با آدرس دهی می شود تقسیم و خارج قسمت در AL و باقیمانده در AH خواهد بود

در یک تقسیم ۸ بیتی عدد ۸ بیتی باید به ۱۶ بیت تبدیل شود. یک دستور در ریزپردازنده وجود دارد بنام CBW که یک بایت را به WORD تبدیل می کند. تبدیل توسط این دستور انجام می گیرد. اگر یک عدد علامت دار در AL داشته باشیم وقتی تبدیل به ۱۶ بیت می شود بصورت ۱۶ بیتی علامت دار در AX قرار می گیرد. این دستور همواره قبل از تقسیم ۸ بیتی بکار می رود. مثال ۱۲-۶ قطعه برنامه ای را نشان می دهد که عدد بدون علامت 12H را برابر ۳ تقسیم می کند. توجه دارید که دستور CBW در این مثال بکار نرفته است. به جای آن از دستور MOV AH,0 استفاده شده است. اگر از دستور تبدیل هم استفاده می شد در AH مقدار 00H قرار می گرفت. پس از اجرای تقسیم عدد 04H در AL و 00 در AH خواهد بود

Example 6-12
 MOV AL,12H
 MOV CL,3
 MOV AH,0
 DIV CL

۱-۲-۲-۶- دستور تقسیم ۱۶ بیتی :

تقسیم ۱۶ بیتی نیز مثل ۸ بیتی است با این تفاوت که مقسم ۱۶ بیتی به ۳۲ بیت تبدیل می شود و بخش با ارزش تر درون DX و کم ارزشتر درون AX خواهد بود. بعد از اجرای دستور خارج قسمت در AX و باقیمانده در DX قرار خواهد گرفت.

جدول ۶-۱۱ بعضی از دستورات مهم تقسیم اعداد ۱۶ بیتی را نشان می‌دهد. همانگونه که در تقسیم ۸ بیتی یک دستور وجود داشت تا عمل تبدیل بایت را به ۱۶ بیت انجام دهد، در این جانیز دستور CWD عمل تبدیل ۱۶ بیت به ۳۲ بیت را انجام می‌دهد.

(CONVERT WORD TO DOUBLE WORD)

جدول ۶-۱۱-۶- دستورات تقسیم ۱۶ بیتی

دستورات	شرح
DIV CX	عدد بدون علامت موجود در AX بر CX تقسیم می‌شود و خارج قسمت در DX و باقیمانده در AX خواهد بود.
IDIV SI	عدد علامت دار موجود در AX بر SI تقسیم و خارج قسمت در AX و باقیمانده در DX است.
DIV DATA	عدد موجود در AX بر یک دنباع اینتی در حافظه که DATA نشان می‌دهد تقسیم و خارج قسمت در AX باقیمانده در DX خواهد بود.

مثال ۶-۱۳-۶ تقسیم دو عدد علامت دار را نشان می‌دهد که عدد ۱۰۰ را در AX و عدد ۹ را در CX قرار داده است. از دستور CWD نیز استفاده شده است.

Example

6-13
 MOV AX,-100
 MOV CX,9
 CWD
 IDIV CX

۶-۶-۳- دستورات ریاضی

ریزپردازندۀ ۸۰۸۶ قادر است که در هر دو کد BCD (Binary Code Decimal) و ASCII (American Standard Code for Information Interchange) عملیات ریاضی انجام می‌دهد و نتیجه را در کد مربوطه اعلام نماید.

کاربرد عملیات در کد BCD در مواردی است که محاسبات کمی دارند، مثل ترمینالهای فروش فروشگاهها. اما کد ASCII در سیستمهای کاربرد دارد که برای نمایش کاراکترها و دیتا از کد ASCII استفاده می‌کنند و دیتا بصورت ASCII در سیستم ذخیره شده و پردازش می‌شود.

۶-۳-۱- دستورات عملیات BCD

چهار دستور در عملیات BCD وجود دارد و آنها عبارتند از :

- الف - تنظیم دیتا در پایه ۱۰ بعد از انجام عمل جمع (Decimal Adjust after Addition) DAA
- ب - تنظیم دیتا در پایه ۱۰ بعد از انجام عمل تفریق (Decimal Adjust after Subtraction) DAS
- ج - تنظیم حاصلضرب بعد از عمل ضرب (Adjust result of BCD Multiplication) AAM
- د - تنظیم دیتا در کد BCD قبل از عمل تقسیم (Adjust before BCD Division) AAD

فصل ششم

سه دستور AAM, DAS, DAA بعد از اینکه عمل جمع یا تفریق یا ضرب انجام شد کاربرد دارند. اما دستور AAD قبل از عمل تقسیم کاربرد دارد. در عملیات جمع و تفریق دیتاها که باید در جمع یا تفریق شرکت کنند دو رقم در هر بایت در AL دخیره می شوند، که به این عمل دو رقم در بایت کد BCD فشرده شده (Packed BCD format) می گویند. عنوان مثال اگر بیگ بایت شامل H 34 باشد و آنرا بصورت کد BCD نمایش دهنده دارای دو رقم 3 و 4 خواهد بود که این از نظر ارزش با 34H متفاوت است.

در تقسیم و ضرب، اعداد بصورت یک رقم در بایت ذخیره می شوند که بصورت کد BCD فشرده شده نیست. عنوان مثال اگر در AL مقدار 01 باشد یعنی معادل 1 کد (Unpacked BCD Format) است. که در بایت 01 نوشته می شود، بنام کد BCD فشرده نشده می گویند.

۱-۳-۶- دستور DAA :

این دستور العمل پس از جمع دو عدد در کد BCD برای تبدیل حاصل جمع به همان فرم BCD بکار می رود این دستور در صورتیکه 4 بیت کم ارزش AL بیش از 9 باشد و یا AF=1 و واحد به آن می افزاید. سپس 6 واحد به چهار بیت بالاتر AL خواهد افزود البته به شرطی که مقدار آن از 9 بزرگتر باشد و یا CF=1 باشد، به مثال ۶-۱۴ توجه کنید.

Example 6-14
 MOV AL,47H ; AL=01000111
 ADD AL,38H ; AL=47H+38H=7FH در BCD معنی ندارد
 DAA ; F+6=15 \Rightarrow 5 & 7+1=8 \Rightarrow 85

چون رقم اول بیش از 9 است با 6 جمع می شود Carry حاصله با رقم بعدی جمع خواهد شد. مثال شماره ۶-۱۵ را دقیت کنید از آنجائیکه دستور DAA فقط روی AL عمل تنظیم دیتا را انجام می دهد. لذا یک انتقال از BL به AL داریم. عیناً مثل BH به AL لازم به ذکر است که دستور DAA روی پرچمها اثر می گذارد.

Example 6-15
 MOV DX,1234H
 MOV BX,3099H
 MOV AL,BL
 ADD AL,DL
 DAA
 MOV CL,AL
 MOV AL,BH
 ADC AL,DH
 DAA
 MOV CH,AL

۱-۲-۳-۶- دستور DAS

این دستور تنظیم دهدی پس از تفریق است. شبیه DAA عمل می کند. فقط عمل تنظیم را روی AL انجام می دهد. همانگونه که DAA بعد از یک دستور جمع اعمال می شد این دستور بعد از عمل تفریق اجرا می شود. مثال ۱۵-۶-۱۶ عملًا مثل مثال ۱۵-۶ کار می کند.

نحوه عمل به این شکل است که اگر چهار بیت اول AL بزرگتر از ۹ باشد و یا $AF=1$ عدد ۶ را از رقم چهار بیت اول کم می کند و همچنین اگر ۴ بیت بالاتر AL رقیقی بیش از ۹ داشته باشد و یا $CF=1$ باشد عدد ۶ را از رقم چهار بیت بالا کم می کند.

Example

6-16
 MOV DX,1234H
 MOV BX,3099H
 MOV AL,BL
 SUB AL,DL
 DAS
 MOV CL,AL
 MOV AL,BH
 SBB AL,DH
 DAS
 MOV CH,AL

لازم به ذکر است که تمام پرچمها را متأثر می کند ولی OF نامشخص است.

۱-۲-۳-۶-۷- دستور AAM

دستور AAM بعد از ضرب دو عدد تک رقم در کد BCD که بصورت BCD فشرده نشده است بکار می رود مانند مثال ۱۶-۶-۱۷ اگر $CL=05$ و $AL=05$ باشد، ثبات AX شامل عدد ۰۰۱۹H می شود حال باید با دستور AAM آنرا بصورت ۰۲۰۵ یا بصورت ۲ رقم BCD فشرده نشده مثل ۲۵ درآید. در واقع AAM فقط AL را اصلاح می کند و هر رقم بزرگتر از ۹ در AH ذخیره می شود.

Example

6-17
 MOV AL,5
 MOV CL,5
 MUL CL
 AAM

۱-۲-۳-۶-۸- دستور AAD

دستور AAD قبل از دستور DIV بکار می رود در صورتیکه سه دستور قبلی بعد از عمل جمع و تفریق و ضرب بکار می رفته و حاصل عملیات را بصورت BCD در می آورند. این دستور ثبات AX را به دو رقم BCD فشرده نشده تبدیل می کند که با ارزش ترین رقم در AH و کم ارزش ترین در AL قرار می گیرد.

پس ابتدا AAD بکار می رود پس از آن دستور تقسیم مورد استفاده قرار می گیرد و خارج قسمت در AL و باقیمانده در AH قرار می گیرد. مثال شماره ۶-۱۸ چگونگی تقسیم عدد ۷۲ در کد BCD را بر ۹ نشان می دهد. که چگونه عدد ۸ بدهست می آید. عدد ۷۲ در AX ریخته می شود و عدد ۹ هم در BL قرار می گیرد. دستور AAD عدد ۰۷۰۲H را به ۰۰۴۸H تبدیل می کند (توجه دارید که این دستور عدد BCD فشرده نشده را به باینری تبدیل می کند) دستور DIV دیتای باینری (۴۸H) را بر ۹ تقسیم می کند خارج قسمت در AL و باقیمانده در AH قرار می گیرد.

Example 6-18

```

MOV AX,0702H
MOV BL,9
AAD
DIV BL

```

۶-۳-۲- دستور محاسباتی در کد ASCII

دستورات محاسباتی ASCII بدسټورات اطلاع می شود که روی دیتاهایی که در کد ASCII هستند عمل پردازش انجام دهنده و نتیجه را نیز در کد ASCII ارائه نمایند. رنج دیتاهایی که در کد ASCII هستند عبارت است از ۳۰H تا ۳۹H و برای نمایش ارقام دسیمال ۰ تا ۹ مورد استفاده قرار می گیرند. دو دستور داریم که روی داده های در کد ASCII کاربرد دارند، اول دستور AAS (Adjust for Ascii Subtraction) و دوم دستور AAA (Adjust for Ascii Addition) این دستورات همیشه روی ثبات AX بمنزله منبع اطلاعات قبل از اعمال دستور لازم و همچنین مقصد اطلاعات تنظیم شده عمل می نمایند.

۶-۳-۲-۱- دستور AAA

اولاً این دستور روی پرجمهای AF، CF، دستور بعد از یک ADD که دو رقم اسکی را جمع کرده بکار می رود که عمل جمع دیتا در کد اسکی را بدون دخالت رقم دوم که ۳ است فراهم نماید. این دستور فقط ثبات AL را اصلاح می کند، بعنوان مثال اگر عدد ۳۱H با عدد ۳۹H جمع شود نتیجه ۶AH خواهد بود. جمع دو عدد در کد اسکی مثل ۳۱H و ۳۹H باید جوابی هم در کد اسکی ارائه نماید. این جواب دو رقمی و معادل ۱۰ دسیمال است. که در کد اسکی بصورت ۳۱H و ۳۰H خواهد بود. نکته ای که باید دقت کرد اینکه اگر حاصل جمع کمتر از ۱۰ باشد باید AH=0 شود و اگر حاصل جمع بیش از ۹ باشد باید AH=1H شود.

اگر AAA بدنیال یک دستور جمع بیاید مثال بالا را اینگونه اصلاح می کند که در AH عدد ۰۱H و در AL عدد ۰۰H قرار می گیرد. البته دقت دارید که هنوز این جواب در کد اسکی نیست.

براحتی می‌توان آن را به اسکی تبدیل کرد. اگر عدد $3030H$ را به AX اضافه کنیم حاصل در کد اسکی خواهد بود. زیربرنامه‌ای که عملیات بالا را انجام می‌دهد در مثال شماره ۱۹-۶ نشان داده شده است. دقت دارید که اعداد باید در AL جمع شوند تا AAA آنرا تنظیم کند بعد از اعمال دستور AAA محتوای $AX=0100H$ خواهد شد. جواب دارای دو رقم بدست آمده است البته جواب بدست آمده در کد اسکی نیست. زیربرنامه زمانی کامل می‌شود که دستور جمع ADD AX,3030H را اجرا کنیم. در نتیجه جواب $3130H$ می‌شود که معادل 10 دسیمال است.

Example

```
6-19
MOV AX,0031H
ADD AL,39H
AAA
ADD AX,3030H
```

۲-۳-۶- دستور AAS

مانند دستور AAS، دستور AX عمل تنظیم را بعد از یک تفريق در کد اسکی بهده دارد. بعنوان مثال تصور کنید که عدد $35H$ از عدد $39H$ در کد اسکی تفريق شود، حاصل عدد ۴ خواهد بود. که نياز به تصحیح هم ندارد. لذا شما باید مواظب عدد درون ثبات AH باشید. عبارت دیگر اگر $37H-38H$ کنیم درون AL مقدار $09H$ را خواهیم داشت و عدد درون AH باید یکی کاسته شود، این کم کردن از AH اجازه می‌دهد که چند رقم اسکی را از هم کم کنیم.

Example

6-20

MOV AL,32H ;	عدد اسکی ۲ را در ثبات AL می‌گذارد
MOV DH,37H ;	عدد اسکی ۷ را در ثبات DH می‌گذارد
SUB AL,DH ;	تمم دو عدد $-5=32-37=FBH$ است
AAS ;	حال AL=5 و CF=1 می‌شود

مجددآ اگر قرار است بصورت اسکی درآید باید محتوای AL با عدد $30H$ جمع شود. یا با عدد $30H$ با هم OR می‌شود.

۴-۶- دستورات پایه منطقی :

منظور از دستورات پایه منطقی NOT, XOR, OR, AND است. بعلاوه اینکه در این بخش راجع به دستور TEST (AND مخصوص) و NEG نیز صحبت خواهد شد. البته دستور NEG واقعاً یک دستور محاسباتی ریاضی است اما چون شبیه دستور NOT منطقی عمل می‌کند در این بخش مورد بحث قرار می‌گیرد. کاربرد دستورات منطقی معمولاً برای کنترل یک بیت در نرم افزار سطح پایین (Low Level Software) می‌باشد. مثلاً برای آن که یک بیت خاصی در یک ثبات را ۰ یا ۱ یا متمم کرد بکار می‌رond، تمام دستورات منطقی روی پرچم ها اثر می‌گذارند و رقم

فصل ششم

نقلی (Carry) و پرچم سرفتگی (Overflow) را صفر می کنند بنابر پرچم ها بگونه ای تغییر می یابند که وضعیت عمل منطقی را نشان دهند.

۶-۴-۱- دستور AND

دستور AND روی دو داده بیت با بیت متاظر مطابق جدول صحت زیر عمل می کند و حاصل را اگر با $X = A_i \wedge B_i$ نشان دهیم، می توان گفت هر گاه یکی از ورودیها صفر باشد حاصل AND هم صفر است.

A_i	B_i	X_i
0	0	0
0	1	0
1	0	0
1	1	1

مهمترین کاربرد دستورات منطقی دستکاری یک بیت در یک کلمه است. بعنوان مثال اگر بخواهیم بیت سوم دیتائی را یا محتوای ثباتی را صفر کنیم باید ثبات یا دیتای مورد نظر را با عدد مثل 11111011 AND کرد. و حاصل را در آن ثبات ریخت تا بیت سوم آن صفر شود. این عمل را پوشاندن بیت (Bit Masking) گویند. عکس عمل فوق را یعنی قرار دادن یک '1' در بیت خاصی (Bit Inserting) می گویند. در شکل شماره ۶-۱ یک نمونه از عمل پوشاندن بیت را نشان می دهد. توجه دارید منظور از X یعنی هرچه باشد صفر یا یک وقتی که با ۱، AND شود تغییری نمی کند ولی اگر با صفر AND حاصل صفر خواهد شد.

X	دیتای ناشناخته ای							
0	0	0	0	1	1	1	1	الگوی پوشاننده
0	0	0	0	X	X	X	X	حاصل بدست آمده

شکل شماره ۶-۱- چگونگی پوشاندن بیت بایت های یک دیتا

برای عمل AND هم از هر مد آدرس دهی بغیر از ثبات های سگمنت و AND کردن حافظه با حافظه مجاز است. جدول ۶-۱۲ لیست بعضی از دستورات AND آورده شده است. یک دیتا در کد ASCII توسط عمل AND به آسانی قابل تبدیل به BCD می باشد. با پوشاندن ۴ بیت با ارزش هر عددی در کد ASCII آن عدد به معادل BCD آن تبدیل می شود مثال شماره ۶-۲۱ را دقت کنید.

جدول شماره ۱۲-۶ دستورات AND

دستورات	شرح
AND AL,BL	$AL \leftarrow AL \wedge BL$
AND CX,DX	$CX \leftarrow CX \wedge DX$
AND CL,33H	$CL \leftarrow CL \wedge 33H$
AND DI,4FFFH	$DI \leftarrow DI \wedge 4FFFH$
AND AX,[DI]	$AX \leftarrow AX \wedge M[DS \times 10H + DI]$
AND Array[SI],AL	AL با محلی از حافظه که آدرس آن برابر DS×10H+Array+SI باشد AND می شود و حاصل در همان محل ریخته خواهد شد

Example

6-21.

```
MOV DX,3135H
MOV AX,BX
AND AX,0F0FH
```

دقیق کنید حاصل برنامه در ثبات AX بصورت ۰۱۰۵ خواهد بود که دو رقم BCD هستند.

۶-۴-۲- دستور OR

دستور OR در واقع جمع منطقی است. که کمی با جمع ریاضی فرق می کند و آن فرق در این است که در دستور OR حاصل زمانی صفر می شود که هر دو متغیر صفر باشند، در غیر اینصورت حاصل یک خواهد بود. لذا حاصل از جدول صحت زیر تبعیت می کند که A و B ورودیها و حاصل یا خروجی X می باشد.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

کاربرد مهم OR مانند AND است با اندکی تفاوت زمانی که بخواهیم یک بیت از دیتا را یک کنیم می توانیم آن دیتا را با عددی که همه بیت های آن صفر است و فقط آن بیت متناظر یک است OR کنیم و در آن ظرف بریزیم عمل مورد نظر انجام می شود به این عمل اعمال یک '۱' در دیتا گفته می شود (Bit Inserting). شکل ۶-۲ عمل اعمال یک بیت یا چند بیت درون دیتائی را نشان می دهد.

فصل ششم

X	X	X	X	X	X	X	X
0	0	0	0	1	1	1	1
X	X	X	X	1	1	1	1

دیتای ناشناخته ای

الگوی پوشاننده

حاصل بدست آمده

شکل شماره ۶-۲-۶- اعمال چهار بیت ۱ درون لغت

دستور OR نیز می‌تواند با هر مدل آدرس دهی مورد استفاده قرار بگیرد. بغير از ثبات سکمنت و OR کردن دو محل حافظه در بقیه موادر مثل AND عمل می‌کند.

فرض کنید دو عدد را در کد BCD ضرب کنید بعد با دستور AAM در کد BCD تنظیم نمائید

حال اگر بخواهیم آنرا در کد ASCII نمایش دهید با دستور OR خیلی کار آسان می‌شود

مثال ۶-۲۲ جدول ۶-۱۳ تعدادی از دستورات OR را آورده که می‌توانید ملاحظه فرمائید :

جدول شماره ۶-۱۳ دستورات OR

دستورات	شرح
OR AH,BL	AH \leftarrow AH \vee BL
OR SI,DX	SI \leftarrow SI \vee DX
OR DH,A3H	DH \leftarrow DH \vee A3H
OR SP,990DH	SP \leftarrow SP \vee 990DH
OR DX,[BX]	DX \leftarrow DX \vee M[DS \times 10H + BX]
OR DATE[DI+2],AL	M[DATE + DS \times 10H + DI + 2] \leftarrow M \vee AL

Example 6-22
 MOV DX,05
 MOV BL,07
 MUL BL
 AAM
 OR AX,3030H

XOR ۶-۴-۳- دستور

دستور OR انحصاری (EXCLUSIVE-OR) مطابق جدول صحت زیر عمل می‌کند :

$$X = A \oplus B$$

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

با دقت به جدول صحت آن متوجه انحصاری شدن آن می‌شود. انحصار در حالت ۱ و ۰ می‌باشد که در XOR برابر صفر می‌شود. از طرفی به آن دروازه نامتعادل گویند زیرا وقتی ورودی‌های آن نابرابر باشند خروجی اش یک خواهد بود، و هنگامیکه ورودی‌ها برابر باشند خروجی صفر می‌شود.

جدول شماره ۶-۱۴ بعضی از دستورات XOR را آورده است. بدینهی است که تمام دستورات نیز همانند OR, AND XOR نیز همانند برای تغییر یک بیت انتخاب شده در یک دیتا کاربرد دارد. اگر بیت معینی با صفر XOR شود هیچ تغییری نمی‌کند، و اگر با یک XOR شود متمم می‌شود.

شکل ۶-۳ چگونگی متمم کردن بیت خاص در یک لغت را نشان می‌دهد :

جدول شماره ۶-۱۴ دستورات XOR

دستورات	شرح
XOR CH,DH	$CH \leftarrow CH \oplus DH$
XOR SI,BP	$SI \leftarrow SI \oplus BP$
XOR AH,0EEH	$AH \leftarrow AH \oplus 0EEH$
XOR SI,00DDH	$SI \leftarrow SI \oplus 00DDH$
XOR BX,[SI]	$BX \leftarrow BX \oplus M[DS \times 10H + SI]$
XOR DATE[DI+2],AL	$M[DATE + DS \times 10H + DI + 2] \leftarrow M \oplus AL$

مالحظه می‌فرمائید که چهار بیت سمت راست معکوس می‌شوند چون دیتا با چهار عدد یک XOR می‌شود، و بقیه بیت‌ها تغییر نمی‌کنند چون با صفر XOR می‌شوند :

X X X X X X X X	دیتای ناشناخته ای
0 0 0 0 1 1 1 1	الگوی پوشاننده
X X X X X X X X	حاصل بدست آمده

شکل شماره ۶-۳- متمم کردن بیت‌های مورد نظر با XOR

فرض کنید ۰۰ بیت اول ثبات AX را بخواهیم متمم کنیم و مابقی بیت‌ها بدون تغییر بمانند یک دستور XOR بصورت زیر مورد نیاز است :

XOR AX,03FFH

در واقع ده بیت اول الگو یک هستند و مابقی صفر طبق خواسته ما عمل می‌کنند، یعنی ۰۰ بیت متمم می‌شوند و مابقی بدون تغییر می‌مانند.

۴-۶-۴- دستور TEST

دستور TEST عمل AND را روی دو دیتای داده شده پیاده می کند، اما با AND فرقی دارد و آن این است که AND هم روی دیتا و هم روی پرچمها اثر می گذارد ولی دستور TEST فقط پرچمها را متاثر می سازد و هیچکدام از اپرندتها تغییر نمی کند. در دستور TEST هم مثل ثبات های سگمنت نمی توانند اپرند باشند و از تمام مدهای آدرس دهی که AND استفاده می کرد می توانند استفاده نمایند.

دستور TEST اغلب بهمان سبکی که دستور CMP بکار برده می شود مورد استفاده قرار می گیرد. اما در این دستور بررسی یک بیت آسان تر صورت می گیرد. عنوان مثال یک انتخاب منطقی برای بررسی بیت سمت راست ثبات AX می باشد، زیرا اگر از دستور TEST AX,1 استفاده کنید بیت سمت راست AX با یک مقایسه می شود اگر پرچم Z=1 شود بیت سمت راست AX برابر صفر است و در غیر اینصورت بیت سمت راست AX برابر یک است.

۴-۶-۵- دستورات NOT, NEG

عمل معکوس سازی منطقی (NOT) یا متمم یک و معکوس سازی عدد علامت دار (NEG) یا متمم دو دو دستور آخر منطقی هستند البته بجز دستورات جابجایی (Sift) و چرخشی (Rotate) این دو دستور، روی یک اپرند عمل می کنند. در جدول ۱۵-۶ دستورات NOT, NEG را نشان می دهد. برخلاف سایر دستورات منطقی این دو دستور بر روی پرچمها اثر می گذارد. دستور NOT یا متمم یک تمام بیت های یک عدد را تغییر می دهد. لذا عدد ۰۰H را به FFH تبدیل می کند. دستور NEG یا متمم دو یک عدد دقیقاً متمم یک بعلاوه عدد یک است. اگر بخواهیم عدد ۸۸H را تحت تأثیر دستور NEG قرار دهیم اول NOT می کنیم، ۸۸H خواهد شد H ۷۷H و پس از آن ۷۷H+1 مقدار ۷۸H بدست می آید.

جدول شمار ۱۵-۶ دستورات NOT, NEG

دستورات	شرح
NOT CH	را متمم یک می کند.
NEG CH	را متمم دو می کند.
NEG AX	مقدار AX متمم دو می شود.
NOT TEMP	حافظه به آدرس TEMP متمم یک می شود.
NOT BYTE PTR[BX]	یک محل حافظه به مقدار یک بایت به آدرس Offset بیت BX متمم یک می شود

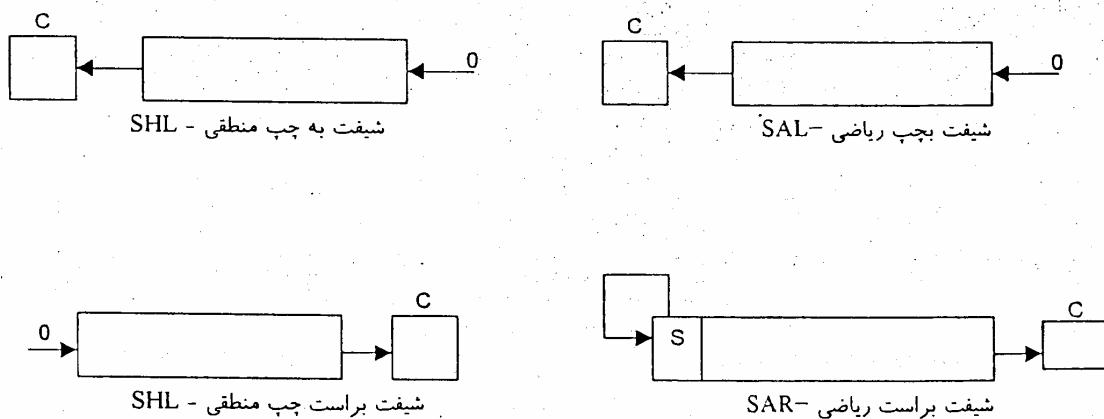
فصل ششم

۵-۶- دستورات جابجایی (SHIFT) یا چرخشی (ROTATE)

دستورات شیفت و چرخشی اطلاعات باینری را به اندازه یک بیت جابجا می کنند. کاربرد آنها بیشتر در کنترلهای سطح پایین ورودی و خروجی سیستم است. ریزپردازنده ۸۰۸۶ یک مجموعه کاملی از این نوع دستورات دارد که به شرح آنها می پردازیم.

۱-۵-۶- دستورات جابجایی (SHIFTS)

این دستورات اطلاعات موجود در حافظه و یا ثبات را جابجا می کنند. این دستورات همچنین قادرند عملیات ریاضی ساده مثل ضرب در^۲ (با عمل شیفت به راست) را انجام دهند، ریزپردازنده ۸۰۸۶ دارای چهار نوع مختلف شیفت است. دو نوع شیفت منطقی دارد و دو نوع شیفت حسابی (ریاضی) دارد. که چهار نوع آن در شکل ۵-۶ نشان داده شده است. توجه داشته باشید که در شکل ۵-۶ دو نوع مختلف شیفت به راست و دو نوع هم شیفت به چپ است.



شکل ۵-۶ چهار نوع شیفت (چپ و راست - ریاضی و منطقی)

ملاحظه می فرمائید در شیفت منطقی اعم از اینکه به چپ باشد یا به راست از سمت دیگر طرف دیگری که شیفت داده می شود صفر وارد خواهد شد.

اما در شیفت ریاضی وقتی که به چپ شیفت داده می شود از سمت راست صفر وارد می شود اما زمانی که شیفت به راست داده می شود باید برای عدد مثبت صفر و برای عدد منفی یک وارد شود. خوبیختانه وقتی عدد مثبت است بیت علامت ۰ و وقتی که عدد منفی است بیت علامت یک است. لذا می توان گفت بیت علامت را داخل ظرف سیستم سرایت می دهیم.

ضمناً دقت دارید که یک بار شیفت ریاضی به چپ عدد ضربدر دو می شود و اگر دوباره شیفت به چپ تکرار شود مجدداً ضربدر ۲ می شود، در مجموع یعنی ضربدر^۲ می شود و لذا می گوئید با

فصل ششم

شیفت بچپ می توان عدد را ضربدر² کرد. در مورد شیفت به راست عدد مورد نظر تقسیم بر دو و با تکرار تقسیم بر² و خواهد شد. جدول ۶-۱۶ دستورات شیفت ریاضی و منطقی را همراه با بعضی از مدهای آدرس دهی مجاز نشان می دهد. دقت کنید اگر در دستور شیفت ثبات CL را نیز وارد کنیم CL می تواند تعداد دفعات شیفت دادن را نگهدارد. مثال ۶-۲۴ چگونگی شیفت دادن ثبات DX را به تعداد ۱۴ دفعه نشان می دهد.

جدول شمار ۶-۱۶ - دستورات شیفت ریاضی و منطقی

دستورات	شرح
SHL BL	ثبات BL را یک بیت به سمت چپ شیفت (منطقی) می دهد
SHR AX	ثبات AX را یک بیت به سمت راست شیفت (منطقی) می دهد
SAL BYTE PTR[BX],CL	یک محل ۸ بیتی از حافظه را که BX آدرس می دهد به اندازه عدد موجود در CL شیفت به چپ (ریاضی) می دهد
SAR SI,CL	ثبات SI را به اندازه عدد موجود درون CL شیفت (ریاضی) به راست می دهد

Example 6-24

```
MOV CL,14
SHL DX,CL
```

فرض کنید محتوای AX را بخواهیم ضربدر ۱۰ کنیم. می توانیم مطابق مثال ۶-۲۵ عمل کنیم.

Example 6-26

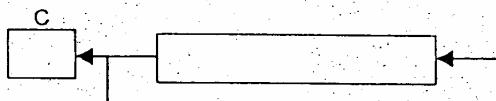
```
SHL AX ; AX=2*AX
MOV BX,AX ; BX=AX
SHL AX ; AX=4AX
SHL AX ; AX=8AX
ADD AX,BX ; AX=8AX+2AX=10AX
```

6-۵-۶- دستورات چرخشی (ROTATE)

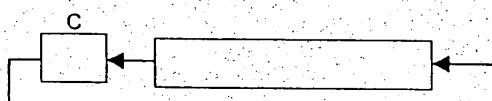
دستور چرخشی دیتای موجود در یک محل حافظه را به اندازه یک بیت جابجا می کند. و بیت خارج شده از یک طرف را به سمت دیگر وارد می کند. این عمل بدون قرار دادن فیلیپ فلاپ Carry در حلقه چرخش و یا با قرار دادن آن در حلقه چرخش انجام می شود. شکل ۶-۶ هر دو صورت آن را نشان می دهد. برنامه نویس باید دقت داشته باشد که کدام فرم چرخشی را استفاده می کند.

ROR و ROL دو دستور چرخشی به چپ و راست هستند که C در حلقه چرخش نیست. البته بیت خارج شده از یک طرف علاوه بر اینکه به طرف دیگر وارد می شود به C نیز وارد خواهد شد. دو دستور RCR، RCL عمل چرخش را همراه با قرار دادن C در حلقه چرخش انجام می دهند. در دستورات چرخشی نیز ثبات CL به منزله شمارنده تعداد دفعات چرخش مورد

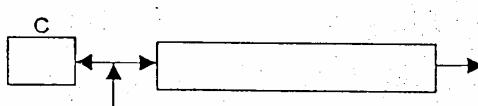
استفاده قرار می‌گیرد. جدول شماره ۱۷-۶ دستورات چرخشی را با بعضی از مدهای آدرس دهی نشان می‌دهد.



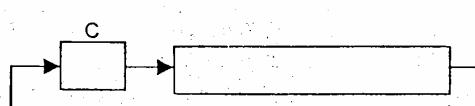
شیفت چرخشی به چپ بدون C



شیفت چرخشی به چپ با C



شیفت چرخشی به راست بدون C



شیفت چرخشی به راست با C

شکل ۶-۶ چهار نوع چرخش به راست و چپ بدون یا با فلیپ فلاب Carry

اگر بخواهیم عددی را که تعداد بیت‌های آن بیش از یک کلمه سیستم باشد شیفت چرخشی به چپ یا به راست بدھیم می‌توانیم از قطعه برنامه مثال ۶-۲۶ استفاده کنیم. ۴۸ بیت دیتا داریم در سه ثبات DX, BX, AX قرار دارند، این دیتای ۴۸ بیتی یک بیت شیفت داده می‌شود.

Example 6-26
SHL AX
RCL BX
RCL DX

۶-۶- دستورات مقایسه ای رشته‌ای (String Comparisons):

همانگونه که در فصل ۵ مشاهده کردید که دستورات رشته‌ای ریزپردازنده ۸۰۸۶ چقدر قوی عمل می‌کنند و چگونه یک بلوك دیتا را از محلی به محل دیگر حافظه منتقل می‌کنند، در این قسمت دو دستور دیگر که عمل مقایسه یا جستجوی دیتائی را انجام می‌دهند معرفی می‌کنیم. در واقع این دو دستور عمل جستجوی یک دیتای خاصی را در یک بخش و یا عمل مقایسه محتوای دو بخش از حافظه که آیا با هم تطبیق دارند یا خیر را انجام می‌دهند و این دستورات عبارتند از:

SCAS = Scan String, Cmps= Compare String

۱-۶-۶- دستور SCAS

دستور SCAS ثبات AX را با یک بلوک حافظه بصورت بایت به بایت و یا ثبات AX را با یک بلوک از حافظه بصورت کلمه (۱۶ بیتی) به کلمه مقایسه می کند به این ترتیب که عمل تفریق روی AX و حافظه یا AX و حافظه انجام می دهد بدون اینکه تأثیری روی ثبات AX یا AL و حافظه داشته باشد. اگر AX را با یک محل حافظه ۸ بیتی مقایسه کنید SCASB بکار می رود ولی اگر AX را با یک کلمه ۱۶ بیتی مقایسه کند SCASW مورد استفاده قرار می گیرد. دقیقاً مثل STOS, LODS, MOVS که در فصل ۵ مطالعه شد.

دستور SCAS نیز از پرجم D (Direction Flag) جهت خود افزایشی یا خود کاهشی ثبات های SI و DI استفاده می کند. همچنین اگر از پیشوند REP استفاده شود این دستور نیز قابل تکرار است. باز یادآور می شوم که DI همیشه دیتا را در بخش اکسترا (ES) آدرس دهی می کند و SI در بخش دیتا (DS) آدرس دهی می کند.

فرض کنید بخواهید یک بلوک دیتا به حجم ۱۰۰ بایت را که شروع آن Block می باشد را بررسی که آیا دارای ۰۰H هست یا خیر؟ قطعه برنامه شماره ۶-۲۷ چگونگی این جستجو یا بررسی را نشان می دهد. ~~یادآور مردم که بایت یا کلمه ای باید Scan شود باید در اکسرا لذت~~

Example:

```
6-27
MOV    DI,OffsetBlock
CLD
MOV    CX,100
MOV    AL,0
REPNE  SCASB
```

لطفاً را تو مانگ باز برمی برد
DI لطفاً را تو مانگ باز برمی برد
Z=1 می شود یعنی دو دیتا مساوی بوده اند پس هم REPNE و هم می توان REPNZ قرار داد.

در برنامه بالا زمانی عمل مقایسه را متوقف می کند که یا CX=0 باشد یا محلی از حافظه بیاپد که دارای مقدار ۰۰H باشد.

اینگونه دستورات (رشته ای) می توانند با پیشوند REPE یا REPNE بکار روند. در مثال ۶-۲۷ عمل SCAS بوسیله تفریق انجام می شود و اگر حاصل تفریق صفر شود پرجم Z=1 می شود یعنی دو دیتا مساوی بوده اند پس هم REPNE و هم می توان REPNZ قرار داد.

۱-۶-۶- دستور CMPS

دستور CMPS همواره دو محل حافظه را یا بصورت بایت به بایت یا بصورت کلمه به کلمه (۱۶ بیتی) با هم مقایسه می کند. برای حالت تطبیق داشتن دو مقدار یا تطبیق نداشتن دو مقدار هیچ تأثیری روی محلهای حافظه نمی گذارد، فقط پرجمها (Flags) متأثر می شوند. البته عمل مقایسه در اینجا هم با تفریق دو مقدار صورت می گیرد. لازم به ذکر است که دو محل حافظه

فصل ششم

در این دستور نیز توسط DI در اکسترا سگمنت (ES) و SI در دیتا سگمنت (DS) آدرس دهی می شوند و با بکار گرفتن پرجم D جهت افزایش یا کاهش آنها مشخص می شود. همانند دستور SCAS با پیشوندهای مذکور می توان این دستور را تکرار کرد. مثال ۶-۲۸ دو بلوک حافظه را با هم مقایسه می کند در صورتیکه اختلافی مشاهده کرد متوقف می شود.

Example 6-28

```
MOV SI,OffsetLine
MOV DI,OffsetTable
MOV CX,10
CLD
REP CMPSB
```

سؤالات دوره ای فصل ۶

- ۱- اشکال دستور ADD CL,AX چیست؟
- ۲- آیا ممکن است CX را با DS جمع کنیم؟
- ۳- اگر AX=1001H و DX=20FFH باشد پس از اجرای دستور ADD AX,DX مطابقت محتوای هر یک از ثباتهای DX, AX و پرجمها؟
- ۴- دستوری انتخاب کنید که DX و BX را با هم جمع کنید و محتوای فلیپ فلاپ Carry را به آن بیافزاید؟
- ۵- اشکال دستور INC [BX] چیست؟
- ۶- قطعه برنامه ای بنویسید که محتوای ثبات BP, SI, DI را از محتوای ثبات AX کم کند و حاصل تفریق را در BX ذخیره کند.
- ۷- دستوری که بتواند عدد یک را از محتوای BL کم کند کدام است؟
- ۸- وقتی که دو عدد ۸ بیتی ضرب می شوند حاصلضرب در چه ثباتی است؟
- ۹- فرق بین دستور MUL و IMUL چیست؟
- ۱۰- یک قطعه برنامه بنویسید که مکعب عدد ۸ بیتی ای که ثبات DL می باشد را محاسبه کند. بعنوان تمرین فرض کنید DL=5 باشد، تحقیق کنید حاصلضرب عدد ۱۶ بیتی می شود.
- ۱۱- اگر عمل تقسیم انجام دهیم خارج قسمت در چه ظرفی خواهد بود؟
- ۱۲- چه دستوراتی بکار می روند تا حاصل عملیات در کد BCD تصحیح شود؟
- ۱۳- قطعه برنامه ای بنویسید که سه بیت سمت چپ DH بدون تغییر سایر بیت ها صفر شوند.
- ۱۴- شرح دهید اختلاف دستور AND و TEST چیست؟
- ۱۵- فرق بین دو دستور NOT و NEG در چیست؟
- ۱۶- نقش پرجم D چیست؟

فصل ششم

- ۱۷- پیشوند REPE چه عملی انجام می دهد.
- ۱۸- چه شرطی باعث ادامه و توقف REPNE SCASB می شود؟
- ۱۹- قطعه برنامه ای بنویسید که در یک بلوک دیتا که با LIST شروع می شود جستجو کند آیا عدد 66 وجود دارد یا خیر؟
- ۲۰- عددی ۱۶ بیتی در خانه ۱۰۰۰ حافظه قرار دارد که شماره (آدرس) پورتی است، برنامه ای بنویسید که این عدد را خوانده و اطلاعات خانه های ۲۰۰۰ تا ۲۰۰FF را روی آن پورت بنویسید.
- ۲۱- برنامه بنویسید که عدد موجود در خانه ۱۰۰۰ حافظه را که یک عدد باینری است به معادل BCD آن تبدیل و در خانه های ۱۰۰۱ و ۱۰۰۲ قرار دهد.
- ۲۲- برنامه بنویسید که دو عدد ۳۲ بیتی را که در خانه های ۱۰۰۰:۰۰۰۰~۱۰۰۳ و ۱۰۰۰:۰۰۰۴~۱۰۰۷ می باشد جمع کند و نتیجه را در ۱۰۰۰:۰۰۰۸~۱۰۰۰B ذخیره کند.

فصل هفتم

«PROGRAM CONTROL INSTRUCCION» دستورات کنترل برنامه

مقدمه :

کامپیوتر زمانی ارزش واقعی خود را پیدا می کند که دستورات کنترل برنامه ای داشته باشد که بتواند آنرا در حین اجرای برنامه راهنمایی کند. در غیر اینصورت باید قدم به قدم به اجرای برنامه پردازد. اما با این گونه دستورات بطور اتوماتیک از یک بخش برنامه به بخش دیگری از برنامه بدون دخالت اپراتور منتقل می شود و حتی می تواند مجدداً به بخش قبلی برگردد. در این فصل سعی می شود تمام دستورات کنترل برنامه ای معرفی شوند. این دستورات **INTERRUPT, RETUR, CALL, JUMP** می نامند.

۱-۷-۱- دستورات گروه پرش (JUMP GROUP)

اصلی ترین دستورات کنترل برنامه ای دستورات پرش (JMP) هستند. که اجازه می دهد کنترل ماشین از یک بخش برنامه به بخش دیگری از برنامه پرش کند. یعنی از یک دستور به **n** دستور بعد یا قبل پرش نماید. و پرش های شرطی اجازه می دهد که برنامه ریز بر اساس یک داده یا نتیجه ای تصمیم گیری نماید. در این قسمت ما راجع به تمام دستورات JMP صحبت خواهیم کرد و کاربرد آنها را تحت عنوان نمونه برنامه های نشان خواهیم داد. همچنین سعی می کنیم راجع به دستورات LOOP نیز مرور دیگری داشته باشیم.

۱-۷-۱-۱- دستورات پرش غیر شرطی (JMP)

با توجه به شکل ۷-۱ در ریزبردازندۀ ۸۰۸۶ سه نوع JMP وجود دارد. و آن سه نوع عبارتند از JMP کوتاه (Short JMP)، نزدیک (Near JMP) و دور (Far JMP). کوتاه یک دستوری است که دارای دو بایت است که اجازه می دهد برنامه نویس به اندازه $+127$ یا -128 محل حافظه را پرش نماید. JMP نزدیک دارای سه بایت است اجازه می دهد کنترل ماشین به هر محلی از همان کد سگمنت که در آن وجود دارد پرش نماید. و بالاخره JMP دور دارای ۵

فصل هفتم

بایت طول می باشد و اجازه می دهد که کنترل ماشین به هر محلی از حافظه که برنامه نویس بخواهد پرش نماید. عموماً به JMP کوتاه و نزدیک و یک پرشهای داخل سگمنت InteraSegment و به JMP دور InterSegment گفته می شود.

۱-۱-۱-۷-پرش کوتاه (ShortJMP)

دستور JMP کوتاه را عموماً پرش (JMP) نسبی نیز می گویند زیرا محل پرش از همان جایی محاسبه می شود که دستور JMP را از حافظه خوانده ای. لذا به اندازه ۱۲۷ محل بعد یا به اندازه ۱۲۸ محل به قبل می تواند برگردد. در واقع آدرس با OP-CODE ذخیره نمی شود بلکه عددی ذخیره می شود (یک بایت) که می تواند مثبت یا منفی باشد. این عدد با IP جمع می شود و IP جدید را برای خواندن دستور العمل جدید ارائه می نماید.

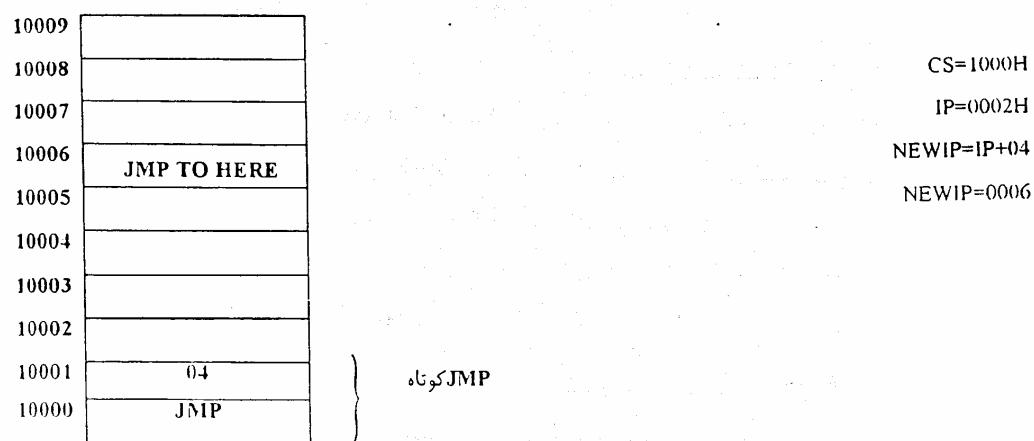
JMP کوتاه	OP-CODE	DISP
	EBH	(a)

JMP نزدیک	OP-CODE	IP-LOW	IP-HIGH
	E9H		(b)

JMP دور	OP-CODE	IP-LOW	IP-HIGH	CS-LOW	CS-HIGH
					(c)

شکل ۱-۷-۱-۷-انواع دستور پرش : (a) پرش کوتاه، (b) پرش نزدیک، (c) پرش دور

اگر ریزپردازندۀ یک دستور JMP کوتاه اجرا نماید. مطابق شکل ۱-۷-۲ محتوای محل بعد از کد عملیاتی (DISP) به مقدار IP اضافه می شود. تا آدرس جدید را تولید کند.



شکل ۷-۲- نمایش یک پرش کوتاه (ShortJMP) به ۴ محل بعد از دستور بعدی

آدرس جدید تولید شده همان آدرس JMP است که باید از آنجا به اجرای برنامه پردازد.

مثال ۷-۱ نیز چگونگی استفاده از این دستور را در برنامه نمایش می‌دهد. در این مثال همچنین نحوه استفاده از Label جدید را آورده است.

Example	7-1
START:	MOV AX,1 ADD AX,BX JMP NEXT ⋮
NEXT:	MOV BX,AX JMP START

در زبان اسمبلی برچسبی (Label) که در محلی بکار می‌رود، دقیقاً آدرس آن محل حافظه است.

دو راه برای معرفی Label برای زبان اسمبلی ۸۰۸۶ وجود دارد.

۱- برچسبی که بیانگر Label نزدیک است.

۲- برچسبی که بیانگر Label دور است.

برچسب نزدیک برای ShortJMP و NearJMP مورد استفاده قرار می‌گیرد. برچسب دور را برای FarJMP کاربرد دارد. فرق برچسب نزدیک با برچسب دور در این است که برچسب نزدیک به ستون (COLON) : ختم می‌شود مثل :NEXT:, START:, LABEL: نمونه هایی از برچسب نزدیک هستند. ولی برچسب دور دارای : نیست این مطلب وجه تمایز دو برچسب است. بخاطر داشته باشید که فقط زمانی بعد از یک برچسب قرار می‌گیرد که منزله آدرس مورد استفاده قرار گرفته است، نه وقتی که بصورت یک اپرند آورده می‌شود. دستور JMP LABEL ایجاد خطای کند، زیرا اینجا LABEL یک اپرند است.

۷-۱-۲- پرش نزدیک (NEAR JMP)

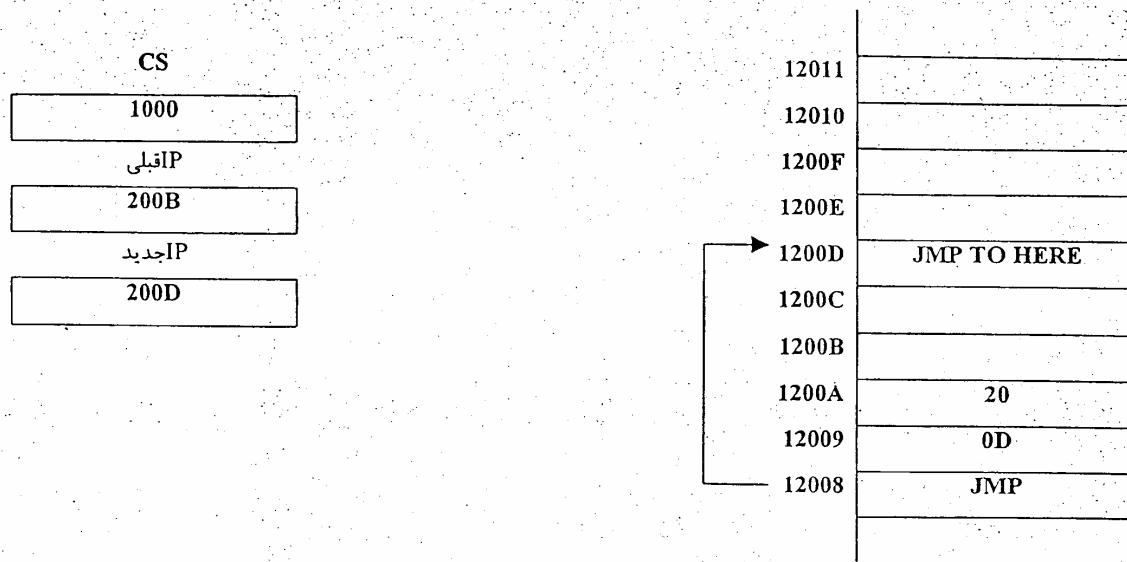
پرش نزدیک هم شبیه پرش کوتاه است با این تفاوت که فاصله پرش بیش از قبلی است یعنی در این پرش دو بایت فضای بعد از کد عملیاتی می‌تواند هر آدرسی را در همان کد سگمنت قبلی ارائه نماید. وقتی که دستور JMP اجرا شود دو بایت بعد از کد عملیاتی درون IP ریخته می‌شود. شکل ۷-۳ نحوه اجرای پرش نزدیک را نشان می‌دهد.

۷-۱-۳- پرش دور (FAR JMP)

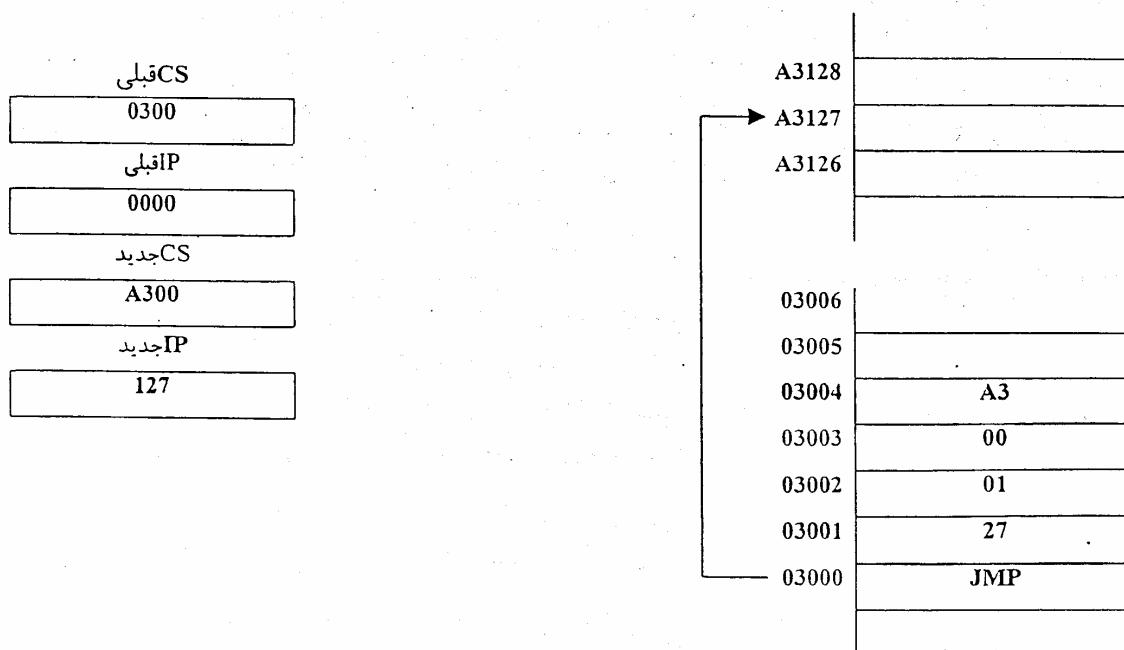
در پرش دور، پرش به هر محلی از حافظه امکان پذیر است زیرا دستور ۵ بایت طول دارد هم سگمنت جدید را معرفی می‌کند و هم IP جدید را ارائه می‌نماید. شکل ۷-۴ چگونگی اجرای

فصل هفتم

آن را نشان می دهد. برخلاف پرش کوتاه و نزدیک در پرش دور از برجسته استفاده می شود که ندارد.



شکل ۷-۳- یک پرش نزدیک محتوی دو محل بعد از کد عملیاتی درون IP ریخته می شود.



شکل ۷-۴- یک پرش دور که هم محتوای IP و CS را عرض می کند.

فصل هفتم

۴-۱-۷-۱-۴- پرش غیر مستقیم ثباتی

در دستور پرش یک ثبات می تواند بعنوان اپنندی که آدرس را داراست استفاده شود. برای این دستور یک آدرس دهی غیر مستقیم برای پرش نزدیک استفاده شده است. آدرس مورد نظر درون ثباتی که در مقابل دستور `JMP` قرار می گیرد. مثل `AX JMP TABLE` محتوای چنین ثباتی دورن IP ریخته خواهد شد. این نوع پرش وقتی که در `JMP TABLE` استفاده می شود کاربرد زیادی دارد، زیرا سرعت اجرا زیاد می شود. مثال شماره ۷-۲ یک جدول پرش را تنظیم کرده است. که می توان به یکی از موارد دورن جدول منتقل شد. فرض کنید محتوای ثبات BX از یک منوی دیتا دریافت شود. اپراتور عدد ۲ را می نویسد و نرم افزار آنرا از صفحه کلید می خواند و به بازنگشی تبدیل می کند و در BX قرار می دهد. و یکی از آن کم می شود و دوباره عدد یک که محتوای BX است به SI افزوده می شود.

بعد از آن دوباره افزوده شدن در SI مقدار `TABLE+2` را داریم و با دستور `MOV AX,[SI]` آدرس TWO درون AX قرار می گیرد و با `JMP AX` محتوای AX که TWO است دورن IP ریخته می شود و کنترل برنامه به آنجا منتقل می شود.

Example	7-2	<pre> MOV SI,OFFSET TABLE DEC BX ADD SI,BX ADD SI,BX MOV AX,[SI] JMP AX . . . </pre>						
	TABLE:	<table border="0" style="width: 100%;"> <tr> <td style="width: 20%;">DW</td> <td style="width: 20%;">ONE</td> </tr> <tr> <td>DW</td> <td>TWO</td> </tr> <tr> <td>DW</td> <td>THREE</td> </tr> </table>	DW	ONE	DW	TWO	DW	THREE
DW	ONE							
DW	TWO							
DW	THREE							

۴-۱-۷-۱-۵- پرش غیر مستقیم حافظه ای

اگر قرار باشد از پرش غیر مستقیم حافظه استفاده کنیم باید از تعاریف NEAR PTR و FAR PTR استفاده کرد. مثال ۷-۳ نشان می دهد که چگونه می توان آدرس دهی غیر مستقیم را استفاده کرد. در JMP اول که آدرس ADDR1 را می دهد بایت آدرس ذخیره کرده چون DD تعریف کرده است یعنی Double Word که هم CS و هم IP بعدی را خواهد داشت. اما در JMP دوم فقط دو بایت که محتوای آن دو بایت IP بعدی خواهد بود.

Example	7-3	<pre> JMP ADDR1 . . . MOV DI,Offset ADDR1 JMP [DI] . . .</pre>								
		<table border="0" style="width: 100%;"> <tr> <td style="width: 30%;">ADDR1:</td> <td style="width: 20%;">DD</td> <td style="width: 30%;">LOOP</td> <td style="width: 10%; text-align: right;">; Far ADDRESS</td> </tr> <tr> <td>ADDR2:</td> <td>DW</td> <td>NEXT</td> <td style="text-align: right;">; Near ADDRESS</td> </tr> </table>	ADDR1:	DD	LOOP	; Far ADDRESS	ADDR2:	DW	NEXT	; Near ADDRESS
ADDR1:	DD	LOOP	; Far ADDRESS							
ADDR2:	DW	NEXT	; Near ADDRESS							

۱-۷-۲- پرشهای شرطی (CONDITIONAL JUMP)

دستورات JMP روی پرچمها اثر نمی‌گذارد. تمام پرشهای شرطی از نوع پرش کوتاه (Short JMP) خواهد بود و حداقل میزان پرش آنها ۱۲۷+ یا -۱۲۸ خواهد بود. شرایط پرش‌های شرطی را به سه گروه می‌توان دسته‌بندی کرد:

- ۱- پرش با بررسی پرچمها
- ۲- پرش با مقایسه اعداد بدون علامت
- ۳- پرش با مقایسه اعداد علامت دار

۱- پرش که روی پرچمها عمل می‌کند به پرچم خاصی اشاره می‌کند، اگر مقدار آن یک (TRUE) شد عمل پرش به آن آدرس داده شده انجام می‌شود در غیر اینصورت به دستور بعد از پرش می‌رود. جدول ۱-۷ دستورات پرش شرطی نوع اول نشان می‌دهد.

جدول شماره ۱-۷- لیست دستورات پرش شرطی روی پرچمها

دستور	شرح دستور	شرایط
JC	JUMP Carry	IF CF=1 THEN JUMP
JNC	JUMP NoCarry	IF CF=0 THEN JUMP
JP	JUMP Parity	IF PF=1 THEN JUMP
JNP	JUMP NoParity	IF PF=0 THEN JUMP
JZ	JUMP Zero	IF ZF=1 THEN JUMP
JNZ	JUMP NoZero	IF ZF=0 THEN JUMP
JS	JUMP Sign	IF SF=1 THEN JUMP
JNS	JUMP NoSign	IF SF=0 THEN JUMP
JO	JUMP OverFlow	IF OF=1 THEN JUMP
JNO	JUMP NoOverFlow	IF OF=0 THEN JUMP

۱- در پرش نوع دوم ابتدا دو عدد بدون علامت مقایسه می‌شوند و دو پرچم ZF, CF مورد بررسی قرار می‌گیرند. نحوه مقایسه به این شکل است که :

CMP DES(منبع)، SORCE(مقصد)

سه حالت اتفاق می‌افتد که بصورت زیر قابل بیان است :

CF	ZF	مقصد	مبدأ
0	0	مقصد	مبدأ
0	1	مقصد	= مبدأ
1	0	مقصد	> مبدأ

با توجه به نتیجه حاصل دستورالعمل‌های شرطی آن در جدول شماره ۱-۷ آورده شده است:

فصل هفتم

جدول شماره ۷-۲- لیست دستورات شرطی که بعد از مقایسه دو عدد بدون علامت می‌آیند

دستور	شرح دستور	شرایط
JA	JUMP ABOVE	IF CF=0 AND Z=0 THEN JUMP
JAE	JUMP ABOVE OR EQUAL	IF CF=0 THEN JUMP
JB	JUMP BELOW	IF CF=1 THEN JUMP
JBE	JUMP BELOW OR EQUAL	IF CF=1 OR ZF=1 THEN JUMP
JE	JUMP EQUAL	IF ZF=1 THEN JUMP
JNE	JUMP NOTEQUAL	IF ZF=0 THEN JUMP

۳- در پرش نوع سوم که در آن شرط به مقایسه اعداد علامت دار ارجاع داده می‌شود. در حالت مقایسه اعداد علامت دار، اگر چه از همان دستور **CMP DES,SOURCE** استفاده می‌شود ولی پرچم‌های بکار رفته برای اعلام نتیجه بقرار زیرند :

OF=SF	با	Z=0	مقدادا	>	مقدادا
ZF=1			مقدادا	=	مقدادا
SF معکوس OF			مقدادا	<	مقدادا

در نتیجه دستور العمل‌های پرش شرطی جداگانه‌ای بکار می‌رود که به شرح زیر خواهد بود :

جدول شماره ۷-۳- لیست دستورات شرطی که بعد از مقایسه دو عدد علامت دار می‌آیند

دستور	شرح دستور	شرایط
JG	JUMP Greater or Equal	IF ZF=0 OR OF=SF THEN JUMP
JGE	JUMP Greater or Equal	IF OF=SF THEN JUMP
JL	JUMP Less	IF OF≠SF THEN JUMP
JLE	JUMP Less or Equal	IF ZF=1 or SF≠OF THEN JUMP
JE	JUMP Equal	IF ZF=1 THEN JUMP

یک دستور العمل پرش شرطی دیگری هم وجود دارد به صورت زیر است :

JCXZ IF CX=0 THEN JUMP

ممکن است سؤال شود اگر بخواهیم پرشی به بیش از محدوده $-128 \sim +127$ داشته باشیم چه باید کرد؟ بدیهی است باید از دستورات پرش غیر شرطی استفاده کرد. فرض کنید از دستور رشته‌ای SCASB برای پیدا کردن عدد 0AH در یک بلوک بطول ۱۰۰ بایت استفاده کنیم. (مثال شماره ۷-۴) در دو صورت از حلقه جستجو خارج می‌شود یکی اینکه دیتای مورد نظر (0AH) را بیابد و دیگر اینکه چنین دیتائی در بلوک پیدا نکند و با **CX=0** خارج شود، با استفاده از دستور JCXZ می‌توان مشخص کرد علت خروج کدام یک از دو حالت فوق بوده است.

Example 7-4
 SCAN: MOV DI,OFFSET TABLE
 MOV CX,100
 MOV AL,0AH
 CLD
 REPNE SCANSB
 JCXZ NOT-FOUN

۳-۱-۷- دستور LOOP

دستور LOOP ترکیبی است از یک پرش شرطی و دستور کم کردن از CX در واقع یک واحد از CX کم می کند اگر ثبات CX صفر نشده باشد به برجسبی که در مقابل دستور LOOP نوشته شده است پرش می کند اما اگر CX=0 شده باشد به دستور بلافصله بعد از LOOP رفته و اجرای برنامه ادامه پیدا می کند.

مثال شماره ۷-۵ چگونگی جمع کردن دیتاها ای از یک بخش حافظه (BLOCK1) را با دیتاها در بخش دیگر حافظه (BLOCK2) را نشان می دهد و بکار گرفتن دستور LOOP برای کنترل تعداد جمع ها می باشد.

Example	7-5
	MOV CX,100
	MOV SI,OFFSET BLOCK1
	MOV DI,OFFSET BLOCK2
AGAIN:	LODSW
	SEG ES:
	ADD AX,[DI]
	STDSW
	LOOP AGAIN

نکته دیگری که در مثال ۷-۵ قابل توجه است تغییر سگمنت است که توسط پیشوند SEG ES: آورده شد، در واقع جمع AX با دیتائی است در اکسترا سگمنت که DI آدرس آنرا می دهد.

۱-۳-۱-۷- دستور حلقه شرطی

دستور LOOP صور دیگری نیز دارد که به آنها (Conditional Loop) حلقه شرطی نیز می گویند مثل LOOPNE, LOOPNE اولی می گوید حلقه ایجاد کن اگر مساوی هستند یعنی اگر CX برابر صفر نشده و شرط تساوی برقرار است به آدرس داده شده مقابل دستور LOOP می رود در غیر اینصورت از LOOP خارج شده و به دستور بلافصله بعد از LOOP رفته و برنامه ادامه پیدا می کند. دستور LOOPNE عکس قبلی عمل می کند.

دستور دیگری که در این زمینه وجود دارد LOOPZ است البته همانند دستورات قبلی LOOPNZ نیز وجود دارد، ناگفته نماند که LOOPNE همانند LOOPZ و LOOPNE همانند LOOPNZ می باشد.

۲- زیربرنامه (SUBROUTINE)

زیربرنامه یک بخش خیلی مهم هر ساختار نرم افزاری یک کامپیوتر محسوب می شود. یک مجموعه دستوراتی که یک عمل مشخصی را انجام می دهند بصورت زیربرنامه در می آورند. اهمیت آن از این نظر است که ممکن است شما چندین بار این قسمت را بکار بگیرید، یک بار نوشتن این بخش و چند بار استفاده کردن مسلم است که نقش مهمی خواهد داشت. هم حافظه

فصل هفتم

صرفه جوئی می شود، هم برنامه شما ساده تر می شود. زیربرنامه توسط دستوری بنام CALL مورد خطاب قرار می گیرد و کنترل ماشین توسط دستور RET از زیربرنامه به برنامه اصلی بر می گردد. هنگام رفتن سراغ زیربرنامه از پشته (STACK) جهت ذخیره کردن آدرس محل برگشت به برنامه اصلی استفاده می شود. تا با اجرای دستور RET به دستور بلا فاصله بعد از CALL برگردد.

در زبان اسambilی ۸۰۸۶ قواعد محدودی برای ذخیره کردن زیربرنامه وجود دارد که ذیلاً ذکر می شود. اول اینکه مثل زبان های سطح بالا (عنوان مثال (PASCAL) زیربرنامه را رویه (PROCEDURE) می نامند.

مثال ۷-۶ دو نوع رویه نزدیک و دور را نشان می دهد (Far Procedure-Near Procedure). اگر دستور CALL زیربرنامه ای را صدا بزنند که در همان کد سگمنت باشد آنرا رویه نزدیک می گویند. اما اگر رویه ای که دستور CALL آنرا مورد خطاب قرار داده در کد سگمنت دیگری باشد آنرا رویه دور می گویند.

به مثال فوق دقت کنید دو شبه دستور (Pseudo-Opcode) مشاهده می شود که یکی PROC و دیگری ENDP است. PROC برای نشان دادن محل شروع زیربرنامه است که توسط اسمی که برای زیربرنامه انتخاب می کنید برچسب زده می شود. نوع زیربرنامه بعد از PROC (نزدیک یا دور بودن آن) آورده می شود.

Example 7-6

NAME	PROC	FAR	:	
	MOV	AX,BX		
	ADD	AX,CX		
	RET			
NAME	ENDP		:	

این دستور اسم زیربرنامه و نوع آن و شروع آنرا مشخص می کند

این دستور برگشت به برنامه اصلی است

این دستور به اسambilر می گوید اینجا پایان زیربرنامه تعریف شده است

LABEL	PROC	NEAR	:	
	MUL	BX		
	MUL	CX		
	MUL	DX		
LABEL	ENDP			

شروع زیربرنامه با نام LABEL او از نوع نزدیک است

پایان زیربرنامه LABEL

در مثال بالا دو نوع زیربرنامه یکی از نوع نزدیک و یکی از نوع دور معرفی شده است، چگونگی شروع و پایان آنها نیز مشخص گردیده است. اغلب زیربرنامه ها که بخشی از کتابخانه سیستم محسوب می شوند باید بصورت رویه دور (Far Procedure) نوشته شوند. بعلاوه هر JMP درون هر زیربرنامه باید از نوع کوتاه (Short) باشد. برای اینکه بتوانند آنرا براحتی جابجا کنند. زیرا پرش کوتاه است که این خاصیت را دارد. و می توان آنرا به هر برنامه ای اضافه کرد. در یک زیربرنامه عمومی، تمام ثبات هائی که در زیربرنامه بکار گرفته می شوند باید در پشته ذخیره شوند و در انتهای زیربرنامه قبل از برگشت باید از پشته برداشته شوند.

۷-۲-۱- دستورات CALLS

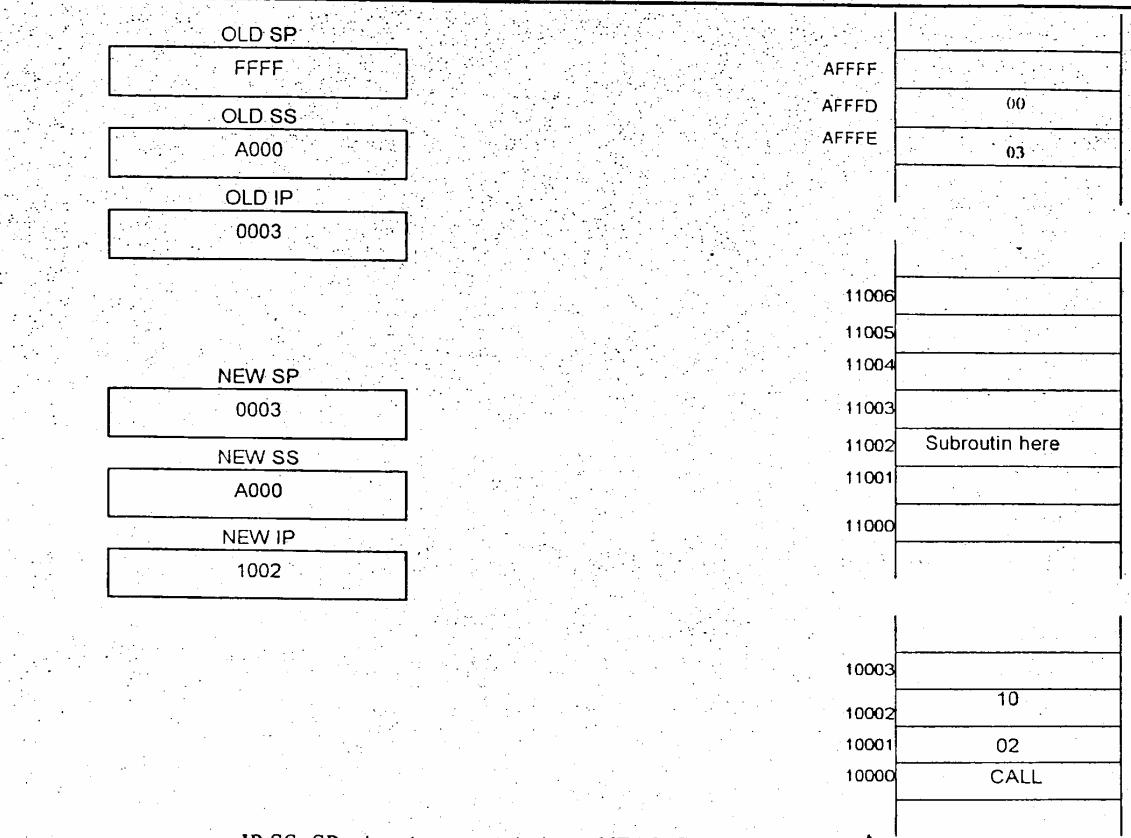
دستور CALL کنترل ماشین را از برنامه به یک زیربرنامه منتقل می‌کند. و این انتقال با انتقالی که در دستور JMP داشتیم متفاوت است، زیرا در اینجا باید محتوای IP را در پشته ذخیره کند (اگر CALL از نوع دور باشد باید IP و CS را در پشته ذخیره کند) آنوقت عمل انتقال صورت بگیرد. هیچگونه تأثیری روی پرچمها ندارد.

۷-۲-۱-۱ NearCall

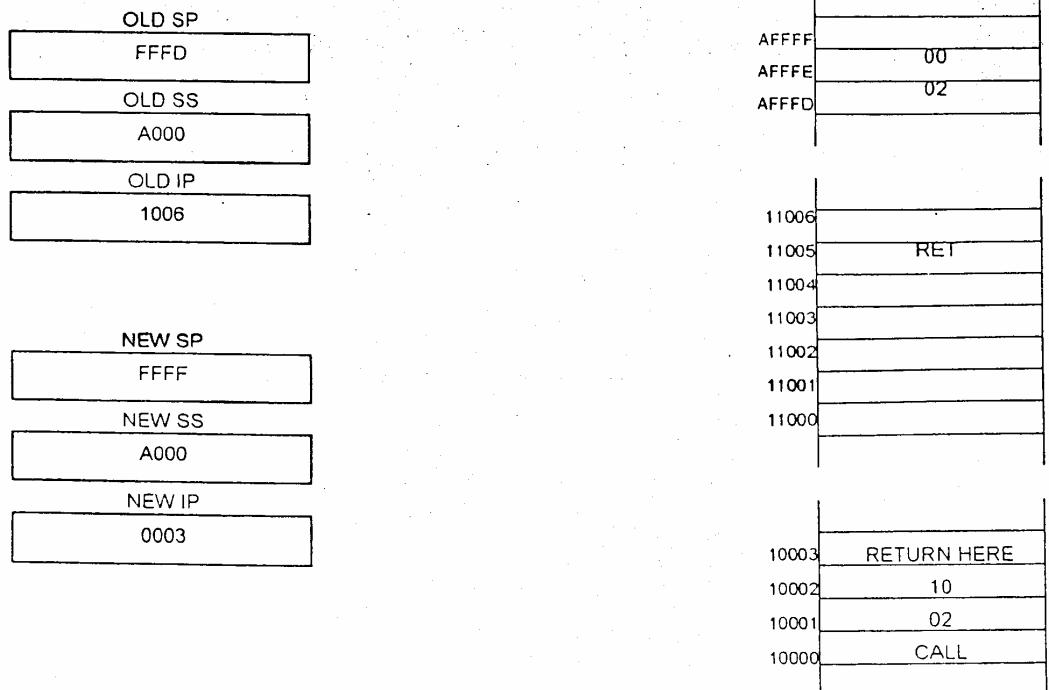
دستور CALL نزدیک ۳ بایت است، بایت اول طبق معمول OP-CODE و بایت دوم و سوم حاوی آفست آدرس محل زیربرنامه می‌باشد لذا پس از ذخیره کردن آدرس دستور بلافاصله بعد از CALL در سازمان پشته بایت دوم و سوم بعد از OP-CODE، درون IP ریخته خواهد شد. ذخیره کردن آدرس برگشت توسط عمل PUSH در سازمان پشته صورت می‌گیرد. شکل ۷-۵ نشان می‌دهد که چگونه آدرس برگشت ذخیره شده و عمل پرس به زیربرنامه صورت گرفته است.

۷-۲-۱-۲ FAR CALL

دستور FAR CALL شبیه FAR JUMP است زیرا امکان صدا زدن زیربرنامه در هر محلی از حافظه را فراهم می‌کند. این دستور ۵ بایت طول دارد که بایت اول OP-CODE و بایت دوم و سوم دارای محتوای IP هستند و بایت چهارم و پنجم کد سگمنت جدید را در بر دارند که باید به ثبات CS ریخته شوند. هنگام اجرای دستور FAR CALL باید آدرس دستور برگشت هم و هم CS در پشته ذخیره می‌شوند.



شکل ۵-۷ تأثیر یک دستور NEAR CALL در سازمان پشتی و نبات های IP,SS, SP



شکل شماره ۷-۶- جگونگی اجرای دستور RET و بارگذشت به برنامه اصلی

فصل هفتم

۷-۲-۱-۳- دستور CALL با مدهای آدرسی دهی مختلف

سه روش برای نوشتن آدرس رویه NEAR و دو روش مختلف برای آدرس دهی FAR وجود دارد که ذیلاً به شرح آنها پرداخته می‌شود:

الف- مستقیم: مانند

```
CALL PROC1
...
...
PROC1 PROC NEAR
...
...
RET
PROC1 ENDP
```

ب- غیر مستقیم ثباتی، مانند

```
CALL [SI]
```

که انتقال به آدرسی که در SI می‌باشد صورت می‌گیرد. مثال ۷-۷ چگونگی استفاده از ثبات را نشان می‌دهد، در این مثال زیر برنامه ای صدای زده می‌شود که شروع آن در محلی است

بنام COMPUTER

ج- غیر مستقیم حافظه ای

```
CALL WORD PTR[DI]
```

در این روش عمل انتقال به محلی صورت می‌گیرد که آدرس آن محل در حافظه است و بوسیله DI محل آدرس در حافظه مشخص می‌شود. این روش در رویه FAR نیز وجود دارد که در معرفی باید DWORD آورد که ذکر خواهد شد.

Example	7-7
	MOV SI,OFFSET COMPUTE
	CALL SI
	⋮
	⋮
COMPUTE	PROC NEAR
	PUSH DX
	MOV DX,AX
	IN AX,DATA
	OUT PORT,AX
	MOV AX,DX
	POP DX
	RET
COMPUTE	ENDP

مثال شماره ۷-۸ چگونگی آدرس دهی غیر مستقیم حافظه ای را نشان می‌دهد. در اینجا فرض شده DI دارای عدد یک یا دو باشد است که هر کدام را به ۰، ۲، ۴ تبدیل و با SI جمع

فصل هفتم

می کند. تولید آدرس می کند، در واقع از تکنیک LOOK UP TABLE استفاده می کند که TABLE این مثال دارای سه آدرس برای سه رویه مختلف است به یکی از آنها دسترسی پیدا می کند.

Example 7-8

```
TABLE: DW ONE
        DW TWO
        DW THREE
```

ابتدا TABLE را تعریف می کند

عمل تبدیل یک یا دو یا سه به 0 یا 2 یا 4 انجام می شود و یکی از سه رویه صدای زده می شود

```
DEC  DI
MOV  SI,OFFSET TABLE
ADD  DI,SI
CALL [DI+SI]
```

```
ONE   PROC NEAR
      :
      RET
ONE   ENDP
;
TWO   PROC NEAR
      :
      RET
TWO   ENDP
;
THREE PROC NEAR
      :
      RET
THREE ENDP
```

زیربرنامه اول

زیربرنامه دوم

زیربرنامه سوم

اما روش های آدرس دهی برای FAR CALL که قبلاً ذکر شد دو روش به شرح زیر می باشد

الف- مستقیم ولی خارج از قطعه، مثل

```
CALL    PROC1
      :
      :
PROC1  PROC FAR
      :
      RETF
PROC1  ENDP
```

ب - غیر مستقیم حافظه ای : این روش شبیه غیر مستقیم حافظه برای NEAR CALL است با این تفاوت که در دستور باید Dword (Double Word) ذکر گردد. مثال :

CALL DWORD PTR[DI]

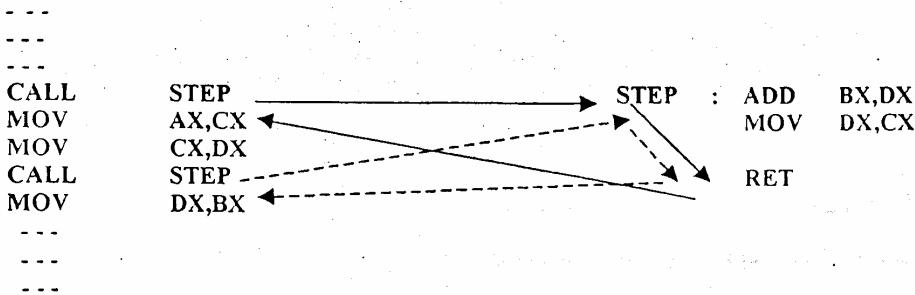
هنگام اجرای دستور کنترل ماشین به محل CS:IP منتقل می شود که مقدار IP آن از محلی از حافظه که DI و DI+1 آدرس دهی می کند پر می شود CS آن از محلی که DI+2 و DI+3 آدرس دهی می کند پر خواهد شد.

۷-۲-۲ - دستورات RETS

این دستور روی پرچمها تأثیری ندارد و مخفف کلمه RETURN است، که معنی برگشت می دهد، اگر از نوع Near باشد ۱۶ بیت تحت عنوان OFFSET آدرس از پشته برداشته و در IP می ریزد و اگر از نوع FAR باشد ۳۲ بیت برداشته ۱۶ بیت در IP و ۱۶ بیت در کد سگمنت CS می ریزد. نوع FAR یا Near بودن آن در تعریف رویه مشخص شده است. و بصورت اتوماتیک در اسمبلر انتخاب می شود.

هنگامیکه IP یا CS, IP از پشته پر شدند آدرس دستور العمل بعدی محل جدیدی خواهد بود که این محل جدید دستور بلا فاصله بعد از CALL است که به این زیر برنامه آمده بود.

شکل ۷-۷ چگونگی صدا زدن یک زیر برنامه بوسیله CALL و چگونگی برگشت از زیر برنامه توسط دستور RET را نشان می دهد.



شکل ۷-۷- یک فقط برنامه کوتاه همراه با زیر برنامه که در برنامه اصلی صدا زده شده و پس از اجرای زیر برنامه به برنامه اصلی مراجعه می کند

دستور برگشت دیگری داریم که به محتوای SP عددی را می افزاید، آنوقت IP را از محلی که SP نشان می دهد پر می کند تا به برنامه اصلی برگردد که بصورت n RET نمایش داده می شود. N همان عددی است که به SP اضافه می شود.

مثال ۷-۹ دستور 4 RET را نشان می دهد که چگونه عمل می کند. این دستور عدد 4 را به SP اضافه می کند. زیرا دو دستور PUSH AX و PUSH BX چهار بایت فضای پشته را پر کرده اند و لذا AX, BX را پاک می کند فقط به POP کردن IP می پردازد البته این یک RET خاص محسوب می شود و ضعیت نرمال RET چنین نیست.

Example 7-9

```

TEST-RIG PROC NEAR
    PUSH AX
    PUSH BX
    -
    -
    -
    RET    4
TEST-RIG ENDP

```

۷-۳-۱-۱ INTRUPT - وقفه

وقفه بمعنی ایجاد توقف در روند جاری برنامه و پرداختن به یک زیربرنامه تا اتمام آن، و پس از آن به برنامه اصلی برگشتن و روند جاری را ادامه دادن است.

وقفه ممکن است توسط سخت افزار تولید شود، ممکن است بواسیله نرم افزار ایجاد شود. در هر صورت برنامه در حال اجرا را متوقف می کند و باعث صدا زدن زیربرنامه سرویس وقفه می شود و آنرا تا آخر اجرا می نماید. درباره وقفه سخت افزاری در فصل ۹ بحث خواهد شد. در این قسمت راجع به وقفه های نرم افزاری صحبت می کنیم، یکی از این وقفه ها همان دستور CALL است که قبل از بحث شد، انواع دستور CALL دیگر در ریزپردازنده ۸۰۸۶ وجود دارد بنام دستورات وقفه که سه نوع آن را بنام INT3، INT0، INT و بردار وقفه و دستور IRET را مورد بحث قرار می دهیم.

۷-۳-۱-۲-۱ بردار وقفه چیست؟ (INTRUPT VECTOR)

بردار وقفه چهار بایت حافظه ای است که در قسمت ۱۰۲۴ محل اول حافظه (یعنی از آدرس 00000H تا 003FFH) بمنظور نگهداری آدرس برنامه سرویس یک وقفه خاص انتخاب شده است. ریزپردازنده ۸۰۸۶ دارای ۲۵۶ نوع وقفه است و لذا ۲۵۶ بردار وقفه باید داشته باشد که آدرس اولین دستور برنامه سرویس وقفه را نگهداری کنند. هم وقفه های سخت افزاری و هم وقفه های نرم افزاری دارای چنین برداری هستند. بردارهای وقفه و توابع آنها و آدرس برنامه هر کدام در جدول ۷-۴ آورده شده اند. هر بردار ۴ بایت طول دارد که دو بایت حاوی مقداری که باید درون IP و دو بایت مقداری که باید درون ثبات CS ریخته شود

۷-۳-۱-۲-۲ دستورات وقفه :

ریزپردازنده ۸۰۸۶ دارای سه نوع مختلف وقفه است. INT3، INT0، INT هر کدام از این سه نوع، یک بردار را از جدول بردارهای وقفه می خواند و پس از آن به زیربرنامه مورد نظر که

آدرسشن را خواند دسترسی پیدا می کند. بعبارت دقیقتر برای اینکه وقفه اجرا شود اتفاقات زیر صورت می پذیرد :

SP-۱ دو واحد کسر می شود و پرچم ها در پشته ذخیره می گردند.

SP-۲ دو واحد کسر می شود و CS در پشته ذخیره می گردد

SP-۳ دو واحد کم شده و IP که آدرس دستور بعد از دستور وقفه را دارد در پشته ذخیره می شود.

۴- شماره نوع وقفه را در چهار ضرب می کند تا آدرس جدول بزرگدار بدهست آید.

با شروع از این آدرس دو بایت اول را در IP و دو بایت بعد را در CS می ریزد، تا از اولین دستور برنامه وقفه شروع به واکشی دستور (Fetch) و اجرا نماید.

IF-۵ را پاک می کند.

جدول شماره ۷-۴- جدول بزرگدارهای وقفه

شماره بزرگدار	آدرس	حالت و منظور از استفاده
0	0~3H	خطای تقسیم
1	4H~7H	اجرای قدم به قدم برنامه
2	8H~BH	وقفه سخت افزاری غیر قابل پوشش (NMI)
3	CH~FH	ایجاد نقطه شکست در برنامه
4	10H~13H	OVER FLOW
5~31	14H~7FH	رزرو برای آینده
32~255	80H~3FFH	وقفه های استفاده کنند.

ملحوظه می کنید که دستور INT نرم افزاری شبیه دستور CALL است با این فرق که در دستور وقفه پرچمها هم در پشته ذخیره می شوند.

گفته شد که ۲۵۶ نوع وقفه داریم و هر وقفه دارای عددی است که ممکن از ۰ تا ۲۵۵ (از H ۰۰H تا FFH) باشد بعنوان مثال وقفه شماره ۱۰۰، بزرگدار وقفه شماره ۱۰۰ را در اختیار می گیرد که هر کدام از دستورات دارای ۲ بایت طول هستند، بغير از دستور وقفه (INT3) که استثنای دارای یک بایت می باشد، نکته دیگری که قابل ذکر است وقتی که وقفه پذیرفته شد فلایپ فلاپ ۱ صفر می شود این بیت کنترل کننده وقفه خارجی است (ینعی پایه INTR) یعنی پایه درخواست کننده وقفه را غیرفعال می کند. وقتی که ۱ دارای مقدار یک منطقی است پایه وقفه فعال می شود. راجع به پرچم T بعداً در مبحث وقفه ها صحبت می کنیم.

وقفه های نرم افزاری معمولاً در سیستم زیربرنامه ای بکار می روند تا وسایل مثل CRT، چاپگر و صفحه نمایش را کنترل کنند.

دستور CALL که معمولاً دارای ۵ بایت طول می باشد برای اینگونه توابع کاربرد دارد و دستورات وقفه دارای دو بایت طول هستند لذا از دستورات وقفه دو بایتی بجای دستورات CALL

فصل هفتم

۵ بایتی استفاده می کنند تا ۳ بایت ذخیره شود. بدینه است که اینگونه ذخیره سازی محل های حافظه در طول یک برنامه قابل ملاحظه خواهد شد.

۷-۳-۳- دستور IRET

دستور IRET معنی برگشت از برنامه وقفه است. کلیه پرچم ها را متأثر می سازد (CF, PF, AF, ZF, SF, TF, IF, DF, OF) یعنی مقدار همه پرچمها همان مقداری که قبل از وقفه داشتند بار می شوند.

دستور IRET هم برای وقفه های نرم افزاری و هم برای وقفه های سخت افزاری کاربرد دارد. هنگام اجرای این دستور :

۱- مقدار IP که در پشته PUSH شده بود POP شده و در IP ریخته می شود.

۲- مقدار CS که در پشته PUSH شده بود POP شده و در CS ریخته می شود.

۳- مقدار ثبات پرچم ها که در پشته PUSH شده بود POP شده و در ثبات پرچم ریخته می شود

در واقع می توان گفت IRET ترکیبی است از دستور RET و POPF

گفته شد با اجرای دستور وقفه پرچمها T, I, صفر می شوند، حال با برگرداندن مقدار پرچمها به ثبات پرچمها مجدداً مقدار قبلی T, I، به حالت قبل از دستور INT برمی گردند یعنی اگر قبل از مجاز بود که به سیستم وقفه وارد شود فقط در طول اجرای برنامه وقفه است که I غیر فعال می شود.

۷-۳-۴- دستور INT3

برخلاف سایر دستورات وقفه که دو بایتی هستند این دستور یک بایتی است و نام آن ایجاد نقطه توقف در برنامه است. یعنی برای رفع عیب برنامه نوشته شده در نقاط معینی دستور وقفه 3 قرار داده می شود برنامه تا آنجا اجرا می شود تا نتیجه وارسی گردد. مجدداً می توان از یک نقطه توقف تا نقطه توقف بعدی پیش روی کرد. هنگام اجرا پس از وارسی می توان یک بایت گذاشته شده را حذف کرد.

۷-۳-۵- دستور INT0

این دستور وقفه سرفتگی است (Overflow) پرچمها IF, TF را متأثر می کند، این دستور در صورتی اجرا می شود که فلیپ فلاپی که برای سرفتگی در نظر گرفته ایم نیز یک شود. حال اگر هنگام اجرای برنامه OF=1 شود، CPU به مکان حافظه به آدرس $10H \times 4 = 40H$ می رود و محتوای مکانهای $10H$, $11H$, $12H$, $13H$ را درون CS ریخته و سراغ قطعه برنامه ای

فصل هفتم

که آدرس اولین دستور آن را برداشته است می‌رود. پس اگر سرفتگی اتفاق نیافتد این دستور اجرا نمی‌شود شیوه $J0$ می‌باشد (یعنی اگر $OF=1$ آنگاه $JMP J0$ کن)

۷-۳-۶- دستور وقفه ۵۰ (INT 50)

فرض کنید در یک بخشی از سیستم اغلب لازم باشد که محتوای BX, BP, SI, DI را جمع کنیم و حاصل جمع را رد AX قرار دهیم. چون می‌خواهیم این برنامه را اغلب داشته باشیم ترجیحاً آنرا بصورت وقفه بنویسیم، البته می‌توان بصورت زیربرنامه هم نوشت اما بصورت وقفه بهتر است مثال ۷-۱۰ این برنامه وقفه را نشان می‌دهد و مشخص شده که چگونه بردار وقفه آدرس برنامه را ارائه می‌نماید. هر وقت که نیاز به این قسمت برنامه باشد بوسیله INT50 این برنامه صدا زده می‌شود

Example	7-10
ADDM :	ADD AX,BX
ADDEM :	ADD AX,SI
	ADD AX,DI
	ADD AX,BP
	IRET

برقراری آدرس بردار به شرح زیر است :

ORG	00088H
DD	ADDEM

شماره بردار وقفه = ۵۰ ;

۷-۳-۷- کنترل وقفه

این بخش نیز شامل وقفه‌های سخت افزاری نمی‌شود. حال ضروری است که دو دستوری که کنترل کننده وقفه‌های سخت افزاری هستند را معرفی کنیم، اولی دستور نشاندن پرچم وقفه (SET INTERRUPT FLAG) که پرچم STI (SET INTERRUPT FLAG) را یک می‌کند که اجازه به ورود پایه سخت افزاری وقفه بدهد (INTR). دوم دستور CLI (CLEAR INTERRUPT FLAG) است پاک کننده پرچم وقفه (CLEAR INTERRUPT FLAG) که صفر در پرچم ۱ قرار می‌دهد و پایه ورودی سخت افزاری را غیر فعال می‌کند.

۴- سایر دستورات کنترل ماشین

دستورات قبلی که راجع به آنها بحث شد راجع به کنترل ماشین بودند. دستوراتی را که حال می‌خواهیم مطرح کنیم کنترل کننده پرچم نقلی (Carry Bit)، نمونه برداری از پایه TEST یا اجرا کننده حالات مختلف دیگر در سیستم می‌باشند که اغلب در کنترل سخت افزاری عمل می‌کند

فصل هفتم

که در انجام فقط نام آنها را می بینم. در مبحثی که راجع به وقته سخت افزاری صحبت خواهیم کرد اینها بطور دقیق و کامل مورد بحث قرار می گیرند.

۱-۴-۷- دستورات کنترلی پرچم نقلی (CONTROLLING THE CARRY FLAG BIT)

پرچم نقلی در جمع و تفریق های چند کلمه ای، نشان دهنده خطأ در زیر برنامه کاربر دارد. سه دستور داریم که اجازه می دهد برنامه ریز محتوای پرچم نقلی را تغییر دهد. CMC، STC، CLC که به ترتیب در قیلیپ فلاپ Carry صفر و یک می گذارد و یا متضم می کند.

۲-۴-۲- دستور WAIT

این دستور ریز پردازنده را به حالت انتظار می برد. روی پرچم ها تأثیری ندارد. به این نحو عمل می کند که پایه سخت افزاری TEST را بررسی می کند، اگر مقدار آن یک باشد در حالت WAIT می ماند این حالت تداوم دارد تا در پایه TEST یک صفر ملاحظه کند، آنوقت از WAIT خارج می شود. این پایه معمولاً توسط ریز پردازنده ۸۰۸۷ ۸۰۸۶ که کمکی ۸۰۸۶ محسوب می شود مورد استفاده قرار می گیرد.

۳-۴-۳- دستور HLT

محفظ کلمه HALT است. این دستور موجب می شود که ریز پردازنده اجرای دستورات را متوقف کند و به حالت کما برود. و توسط INTR یا RESET یا NMI سخت افزاری از حالت HLT خارج می شود. این دستور در حالت های خاص مثلاً در انتهای یک برنامه بکار می رود.

۴-۴-۴- دستور NOP:

روی پرچم ها تأثیری ندارد. عمل فتح را انجام می دهد IP را بهنگام می کند که آماده رفتن به دستور بعدی باشد ولی هیچ کاری انجام نمی دهد. برای تنظیم تأخیرها در اتصال پالس های ساعت بکار می رود سه پالس ساعت زمان می برد.

۵-۴-۵- دستور LOCK

این دستور بنام پیشوند قفل گذرگاه سیستم معروف است این دستور یک بایت است که قبل از هر دستور ۸۰۸۶ قرار می گیرد تا از دسترسی ریز پردازنده دیگر موجود در سیستم به باس(CO-PROCESSOR) ممانعت بعمل آورد.

۶-۴-۷- دستور ESC (ESCAPE INSTRUCTION)

روی پرچمهای تأثیری ندارد و باعث عبور اطلاعات به پردازنده کمکی (ریزپردازنده ۸۰۸۷) می شود. در واقع استفاده از کمک پردازنده ها که باید گذرگاه های آدرس و داده را بصوزت مشترک استفاده کنند را ممکن می سازد. در واقع هر وقت دستور ESC اجرا می شود ریزپردازنده ۸۰۸۶ عمل NOP را اجرا می کند. و یاس ها در اختیار ریزپردازنده کمکی قرار می گیرد تا در حافظه نویسید یا از حافظه بخواند.

۷- برنامه های نمونه

در اینجا به معرفی چند برنامه نمونه می پردازیم تا با سازمان دهی برنامه برای IBM-PC ها یا نوشتن برنامه به زبان اسمنلی آشنا شویم. همچنین بعضی از تکنیک های برنامه نویسی را که کاربرد زیادی دارند معرفی می کنیم.

۱-۵-۷- اماه کردن یک برنامه برای یک فایل اجرائی (DISK OPERATING SYSTEM)

یک فایل اجرائی عملاً یک فرمان برای سیستم عامل DOS می باشد بعنوان مثال اگر شما یک فایل اجرائی بسازید و اسم آنرا DOG.EXE بگذارید. آنوقت باید نویسید که اسم این برنامه DOG است و باید آنرا اجرائی کنید. راه اندازی یک برنامه برای اسمنلر سیستم های IBM-PC کار آسانی نیست اگر چه ممکن است برای بعضی سیستمهای دیگر آسان باشد. هر بخش باید توضیح داده شود. بخشها مرتب شوند. مثال ۵-۱۱ یک الگویی را ارائه می کند که در اغلب برنامه های به زبان اسمنلی کاربرد دارد و همه زیر برنامه ها و برنامه اصلی آورده شده است. در این مثال وقتی که اسم صدا زده می شود وارد دیسک می شود و همراه با ادیتور برنامه را طبق مراحل زیر ترجمه می کند.

۱- اسم ماکرو اسمنلر یا اسم اسمنلر (بر حسب اینکه کدام یک را دارید) را می نویسند که برنامه

را ترجمه کند. این عمل فایل جدیدی می سازد که همان اسم شما را با پسوند .OBJ.

(name.OBJ) انتخاب می کند که این برنامه قابل اجرا نیست.

۲- یک اسمی که پیونددنه این برنامه با قسمت های دیگر ش اگر وجود داشته باشد

بنام LINKname می نویسید که برنامه را به فایل قابل اجرا تبدیل می کند و اسمی با

پسوند .EXE. برای آن انتخاب می کند.

۳- اسم برنامه را می نویسید تا برنامه اجرا شود (البته اسمی که پسوند .EXE. دارد) شما روی

صفحه نمایش (پس از گذاشتن از ۵ خط اول) چه باید بینید؟

GARRAEE.HASSAN
WAS HERE !

فصل هفتم

۱-۱۰-۵-۷- معادل سازی برنامه :

اولین بخش هر برنامه لیستی معرفی می شود که معادل سازی برنامه نامیده می شود. این بخش بکار می رود که با استفاده از راهنمای EQU برجسب های شما را با مقادیر مختلف بکار رفته در طول برنامه هم ارز یا معادل می سازد، راهنمایی EQU دستوراتی را توصیف می کند که هم در برنامه اصلی و هم در زیر برنامه ها مورد استفاده قرار گرفته اند و داده های را معرفی می کند که در برنامه استفاده شده اند. هم ارز سازها (معادل سازها) انتخابی هستند ولی برنامه را خیلی راحت قابل فهم می سازند و برنامه توأم با مستندات می شود. توجه کنید وقتی انتهای پیغام را \$ معرفی می کنید چقدر راحتتر می شود تا اینکه بخواهیم بنویسید : MESAGE-END

یک نمونه برنامه که چند پیام را روی صفحه نمایش ارسال خواهد کرد.

Example 7-11

```

;-----;
;      PROGRAM   EQUATES
;-----;

MESAGE-END    EQU     '$'      ;      بایان پیغام هم ارز $ است
CR           EQU     0DH      ;      هم ارز ENTER
LF           EQU     0AH      ;      هم ارز LineFeed
SP           EQU     ' '      ;      معادل یک فاصله SPACE
ZERO         EQU     0        ;
NINE          EQU     0        ;

;-----;
;      STACK SEGMENT
;-----;

STACK-SEG      SEGMENT PARA PUBLIC
DB             256 DUP(?) ;      ۲۵۶ بایت برای پشته ذخیره می کند
STACK-SEG      ENDS

;-----;
;      DATA   SEGMENT
;-----;

DATA-SEG      SEGMENT PARA PUBLIC
MESAGE-ONE    DB      CR
                DB      5 DUP(0AH)
                DB      'GHARAEEL'      'HASSAN'
                DB      CR
                DB      LF
                DB      MESAGE-END
MESAGE-TWO    DB      'WAS HERE?!!'
                DB      MESAGE-END
DATA-SEG      ENDS

```

```

;      CODE SEGMENT
;
CODE-SEG      SEMENT PUBLIC
ASUME          CS:CODE-SEG      DS:DATA-SEG
;
;      MAIN PROGRAM PROCEDURE
;
MAIN          PROC FAR
PUSH SS
MOV AX,ZERO
PUSH AX
MOV AX,DATA-SEG
MOV DS,MESAGE-ONE
CALL DISPLAY-MESSAGE
MOV DX,MESAGE-TWO
CALL DISPLAY-MESSAGE
RET
MAIN          ENDP
DISPLAY-MESSAGE PROC NEAR
MOV AH,NINE
INT 21H
RET
DISPLAY-MESSAGE ENDP
CODE-SEG       ENDS
END             MAIN

```

۷-۵-۱-۲- بخش پشته (STAK-SEGMENT)

بخش بعدی برنامه قسمت پشته است. در واقع اینجا پشته برقرار و تعریف می شود و اجازه می دهد که شما از دستورات RET, CALL و یا هر دستور دیگری در رابطه با POP, PUSH استفاده کنید.

اولین بخش در اینجا شبیه عمل SEGMENT که معرفی کننده برای زبان اسembly است نه کد ماشین که عمل انجام می دهد. یعنی شبیه عمل SEGMENT شروع را برای زبان اسembly تعریف می کند یعنی می گوید شروع بخش است (Paragraph Boundary=PARA)

علاوه کلمه SEGMENT به بخش متصل کننده اسembly (STACK-SEG IS PUBLIC) و لذا بخش متصل کننده می تواند با سایر بخش ها بهمان نام آنرا متصل کند دستور العمل بعدی (DB DUP(?) 256 است، که 256 بایت برای پشته جا در نظر می گیرد.

وقتی که متصل کننده (LINKER) فایل را به یک فایل EXE تبدیل کرد بطور اتوماتیک (SETUP) برقار می شود. هم SP, SS مقادیر خود را دریافت می کند و آماده دریافت اطلاعات و

فصل هفتم

تحویل آن می شوند، بخش دیگر پشته دستور ENDS می باشد که پایان پشته را اعلام می کند.
البته مشاهده می کنید که برای اعلام ENDS از برجسب STACK-SEG نیز استفاده می شود.

۷-۵-۳- بخش دیتا (DATA-SEGMENT)

این قسمت با نام DATA-SEG شروع می شود. هر اسمی که برنامه زیر بخواهد می تواند برای آن انتخاب کند و صدا بزند. همانند بخش پشته با اسم DATA-SEG شروع می شود و با ENDS پایان می پذیرد. برنامه زیر همچنین آزاد است که نام اکسپترا سگمنت را برای آن انتخاب کند.

۷-۵-۴- بخش کد سگمنت :

این بخش شامل برنامه است و امکان تغییر اسن آن وجود ندارد انتخاب شده است. کد سگمنت هم با عبارت سگمنت (SEGMENT) شروع می شود و همچنین به ENDS ختم می شود.
جمله دوم در کد سگمنت جمله ASSUME است که به اسملر می گوید که نامی که برای کد سگمنت انتخاب کرده اید CODE-SEGMENT است و بخش دیتا DATA-SEG است و همچنین اگر از اگسترا سگمنت هم استفاده می کنید باید آورده شود. بخش بعد از نظر فرم افزاری زیربرنامه اصلی است (MAIN PROCEDURE) معمولاً این زیربرنامه اصلی بصورت FAR انتخاب می شود و چند دستور اول این زیربرنامه برقراری شرایط برای برگشت به DOS است مثل DS PUSH زیرا که کد سگمنت DOS را دارد. و پس از آن انتقال پیغام ها به صفحه نمایش است.

۷-۵-۲- برنامه نمونه ورودی و خروجی :

این مسئله، کامپیوتر شما را مثل یک ماشین تحریر در می آورد، برنامه فوق العاده ساده ای است. به شما نشان می دهد که چگونه بخش اول برنامه ۷-۱۱ را با اندکی تغییر می توانید برای هر برنامه ای بکار بگیرید.

در این برنامه بخش هم ارزسازی فقط سه جمله است در یک برنامه بزرگ ممکن است دارای ۱۰۰ جمله هم ارز باشد. جالب است بدانید این برنامه بخش دیتا سگمنت هم ندارد. زیرا قرار نیست دیتائی در حافظه ذخیره کند یا قرار نیست دیتائی از حافظه خوانده شود. فقط قرار است یک کلید ساده خوانده شود و این عمل تکرار شود تا کارکتر \$ تایپ شود. آنوقت کامپیوتر را به محیط DOS برگرداند. بطوریکه بتوانید برنامه اجرائی دیگری را، اجرا نمایید. در زبان اسملی پایه این برنامه را بصورت زیربرنامه ای که خواندن صفحه کلید نام دارد صدا می زند.

فصل هفتم

این برنامه از صفحه کلید اطلاعات دریافت می‌کند و در مونیتور نشان می‌دهد تایپ '\$' تایپ شود.

```
;;
;      PROGRAM EQUADES
;;

ZERO      EQU    0
TWO       EQU    2
DOLAR-SIGN EQU    '$'

;;
;      STACK-SEGMENT
;;

STACK-SEG SEGMENT PARA PUBLIC
DB        256 DUP(?)
STACK-SEG ENDS

;;
;      CODE-SEGMENT
;;

CODE-SEG SEGMENT PARA PUBLIC
ASSUME : CS : CODE-SEG

;;
;      MAIN PROGRAM PROCEDURE
;;

MAIN      PROC   FAR
          PUSH   DS
          MOV    AX,ZERO
          PUSH   AX

AGAIN :   CALL   READ-KEY
          CMP    AL,DOLAR-SIGN
          JNE    AGAIN
          RET
MAIN     ENDP

READ-KEY PROC   NEAR
          MOV    AH,TWO
          INT    21H
          RET
READ-KEY ENDP

CODE-SEG ENDS
END     MAIN
```

تابع شماره ۲ در انتظار زدن دکمه کلید می‌ماند. کلید زده شده را می‌خواند و با قرار دادن آن در AL برمی‌گردد و آنرا نمایش می‌دهد.

فصل هفتم

وقتی که یک کارکتر تایپ شد برنامه اصلی زیربرنامه خواندن کلید را صدای می‌زند و آنرا با \$ چک می‌کند اگر \$ تایپ شده بود به DOS برمی‌گردد و لایه زیربرنامه خواندن کارکتر دیگر از صفحه کلید بر می‌گردد.

۷-۵-۲-۱- نمونه برنامه‌ای که دو عدد را جمع می‌کند :

این برنامه دو عدد را از صفحه کلید دریافت می‌کند آنها را با هم جمع می‌کند و در صفحه نمایش نشان می‌دهد، بمراتب پیچیده تر از برنامه قبلی است، مثال شماره ۷-۱۳ را بینید.
در این برنامه هم از بخش دیتا (DS) و هم از بخش اکسترا (ES) استفاده شده است. زیرا می‌خواهیم از عملیات رشته‌ای استفاده کنیم.

برنامه دو عدد تا ۴۰ رقم را از صفحه کلید می‌پذیرد. طول اولین عدد ادامه پیدا می‌کند تا اپراتور + را تایپ کنید، پس از آن عدد دوم شروع می‌شود و ادامه پیدا می‌کند تا اپراتور = تایپ شود. وقتی که علامت = تایپ شد، برنامه دو عدد را با هم جمع می‌کند (با بکار گرفتن جمع در ASCII) و حاصل جمع در یک بافری بنام ANSWER قرار داده می‌شود. و آنرا روی مونیتور می‌فرستد. فرم وارد کردن دیتا بصورت زیر است :

$$123 + 333 = 456$$

برنامه‌ای که دو عدد تا ۴۰ رقم را دریافت می‌کند و حاصل جمع را روی مونیتور نمایش می‌دهد :

```
PROGRAM EQUATION
;
ZERO      EQU    0
TWO       EQU    2
EIGHT     EQU    8
NINE      EQU    9
FPRTY    EQU    40
CR        EQU    13
LF        EQU    10
EDM       EQU    '$'
DOS-FUNCTION EQU    21H
NO        EQU    'N'
YES      EQU    'Y'
PLUS     EQU    '+'
EQUAL    EQU    '='
NINE-ASCII EQU    '9'
ZERO-ASCII EQU    '0'

STACK SEGMENT
;
STACK-SEG SEGMENT PARA PUBLIC
           DB      256 DUP(?)
STACK-SEG ENDS
```

; DATA SEGMENT

DATA-SEG	SEGMENT PARA PUBLIC
SIGN-ON	DB CR,LF,LF
	DB 'ADD TWO NUMBER(Y/N)?'
	DB EDM
NUMB1	DB 40 DUP(?)
OVER	DB (?)
NUMB2	DB 40 DUP(?)
ANSWER	DB 41 DUP(?)
END-ANS	EQU 'THIS BYTE'
	DB EDM

; CODE SEGMENT

CODE-SEG	SEGMENT PARA PUBLIC
	ASSUME CS:CODE-SEG DS : DATA-SEG ES : DATA-SEG

; MAIN PROGRAM PROCEDURE

MAIN	PROC FAR
	PUSH DS
	MOV AX,ZERO
	PUSH AX
	MOV AX,DATA-SEG
	MOV DS,AX
	MOV ES,AX
TOF :	MOV DX,OFFSET SIGN-ON
	CALL OUT-MESEGE
AGAIN :	CALL READ-KEY
	CMP AL,NO
	JE MAIN-END
	CMP AL,YES
	JNE AGAIN
	CALL OUT-CHAR
	MOV BL,PLUS
	MOV DI,OFFSET NUMB1
	CALL READ-NUMB
	MOV BL,EQUAL
	MOV DI,OFFSET NUMB2
	CALL READ-NUMB2

; READ-KEY PROCEDURE

READ-KEY	PROC NEAR
	PUSH DI
	PUSH BX
	MOV AH,EIGHT
	INT DOS-FUNCTION
	POP BX
	POP DI
READ-KEY	RET ENDP

; DISPLAY CHARACTER PROCEDURE

```
OUT-CHAR PROC NEAR
    PUSH DI
    PUSH BX
    MOV AH,TWO
    MOV DL,AL
    INT DOS-FUNCTION
    POP BX
    POP DI
    RET
OUT-CHAR ENDP.
```

; READ NUMBER PROCEDURE

```
READ-NUMB PROC NEAR
    CALL CLEAR
    MOV BH,ZERO
    READ-NUMB1: CALL READ-KEY
    CALL CHAR-NUMB
    JC READ-NUMB2
    PUSH AX
    CALL OUT-CHAR
    INC BH
    JMP READ-NUMB1
    READ-NUMB2: CMP AL,BL
    JNE READ-NUMB1
    CALL OUT-CHAR
    MOV CL,BH
    MOV CH,ZERO
    CLD
    READ-NUMB3: POP AX
    STOSB
    LOOP READ-NUMB3
    READ-NUMB ENDP
```

; CLEAR PROCEDURE

```
CLEAR PROC NEAR
    PUSH DI
    MOV CX,FORTY
    CLD
    MOV AL,ZERO-ASCII
    REP
    STOSB
    POP DI
    RET
CLEAR ENDP
```

این قسمت بافر را پاک می کند

; CHK-NUMB PROCEDURE

```
CHK-NUMB PROC NEAR
    CMP AL,ZERO-ASCII
    JB CHK-ERR
```

این قسمت شماره زیربرنامه را جک می کند

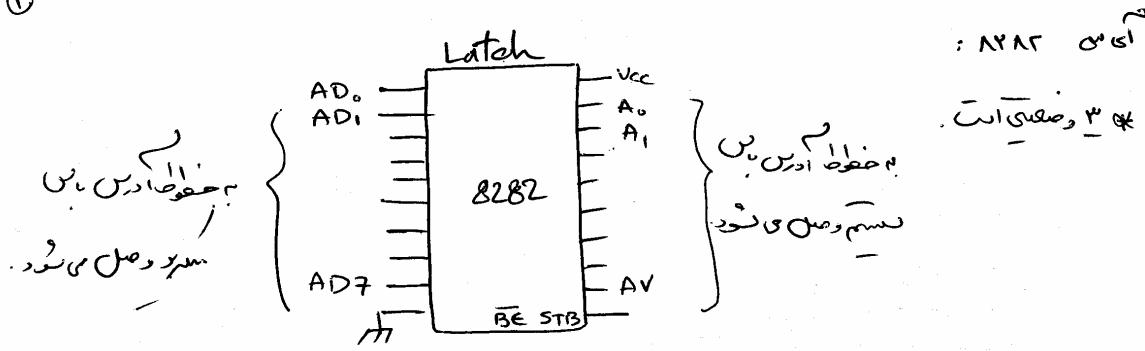
	CMP	AL,NINE-ASCII
	JA	CHK-ERR
	CLC	
	JMP	CHK-END
CHK-ERR :	STC	
CBH-END	RET	
CHK-NUMB	ENDP	
CODE-SEG	ENDS	
	END	MAIN
	MOV	DI,OFFSET ANSWER
	CALL	CLEAR
	MOV	SI,OFFSET NUMB1
	MOV	DI,OFFSET END-ANS
	MOV	BX,OFFSET NUMB1
	MOV	CX,FORTY
	MOV	OVER-ZERO-ASCII
MAIN1 :	CLD	
	LODSB	
	ADD	AL,[BX]
	MOV	AH,ZERO
	AAA	
	ADD	[DI],AH
	STD	
	ADD	AL,ZERO-ASCII
	STOSB	
	INC	BX
	LOOP	MAIN1
	MOV	AL,OVER
	STOSB	
	MOV	DI,OFFSET ANSWER
	MOV	AL,ZERO-ASCII
	MOV	CX,FORTY
	REPE	
	SCASB	
	CALL	OUT-MESSAGE
	JMP	TOP
MAIN-END :	CALL	OUT-CHAR
	RET	
	ENDP	

; OUT MESSAGE PROCEDURE

پیغامی که باید به بیرون ارسال شود

OUT-MESSAGE	PROC	NEAR
	MOV	AH,NINE
	INT	DOS-FUNCTION
	RET	
OUT-MESSAGE	ENDP	

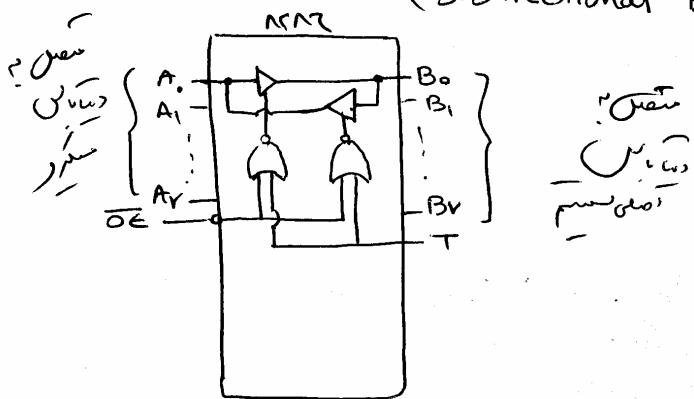
①



۱۱. الگوریتم معرفی شده در ماده ایشان، میتوان OE را مستقیماً به GND وصل کرد.
برای اینجا ۸۲۸۲ استفاده شود (راهن پوشش داده شد).

: (Transceiver) ۸۲۸۲ گیگ
Transmitter + Receiver

(BiDirectional Buffer) کیفیت ایجاد کننده



. جهازه DT/R را ترتیب دهید
. DEN را به OE وصل کنید.

Key Board / Display controller

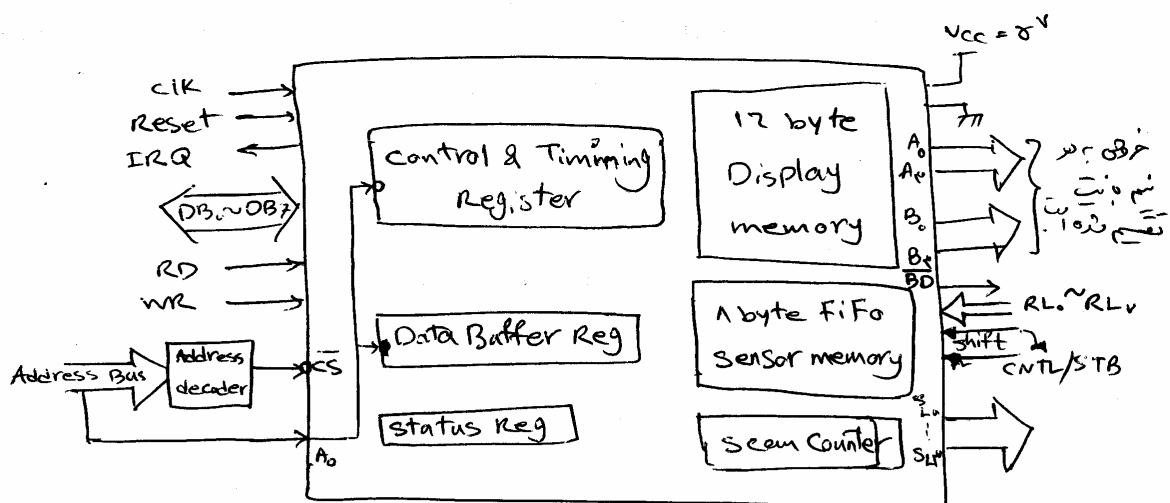
: ۸۲۷۹ گیگ

. ۱۲ بیتی را ایجاد کنید.

. ۱۱ \times ۸ تایی fxf KeyBoard

. ۱۱ LSI یافته

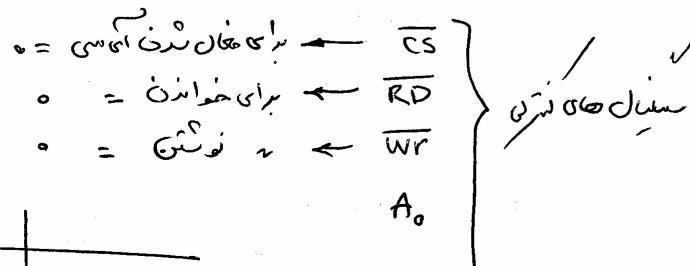
(5)



// (پیش بون نام) Pollling - ۱ } اطلاعات زیع
 interrupt - ۲ } سترک اطلاعات نظری
 اطلاعات داده
 اخراج آنلاین

← بخش سترک با آدرس زیع و کنٹرول سیگنال های آدرس خروجی خالی می شود.
 و سی و ای ایندیکیشن عدایاول بـ $\overline{BD} = 0$ & Display: Blank Display

و سی و ای ایندیکیشن دهن است (۰۰۰۰۰۰۰۰۰۰۰۰۰) : برای آدرس دهن است $SL \sim SH$
 + بایک دو مردم در مردم



\overline{CS}	RD	WR	A ₀	
0	1	0	0	data Bus send to Buffer Register
0	1	0	1	" " " " control
0	0	1	0	" BufferReg " Data Bus
0	0	1	1	control Reg " " " "

④



محتوا نوشته
۱۸۷۹

1- Key Board Display Mode Set

ooo DDD KKK ← امروز صدرازده بود، حذف می شوند.
٠٠٠ ← این کیبر دک ترور آلم ۲ کلید هر کیبورد را در می شوند، حذف می شوند.
٠٠١ ← این کیبر دک ترور آلم ۲ کلید هر کیبورد را در می شوند، حذف می شوند.
٠١٠ ← این کیبر دک ترور آلم ۲ کلید هر کیبورد را در می شوند، حذف می شوند.
٠١١ ← این کیبر دک ترور آلم ۲ کلید هر کیبورد را در می شوند، حذف می شوند.
١٠٠ ← این کیبر دک ترور آلم ۲ کلید هر کیبورد را در می شوند، حذف می شوند.
١٠١ ← این کیبر دک ترور آلم ۲ کلید هر کیبورد را در می شوند، حذف می شوند.
١١٠ ← درود strobe گردانند، همه کلید های کیبورد را در می شوند.
١١١ ← درود strobe گردانند، همه کلید های کیبورد را در می شوند.

DDD

٠٠٠ ← عالی دسترسی درودی ۸ بیتی
٠٠١ ← ۱۲ بیتی
٠١٠ ← ۱۰ بیتی
٠١١ ← ۱۱ بیتی

اعتنی کنید ۱۴

۳۰ نظره ساده بخشی - بخش اصلی