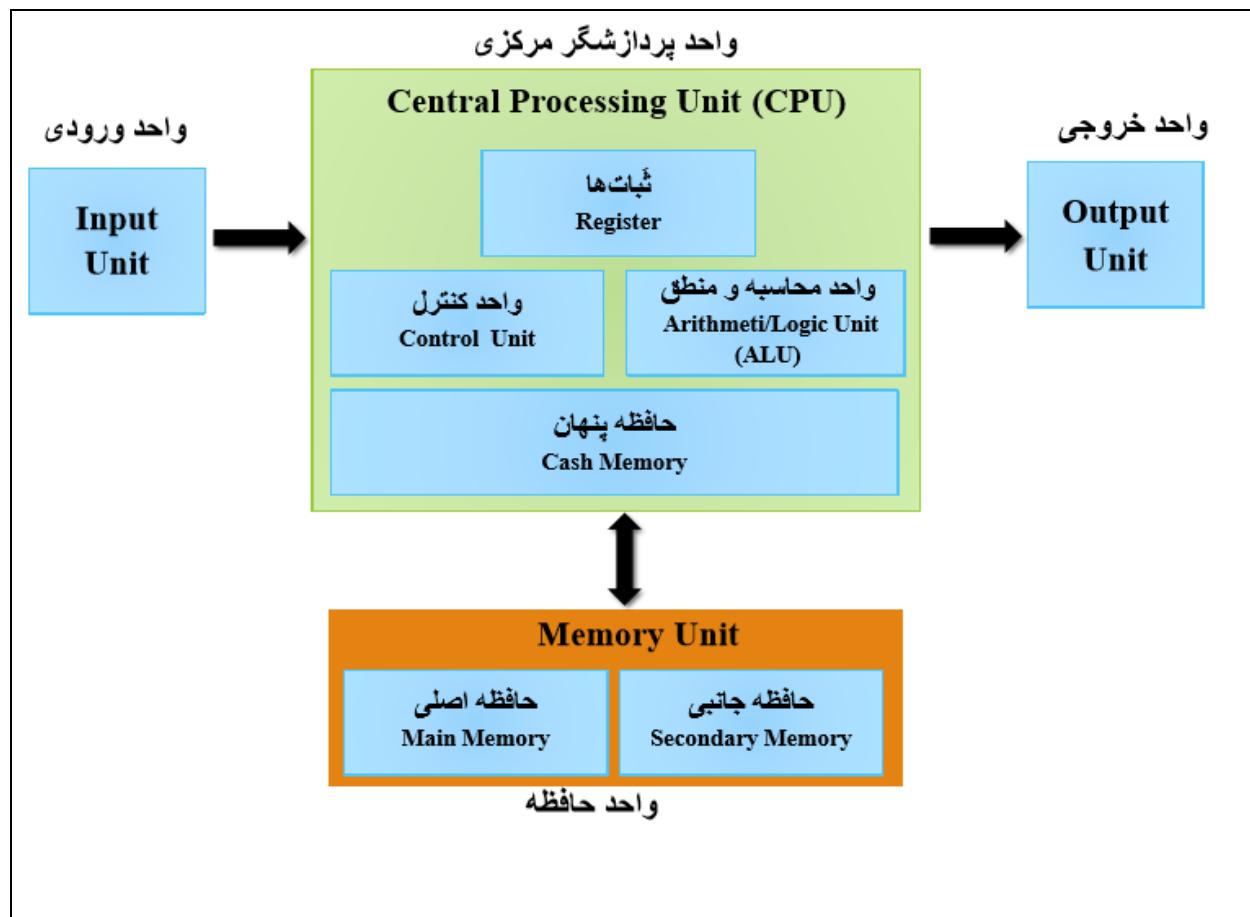


## جلسه چهارم:

ساختار و اجزای کامپیوتر:

- ۱ - واحد پردازشگر مرکزی (cpu)
- ۲ - سیستمهای ورودی و خروجی (i/o)
- ۳ - حافظه (memory)
- ۴ - گذرگاه (bus)

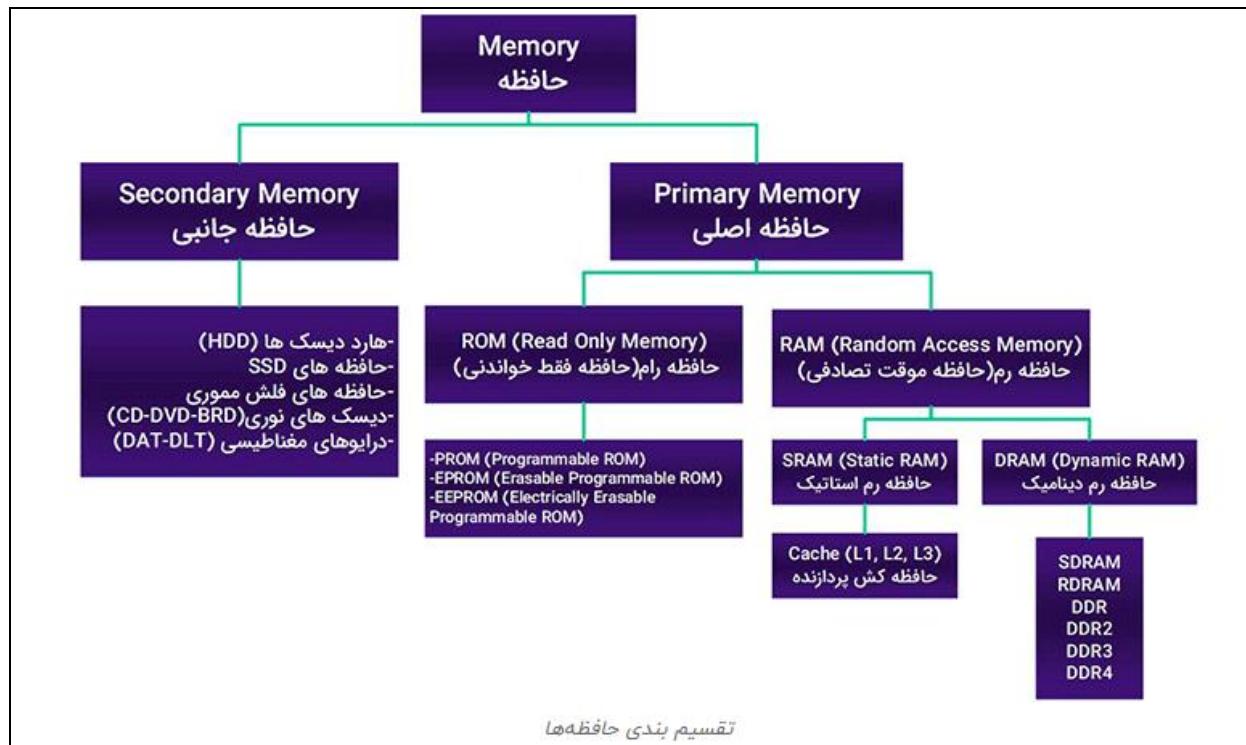


سیستمهای ورودی و خروجی:

ADC: هر نوع ولتاژ انالوگ را به دیجیتال تبدیل می کند

DAC: خروجی دیجیتال از ریزپردازنده را به یک ولتاژ انالوگ تبدیل می کند.

حافظه‌ها:



### ۱- حافظه اصلی (سرعت و قیمت بالا)

• RAM: حافظه موقتی برای نگهداری داده‌ها و اطلاعات است. دسترسی به خانه‌های حافظه به صورت تصادفی است. اطلاعات ذخیره شده در ram قابل پاک شدن هستند. یک حافظه فرار است چون با قطع برق تمام محتویات آن از بین می‌رود.

:RAM انواع

• Dynamic ram: این مدل چندین سلول حافظه دارد که هر کدام از آنها دارای یک ترانزیستور و یک خازن است. ما معمولاً از این مدل به عنوان رم اصلی رایانه نام می‌بریم. ظرفیت آن بسیار بالا است و به همین دلیل است که مورد توجه بسیاری قرار گرفته است و همواره رو به پیشرفت است. برای سیستم‌های بزرگ حافظه. اطلاعات را به صورت دودویی در خازن ترانزیستور ذخیره می‌کند. مزایای حافظه‌ی پویا: ۱- مصرف توان کم ۲- تعداد زیاد سلول حافظه

:Static ram •

• این نوع حافظه که عموماً به عنوان کش در پردازنده‌ها بکار می‌رود از ۶ عدد ترانزیستور برای نگهداری هر بیت استفاده می‌کند. این خازن‌ها به صورتی کنار هم قرار می‌گیرند که تشکیل یک

FLIP-FLOP را می‌دهند و امکان ذخیره اطلاعات را فراهم می‌کنند. این حافظه‌ها سرعت خیلی بالایی دارند ولی هزینه ساخت آن‌ها خیلی بیشتر از حافظه‌های داینامیک است همچنین این حافظه‌ها فضای بیشتری نسبت به حافظه‌های داینامیک اشغال می‌کنند. ظرفیت حافظه و تعداد سلول‌های حافظه آن کم است. قیمت‌ش نسبت به DRAM بالاتر می‌باشد. هزینه ساخت بالا و اشغال فضای بیشتر نسبت به حافظه داینامیک دو دلیل استفاده نکردن حافظه استاتیک به جای حافظه داینامیک است. اگر قرار بود در خرید کامپیوتر از رم‌های استاتیک به جای داینامیک استفاده کنیم، اکثر هزینه را باید برای رم می‌پرداختیم. کار با این حافظه ساده و زمان خواندن و نوشتن در این نوع حافظه کوتاه است.

### توضیحات تکمیلی در مورد عملکرد Flip-Flop static ram درون

فلیپ فلاب چیست؟ فلیپ فلاب یک مدار الکترونیکی است که می‌تواند یکی از دو حالت پایدار را داشته باشد. این حالت‌ها معمولاً با ارقام باینری ۰ و ۱ نمایش داده می‌شوند و برای ذخیره‌سازی یک بیت اطلاعات استفاده می‌شوند. فلیپ فلاب‌ها جزء اصلی تشکیل‌دهنده‌ی مدارهای حافظه هستند.

استفاده از ترانزیستورها و خازن‌ها :

- ❖ ترانزیستورها به عنوان سوئیچ‌های الکترونیکی عمل می‌کنند و جریان الکتریکی را کنترل می‌کنند.
- ❖ خازن‌ها انرژی الکتریکی را به صورت بار الکتریکی ذخیره می‌کنند.
- ❖ ترکیب این دو عنصر، امکان ایجاد یک مدار پایدار را فراهم می‌کند که می‌تواند یکی از دو حالت را حفظ کند.

### تشکیل فلیپ فلاب :

- ✓ ترانزیستورها و خازن‌ها به گونه‌ای کنار هم قرار می‌گیرند که یک مدار فلیپ فلاب را تشکیل می‌دهند.
- ✓ این مدار می‌تواند یکی از دو حالت ۰ یا ۱ را داشته باشد و این حالت تا زمانی که یک سیگنال خارجی به آن اعمال نشود، حفظ می‌شود.

ذخیره اطلاعات :

- هر بیت اطلاعات در حافظه کش، توسط یک فلیپ فلاب ذخیره می‌شود.
- وقتی یک فلیپ فلاب در حالت ۱ قرار دارد، به معنای آن است که بیت مربوطه مقدار ۱ را دارد و بر عکس.

### چرا حافظه کش به عنوان فلیپ فلاپ عمل می‌کند؟

- سرعت بالا: فلیپ فلاپ‌ها بسیار سریع عمل می‌کنند و می‌توانند اطلاعات را به سرعت خوانده و نوشته شوند. این ویژگی برای حافظه کش که باید به سرعت به درخواست‌های پردازنده پاسخ دهد، بسیار مهم است.
- پایداری: فلیپ فلاپ‌ها می‌توانند اطلاعات را به صورت پایدار ذخیره کنند تا زمانی که یک سیگнал خارجی به آن‌ها اعمال شود.
- سادگی: ساختار فلیپ فلاپ نسبتاً ساده است و از تعدادی ترانزیستور و خازن تشکیل شده است.

### مزایای استفاده از حافظه کش به عنوان فلیپ فلاپ

- سرعت دسترسی بالا: به دلیل استفاده از فلیپ فلاپ‌ها، دسترسی به اطلاعات ذخیره شده در حافظه کش بسیار سریع است.
- مصرف انرژی کم: فلیپ فلاپ‌ها نسبت به سایر انواع حافظه‌ها، مصرف انرژی کمتری دارند.
- تراکم بالا: می‌توان تعداد زیادی فلیپ فلاپ را در یک تراشه قرار داد و در نتیجه ظرفیت حافظه کش را افزایش داد.

در نتیجه، می‌توان گفت که استفاده از فلیپ فلاپ‌ها در ساخت حافظه کش، باعث شده است که این حافظه‌ها دارای سرعت بالا، پایداری و تراکم بالایی باشند و به عنوان یک واسطه سریع بین پردازنده و حافظه اصلی عمل کنند.

---

- حافظه SAM: حافظه دسترسی ترتیبی (یعنی باید به ترتیب از اطلاعات بگذریم تا به اطلاعات مورد نظر برسیم مانند نوار کاست یا دیسک مغناطیسی)
- RWM: حافظه قابل خواندن و نوشت: به حافظه‌ای گفته می‌شود که در آن می‌توان هم داده‌ها را خواند و هم داده‌های جدید را در آن نوشت. مانند RAM، Flash memory و حافظه‌های سخت‌افزاری قابل برنامه‌ریزی (FPGA)

- ROM: حافظه فقط خواندنی

انواع ROM:

✓ **PROM:** نوعی حافظه فقط خواندنی است که پس از برنامه‌ریزی اولیه، محتوای آن دیگر قابل تغییر نیست. به عبارت ساده‌تر، شما می‌توانید اطلاعات را یک بار در آن بنویسید و پس از آن فقط قادر به خواندن آن خواهید بود. معمولاً برای ذخیره اطلاعاتی که نیازی به تغییر مداوم ندارند استفاده می‌شود. برخی از این موارد عبارتند از: ۱- بایوس(BIOS) یرنامه‌ای که هنگام روشن شدن کامپیوتراجرامی شود و سخت‌افزارهای سیستم را راهاندازی می‌کند. ۲- جدول‌های جستجو: برای عملیات‌هایی مانند تبدیل کدهای کاراکتر به کدهای باینری یا محاسبات ریاضی ساده.

چرا از حافظه PROM استفاده می‌کنیم؟

- **اطلاعات غیر قابل تغییر:** برای اطمینان از اینکه اطلاعات مهم به صورت تصادفی تغییر نمی‌کنند.
- **سرعت بالا:** حافظه PROM معمولاً سرعت دسترسی بالایی دارد.
- **صرف انرژی کم:** حافظه PROM به دلیل سادگی ساختار، مصرف انرژی کمی دارد.
- **قابلیت اطمینان بالا:** حافظه PROM در برابر نویز و اختلالات الکتریکی مقاوم است.

✓ **EPROM:** قابل برنامه ریزی هست. تحت تاثیر اشعه ماوراء بنفس قابل پاک شدن است و مجدد برنامه ریزی می‌شود. در مواردی که نیاز به ذخیره اطلاعاتی است که به ندرت تغییر می‌کنند، EPROM می‌تواند یک گزینه مناسب باشد. به عنوان مثال، برخی از دستگاه‌های الکترونیکی از EPROM برای ذخیره تنظیمات کارخانه استفاده می‌کنند. سرعت برنامه ریزی پایین، محدودیت تعداد برنامه ریزی و حساسیت به اشعه ماوراء بنفس باعث شده این نوع حافظه به صورت گسترده استفاده نشود.

✓ **EEPROM :** از سیگنال الکتریکی برای پاک کردن استفاده می‌شود و قابل برنامه ریزی مجدد است. بسیاری از دستگاه‌های الکترونیکی از EEPROM برای ذخیره تنظیمات کاربری استفاده می‌کنند. این تنظیمات ممکن است شامل تاریخ و زمان، تنظیمات شبکه، تنظیمات صدا باشد.

۲- جانبی(سرعت و قیمت پایین): hard cd/dvd flash

۳- حافظه نهان یا cache: رابط بین ram و cpu

برای حافظه دو پارامتر حجم و سرعت مهم است: حجم و سرعت در جهت عکس هم هستند

Hard → RAM → cache → register

تفاوت حافظه های جانبی و اصلی:

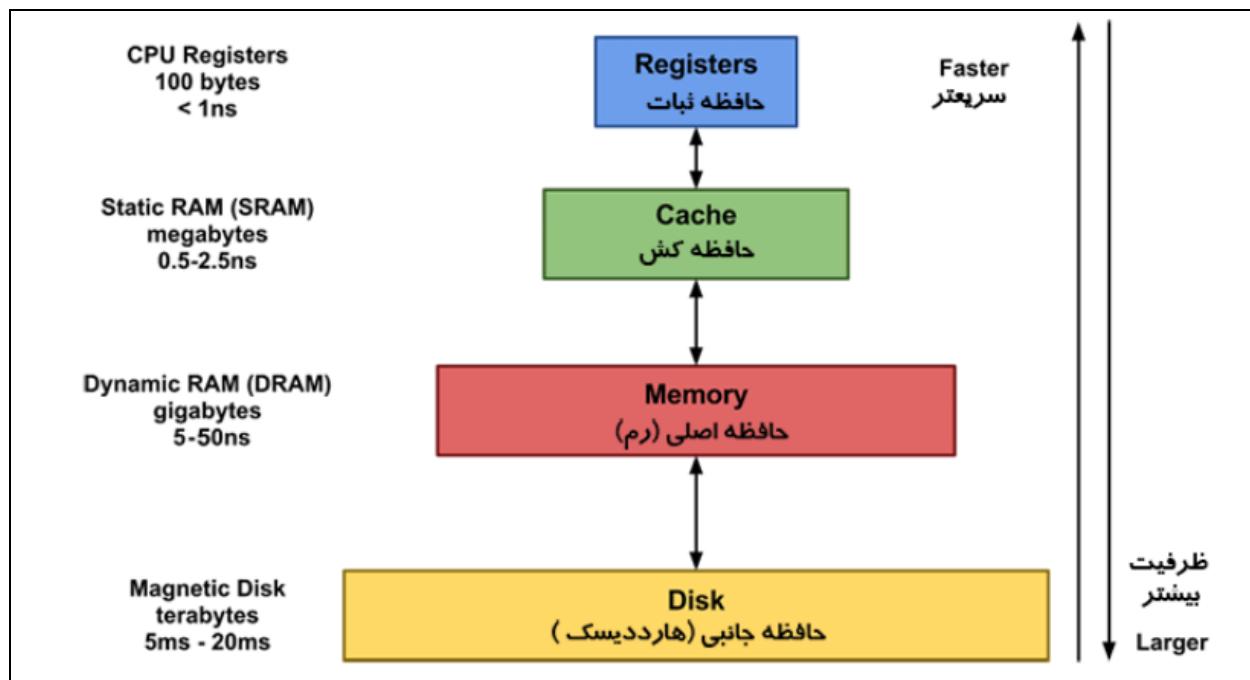
۱- حافظه های اصلی برای پردازش سیستم ضروری هستند ولی حافظه های جانبی در پردازش سیستم نقشی ندارند.

۲- حافظه های اصلی توسط سیستم مدیریت می شوند ولی حافظه های جانبی توسط کاربر مدیریت می شوند.

۳- داده ها در حافظه های اصلی موقت و برای امر پردازش استفاده می شوند ولی داده های حافظه جانبی به صورت بلند مدت ذخیره می شوند و فقط در صورت نیاز برای پردازش خوانده می شوند.

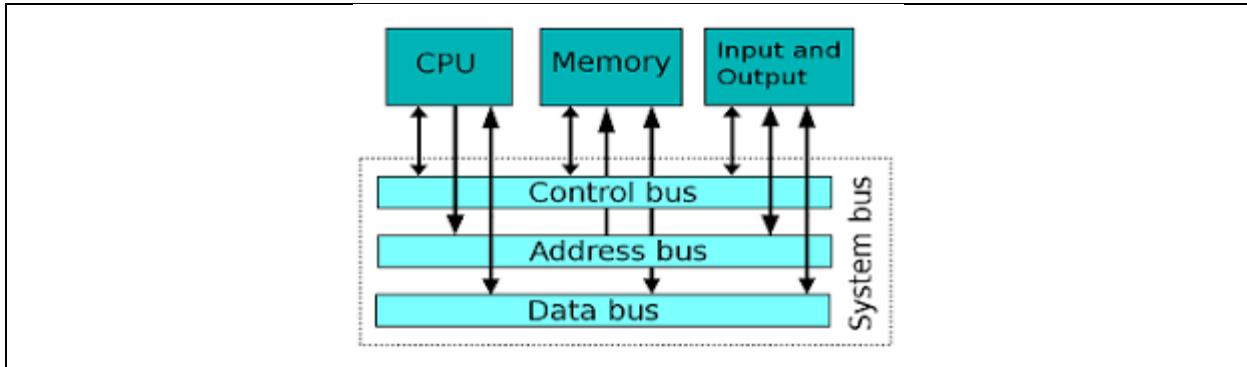
دلیل استفاده از حافظه RAM:

دلیل اصلی تفاوت خیلی زیاد سرعت پردازنده و حافظه جانبی است. به دلیل نوع ساختار این دو حافظه سرعت جابجایی اطلاعات در حافظه جانبی فوق العاده پایین تر از پردازنده است. بنابراین اگر پردازنده بدون واسطه (RAM) از حافظه جانبی استفاده کند، شاهد تاخیر خیلی زیادی خواهیم بود.



### انواع گذرگاه(BUS):

- ۱ - گذرگاه داده: برای انتقال اطلاعات بین CPU و حافظه و دستگاههای ورودی و خروجی است
- ۲ - گذرگاه آدرس: آدرس خانه های حافظه را به بخش‌های مختلف CPU و سیستم‌های ورودی و خروجی منتقل می کند.
- ۳ - گذرگاه کنترل: برای کنترل سیستم‌های حافظه، سیستم‌های ورودی و خروجی و CPU استفاده می‌شود



نکته: باس داده یک باس دو طرفه است.

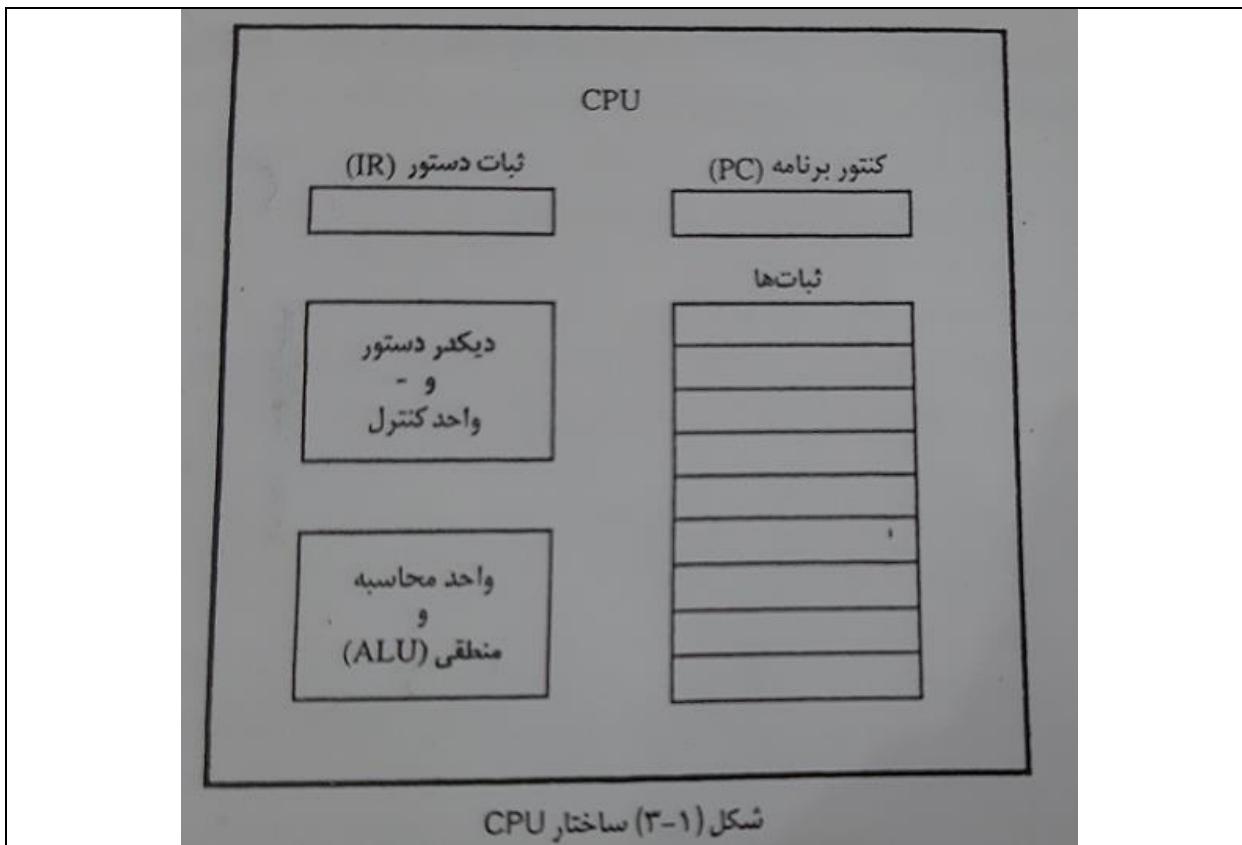
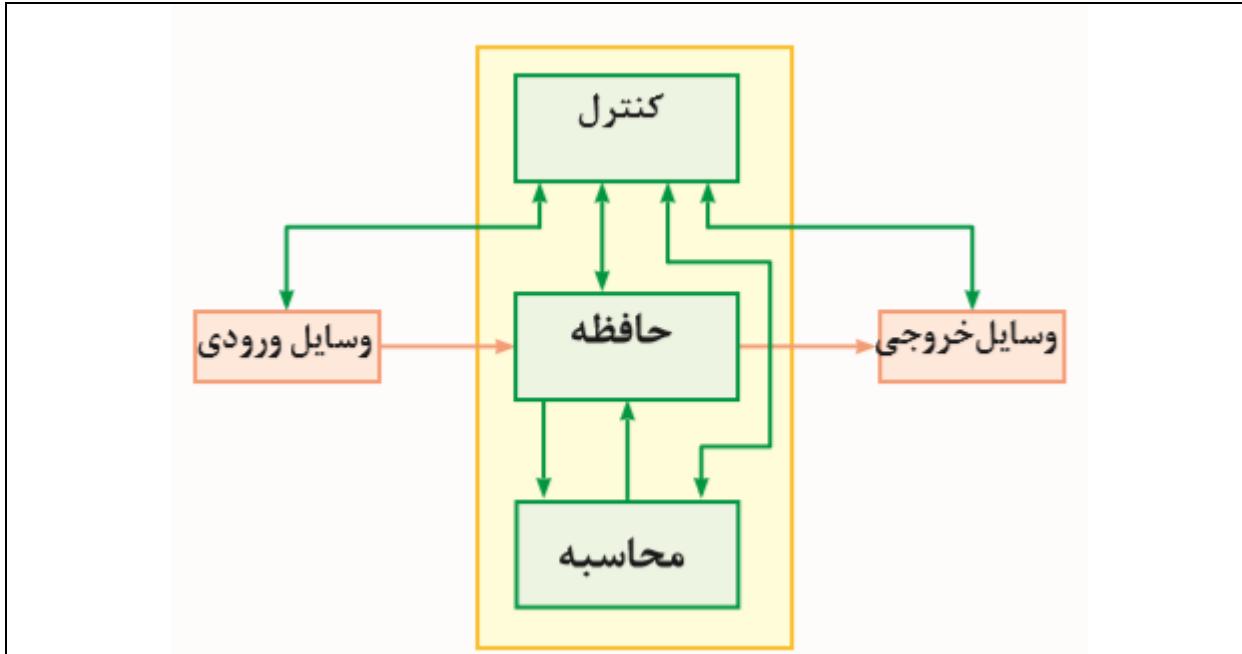
باس آدرس یک طرفه است و همیشه توسط CPU ارسال می‌شود.

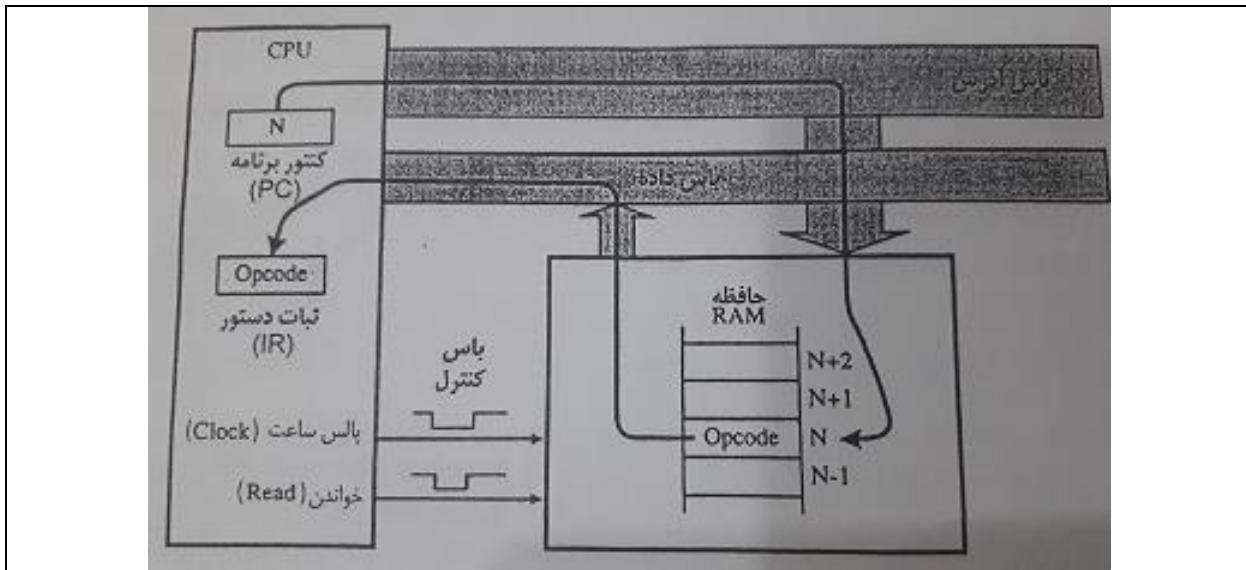
### انواع ساختار گذرگاه

- ۱ - گذرگاه نوع اول: گذرگاهی که یک فرستنده و یک گیرنده دارد.
- ۲ - گذرگاه نوع دوم: چند فرستنده و یک گیرنده دارد
- ۳ - گذرگاه نوع سوم: گذرگاهی که چند فرستنده و چند گیرنده دارد
- ۴ - گذرگاه دو طرفه : با یک پایه کنترل می توان محل ارسال و دریافت را تعیین کرد.

### جلسه پنجم

میکروپروسسور: همان مغز کامپیوتر یا همان CPU است

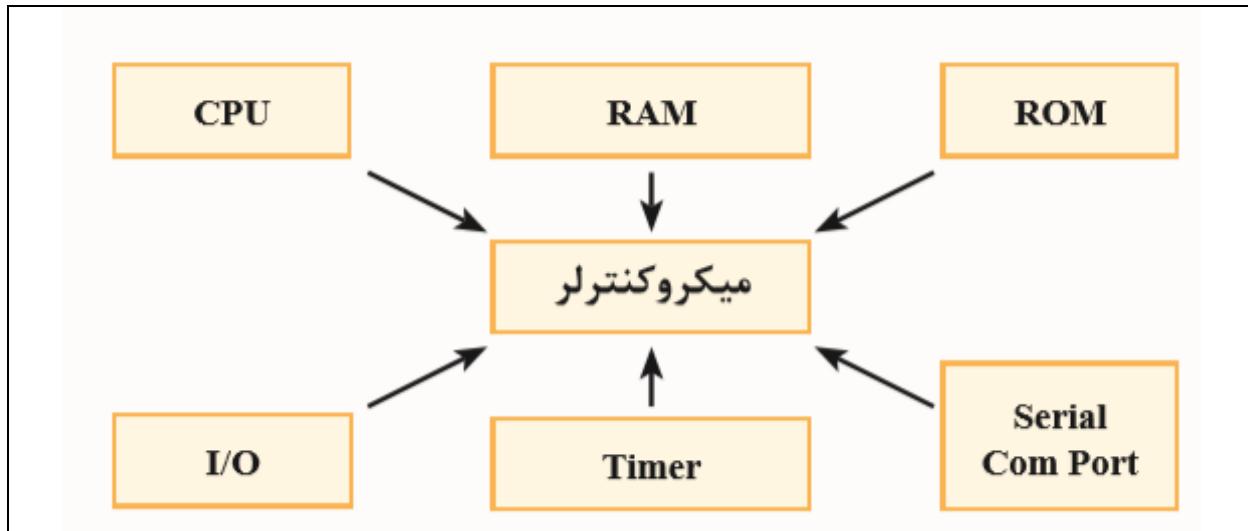




مهمترین عملیات cpu واکشی از حافظه rom و ram است

- ۱- محتوای ثبات pc بر روی باس آدرس قرار میگیرد
- ۲- سیگنال read فعال میشود
- ۳- داده یا کد دستور از حافظه خوانده شده و در باس داده قرار میگیرد
- ۴- کد دستور در ثبات دستور IR داخل cpu قرار میگیرد
- ۵- دیکدر دستور و واحد کنترل آن را تجزیه و تحلیل می کنند و نوع عملیات مشخص میشود
- ۶- واحد ALU پردازش را انجام میدهد. (کمترین زمان در این مرحله است. عملیات محاسباتی (جمع و تفریق به صورت مکمل دو) عملیات منطقی (and or xor....))
- ۷- محتوای ثبات pc یک واحد اضافه میشود.

میکروکنترلر: علاوه بر cpu اجزای دیگری هم دارد



اجزای میکروکنترلر:

- ۱- Cpu
- ۲- حافظه
- ۳- ورودی و خروجی
- ۴- پورتهای سریال: برای ارتباط با لوازم جانبی
- ۵- تایмер:
- ۶- مبدل آنالوگ به دیجیتال
- ۷- مبدل دیجیتال به آنالوگ
- ۸- کنترل کننده: وظیفه کنترل تاخیر برای برنامه در حال اجرا را دارد
- ۹- بلوک عملکرد خاص: این بلوک پورتهای اضافی جهت انجام عملیات خاص را دارد.

تقسیم بندی انواع میکروکنترلر

- ۱- بیت: (۸ بیتی ۱۶ بیتی یا ۳۲ بیتی): یعنی bus data ۸ بیتی، رجیسترها ۸ بیتی، خانه های حافظه ۸ بیتی و CPU عملیات ریاضی را بر روی داده های ۸ بیتی انجام میدهد.
- ۲- حافظه

- External memory
- Embedded memory
- از لحاظ نوع دستور العمل

**RISC و CISC** دو معماری اصلی پردازنده هستند که بر اساس مجموعه دستورالعمل های آنها تعریف می شوند. هر کدام مزايا و معایب خاص خود را دارند و برای کاربردهای مختلفي مناسب هستند.

### (Complex Instruction Set Computer) CISC

- دستورالعمل های پیچیده : پردازنده های CISC از مجموعه دستورالعمل های پیچیده ای برخوردارند که می توانند عملیات های مختلفی را در یک دستورالعمل انجام دهند. این دستورالعمل ها اغلب به زبان های سطح بالا شباهت دارند و برنامه نویسی را ساده تر می کنند.
- تعداد دستورالعمل های زیاد : معماری CISC دارای تعداد زیادی دستورالعمل است که هر کدام برای انجام کار خاصی طراحی شده اند.
- طول متغیر دستورالعمل ها : طول دستورالعمل ها در معماری CISC متفاوت است و این می تواند پیچیدگی طراحی پردازنده را افزایش دهد.
- مثال : پردازنده های  $\times 86$  که در رایانه های شخصی استفاده می شوند، نمونه ای از پردازنده های CISC هستند.

### CISC: مزايا

- برنامه نویسي آسان تر : به دليل شباهت دستورالعمل ها به زبان های سطح بالا، برنامه نویسي برای پردازنده های CISC ساده تر است.
- کدهای برنامه کوتاه تر : به دليل وجود دستورالعمل های پیچیده، کدهای برنامه کوتاه تر می شوند.

### CISC: معایب

- پیچیدگی طراحی پردازنده : طراحی پردازنده های CISC پیچیده تر است و باعث افزایش هزینه تولید می شود.
- سرعت پایین تر : به دليل پیچیدگی دستورالعمل ها، پردازش هر دستورالعمل زمان بیشتری می برد.
- مصرف انرژي بیشتر : پردازنده های CISC به دليل پیچیدگی معماری، انرژي بیشتری مصرف می کنند.

### (Reduced Instruction Set Computer) RISC

- دستورالعمل های ساده : پردازنده های RISC از مجموعه دستورالعمل های ساده و یکسانی برخوردارند. هر دستورالعمل معمولاً یک کار ساده را انجام می دهد.
- تعداد دستورالعمل های کم : معماری RISC دارای تعداد محدودی دستورالعمل است.

- طول ثابت دستورالعمل‌ها: طول همه دستورالعمل‌ها در معماری RISC یکسان است و این باعث ساده‌تر شدن طراحی پردازنده می‌شود.
- مثال: پردازنده‌های ARM که در دستگاه‌های موبایل استفاده می‌شوند، نمونه‌ای از پردازنده‌های RISC هستند.

### RISC: مزایای

- سادگی طراحی پردازنده: طراحی پردازنده‌های RISC ساده‌تر است و باعث کاهش هزینه تولید می‌شود.
- سرعت بالاتر: به دلیل سادگی دستورالعمل‌ها، پردازش هر دستورالعمل زمان کمتری می‌برد.
- مصرف انرژی کمتر: پردازنده‌های RISC به دلیل سادگی معماری، انرژی کمتری مصرف می‌کنند.

### RISC: معایب

- برنامه‌نویسی پیچیده‌تر: به دلیل سادگی دستورالعمل‌ها، برنامه‌نویسی برای پردازنده‌های RISC پیچیده‌تر است و نیاز به نوشتن کدهای طولانی‌تر دارد.
- کدهای برنامه طولانی‌تر: به دلیل سادگی دستورالعمل‌ها، کدهای برنامه طولانی‌تر می‌شوند.

### CISC و RISC: تفاوت

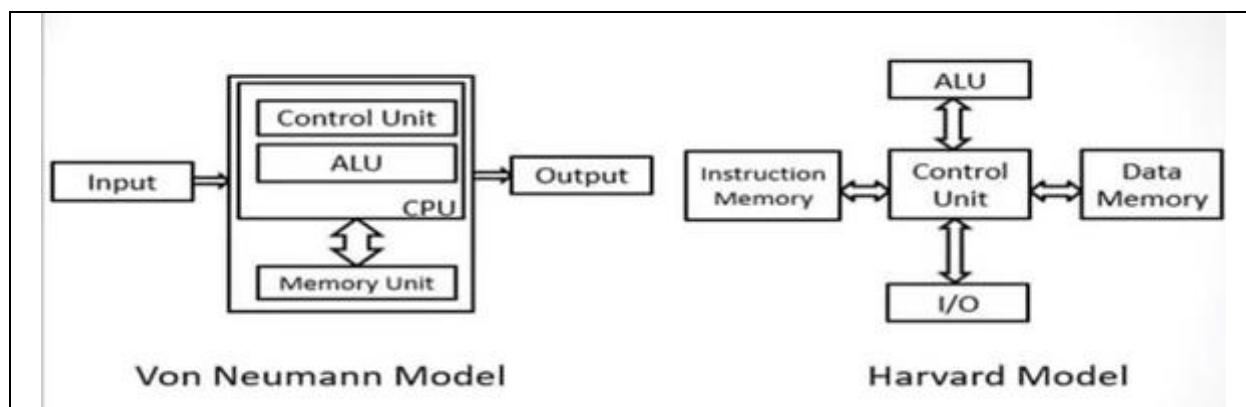
- تعداد و اندازه دستورات در RISC کمتر از CISC است در نتیجه سرعت RISC بیشتره
- در معماری RISC انتقال داده از رجیستر به رجیستر و یا حافظه به حافظه صورت نمی‌گیرد
- تعداد رجیسترها در معماری RISC بیشتره
- برنامه نویسی به زبان اسمنلی در معماری RISC پیچیده‌تر از CISC است
- اکثر دستورات در معماری RISC در یک کلک سیکل اجرا می‌شود
- مصرف توان معماری CISC بیشتر از RISC است

RISC	CISC	ویرگی
کم	زیاد	تعداد دستور العملها
ساده	بیچاره	بیچارگی دستور العملها
ثابت	متغیر	طول دستور العملها
بالاگر	پالینگر	سرعت
کمتر	بیشتر	صرف انرژی
کمتر	بیشتر	هزینه تولید
بیچاره	آسان‌تر	برنامه‌نویسی
طوالانی‌تر	کوتاه‌تر	اندازه کد برنامه

#### ۴- از لحاظ معماری ساخت:

Harvard: حافظه به دو بخش برای داده و برنامه تقسیم می‌شود و امکان دسترسی همزمان به دستورات و داده‌ها وجود دارد مزایا: دسترسی همزمان به حافظه داده و برنامه. معایب: تعداد پایه‌های زیاد

Von neumann : برای داده و دستورات یک حافظه در نظر گرفته شده و عمل استخراج و انتقال داده همزمان انجام نمی‌شود چون باس داده و دستورات مشترک است. سرعت آن پایینتر است.



#### ۵- تعداد پین های ورودی و خروجی:

## جلسه ششم:

در حافظه دو مولفه اهمیت دارد:

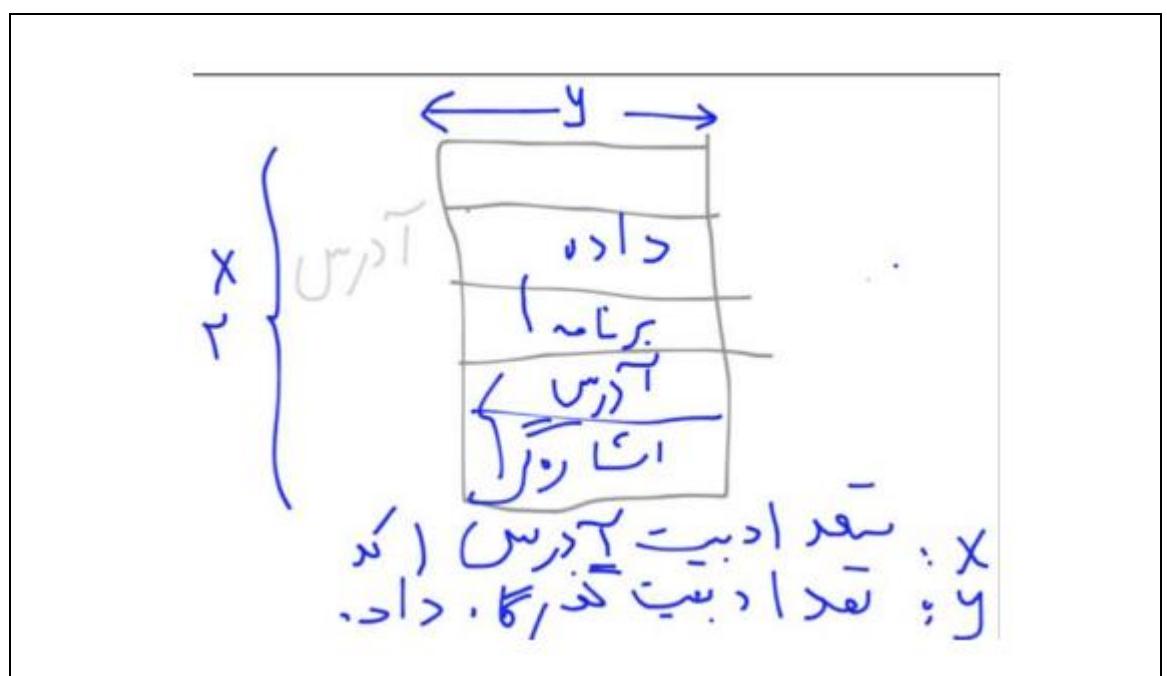
۱- ظرفیت حافظه: حافظه را بر اساس مضربی از بایت بیان میکرد

۲- سازمان حافظه: حافظه را بر اساس بیت بیان میکرد

هر حافظه یک عرض  $l$  که تعداد بیتهای گذرگاه داده است و یک طول که با  $2$  به توان  $X$  که  $X$  تعداد بیت گذرگاه آدرس است و تعداد خانه های حافظه را نشان می دهد.

سازمان حافظه : حاصلضرب  $2$  به توان  $X$  در  $l$  سازمان حافظه بر اساس بیت بیان میشود

ظرفیت حافظه : بیان سازمان حافظه بر اساس بایت ظرفیت حافظه بر اساس مضربی از بایت بیان میشود.



سوال: یک گذرگاه آدرس ۱۶ بیتی و یک گذرگاه داده ۸ بیتی. سازمان و ظرفیت حافظه چیست؟

سازمان حافظه:  $8 * 2^{16}$

ظرفیت حافظه:  $64kB$

سوال: سازمان حافظه و تعداد گذرگاه آدرس را برای یک حافظه ۵۱۲ کیلو بایتی را پیدا کنید در صورتی که تعداد گذرگاه داده ۱۶ بیت است؟ جواب = ۱۸

حل:

<pre> 2^9*2^10*8 2^9*2^10*2^3 2^22 2^18 *2^4 2^18*16 x=18,y=16 </pre>	$512 = 2^9$ $= \text{کیلو}$ $= 8 \text{ بایت}$ $2^9 \times 2^{10} \times 2^3 = 2^{22}$ $\leftarrow 2^{22} = 2^{18} \times 2^4$ <p>چون گفته گذرگاه داده ۱۶ بیتی است. یعنی <math>y = 16</math>.</p> <p>پس باید از <math>2^{22}</math> عدد ۱۶ در بیاریم پس می‌نویسیم</p> $2^{18} \times 16$
---	--

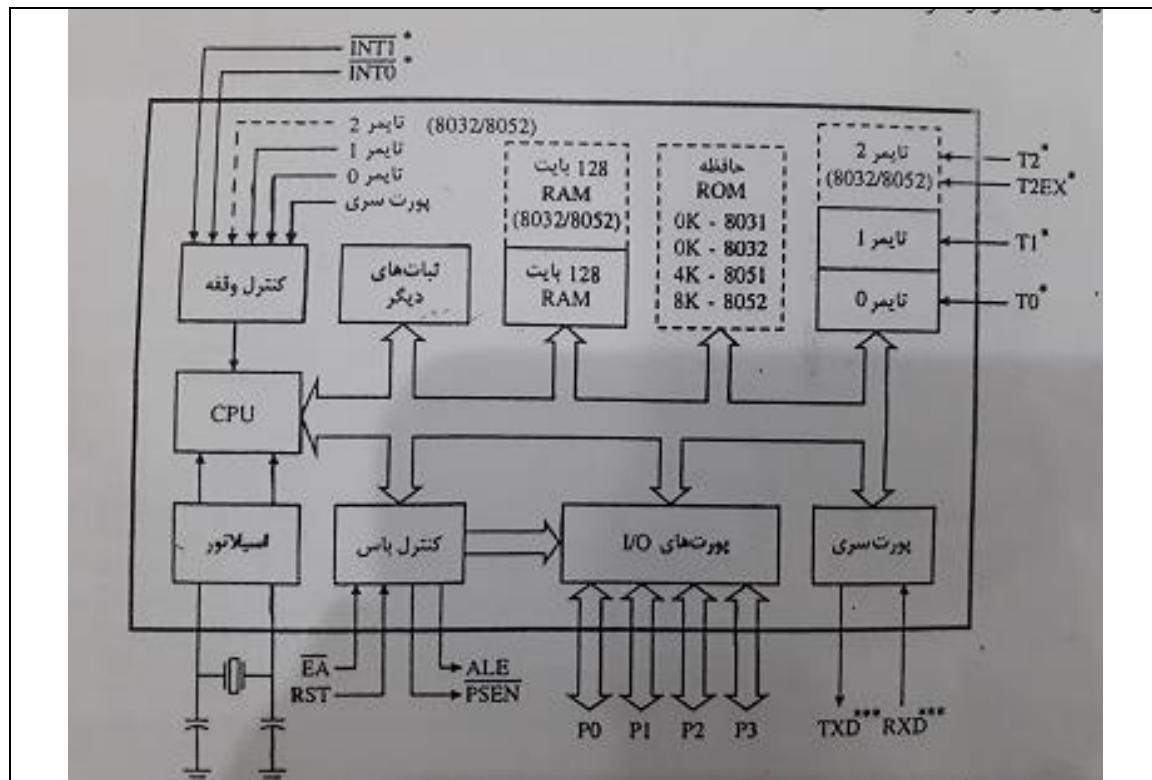
انواع حافظه در میکروکنترلرهای:

- ۱- حافظه برنامه (program memory): برنامه نوشته شده توسط کاربر و برنامه boot در این حافظه ذخیره میشود این حافظه معمولاً از نوع فلش (نوعی از eeprom) است و با سرعت و چندین بار قابل نوشتن و پاک کردن است. حافظه های rom و eprom در میکروکنترلرهای به عنوان حافظه برنامه مورد استفاده قرار میگیرند
- ۲- حافظه داده (data memory): برای ذخیره متغیرها از حافظه sram استفاده میشود رجیسترهای عمومی و  $0/1$  در این حافظه قرار دارند. برای ذخیره مقادیری که نباید با قطع برق از بین رود از حافظه eeprom استفاده میشود. حافظه eeprom در برخی میکروکنترلرهای به صورت داخل تراشه ای و در برخی در خارج از تراشه قرار دارد.

انواع میکروکنترلرهای با توجه به ۵ فاکتور ذکر شده به ۴ دسته ۸۰۵۱، AVR، PIC و ARM تقسیم میشوند.



خانواده ۸۰۵۱ : اولین میکروکنترلر ساخت بشر توسط شرکت intel است. با گذشت زمان میکروکنترلرهای ۸۰۵۱ پیشرفت زیادی نداشته است. نسبت به انواع دیگر امکانات ضعیفتری دارد تقریبا حافظه ای برابر ۱۰۰۰۰ حافظه avr و سرعت آن ۴ برابر کمتر از PIC و ۱۲ برابر کمتر از avr است. برای کارهای ساده با پیچیدگی کم و هزینه کم مناسب است. از زبان C و اسembly پشتیبانی میکند و زبان اصلی آن اسembly است. یک تراشه ۸ بیتی (تعداد بیت گذرگاه داده، رجیسترها، حافظه ها و عملیات محاسباتی به صورت ۸ بیتی است). ۴۰ تا پایه ورودی دارد. ۴ کیلو بایت ram ۱۲۸ rom باشد. حافظه ۶۴ کیلوبایتی خارجی میتواند به آن متصل شود. ۴ پورت موازی ۸ بیتی دارد. یک اسیلاتور یا تولید کننده نوسان داخل تراشه ای دارد که فرکانسی معادل ۱۲ مگا هرتز را میتواند تولید کند. ۲ تایمر ۱۶ بیتی دارد که به عنوان تایمر برای عملکرد داخلی و به عنون شمارنده برای عملکرد خارجی استفاده میشود. شامل ۵ تا منبع وقفه با نامهای وقفه پورت سریال، وقفه تایمر ۱، وقفه خارجی ۰، وقفه تایمر ۰ و وقفه خارجی ۱ است.



تعداد تایمروها	حافظه داده داخلی توابعه	حافظه برنامه (کد) داخلی تراشه	تعداد IC	تعداد
2	بایت 128	4K ROM	8051	1
2	بایت 128	0K	8031	1
2	بایت 128	4K EPROM	8751	1
3	بایت 256	8K ROM	8052	1
3	بایت 256	0K	8032	1
3	بایت 256	8K EPROM	8752	1

جلسه هفتم:

خانواده AVR

جز محبوبترین میکروکنترلرها هستند. ساخت شرکت Atmel است. در زمینه برق و الکترونیک پر کاربرد است. به خاطر فرکانس پایین در پروژه‌های حساس به فرکانس بالا استفاده نمی‌شود. تمامی

میکروکنترلهای AVR به زبان C و بیسیک هستند و برای زبان C از نرم افزار Code Vision AVR و برای زبان بیسیک از AVR BASCOM به عنوان کامپایلر استفاده می‌شود. براساس معماری Harvard است. اساس آن کاهش مجموعه دستورالعملها است یعنی RISC است. نوع حافظه در AVR وجود دارد: flash: برنامه اصلی درون آن ذخیره می‌شود. SRAM: برای اجرای برنامه به این حافظه نیاز است. یک حافظه داخلی و دائمی است که داده‌ها در آن ذخیره می‌شوند. دارای 32 تا رجیستر 8 بیتی است که مستقیماً به ALU وصل است. تعداد دستورالعملها آن کم است و معمولاً در یک کلاک انجام می‌شود و AVR به چهار دسته تقسیم می‌شود.

دارای CPU قدرتمند و حافظه 4 کیلو بایتی Flash و EEPROM دارای ATtiny 128 بایتی دارد. به خاطر سایز کوچکش معمولاً در مدارهایی که حجم در آن مهم است استفاده می‌شود.

ATtiny10	ATtiny12	ATtiny15	ATtiny25	ATtiny28	ATtiny85
ATtiny11	ATtiny13	ATtiny22	ATtiny26	ATtiny45	ATtiny2313

میکروکنترلهای سری ATtiny

سری AT90S (سری کلاسیک): امکانات و توانایی بیشتری نسبت به سری ATTiny دارد که می‌توان به افزایش حافظه برنامه و حافظه داده اشاره کرد.

AT90S4434	AT90S8535	AT90S4433	AT90S2323	AT90S1200
AT90S8534	AT90S4414	AT90S8515	AT90S2343	AT90S2313

میکروکنترلهای سری AT90S

سری Atmega: در زمینه‌های جزئی مانند کانالهای ADC، تایمراها و تعداد حافظه با هم متفاوت هستند. عدد نوشته شده در کنار مدل مقدار حافظه Flash موجود در میکروکنترلر را بر اساس کیلو بایت تعیین می‌کند. هر یک از این میکروکنترلهای ذارای دو نوع A و L هستند. که از لحاظ سطح ولتاژ و فرکانس با هم تفاوت دارند.

: دارای ۱۶ کیلو بایت حافظه فلاش - با ولتاژ ۴.۵ تا ۵.۵ کار میکند. با فرکانس ۰ تا ۱۶ مگا هرتز کار میکند.

: دارای ۱۶ کیلو بایت حافظه فلاش - با ولتاژ ۲.۷ تا ۵.۵ کار میکند. با فرکانس ۰ تا ۸ مگا هرتز کار میکند.

ATmega48	ATmega603	ATmega165	ATmega324
ATmega8	ATmega649	ATmega168	ATmega325
ATmega88	ATmega6490	ATmega169	ATmega329
ATmega64	ATmega16	ATmega128	ATmega3290
ATmega640	ATmega161	ATmega1280	ATmega3250
ATmega644	ATmega162	ATmega1281	ATmega256
ATmega6450	ATmega163	ATmega32	ATmega8535
ATmega670	ATmega164	ATmega323	ATmega8515

### میکروکنترلرهای سری ATmega

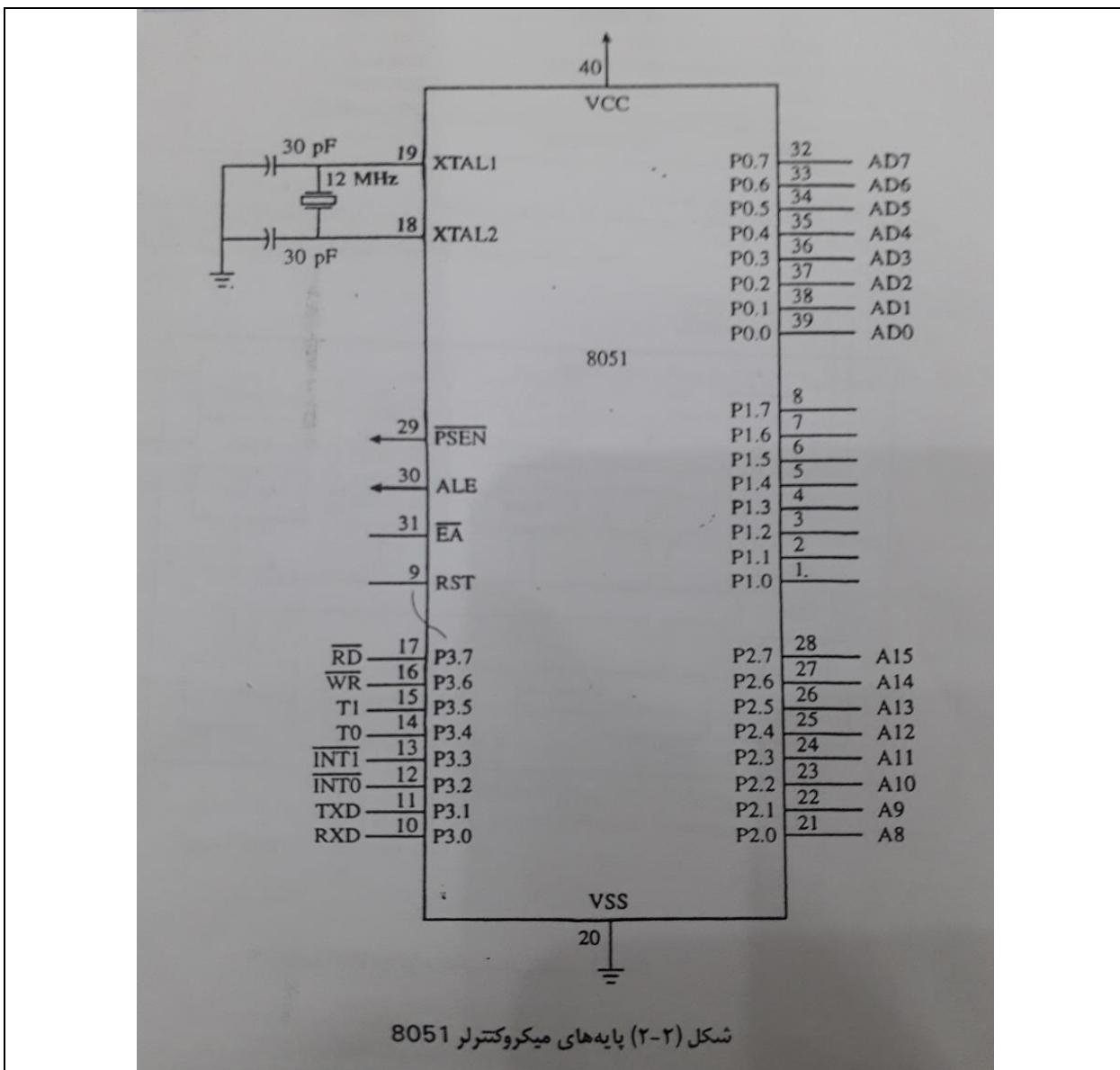
سری ATXmega (سری جدید): جدیدترین و قدرتمند ترین avr است. امکانات جانبی آن هم افزایش پیدا کرده. کارایی بهتر دارد . توان مصرفی کمی دارد. کم مصرف بودن توان باعث میشود برای کاربردهای همراه (دارای باتری) مناسب است. معماری cpu آن مشابه معماری atmega cpu نوع است. با این تفاوت که ATXmega پنهانی رجیسترها و باسهای داده و دستورالعملها ۱۶ بیتی هستند. و در دو حالت ۸ و ۱۶ بیتی پردازش میشوند. میزان حافظه بیشتر و فرکانس بالاتری دارند.

: شرکت mirochip خانواده وسیعی از میکرومنترلهای pic را در دسته های ۱۶ ۸ و ۳۲ بیتی برای کاربردهای مختلف تولید کرده است. این خانواده روز به روز در حال توسعه یافتن است و همین امر موجب شده pic ها در صنعت محبوب باشند. معماری آن harvard است. از زبان C اسambilی و بیسیک پشتیبانی میکند. ویژگی های آن هزینه کم، مصونیت ئر برابر نویز، قابلیت برنامه ریزی سریال و دسترسی گسترده است.

: این خانواده در دسته های ۱۶ ۸ و ۳۲ بیتی موجود هستند. میکروکنترلهای ۸ و ۱۶ بیتی جوابگوی سیستمهای توسعه یافته نیستند و به کارگیری میکروکنترلهای ۳۲ رو به افزایش است. به دلیل مصرف توان اندک، سرعت پردازش زیاد و قیمت بسیار اندک نخستین انتخاب موجود است و بسیاری از تولیدکنندگان از

آن استفاده میکنند. معماری ۳۲ بیتی آن risc است. این خانواده زبانهای برنامه نویسی سطح بالا و پایین را پشتیبانی میکند.

جلسه هشتم:



۳۲ پایه از ۴۰ پایه برای عملیات ورودی و خروجی است. ۲۴ پایه برای دو منظور ورودی- خروجی و هم چنین خطهای کنترل و یا قسمتی از بس آدرس و داده استفاده میشود. هر خط پورت را بطور مستقل برای اتصال به کلید ترانزیستور، موتور و بلندگو میتوان استفاده کرد.

پورت  $\text{port}0$ : در پایه های ۳۹ تا ۳۲ است و برای دو منظور استفاده میشود

در طرحهای کوچک که به حافظه خارجی نیازی نیست برای عملیات ورودی و خروجی استفاده میشود

در طرحهای بزرگتر با حافظه خارجی ، این پورت به عنوان بس آدرس و یا بس داده هم استفاده میشود.

پورت ۱: فقط برای اتصال به دستگاههای ورودی و خروجی استفاده میشود. در میکروکنترلر ۸۰۳۲ و ۸۰۵۲ پایه های ۰ p1.۰ و ۱ p1.۱ برای عملیات ورودی و خروجی و یا به عنوان پالس خارجی تایмер سوم به کار میروند.

پورت ۲: در پایه های ۲۱ تا ۲۸ است و برای دو منظور ورودی و خروجی و یا به عنوان بایت بزرگتر بس آدرس برای طرحهایی که دارای حافظه برنامه و داده خارجی بیش از ۲۵۶ بایت باشند.

پورت ۳: پایه های ۱۰ تا ۱۷ است و برای دو منظور ورودی و خروجی و عملیات خاص در ۸۰۵۱ استفاده میشود.

بیت	نام	ادرس بیت	عملکرد دیگر
P3.0	RXD	B0H*	دریافت داده برای پورت سری
P3.1	TXD	B1H	ارسال داده برای پورت سری
P3.2	INT0	B2H	وقفه ۰ خارجی
P3.3	INT1	B3H	وقفه ۱ خارجی
P3.4	T0	B4H	ورودی خارجی تایمر یا شمارنده ۰
P3.5	T1	B5H	ورودی خارجی تایمر یا شمارنده ۱
P3.6	WR	B6H	سیگنال نوشتن حافظه داده خارجی
P3.7	RD	B7H	سیگنال خواندن حافظه داده خارجی
P1.0	T2	90H	ورودی خارجی تایمر یا شمارنده ۲
P1.1	T2EX	91H	گرفتن و بار کردن مجدد تایمر یا شمارنده ۲

پایه شماره ۲۹ عملکرد سیگنال خارجی 'psen' (فعالساز حافظه برنامه): اگر این پایه ۱ باشد یعنی غیر فعال است و از حافظه برنامه داخلی rom استفاده میشود اگر این پایه صفر باشد یعنی فعال است و به پایه خروجی (فعال صفر) OE وصل میشود و از حافظه برنامه خارجی eprom یک بایت برنامه خوانده میشود و درون بس داده قرار میگیرد و سپس در ثبات IR و ....

پایه شماره ۳۰ ALE: برای جدا کردن بس آدرس و داده استفاده میشود.

پایه شماره ۳۱ سیگنال ورودی دسترسی خارجی(EA'): ورودی EA معمولا به  $+5V$  یا  $0$  زمین متصل است. اگر EA به  $5$  ولت وصل باشد برنامه را از rom داخل اجرا میکند و اگر به  $0$  وصل باشد rom داخلی غیر فعال است و برنامه فقط از حافظه eprom خارجی اجرا میشود. (در این صورت پلیه ۲۹ یا psen برابر  $0$  و فعال است) برای میکروکنترلهای EA ۸۰۳۱ و ۸۰۳۲ قرار داده شود چون حافظه برنامه يا rom داخلی ندارند.

پایه ۹ سیگنال ورودی (RST)reset:

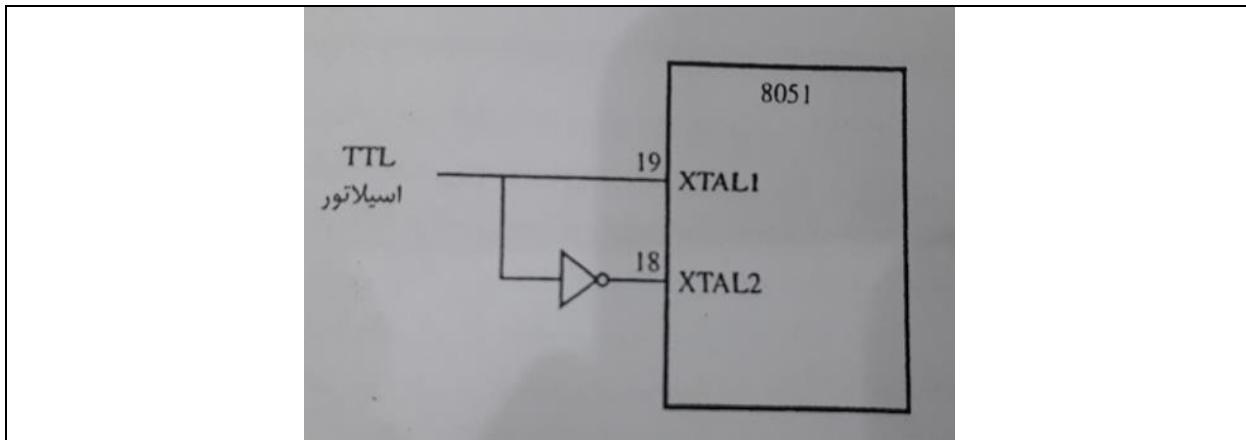
موقعی که این ورودی حداقل برای دو سیکل ماشین فعال و برابر  $1$  باشد تمام ثباتها با مقدارهای مناسب بار میشوند. در حالت عادی این پایه  $0$  یا غیر فعال هست.

پایه ۱۸ و ۱۹ (ورودی های اسیلاتور تراشه): این پایه ها به اسیلاتور وصل میشوند و فرکانسی معادل  $12$  مگا هرتز را تولید می کنند.

هر چه فرکانس بیشتر باشد سرعت عملکرد دستورات بیشتر است. فرکانس بالا را برای CPU های موبایل تبلت و لپ تاپ در نظر میگیرند. چون هر چه فرکانس بالاتر باشد سرعت عملکرد دستورات بیشتر است و زمان انجام کار کوتاهتر است

اتصالات منبع تغذیه:

میکرو کنترلر ۸۰۵۱ با منبع تغذیه  $5$  ولت کار میکند که VCC ( $5$  ولت) به پایه  $۴۰$  (زمین) به پایه  $۲۰$  متصل میشود.



$$T = 1/F \quad F = 1/T$$

فرکانسی که به میکروکنترلر میرسد طبق سخت افزاری که پیاده سازی شده است  $1/12$  فرکانس اسیلاتور است.

$$F = \frac{1}{T} \quad T = \frac{1}{F}$$

$$f_{osc} = 12 \text{ MHz}$$

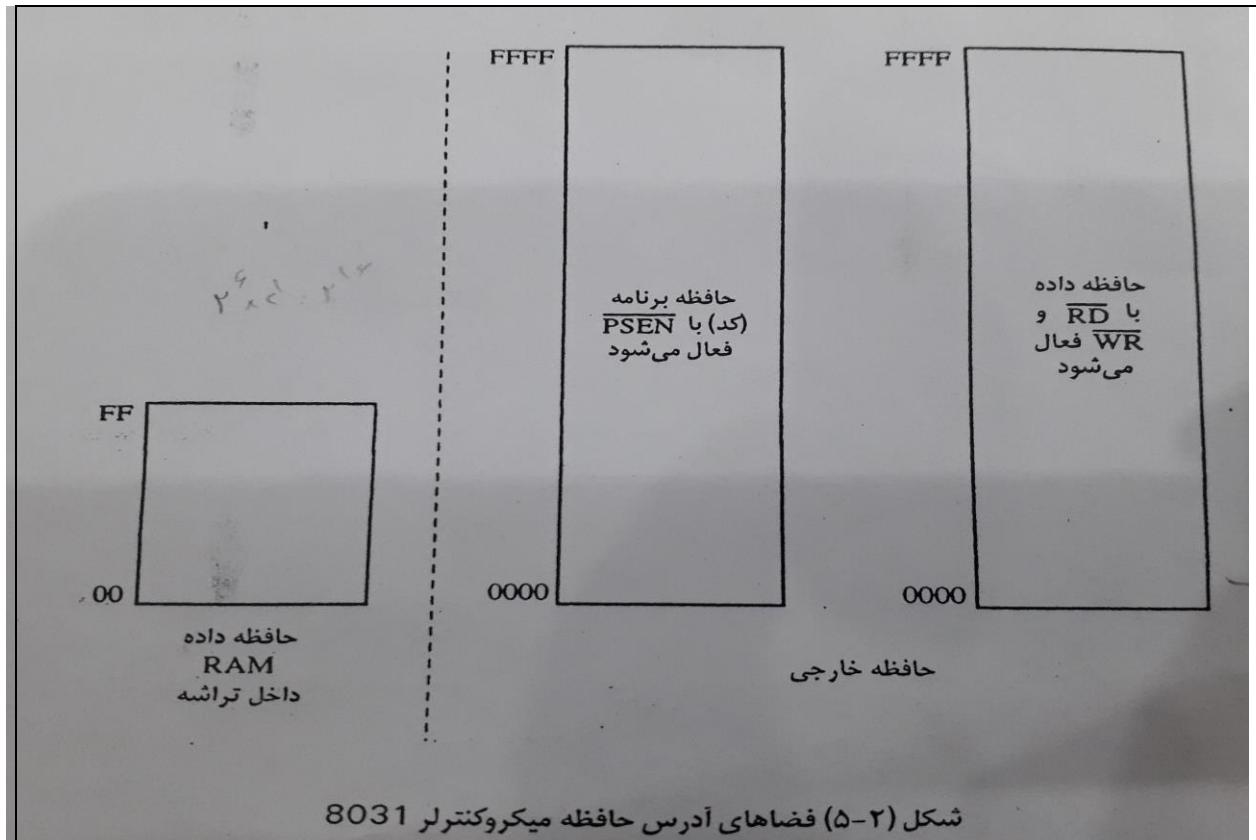
$$f_{micro} = \frac{1}{12} f_{osc} = 1 \text{ MHz}$$

$$T = \frac{1}{F} = \frac{1}{1 \text{ MHz}} = \frac{1}{10^6 \text{ Hz}} = 10^{-6} \text{ s} = 1 \text{ microsecond}$$

قبل از خواندن هر گونه اطلاعاتی از پورتها با ید با دستورات میکروکنترلر بیتها این پورتها یک شوند.

MOV A, #FFH

MOV P0, A



شکل (۵-۲) فضاهای آدرس حافظه میکروکنترلر 8031

حافظه RAM شامل خانه حافظه عمومی داده، حافظه بیتی با بیت آدرس پذیر، بانکهای ثبات و ثباتهای کاربرد خاص (SFR).

بر عکس میکرو پروسسورها، در میکروکنترلرها حافظه پشته، در حافظه RAM داخل تراشه قرار دارد.

آدرس بايت	آدرس بيت	آدرس بايت	آدرس بيت
7F		FF	
RAM	حافظه عمومی داده RAM	F0	F7 F6 F5 F4 F3 F2 F1 F0 مثالی از آدرس بیت F1H ↗
30	7DH مثالی از آدرس بیت	E0	E7 E6 E5 E4 E3 E2 E1 E0 ACC
2F	7F 7E 7D 7C 7B 7A 79 78	D0	D7 D6 D5 D4 D3 D2 - D0 PSW
2E	77 76 75 74 73 72 71 70	B8	- - - BC BB BA B9 B8 IP
2D	6F 6E 6D 6C 6B 6A 69 68	B0	B7 B6 B5 B4 B3 B2 B1 B0 P3 پورت 3
2C	67 66 65 64 63 62 61 60	A8	AF - - AC AB AA A9 A8 IE
2B	5F 5E 5D 5C 5B 5A 59 58	A0	A7 A6 A5 A4 A3 A2 A1 A0 P2 پورت 2
2A	57 56 55 54 53 52 51 50	99	not bit addressable
29	4F 4E 4D 4C 4B 4A 49 48	98	9F 9E 9D 9C 9B 9A 99 98 SBUF } پورت سری SCON }
28	47 46 45 44 43 42 41 40	90	97 96 95 94 93 92 91 90 P1 پورت 1
27	3F 3E 3D 3C 3B 3A 39 38	8D	not bit addressable
26	37 36 35 34 33 32 31 30	8C	not bit addressable
25	2F 2E 2D 2C 2B 2A 29 28	8B	not bit addressable
24	27 26 25 24 23 22 21 20	8A	not bit addressable
23	1F 1E 1D 1C 1B 1A 19 18	89	not bit addressable
22	17 16 15 14 13 12 11 10	88	8F 8E 8D 8C 8B 8A 89 88 TL1 } تایمراه TL0 TMOD TCON PCON
21	0F 0E 0D 0C 0B 0A 09 08	87	not bit addressable
20	07 06 05 04 03 02 01 00	83	not bit addressable
IF	بانک 3	82	not bit addressable
18		81	not bit addressable
17	بانک 2	80	87 86 85 84 83 82 81 80 DPH DPL SP PO پورت 0
10			
0F	بانک 1		
08			
07	پیشفرض برای ثبت های R0 تا R7		
00			
حافظه RAM (الف)		ثبت های کاربرد خاص (SFR) (ب)	

شکل (۲-۶) خلاصه ای از محل های حافظه داده RAM داخل تراشه میکروکنترلر 8051

بانکهای ثبات از آدرس 00 تا 1FH را به خود اختصاص میدهند ۴ بانک ثبات با نام بانک ثبات ۰ تا ۳ داریم هر کدام دارای ۸ ثبات با نام های R0 R1 .....R7 هستند و در حال اولیه بانک ۰ فعال می باشد

حافظه RAM با بیت آدرس پذیر (آدرس 20H تا 2FH)

حافظه عمومی داده RAM(آدرس 30H تا 7FH)

ثباتهای کاربرد خاص (FFH تا 80H)

حافظه عمومی داده RAM:

تعداد ۸۰ بایت از آدرس 30H تا 7FH برای حافظه داده استفاده میشود ولی ۳۲ بایت پایین RAM از آدرس 00 تا 1FH هم میتواند برای این منظور استفاده شود.

به هر محلی از حافظه RAM می توان به طور مستقیم یا غیر مستقیم دسترسی داشت

روش مستقیم:

MOV A,5FH

روش غیر مستقیم

MOV R0 ,#5FH

MOV A, @R0

حافظه RAM با بیت آدرس پذیر

میکرو کنترلر ۸۰۵۱ دارای ۲۱۰ بیت آدرس پذیر هست ۱۲۸ بیت آن از آدرس بایت 20H تا 2FH است و بقیه در ثباتهای کاربرد خاص وجود دارد.

برخی از دستورات بر روی بیت و برخی برروی بایت انجام میشوند.

مثال: برای یک کردن بیت با آدرس 67H از دستور زیر استفاده میشود(بیت 67H بیت پرارزش از بایت 2CH است)

روش بیتی:

SETB 67H

از روش بایتی

MOV A,2CH

ORL A ,#10000000B

MOV 2CH,A

بانکهای ثبات:

۳۲ بایت ابتدای RAM که شامل ۴ بانک هر بانک ۸ ثبات دارد و در حالت RESET بانک ۰ فعال است.

MOV A , R5

MOV A ,05H

در شروع کار و در حالت پیش فرض بانک ثبات ۰ فعال است و دو دستور بالا معادل هم هستند.

برای فعال کردن دیگر بانکها باید بیتهاي انتخاب بانک ثبات تغییر داده شوند.

جلسه دهم:

ثباتهای کاربرد خاص:

از آدرس H 80 تا FFH را شامل میشوند و شامل ۲۱ بایت است که بیتهاي آن هم آدرس پذیر است.

INC A

نام دیگر آکومولاتور ACC هم می باشد

SETB 0EOH

آدرس EOH هم آدرس بیت ۰ از آکومولاتور A هم آدرس بایت آکومولاتور A است که با نوع دستور مورد استفاده بیتی و بایتی این عدد مشخص میشود.

پورت P1 در آدرس 90H قرار دارد و آدرس بیتهاي ان هم از 90H تا 97H است ئو دستور زیر معادل هم هستند

MOV P1,A

MOV 90H,A

این دستورات بايتي هستند

SETB P1.0

SETB 90H

دو دستور بالا معادل هم هستند و بيت کم ارزش پورت P1 را ۱ می کنند و دستورات بیتي هستند.

ثبات وضعیت (PSW):

جدول (۳-۲) خلاصه‌ای از بیتهاي ثبات وضعیت (PSW)

بیت	نمی‌یول	آدرس	توضیح بیت
PSW.7	CY یا C	D7H	بیت پرچم نقلی ۱
PSW.6	AC	D6H	بیت پرچم نقلی کمکی ۲
PSW.5	F0	D5H	پرچم ۰
PSW.4	RS1	D4H	انتخاب ۱ بانک ثبات
PSW.3	RS0	D3H	انتخاب ۰ بانک ثبات
			بانک ۰ = ۰۰: از آدرس 00H تا 07H
			بانک ۱ = ۰۱: از آدرس 08H تا 0FH
			بانک ۲ = ۱۰: از آدرس 10H تا 17H
			بانک ۳ = ۱۱: از آدرس 18H تا 1FH
PSW.2	OV	D2H	بیت پرچم سرریز
PSW.1	--	D1H	رزرو شده
PSW.0	P	D0H	بیت پرچم توازن زوج ۴

بیت پرچم نقلی:

این بیت برای دو منظور استفاده می‌شود

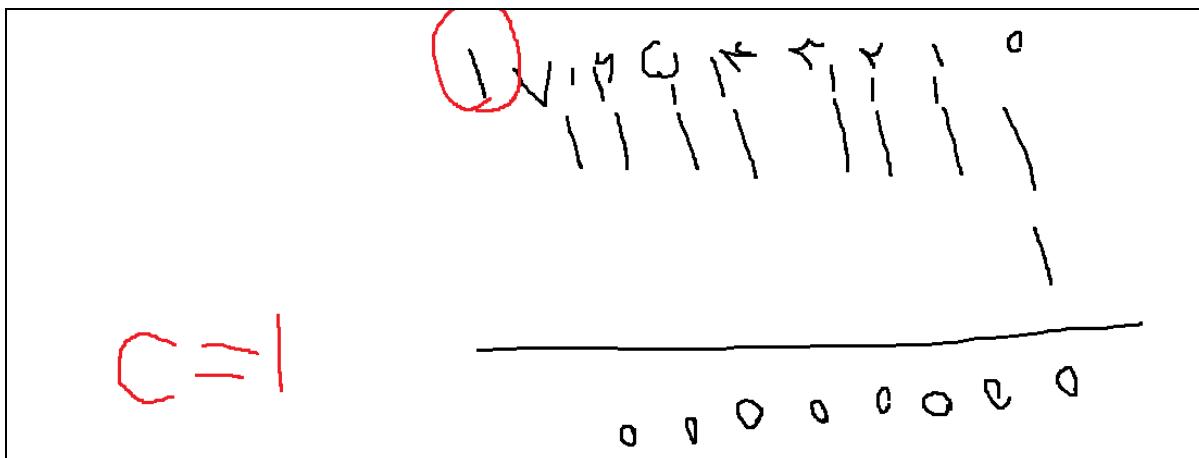
۱ - در موقع محابات عمل جمع کردن (ADD) اگر از بیت ۷ بیت نقلی خارج شود بیت پرچم نقلی شما برابر

۱ می‌شود

۲- اگر در زمان تفریق از بیت ۷ قرض گرفته شود این بیت برابر ۱ میشود.

محتویات A برابر با FFH است

ADD A ,#1



بیت پرچم نقلی کمکی (AC):

موقعی که دو عدد BCD با هم جمع میشوند اگر از بیت ۳ حاصل جمع بیت نقلی خارج شود و یا نتیجه ۴ بیت اول بین AH 0 تا 0FH باشد بزرگتر از ۹ بیت نقلی کمکی برابر ۱ می شود.

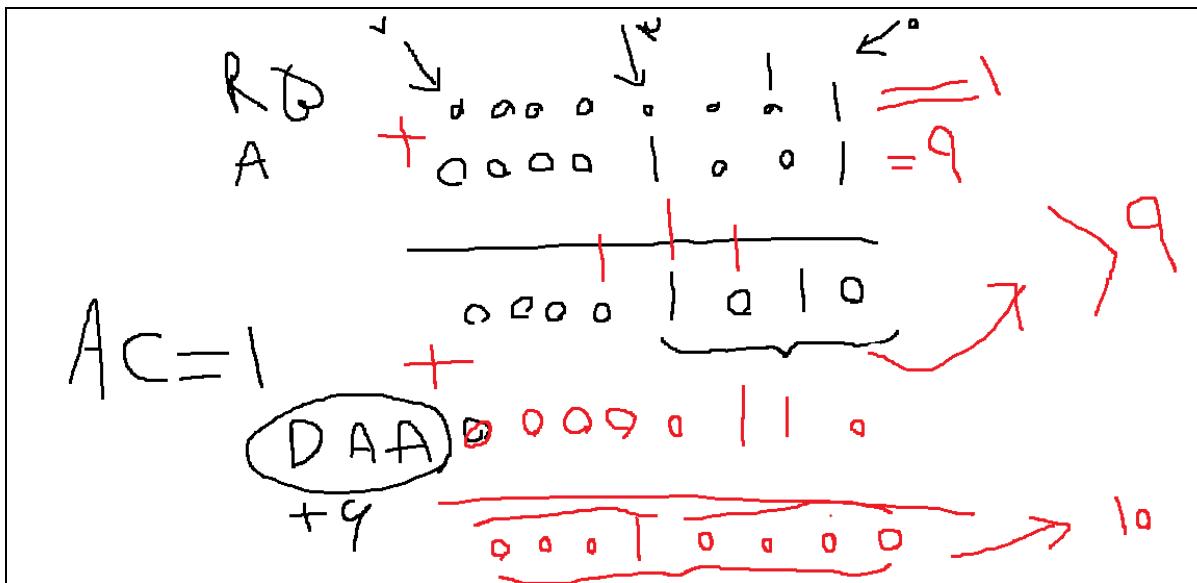
در صورتی که دستور جمع ADD استفاده شده باشد باید بعد از دستور مذکور از دستور DAA استفاده شود تا نتیجه صحیح بین ۰ تا ۹ قرار گیرد.

مثال

MOV R5,#1

MOV A , #9

ADD A,R5



بیت انتخاب بانک ثبات: بیت سوم و چهارم از ثبات وضعیت برای انتخاب بانک ثبات استفاده میشود (RS0 , RS1)

مثال : محتویت ثبات R7 از بانک ثبات ۳ را درون اکومولاتور قرار دهید.

SETB RS1

SETB RS0

MOV A,R7

فعال کردن بانک ثبات ۲

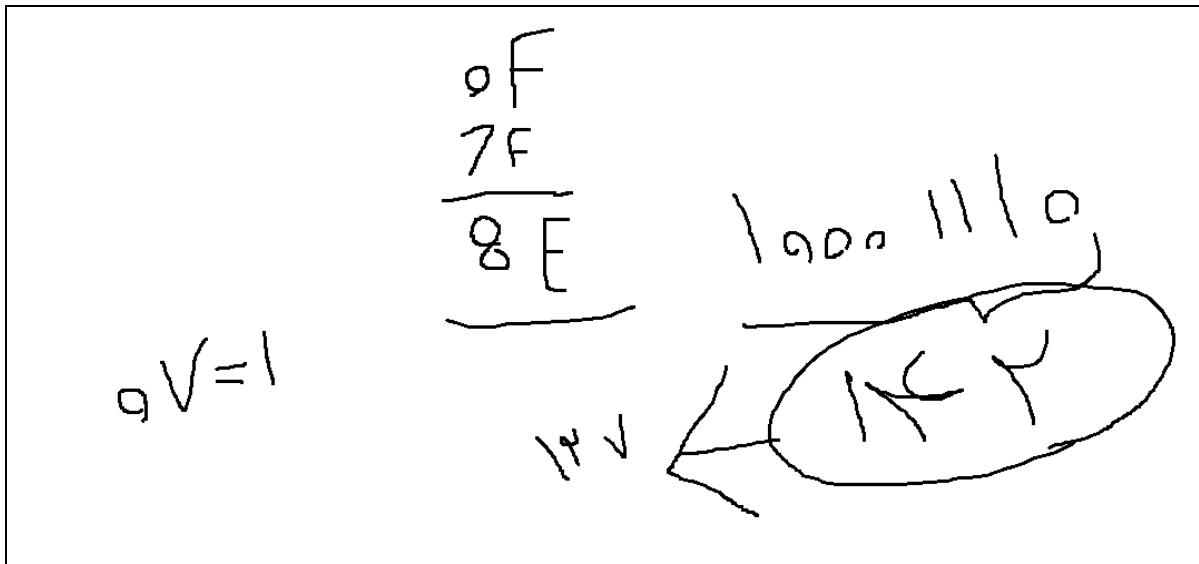
SETB RS1

CLR RS0

بیت پرچم سریز(OV):

بعد از عملیات جمع یا تفریق در صورتی که سریز به وجود آید این بیت برابر ۱ خواهد شد.

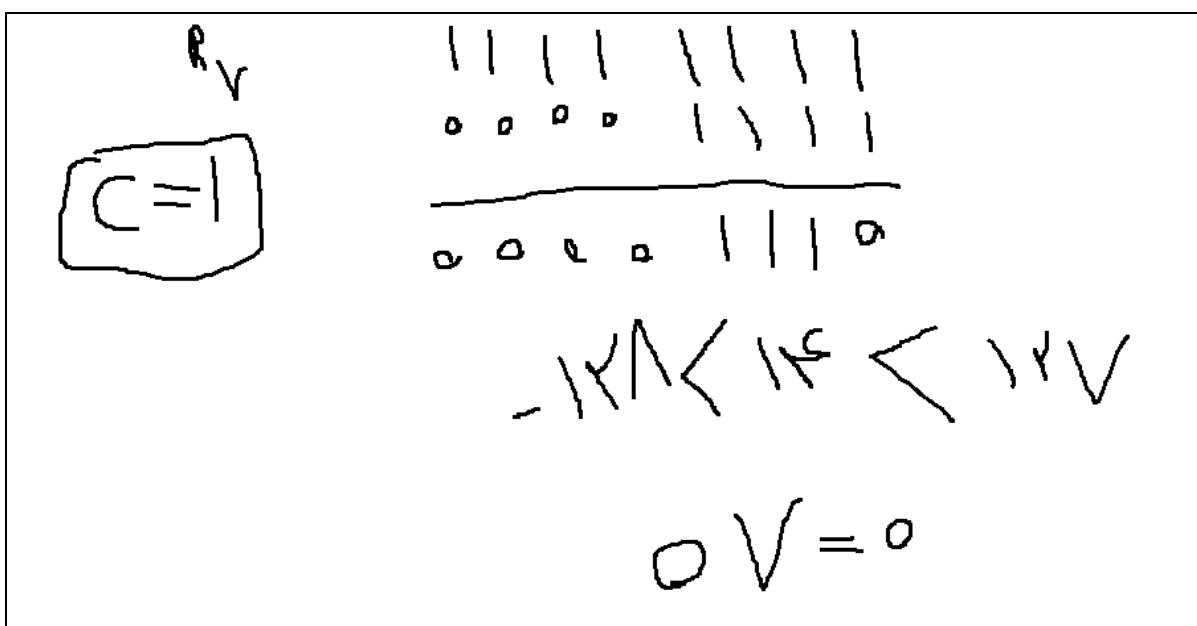
در موقع جمع اگر نتیجه بزرگتر از  $127 +$  و یا کوچکتر از  $128 -$  بشود سریز اتفاق می افتد.



MOV R7,#0FFH

MOV A , #0FH

ADD A , R7



بیت توازن(P):

بیت توازن طوری تولید میشود که تعداد بیتهای ۱ اکومولاتور A به علاوه بیت توازن P همواره زوج شود. لذا بیت مذکور بعد از هر سیکل ماشین به طور اتوماتیک ۱ یا ۰ میشود.

مثال: اگر اکومولاتور A دارای مقدار 10101101 باشد در این صورت بیت توازن برابر ۱ میشود. تا مجموع بیتهای برابر ۶ و عدد زوج شود. بیت توازن اکثرا در روتینهای پورت سری استفاده میشود بطوری که اطلاعات با بیت توازن ارسال میشود و در موقع دریافت اطلاعات بیت توازن تست میگردد.

### جلسه یازدهم

ثبتات یا اکومولاتور B معمولا با اکومولاتور A برای محاسبات ضرب و تقسیم استفاده می شود.

#### MUL AB

محتوای ثبات ۸ بیتی A و ثبات ۸ بیتی B را در هم ضرب کرده و حاصل ۱۶ بیتی را که ۸ بیت کم ارزش آن را در A و ۸ بیت پارازش تر را در B ذخیره می کند

#### DIV AB

محتوای ثبات ۸ بیتی A را بر ثبات ۸ بیتی B تقسیم می کند. خارج قسمت در A و باقی مانده در B ذخیره میشود

از ثبات B به عنوان یک ثبات عمومی برای هر کاربرد دیگری هم می توان استفاده کرد

### شاره گر پشته (SP)

ادرس اطلاعات بالای حافظه پشته را دارا می باشد. عملیات حافظه پشته شامل قرار دادن اطلاعات در بالای پشته (با دستور PUSH) و برداشتن اطلاعات (با دستور POP) از بالای حافظه پشته است.

برای قرار دادن اطلاعات در بالای حافظه پشته ابتدا یک واحد به SP اضافه شده و بعد اطلاعات در بالای حافظه درج میشود

#### SP=SP+1

#### PUSH

برای برداشتن اطلاعات از بالای پشته ابتدا اطلاعات خوانده شده و سپس یک واحد از SP کم میشود

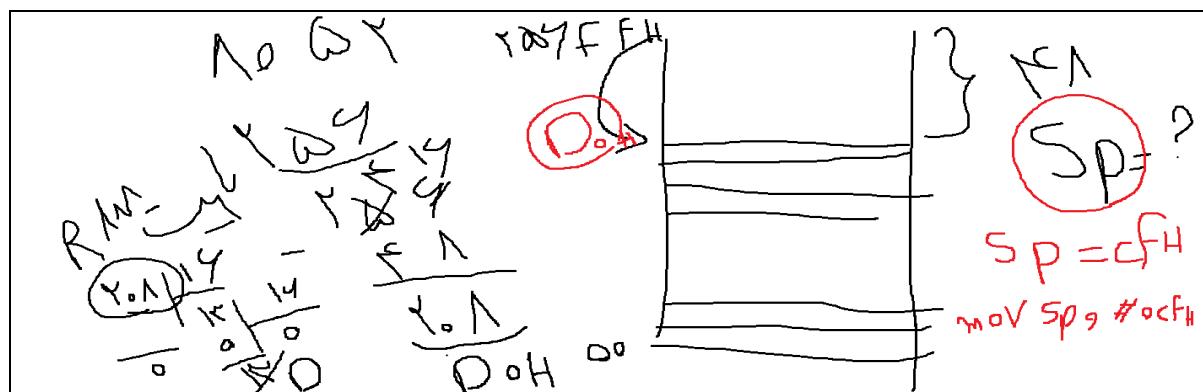
#### POP

$SP=SP-1$

اگر حافظه پشته از ادرس 60H حافظه RAM شروع شود در این صورت SP را با یک واحد کمتر یعنی با مقدار 5FH بار می کنند.

`MOV SP,#5FH`

دستوری بنویسید که اشاره گر پشته در ۲۵۶(۸۰۵۲) بايت حافظه RAM دارد را طوری ادرس دهد که حافظه پشته ای با تا ۴۸ بايت در بالای حافظه داخلی RAM بتواند ساخته شود



(DPTR) گر داده (DPTR)

یک ثبات ۱۶ بیتی است که از بايت کم ارزش DPL به ادرس 82H و بايت پرارزش DPH به ادرس H83H تشکیل شده است. از این اشاره‌گر برای دسترسی به حافظه کد و داده خارجی استفاده می‌شود.

این سه خط مقدار محتویات اکومولاتور A که 55H است را داخل ادرس 1000H از حافظه خارجی قرار می‌دهد.

`MOV A,#55H`

`MOV DPTR ,#1000H`

`MOVX @DPTR,A`

از دستور MOVX برای کار با حافظه خارجی استفاده می‌شود

ثبتات‌های پورت

پورت ۰

پورت ۱

پورت ۲

پورت ۳

زمانی که از حافظه خارجی استفاده می‌کنیم پورت ۰ و ۲ و ۳ برای انتقال ورودی و خروجی استفاده نمی‌شوند بلکه به عنوان بس داده ی ادرس یا برای کاربردهای خاص استفاده می‌شوند. ولی بیتهاي P1.7 تا P1.0 پورت ۱ همیشه برای خطهای ورودی و خروجی استفاده می‌شود نکته: P1.0 و P1.1 برای تایمر ۲ استفاده می‌شوند.

CLR P1.7

CLR 97H

این دو دستور معادل هستند و هر دو به بیت بالارزش پورت ۱ اشاره می‌کنند و ان را ۰ می‌کنند.

ثباتهای تایمر:

میکرو کنترلر ۸۰۵۱ دارای دو تایمر به نام تایمر ۰ و تایمر ۱ است که ۱۶ بیتی می‌باشند و برای اندازه گیری زمان و شمارش رویداد به کار می‌روند. ثباتهای تایمر ۰ به نام TL0 بایت کم ارزشتر تایمر ۰ در ادرس 8AH و TH0 بایت پارازشتر تایمر ۰ در ادرس 8CH است. ثباتهای تایمر ۱ نیز به نام TL1 بایت کم ارزشتر تایمر ۱ در ادرس 8BH و TH1 بایت پارازشتر تایمر ۱ در ادرس 8DH است. طرز کار تایمر با ثبات حالت TMOD در ادرس 89H و هم چنین ثبات کنترل TCON در ادرس 88H مشخص می‌شود. و بیتهاي ثبات کنترل TCON ادرس پذیر هستند.

ثباتهای پورت سری:

میکرومنترلر ۸۰۵۱ دارای پورت سری در داخل تراشه برای اتصال به دستگاههای سری مانند ترمینال مودم و ... است. ثبات بافر SBUF در ادرس 99H اطلاعات ارسالی و دریافتی را نگهداری می‌کند. نوشتن در ثبات

باfr SBUF برای ارسال اطلاعات و خواندن از ثبات باfr SBUF برای خواندن اطلاعات دریافتی می باشد.  
حالتهای مختلف عملکرد پورت سری با برنامه ریزی ثبات کنترل SCON در ادرس 98H انجام می شود.

#### ثباتهای وقفه:

میکرومنترلر ۸۰۵۱ دارای ۵ منبع وقفه با دو سطح اولویت است. وقفه ها بعد از RESET میکروکنترلر غیرفعال میشوند و با نوشتمن در ثبات فعال ساز وقفه (IE) در ادرس A8H فعال میشوند. اولویت وقفه ها نیز با ثبات اولویت وقفه (IP) در ادرس B8H مشخص میشود. بیتهای هر دو ثبات ادرس پذیر هستند.

#### جلسه دوازدهم:

#### ثبات کنترل توان (PCON):

در ادرس 87H است و شامل بیتهای کنترلی مختلف است.

بیت ۸ با نام SMOD: بیت دو برابر کردن پورت سری: اگر این بیت ۱ شود سرعت ارسال و دریافت پورت سری در حالتهای ۱، ۲ و ۳ برابر میشود.

بیت دوم: بیت توان کم: اگر این بیت ۱ شود میکروکنترلر در حالت توان کم قرار میگیرد و فقط با RESET کردن از این حالت خارج میشود

بیت اول: بیت حالت معلق: اگر ۱ شود در حالت معلق قرار میگیرد و فقط با RESET خارج میشود

حالت معلق: در حالت معلق سیگنال ساعت داخلی به CPU اعمال نمیشود. ولی برای تایمیرها وقفه ها و پورت سری استفاده میشود. وضعیت CPU و ثباتها ذخیره میشود. پایه های پورتها مقدار خود را نگه میدارند. سیگنالهای ALE و PSEN برابر ۱ میشوند. حالت معلق توسط هر وقفه که فعال شود با RESET شدن سیستم از بین میرود

#### حالت توان کم:

اسیلاتور تراشه میکروکنترلر متوقف میشود

تمام کارها متوقف میشود

تمام محتویات حافظه RAM نگهداری میشود

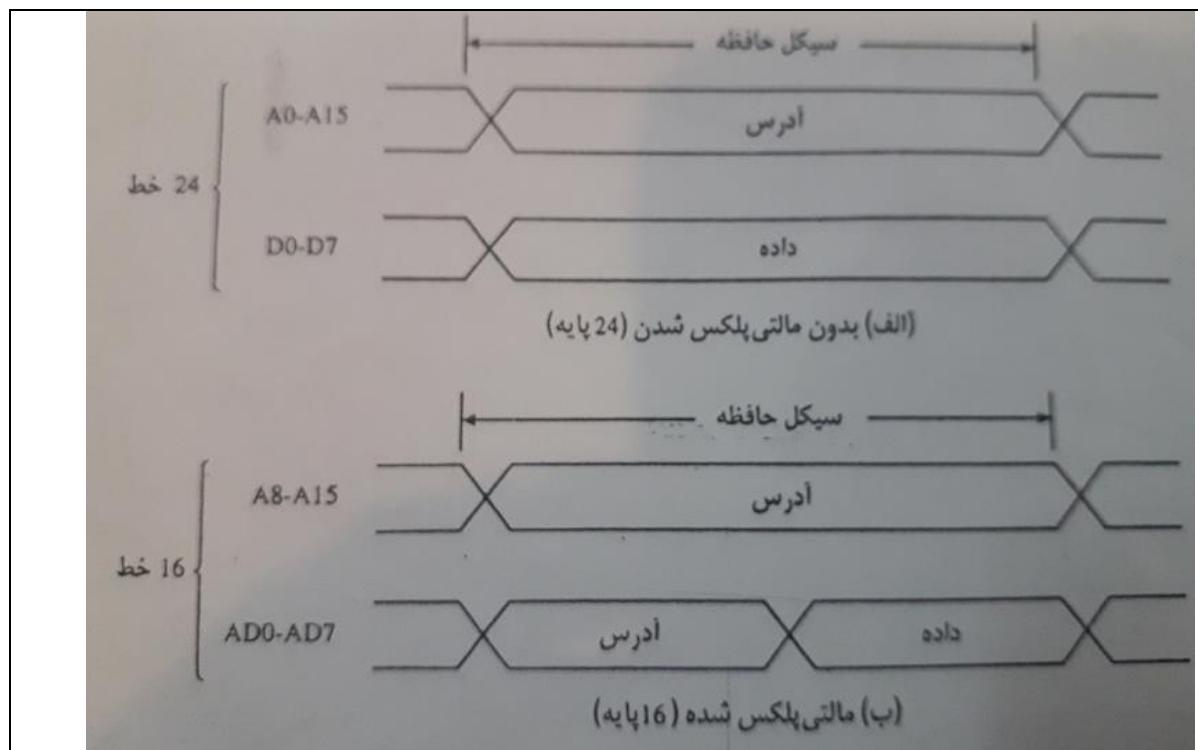
تمام پایه های پورتها مقدار خود را نگه میدارند

سیگنالهای PSEN , ALE برابر میشود.

تنها راه خروج از این حالت RESET کردن میکروکنترلر است.

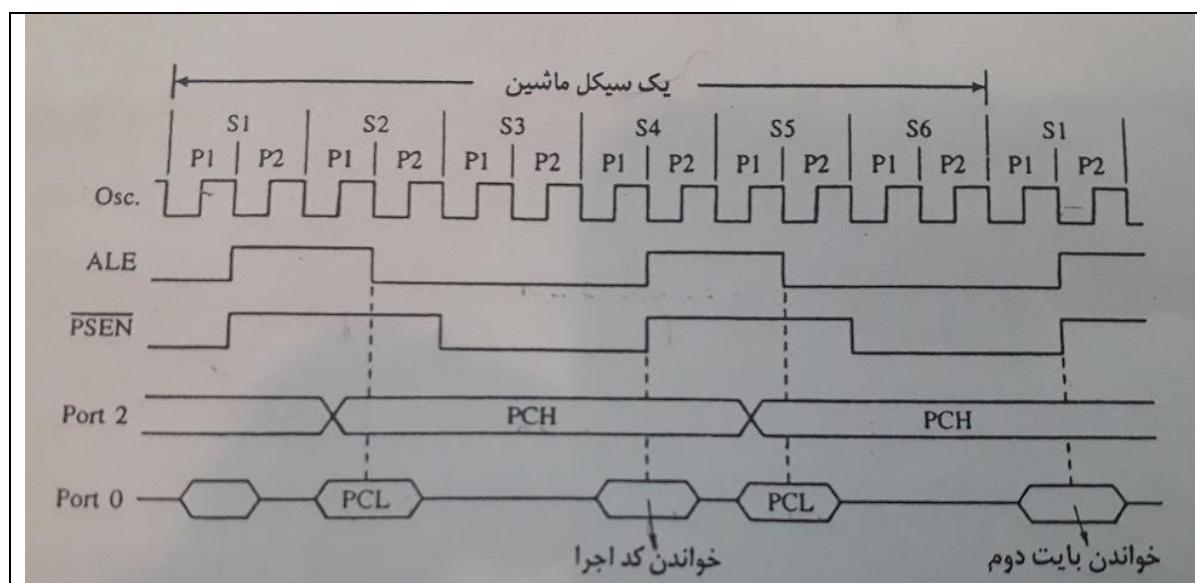
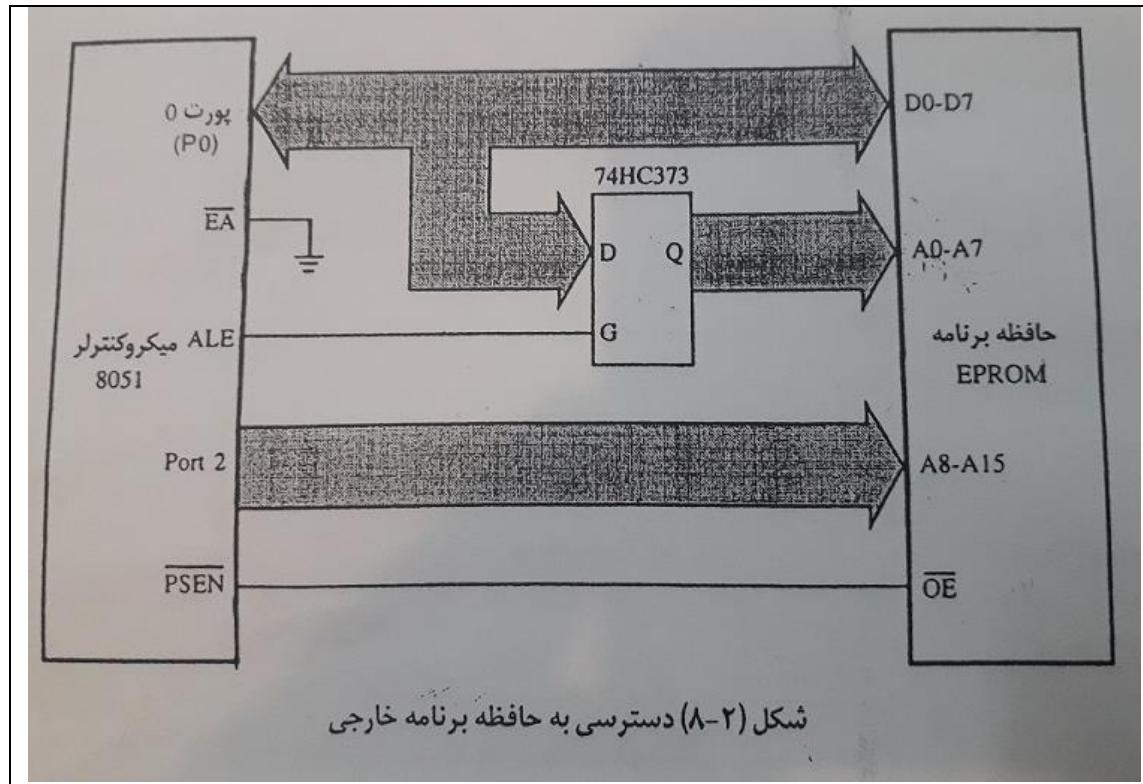
### حافظه خارجی:

برای ارتباط با حافظه خارجی پورت ۰ و ۲ به عنوان ورودی و خروجی استفاده نمیشود بلکه به عنوان باس ادرس و داد استفاده میشود(پورت ۰ هم برای بایت کم ارزش ادرس و هم باس داده و پورت ۲ برای بایت پارازش باس ادرس) اگر از امکان مالتی پلکس شدن پورت ۰ بین باس ادرس و داده که توسط سیگنال ALE ایجاد میشود استفاده کنیم ۱۶ پایه برای ارتباط با حافظه خارجی مورد نیاز است ولی اگر از این امکان استفاده نشود ۱۶ پایه برای باس ادرس و ۸ پایه برای باس داده مورد نیاز است که روی هم به ۲۴ پایه احتیاج است.



### دسترسی به حافظه برنامه یا کد خارجی:

حافظه برنامه خارجی یک حافظه فقط خواندن ROM یا EPROM است که با سیگنال فعال صفر PSEN فعال میشود موقعی که حافظه EPROM استفاده میشود دو پورت ۰ و ۲ برای ورودی و خروجی استفاده نمیشود بلکه برای بس ادرس و داده به کار برده میشود.

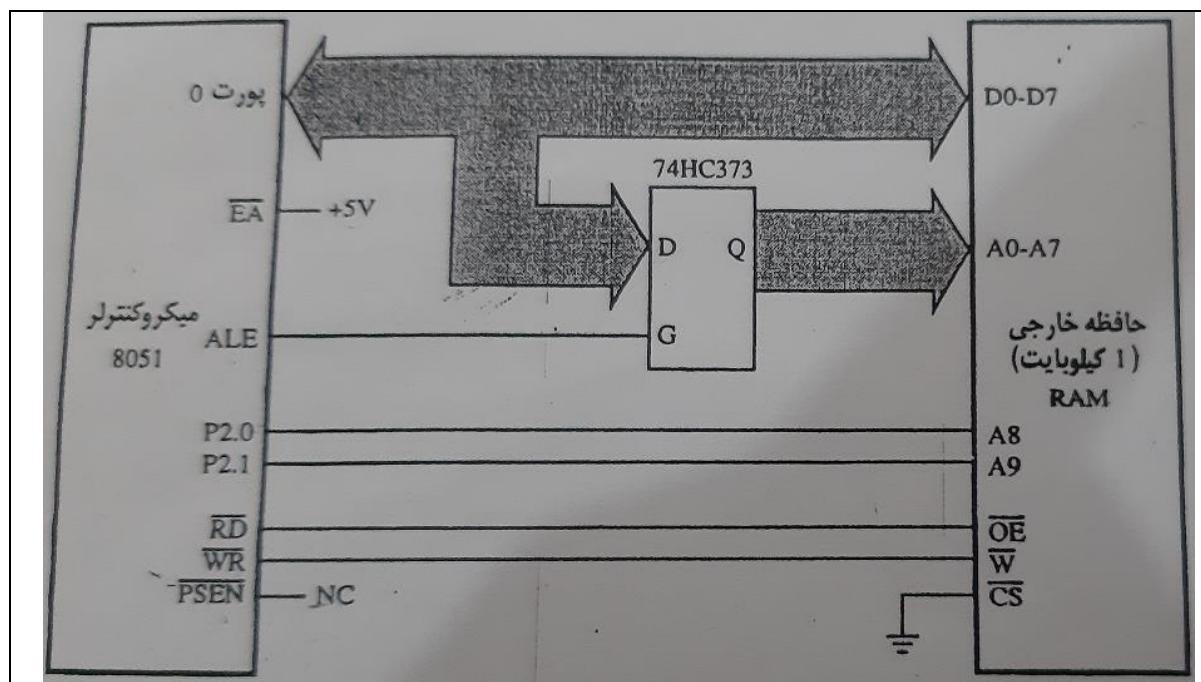


در رابطه با شکل بالا هر سیکل ماشین میکروکنترلر ۸۰۵۱ دوازده پریود اسیلاتور تراشه میکروکنترلر است. اگر فرکانس اسیلاتور ۱۲ مگا هرتز باشد در این صورت یک سیکل ماشین ۱ میکروثانیه است در زمان یک سیکل ماشین دوبار سیگنال ALE پالس می دهد و دو بایت از حافظه خوانده میشود. (در صورتیکه دستور یک بایتی باشد بایت دوم خوانده نمیشود). خواندن از حافظه برنامه خارجی به واکشی دستور معروف است.

### دسترسی به حافظه داده خارجی RAM

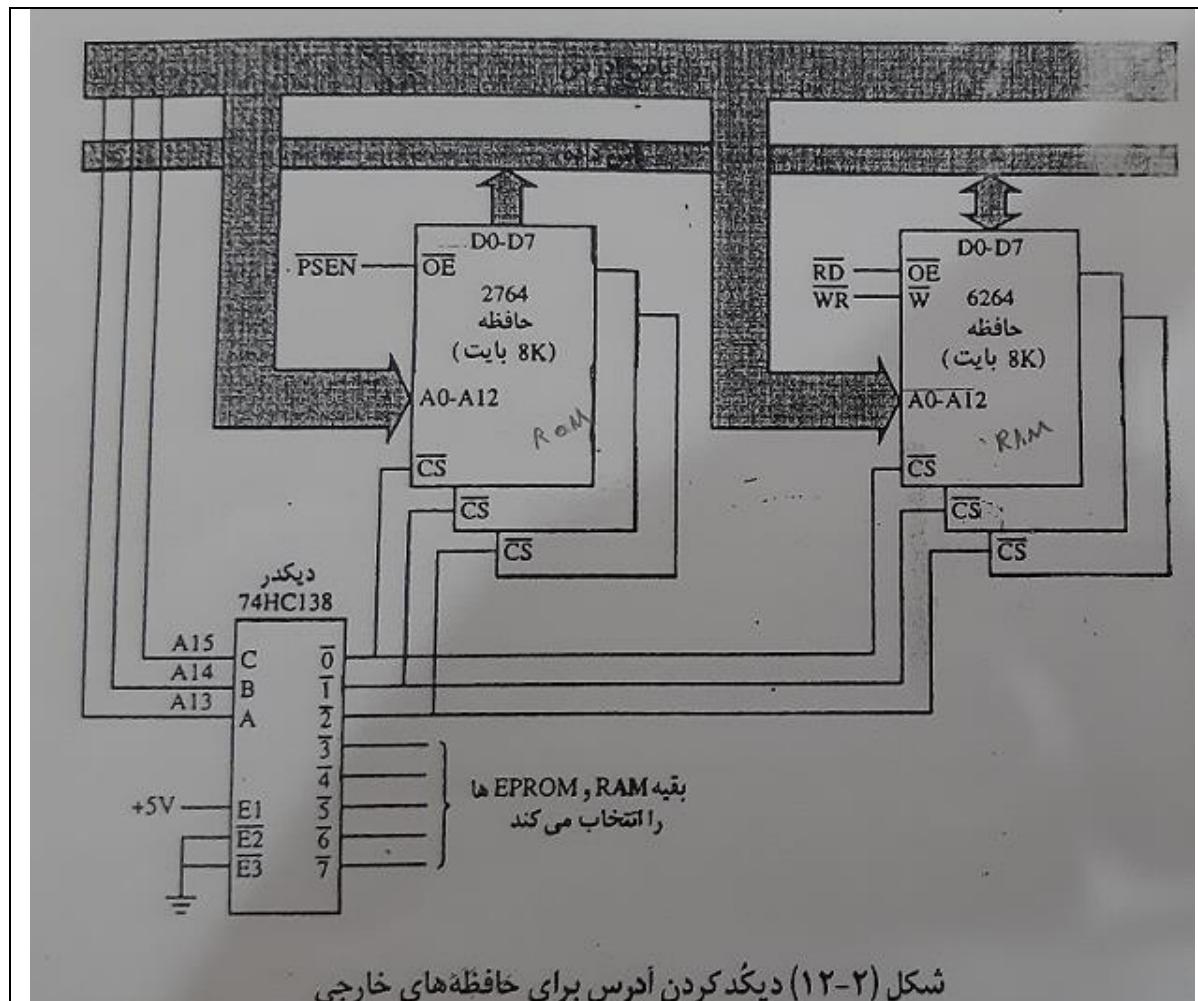
حافظه داده خارجی یک حافظه قابل خواندن و نوشتن RAM است که با سیگنال RD فعال صفر(پایه P3.7) و نوشتن WR فعال صفر(پایه P3.6) فعال میشود. دسترسی به حافظه داده خارجی فقط با دستور MOVX و اشاره گر ۱۶ بیتی D PTR یا ثباتهای R0 و R1 به عنوان ثبات ادرس امکان پذیر است.

حافظه RAM مانند حافظه EPROM به میکروکنترلر ۸۰۵۱ متصل میشود با این تفاوت که سیگنال خواندن و نوشتن میکرومنترلر به ترتیب به خروجی فعالساز (OE فعال صفر) و خط نوشتن (W فعال صفر) حافظه RAM متصل میشود. اتصالات باس ادرس و باس داده حافظه RAM مانند EPROM است و پورتهای ۰ و ۲ می توانند تا 64K بایت حافظه RAM را ادرس دهی کنند.



از ۸ بیت ادرس برای یک صفحه داده حافظه خارجی ۲۵۶ بایتی استفاده میشود. در صورتی که بیش از یک صفحه حافظه لازم باشد چند بیت پورت ۲ برای انتخاب صفحه حافظه استفاده میشود.

### دیکد کردن ادرس حافظه



اگر **EPROM** و **RAM** 8K بایت باشند دراین صورت از ۱۳ پایه برای ادرس استفاده می‌شود و از ۳ بیت پرارزش تر باس ادرس برای دیکدر و انتخاب تراشه ها با پایه CS استفاده می‌شود.

### عملیات RESET کردن

میکروکنترلر ۸۰۵۱ موقعی **RESET** می‌شود که ورودی **RST** آن به مدت حداقل دو سیکل ماشین برابر ۱ و سپس به ۰ برگردد. بعد از عمل **RESET** میکروکنترلر تمام ثباتهای خود را در حالت اولیه قرار میدهد.

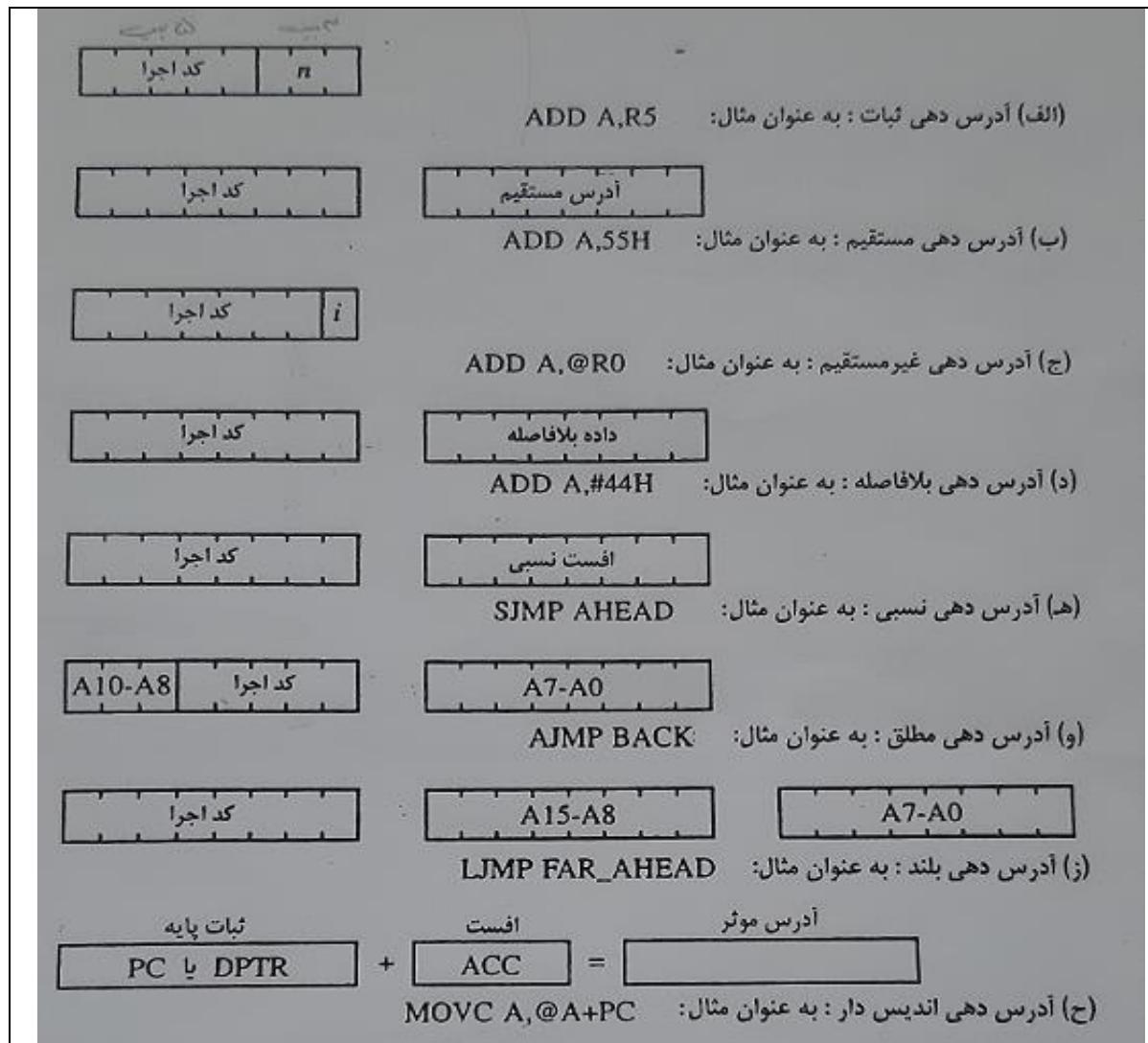
جدول (۶-۲) مقادیر ثبات‌ها بعد از Reset میکروکنترلر

ثبات‌ها	محتوا
کنتور برنامه (PC)	0000H
اکومولاتور (A)	00H
ثبات B	00H
PSW	00H
SP	07H
اشارة گر داده DPTR	0000H
Ports 0-3	FFH
IP (8031/8051)	XXX00000B
IP (8032/8052)	XX000000B
IE (8031/8051)	0XX00000B
IE (8032/8052)	0X000000B
ثبات‌های تایمر	00H
SCON	00H
SBUF	00H
PCON (HMOS)	0XXXXXXXB
PCON (CMOS)	0XXX0000B

### جلسه ۱۳

پروسسور ۸ بیتی میکروکنترلر ۸۰۵۱ ۸۰۵۱ دارای کد اجرایی (opcode) ۸ بیتی است در نتیجه امکان داشتن تا ۲۵۶ دستور را دارد. برخی از دستورات دارای یک بایت یا دو بایت اضافی برای داده و ادرس هستند. در کل تعداد ۱۳۹ دستور یک بایتی ۹۲ دستور دو بایتی و ۲۴ دستور ۳ بایتی در ۸۰۵۱ وجود دارد.

روشهای ادرس دهی:



آدرس دهی ثبات:

میکروکنترلر ۸۰۵۱ به هشت ثبات با نامهای R0 تا R7 دسترسی دارد. در دستورات آدرس دهی ثبات، ۳ بیت کم ارزش تر کد دستور، آدرس ثباتهای مذکور را مشخص می کند. بنابراین کد اجرا و آدرس عملوند یا ثبات در یک بایت قرار می گیرد.

**ADD A,R7**

**00101111**

**MOV A,R7**

**11101111 =FFH**

چهار بانک ثبات کاری در ۸۰۵۱ وجود دارد که هر کدام شامل ثبات‌های R0 تا R7 هستند. در هر لحظه فقط یکی بانک ثبات فعال است و قابل استفاده است. بانکهای ثبات ۳۲ بایت اول ۸۰۵۱ را شامل می‌شوند. موقعی که میکروکنترلر RESET می‌شود بانک ۰ ثبات فعال می‌گردد ولی با تغییربیت‌های ۳ و ۴ ثبات وضعیت PSW می‌توان هر یک از بانکها را انتخاب نمود.

مثال : میخواهیم با ثبات ۵ از بانک ثبات شماره ۳ کار کنیم

MOV PSW #00011000B

MOV A,R5

بعضی دستورات، مخصوص ثبات‌های اکومولاتور A و B و اشاره گر DPTR کنتور برنامه PC و بیت پرچم نقلی C می‌باشند، که ادرس آنها در دسترسی وجود ندارد در این حالت کد اجرا به تنها یک مشخص کننده نوع عملیات و ثبات است مانند

DIV AB

INC DPTR

MUL AB

ادرس دهی مستقیم:

با این روش می‌توان به هر یک از ثبات‌ها را خانه‌های حافظه داخل تراشه دسترسی پیدا کنید. برای این کار یک بایت اضافی(بایت دوم) که دنباله کد اجرا است محل داده را مشخص می‌کند.

بسته به بیت پرارزش ادرس مستقیم یکی از دو بخش حافظه RAM مورد استفاده قرار می‌گرد

اگر این بیت ۰ باشد آدرس مستقیم بین ۰ تا ۱۲۷(127H تا 00H) و اگر این بیت ۱ باشد ادرس مستقیم بین ۱۲۸ تا ۲۵۵(80H تا FFH) است

نیازی به دانستن ادرس پورتها و ثبات‌های خاص نیست چون برنامه مترجم اسمبلر از نام سیمبولیک، آدرس انها را تشخیص می‌دهد.

**MOV P1,A**

دو بایتی است

**MOV SCON,#55H**

**75H 98H 55H**

سه بایتی هست

خود مترجم اسمنلر ادرس مستقیم P1 و SCON را محاسبه می کند.

ادرس دهی غیر مستقیم:

اگر لازم باشد به خانه های پشت سر هم حافظه، جدول حافظه یا رشته ای از اطلاعات دسترسی پیدا کنید باید از آدرس دهی غیرمستقیم استفاده نمود. در این صورت در زمان اجرای برنامه، می توان ادرس را محاسبه یا عوض نمود و به خانه های حافظه مذکوردسترسی پیدا کرد.

برای ادرس دهی غیر مستقیم در ۸۰۵۱ از ثباتهای R0 و R1 به عنوان اشاره گر حافظه استفاده میشود. که محتوای انها ادرس حافظه داخلی RAM را مشخص میکند تا بتوان در این خانه های حافظه نوشت و یا از ان خواند. کم ارزشترین بیت کد دستور، ثبات R0 یا R1 را به عنوان اشاره گر حافظه مشخص می کند. اگر بیت کم ارزش ۰ بود یعنی R0 و اگر ۱ بود یعنی R1 انتخاب شود. در زبان اسمنلی میکروکنترلر ادرس غیرمستقیم با علامت @ که قبل از R0 و R1 قرار میگیرد مشخص میشود.

مثال: ثبات R1 برابر 40H است و محتوای خانه به ادرس 40H برابر 55H است:

**MOV A,@R1**

عدد 55H را به اکومولاتور A منتقل میکند.

کد اجرای دستور زیر چیست؟

**MOV A,@R0**

**11100110=E6H**

مثال: قطعه کدی که خانه های H 60 تا 7FH را برابر ۰ کند؟

**MOV R0,#60H**

**LOOP: MOV @R0,#0**

INC R0

CJNE R0,#80H,LOOP

جلسه ۱۴

ادرس دهی بلاfacسله

در دستور میکروکنترلر موقعی که اپرند منبع یک عدد ثابت باشد یک بایت داده یا اپرند بلاfacسله است که بلاfacسله بعد از کد اجرا می اید و بایت دوم دستور می باشد.

MOV A,#12

اپرند بلاfacسله یک عدد یا یک سیمبول یا عبارت محاسباتی می باشد. در این صورت برنامه مترجم اسمنلر مقدار سیمبول یا عبارت محاسباتی را محاسبه و مقدار عددی آن را در دستور قرار می دهد.

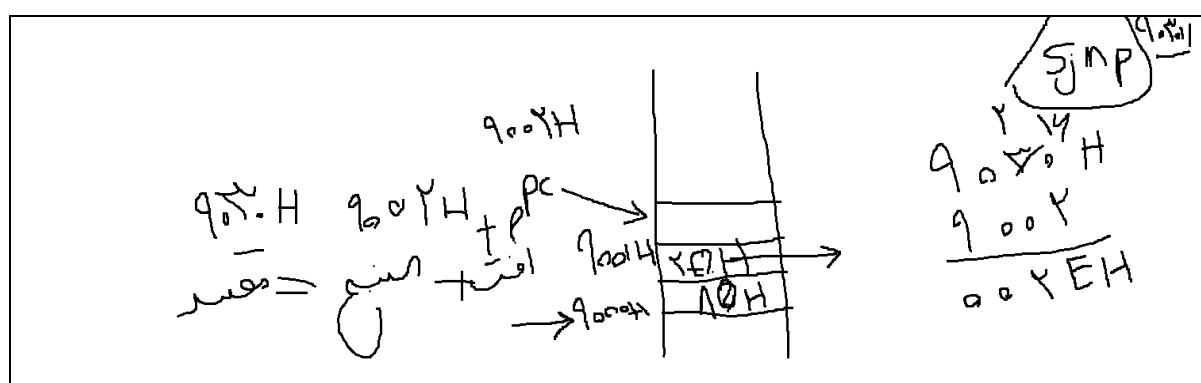
MOV DPTR, #8000H

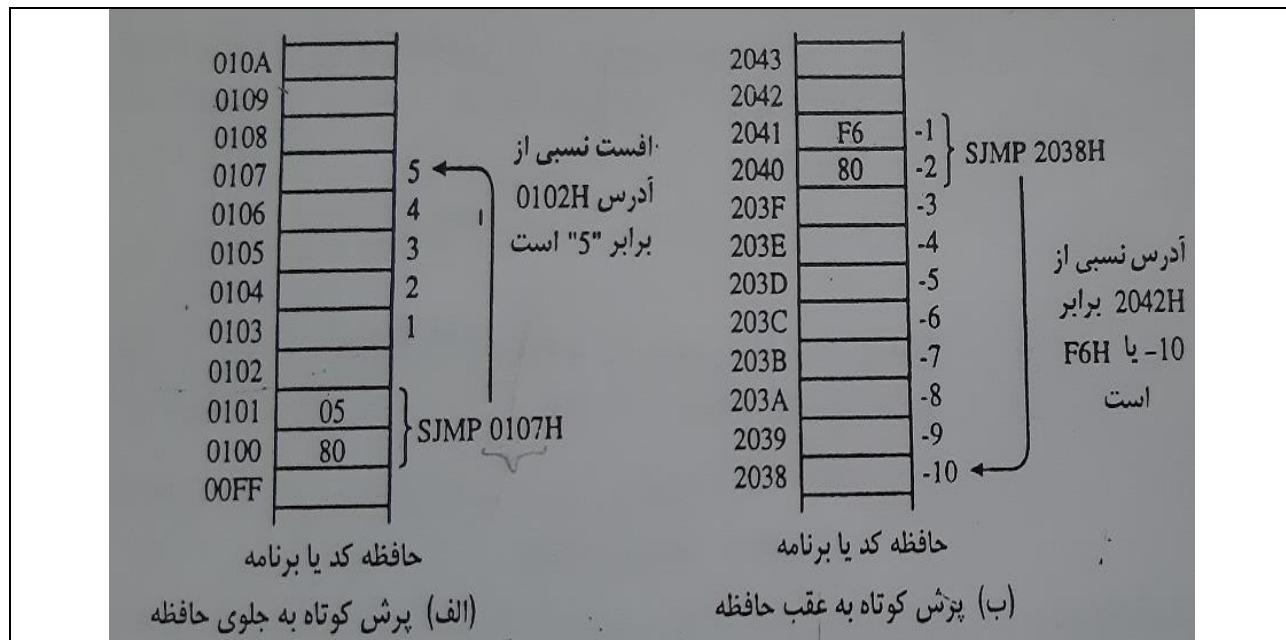
دستوراتی که با ثبات DPTR کار می کنند قسمت اپرند منبع آنها ۱۶ بیتی است.

ادرس دهی نسبی

ادرس دهی نسبی در برخی از دستورات پرس استفاده میشود. ادرس نسبی یا افست یم عدد هشت بیتی علامت دار است و به عنوان یک بایت دوم در ادامه کد اجرا قرار میگیرد و به کنتور برنامه یا PC اضافه میشود تا ادرس دستور بعدی که باید اجرا شود را بسازد. چون افست یک عدد ۸ بیتی علامتدار است عمل پرس ۱۲۷+ تا بلاتر و -۱۲۸ تا پایینتر پرس می کند. چون قبل از اضافه شدن افست، به PC یک واحد اضافه شده است لذا ادرس جدید نسبت به ادرس دستور بعدی پرس است نه نسبت به ادرس خود پرس.

مثال دستور SJMP 9030H درخانه 9000H و 9001H قرار دارد ترجمه این دستور به زبان ماشین چیست؟





مثال دیگر: ترجمه دستور sjmp برابر H 80H,F6H است و در خانه های حافظه 0802H و 0803H قرار دارد  
به چه ادرسی پرش انجام میشود؟

جواب: افست این دستور برابر F6H است که یک عدد منفی می باشد پس پرش به عقب است. ادرس منبع ادرس بعد از پرش یعنی محتوای PC و 0804H است.

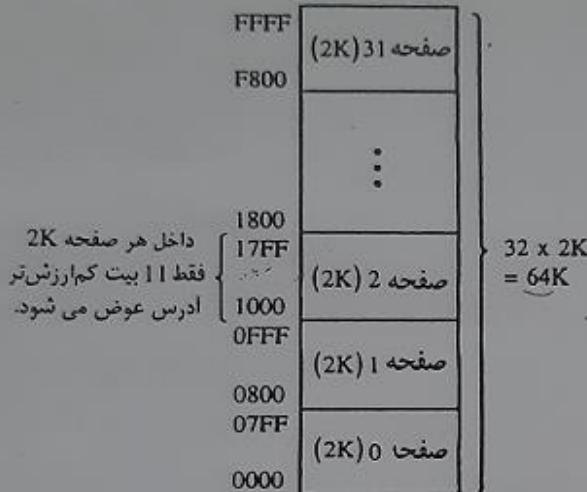
افست+ادرس منبع=ادرس مقصد

$$0804H + FFF6H = 07FAH$$

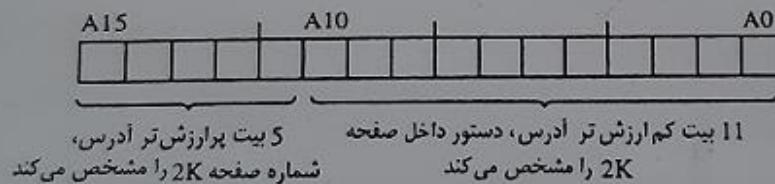
**ادرس دهی مطلق:**

ادرس دهی مطلق، فقط برای دستورات ACALL (فراخوانی سابروتین) و AJMP (پرش مطلق) به کار برد میشود و دو بایتی است. با این دستورات می توان تا 2K بایت، در داخل یک صفحه حافظه برنامه پرش نمود. (با 11 بیت کم ارزشتر ادرس یعنی از A0 تا A7 در بایت دوم دستور و A8 تا A10 سه بیت پرازش باشد اول یعنی کد اجرا).

5 بیت پرازشتر ادرس مقصد (A15 تا A11) همان 5 بیت پرازشتر کنتور برنامه PC است و چون این بیتهاي ادرس تغییر نمی کنند، لذا دستور بعد از دستور انشعاب و ادرس مقصد دستور انشعاب، باید در صفحه 2K بایت باشند.



الف: نقشه حافظه که صفحات 2K بایتی را تشکیل می‌دهد.



ب: در داخل یک صفحه، پنج بیت پرازش تر برای آدرس‌های منبع و مقصد یکسان هستند و ۱۱ بیت کم ارزش تر آدرس، دستور داخل صفحه ۵ بیت پرازش تر آدرس، شماره صفحه 2K را مشخص می‌کند

مثال : اگر برچسب THERE در آدرس 0F46H را نمایش دهد و دستور AJMP THERE

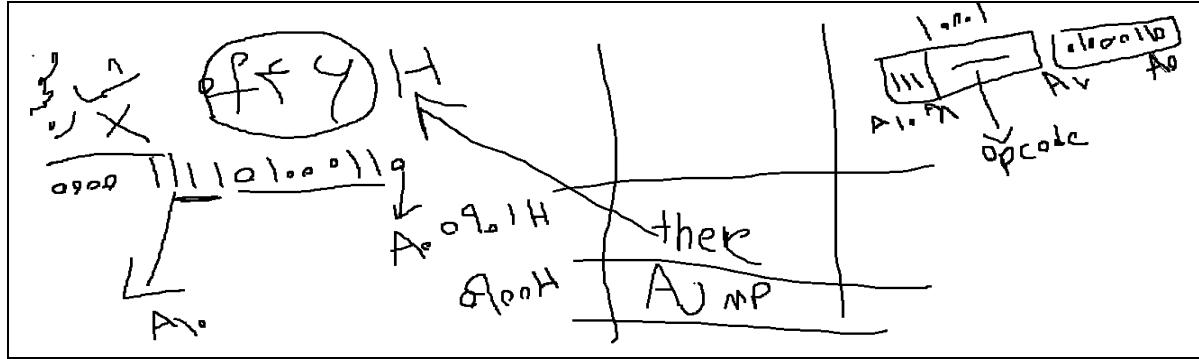
در آدرس 0900H و 0901H قرار داشته باشد در این صورت برنامه مترجم اسمبلي دستور THERE را به صورت زیر ترجمه می‌کند.

**=OF46H=0000111101000110**

00001 آدرس صفحه است که در آدرس دهی مطلق داخل یک صفحه 2K بایتی تغییر نمی‌کند.

11 بیت کم ارزش (A8) بیت اول، A7 در بایت دوم و ۳ بیت پرازش A10 تا A10 در سه بیت پرازش بایت اول به همراه کد اجرایی ذخیره می‌شود.)

**1110000101000110**



یکی از مزیتهای ادرس دهی مطلق این است که دستور ان کوتاه و دوبایتی است ولی اشکال ان این است که ادرس مقصد ان، محدود و در حد  $2K$  بایت است و تابع محل دستور است. ادرس دهی مطلق فقط موقعی که ۵ بیت پر ازش ادرس مقصد و منبع یکسان باشند استفاده می شود یعنی ادرس دستور منبع و دستور مقصد در یک صفحه  $2K$  حافظه باشند.

### ادرس دهی بلند:

ادرس دهی بلند فقط با دستورات فراخوانی سابروتین (LCALL) و دستور پرش بلند (LJMP) بکار برده می شود. این دستورات سه بایتی هستند که در بایت دوم و سوم آنها ادرس  $16$  بیتی مقصد می باشد. یکی از مزیتهای این دستورات این است که میتوان به تمام فضای حافظه برنامه (کد) دسترسی پیدا کرد ولی اشکال آنها این است که  $3$  بایت طول دارند.

مثال: دستور LJMP 2040H ترجمه آن به زبان ماشین چیست

02H 20H 40H

### ادرس دهی اندیس دار

ادرس دهی اندیسدار در دستورات پرش JMP و انتقال MOVC استفاده می شود که محتوای ثبات پایه ای مانند PC، اشاره گر داده DPTR را با محتوای اکومولاتور A جمع می کند تا ادرس موثر یا ادرس مقصد حاصل

شود. با استفاده از این دستورات می توان به نقاط مختلف جدول جستجو در حافظه به اسانی دسترسی پیدا کرد.

مثال:

MOVC A,@A+DPTR

MOVC A,@A+PC

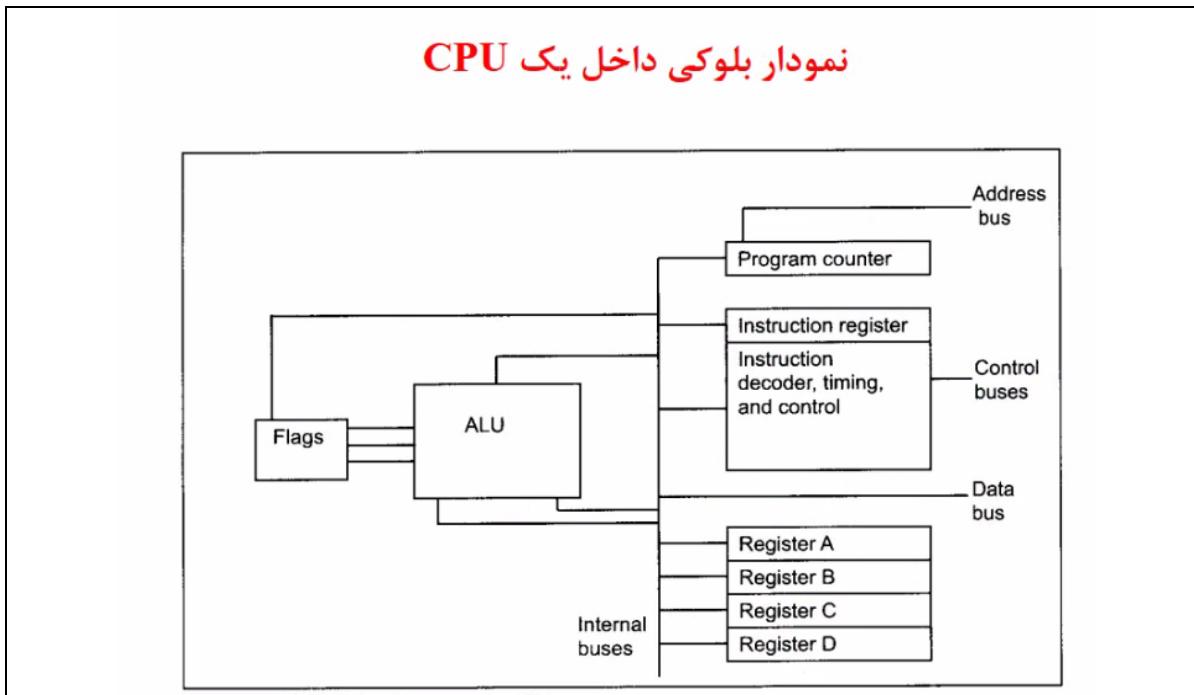
یا

Jmp @A+DPTR

نکته: از دستور MOVC برای انتقال با حافظه برنامه یا کد استفاده میشود.

## جلسه ۱۵:

## نمودار بلوکی داخل یک ریزپردازنده



وظیفه cpu این است که دستوری را با استفاده از ثبات pc از حافظه بردارد (واکشی یا fetch) و آن را اجرا کند.

برای عمل fetch و run در cpu باید ۴ امکان وجود داشته باشد

۱- ثباتها: هر cpu یه تعدادی ثبات دارد ثبات A B C D . ثباتهای می توانند ۸ یا ۱۶ یا ۳۲ یا ۶۴ بیتی باشند. هر چه تعداد ثبات بیشتر و اندازه آنها بزرگتر باشد CPU قدرتمندتر و گران تر است. ثباتها اطلاعات را به صورت اساس تعداد ثبات و اندازه آنها با هم متفاوت هستند و انواع مختلفی دارند. ثباتها اطلاعات را به صورت موقت ذخیره می کنند و سرعت دسترسی به این اطلاعات نسبت به سرعت دسترسی به RAM خیلی بالاتر است نکته: FLAGها جز ثباتها هستند که چند بیت آنها برای کارهای کنترلی قابل استفاده هستند.

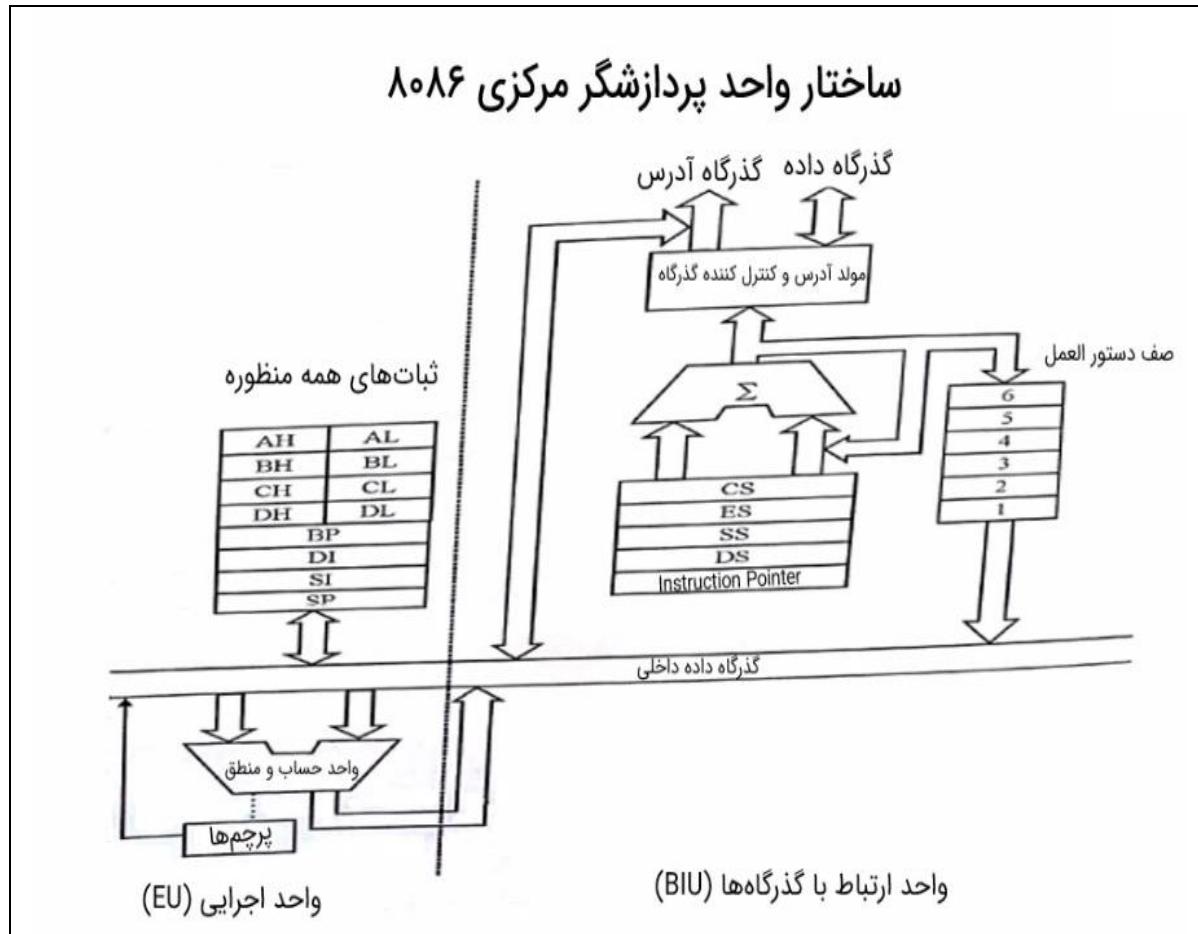
۲- ALU: کارهای ریاضی و منطقی توسط این قسمت انجام میشود

۳- PC: ادرس دستور بعدی که باید اجرا شود درون PC وجود دارد محتویات PC بر روی باس ادرس قرار میگیرد و دستور مورد نظر از حافظه واکشی و اجرامیشود.

۴- دیکدر دستور: کار ان تفسیر دستور واکشی شده است.

FLAG ها جز ثباتها هستند.

درون ۸۰۸۶



ریز پردازنده از دو بخش مهم واحد ارتباط با گذرگاهها(BIU) و واحد اجرایی EXCUTED UNIT(EU) تشکیل شده است.

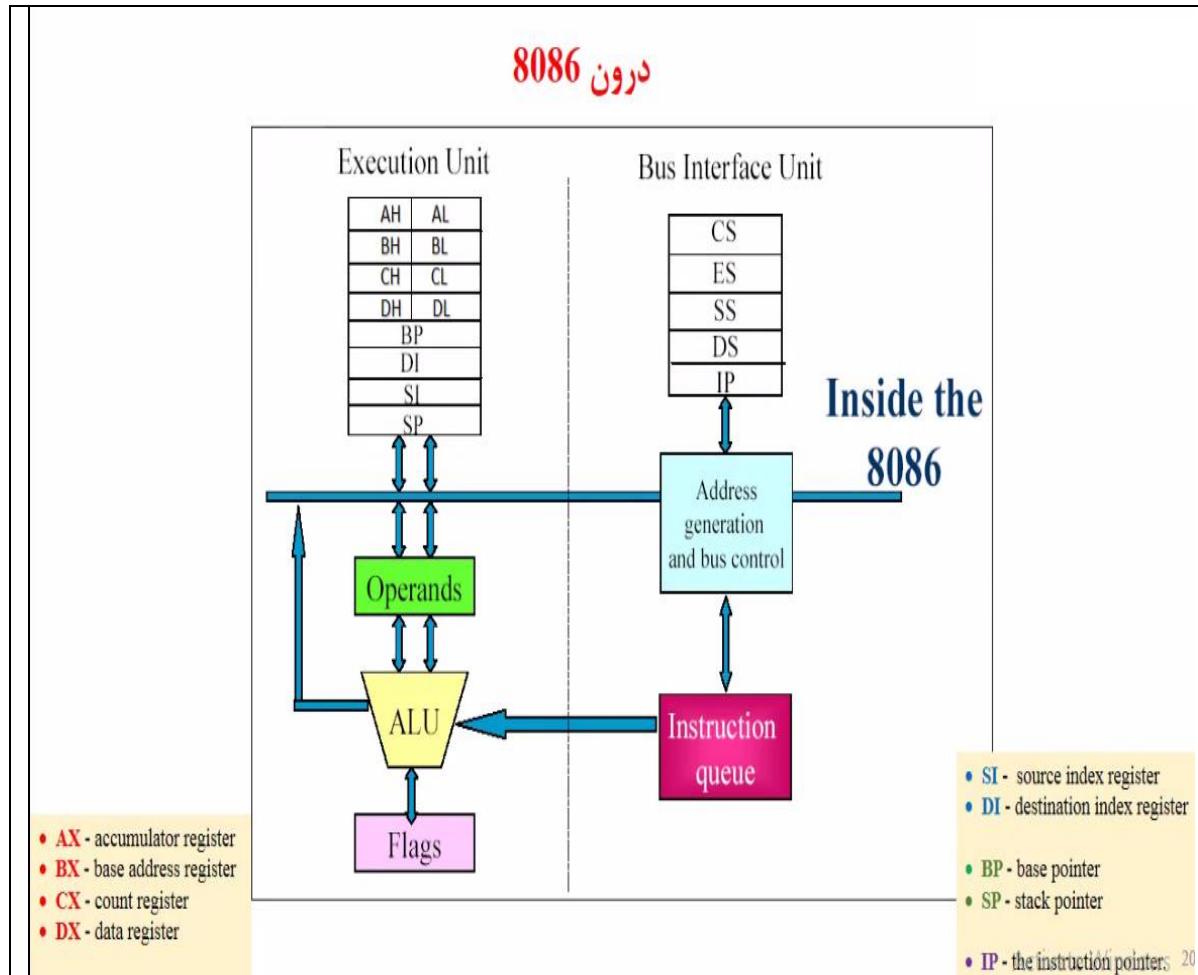
واحد BIU با سه گذرگاه داده(انتقال داده) ، گذرگاه ادرس(مشخص کردن محل مورد نظر حافظه) و گذرگاه کنترل در ارتباط است.

۸۰۸۶ یک گذرگاه داده داخلی هم دارد که پل ارتباطی بین BIU و EU است.

در ۸۰۸۶ تعدادی ثبات همه منظوره وجود دارد که ۱۶ بیتی هستند ولی به ۸ بیت هم قابل تفکیک هستند

واحد BIU: عملیات سخت افزاری هم چون تولید کردن ادرس حافظه  $O/I$  برای انتقال داده بین دنیای بیرون واحد EU و واحد CPU را انجام می‌دهد.

واحد EU: کد دستورالعملهای برنامه و نیز داده‌ها را BIU دریافت می‌کند و دستورات را اجرا می‌کند و نتایج را در ثباتهای عمومی ذخیره می‌کند. واحد EU هیچ ارتباطی با گذرگاههای سیستم ندارد و داده‌ها از طریق BIU دریافت و ارسال می‌شود.



## ثباتها

DATA REGISTER یا ثبات داده یا ثبات همگانی: از ثبات AX ثبات اکومولاتور، ثبات BX ثبات مبنا، ثبات CX ثبات شمارشگر برای استفاده در حلقه‌ها و DX ثبات داده برای اشاره به داده در عملیات ورودی و خروجی تشکیل شد است.

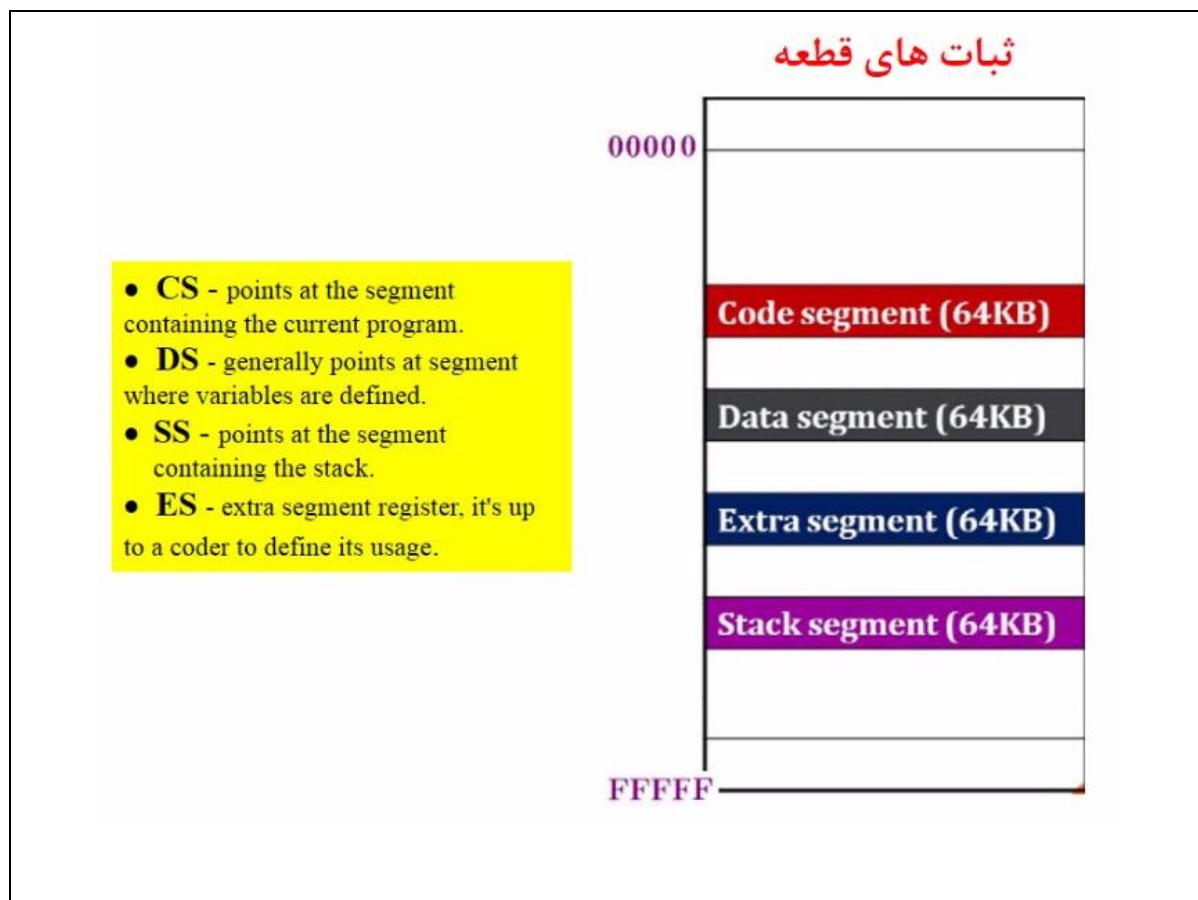
این ثباتها قابلیت تفکیک به دو دسته ۸ بیتی هستند و فقط ثباتهای گروه داده این قابلیت را دارند

$$AX=AH-AL$$

رجیسترهاي گروه اندیس: اشاره گر پشته SP ، اشاره گر مبنای BP ، اشاره گر اندیس مبدا SI ، اشاره گر اندیس مقصد DI و IP که دستوراعملی که باید اجرا شود را در خود ذخیره می کند و توسط کاربر قابل کنترل نیست از ثبات DI و SI برای انتقال بلوکی یا انتقال رشته ای در ۸۰۸۶ استفاده میشود.

رجیسترها سگمنت یا قطعه:

برای بخش بندی کردن حافظه استفاده میشود. شامل سگمنت اضافی ES ، سگمنت داده DS، سگمنت کد CS و سگمنت پشته SS است.



سگمنت یک ناحیه پیوسته از حافظه است که حداقل اندازه آن ۶۴ کیلو بایت است و ادرس شروع هر سگمنت در ثباتهای مربوطه قرار دارد. مثلا در CS ادرس شروع ناحیه ای از حافظه که کدها و دستورات در آن قرار دارد را مشخص می کند. مثلا وقتی برنامه به زبان اسمنلی مینویسید دستورات در این قطعه کد قرار میگیرد و ادرس شروع این قطعه در ثبات CS قرار داده می شود. ناحیه یا قطعه ای از حافظه که دادهها در آن قرار میگیرد سگمنت

داده نام دارد و ثبات DS به ادرس شروع این سگمنت یا قطعه اشاره می کند. وقتی از پشتۀ استفاده می کنیم یک ناحیه پشت سرهم از حافظه برای آن در نظر گرفته می شود که ثبات SS به ادرس شروع این سگمنت اشاره می کند. برای کار با رشته ها یک ناحیه پشت سرهم از حافظه در نظر گرفته شده است که ادرس شروع آن در ثبات ES قرار دارد.

CS ثباتی که به قطعه کد اشاره میکند و ادرس شروع قطعه کد در ان قرار دارد  
ES: ثباتی که به قطعه اضافی اشاره می کند و ادرس شروع این قطعه در ان قرار دارد  
SS: ثباتی که به قطعه پشتۀ اشاره می کند و ادرس شروع این قطعه در ان قرار دارد  
DS ثباتی که به قطعه داده اشاره می کند و ادرس شروع این قطعه در ان قرار دارد.

رجیسترهاي ۸۰۸۶				
Data Registers	AH	AL	AX	Accumulator
	BH	BL	BX	Base
	CH	CL	CX	Count
	DH	DL	DX	Data
Index Registers	SP		-	Stack Pointer
	BP		-	Base Pointer
	SI		-	Source Index
	DI		-	Destination Index
	IP		-	Instruction Pointer
Segment Registers	ES		-	Extra Segment
	DS		-	Data Segment
	CS		-	Code Segment
	SS		-	Stack Segment
Flags Registers	Flags H	Flags L	-	Status And Control Flags

<b>Category</b>	<b>Bits</b>	<b>Register Names</b>
General	16	AX, BX, CX, DX
	8	AH, AL, BH, BL, CH, CL, DH, DL
Pointer	16	SP (stack pointer) , BP (base pointer)
Index	16	SI (source index) , DI (destination index)
Segment	16	CS (code segment) , DS (data segment) SS (stack segment) , ES (extra segment)
Instruction	16	IP (instruction pointer)
Flag	16	FR (flag register)

**۸۰۸۶ انواع****INTEL 8086(1978)**1 MB ADDRESSABLE RAM     $2^{20}$       باس ادرس ۲۰ بیتی هست

16 bit register

16 bit data bus

صف دستور العمل ۶ بایتی

۲۹ هزار ترانزیستور دارد

سرعت آن بین ۵ تا ۱۰ مگا هرتز است

**Intel 8088(1979)**1 MB ADDRESSABLE RAM     $2^{20}$       باس ادرس ۲۰ بیتی هست

16 bit register

**8 bit data bus**

صف دستور العمل ۴ بایتی

۲۹ هزار ترانزیستور دارد

سرعت آن بین ۵ تا ۱۰ مگا هرتز است

مشکل ۸۰۸۶: باس داده ان ۱۶ بیتی بود و با لوازم جانبی ۸ بیتی نمیتوانست ارتباط برقرار کند در ۸۰۸۸ باس داده ۸ بیتی شد و این مشکل رفع شد.

تفاوت موجود میان دو ریزپردازنده ۸۰۸۶ و ۸۰۸۸ تنها در واحد BIU است از انجا که واحد EU در هر دو یکسان است دستورات برنامه نویسی برای هر دو یکسان است بنابراین برنامه های نوشته شده در ۸۰۸۶ بدون هیچ تغییری در ۸۰۸۸ قابل اجرا است.

**Intel 80286(1982)**

16 MB ADDRESSABLE RAM  $2^{24}$  باس ادرس ۲۴ بیتی هست

16 bit register

16 bit data bus

۱۳۰ هزار ترانزیستور دارد

سرعت آن حداقل ۲۰ مگا هرتز است

**Intel 386(1985)**

4 GB ADDRESSABLE RAM  $2^{32}$  باس ادرس ۳۲ بیتی هست

32 bit register

32 bit data bus

۲۷۵ هزار ترانزیستور دارد

سرعت آن حداقل ۳۳ مگا هرتز است

**INTEL 486(1989)**

4GB ADDRESSABLE RAM  $2^{32}$  باس ادرس ۳۲ بیتی هست

32 bit register

32 bit data bus

یک میلیون ترانزیستور دارد

سرعت ان حداکثر تا ۵۰ مگا هرتز است

حافظه نهان 8K cache هم اضافه شده و نوعی ram ایستا است و سرعت دستیابی به آن بسیار زیاد است.

هر چه جلوتر رفتیم گذرگاهها و ثباتها بزرگتر شدند و درنتیجه مقدار ram افزایش یافت و سرعت هم بیشتر شد.

جلسه ۱۶

ثبات flag

### پرچم‌های ۸۰۸۶ (Flags)

Flags H								Flags L							
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

ثبات flag نشان دهنده وضعیت جاری cpu است و برخی از بیتها بدون استفاده و رزرو شده هستند.

بیت cf بیت کری: اگر در عملیات محاسباتی رقم نقلی تولید شود این بیت ۱ میشود

بیت pf بیت توازن(زوج): تعداد یکها را در بایت کم ارزش میشمارد اگر تعداد یکها زوج بود این بیت ۱ میشود به دلیل وجود نویز و برای بررسی صحت نقل و انتقال اطلاعات از این flag استفاده میشود.

بیت af بیت نقلی کمکی: در واقع کری از بیت شماره سوم به بیت شماره چهارم است اگر نقلی به وجود آمد این بیت ۱ میشود.

بیت صفر، پرچم صفر یا ZF: اگر نتیجه محاسبه ما صفر باشد این بیت ۱ میشود.

بیت علامت یا SF: متناسب با مقدار بیت پرارزش این FLAG مقدار میگیرد.

بیت TF اگر این بیت ۱ شود. برنامه به صورت مرحله به مرحله انجام میشود

بیت IF یا وقفه: اگر این بیت 1 شود وقفه ها پذیرفته میشوند.

بیت جهت یا DF: در بحث رشته ها کاربرد دارد و جهت انتقال دیتا را مشخص می کند.

بیت سرریز یا OF: اگر در نتیجه محاسبات علامتدار عدد در رنج قابل نمایش نگنجد این بیت 1 میشود

مثال:

$$\begin{array}{r}
 0101010000111001 + \\
 0100010101101010 \\
 \hline
 1001100110100011
 \end{array}$$

ZF=0

تمام بیت های نتیجه 0 نیستند.

CF=0

در بیت شماره 15 carry وجود ندارد.

SF=1

نتیجه برابر با یک می باشد.

AF=1

در بیت شماره 3 یک carry وجود دارد.

PF=1

تعداد بیت های یک در بایت مرتبه پائین نتیجه زوج است.

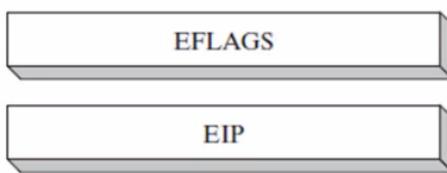
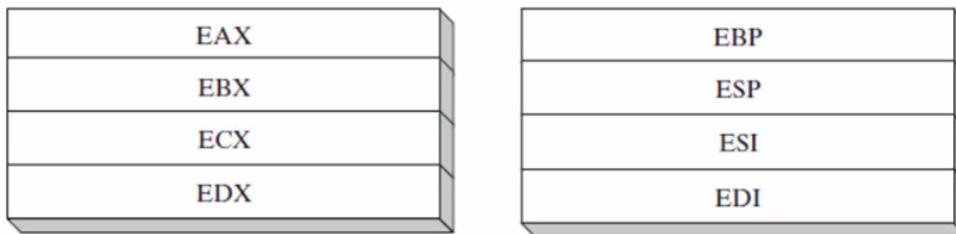
وضعیت پرچم‌ها بعد از اجرای دستور ADD AL, 01 با فرض AL=7FH

AL = 80H	$7FH + 1 = 80H$
CF = 0	در اثر عملیات جمع، بیت نقلی به وجود نیامده است.
PF = 0	عدد 80H تعداد فردی از بیت 1 دارد.
AF = 1	یک بیت نقلی از بیت 3 به بیت 4 وارد شده است (شمارش بیتها از بیت صفر شروع می‌شود).
ZF = 0	نتیجه صفر نیست.
SF = 1	مقدار بیت هفتم، 1 است.
OF = 1	اگر نتیجه را با ۱۲۸ جمع کنیم، از محدوده علامتدار ثبات AL خارج می‌شود.

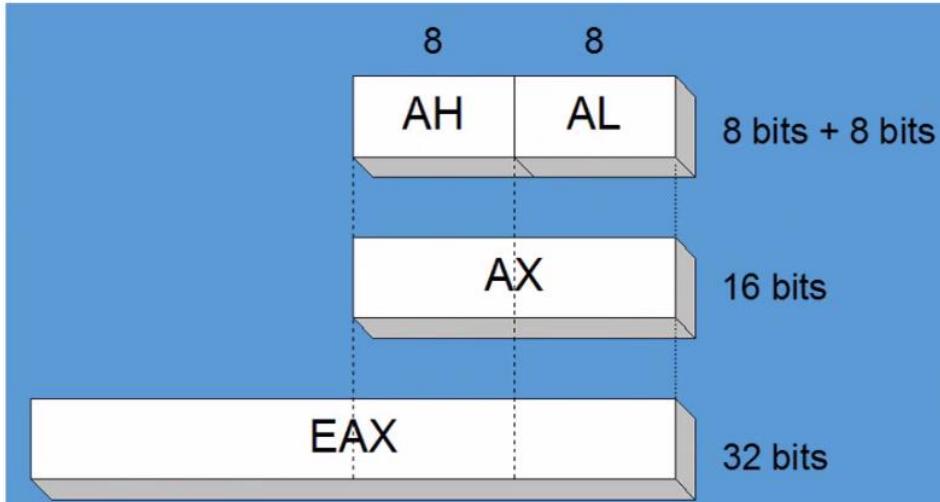
### تفاوت overflow و carry

$  \begin{array}{r}  +9 \\  +12 \\  \hline  \end{array}  $	$  \begin{array}{r}  100 \\  110 \\  \hline  0101  \end{array}  $	$  \begin{array}{r}  +5 \\  -6 \\  \hline  010  \end{array}  $	$  \begin{array}{r}  1010 \\  111 \\  \hline  0011  \end{array}  $	$  \begin{array}{r}  +8 \\  -3 \\  \hline  0010  \end{array}  $	$  \begin{array}{r}  1101 \\  110 \\  \hline  110  \end{array}  $
	$  \text{carry} \rightarrow 1  $	$  \text{C} = 0  $	$  \text{C} = 0  $	$  \text{C} = 1  $	$  \text{C} = 1 \\  \text{O} = 0  $

### ثبتات های ۳۲ بیتی



ثبتات های سگمنت تغییری نکردن و هم چنان ۱۶ بیتی هستند.



## رجیسترهاي Pentium

EAX	31	16	15	0	AX Accumulator
EBX			AH	AL	BX Base
ECX			BH	BL	CX Count
EDX			CH	CL	DX Data
EBP			DH	DL	Base Pointer
ESI			<b>BP</b>		Source Index
EDI			<b>SI</b>		Destination Index
			<b>DI</b>		
EIP	31	16	15	0	Instruction Pointer
ESP			<b>IP</b>		Stack Pointer
EFLAGS			<b>SP</b>		Flags
			<b>FLAGS</b>		
	15		0		
		CS			Code Segment
		DS			Data Segment
		SS			Stack Segment
		ES			Extra Segment
		FS			Extra Segment
		GS			Extra Segment

مقایسه ۸۰۸۶ و پنتیوم

:۸۰۸۶

ثباتها ۱۶ بیتی

خطوط ادرس ۲۰ بیتی

۱ مگا بايت از حافظه قابلیت ادرس دهی دارد

پنتیوم:

ثباتها ۳۲ بیتی

خطوط ادرس ۳۲ بیتی

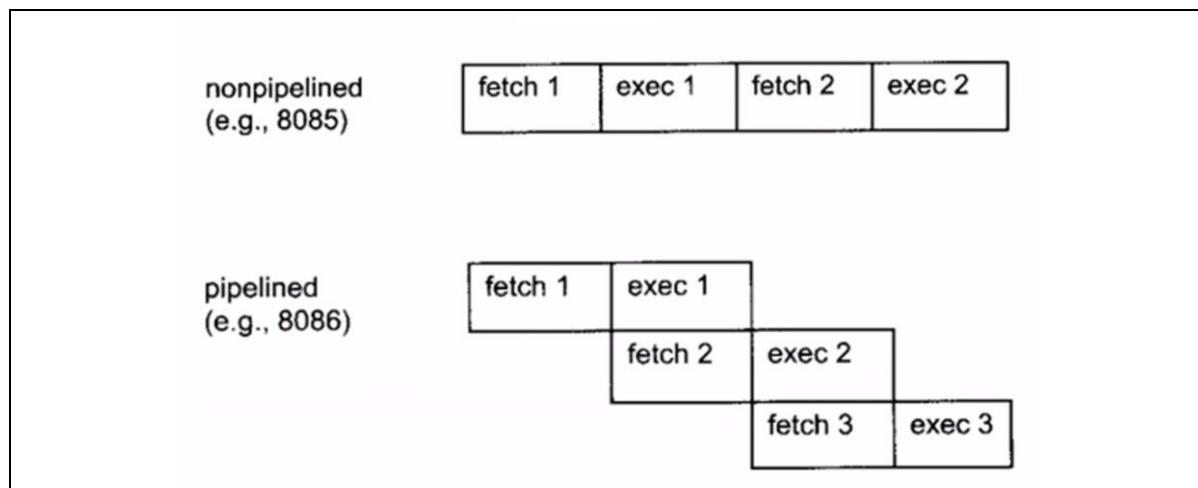
۴ گیگا بايت از حافظه را ادرس دهی میکند.

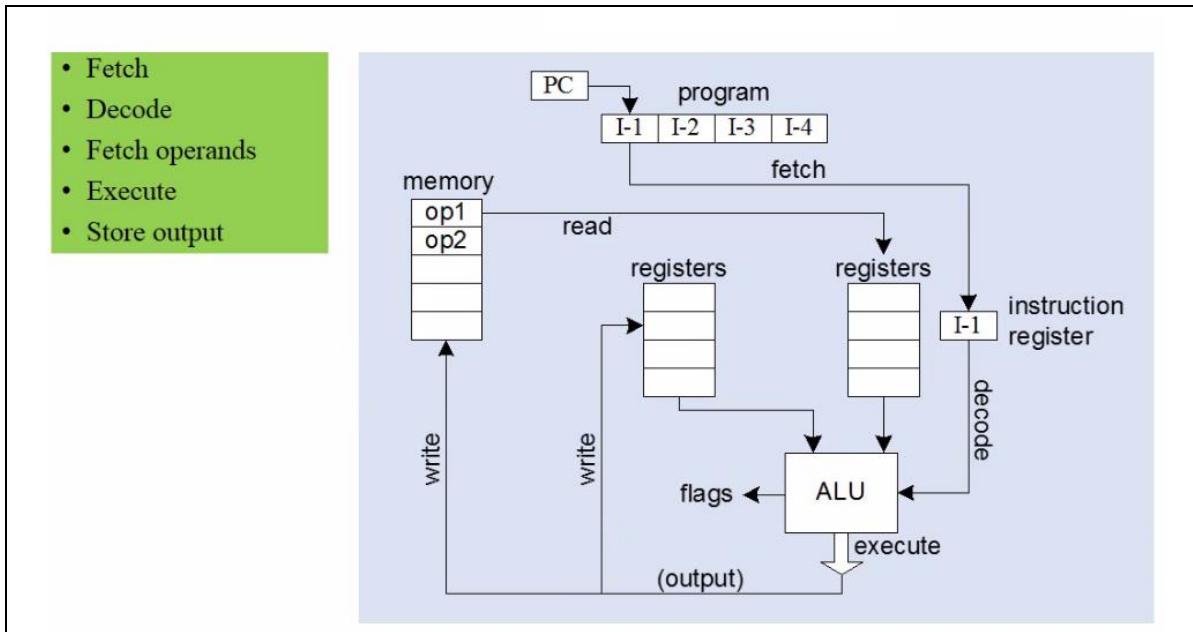
### نحوه تولید آدرس در ریز پردازنده

Pentium	۸۰۸۶
$Physical\ Add. = Reg1 \times 2^n + Reg2 + \text{Constant}$	$Physical\ Add. = SReg \times 2^4 + 0H$
• $n = 1, 2, 4, 8$	آدرس اصلی      آدرس فرعی
• جابجایی = 8 Bit و 32 Bit و هیچکدام	(مثال) DS=0700H , SI=1234H $\rightarrow$ DS:SI
• Reg1 = EAX, EBX, ECX, EDX, EBP, ESI, ESP	$\frac{0700H \times 2^4}{07000H} + 1234H = 08234H$
• Reg2 = EAX, EBX, ECX, EDX, EBP, ESI, ESP, EDI	۲۰ بیت

جلسه ۱۷

خط لوله : pipeline





برنامه نويسی اسمبلي ۸۰۸۶ با نرم افزار emu8086

### مجموعه دستورات زبان اسمبلي

- ۱- دستورات جابه جایی
- ۲- دستورات منطقی
- ۳- دستورات ریاضی
- ۴- دستورات انتقال کنترل
- ۵- دستورات کنترل ریزپردازنده.

Opcode operand1,operand2 اختیاری است) (بسته به نوع دستورات

Name sudoinstruction value

X DB(define byte) 35h

Y DW 05h,40h

U DD 45H,89H,67,45 DOUBLE WORD(32 BIT)

E DQ 56H,90H,34H,..... DIFINE quad word (64 bit)

دستورات جابه جایی:

**mov** مبدأ, مقصد

reg memory

memory reg

reg reg

reg , immediate

s reg ,memory

memory , s reg

reg , s reg

s reg , reg

نکته با دستور **mov** نمیتوان مقداری را درون ثبات ip و CS ریخت

و هر دو اپرند **mov** نمی تواند s reg باشد باید از یک ثبات واسط استفاده کرد

**Mov reg,sreg1**

**Mov sreg2, reg**

دستور **MOV** هیچ FLAG را تغییر نمی دهد.

جلسه ۱۸

**XCHG**

Reg memory

Memory reg

Reg reg

مقادیر دو عملوند را با هم جابه جا می کند.

هر دو عملوند باید هم سایز باشند.

مثال:

MOV AL,02H

MOV AH,05H

XCHG AL,AH

LAHF

این دستور عملوندی ندارد و ۸ بیت کم ارزش ثبات FLAG را درون ثبات AH میریزد

SAHF

این دستور عملوندی ندارد و محتویات ثبات AH را درون ۸ بیت کم ارزشتر ثبات FLAG می‌ریزد.

LEA

Reg memory

این دستور دو عملوندی هست و ادرس متغیر را درون ثبات می‌ریزد

این دستور معادل دستور زیر است

X DW 35

LEA BX , X

یا

MOV BX, OFFSET X

دستورات منطقی

AND

Reg memory

Memory reg

Reg reg

Reg immediate

Memory immediate

بین همه بیتها عملیات and منطقی را انجام می دهد.

TEST

Reg memory

Memory reg

Reg reg

Reg immediate

Memory immediate

همان عمل and منطقی را انجام می دهد فقط بر روی برخی از flagها تاثیر می گذارد مانند CF ZF PF

OR

Reg memory

Memory reg

Reg reg

Reg immediate

Memory immediate

XOR

Reg memory

Memory reg

Reg reg

Reg immediate

Memory immediate

مثال:

MOV AL, 00000111B

XOR AL, 00000010B

NOT

REG

MEMORY

:19 جلسه

دستورات ریاضی

ADD

REG MEMORY

MEMORY REG

REG REG

REG IMMEDIATE

MEMORY IMMEDIATE

دو عملوندی است و عملوند اول را با عملوند دوم جمع کرده و حاصل را در عملوند اول قرار می دهد.

ADC

REG MEMORY

MEMORY REG

REG REG

REG IMMEDIATE

**MEMORY IMMEDIATE**

دو عملوند اول را با عملوند دوم و CF جمع کرده و حاصل درون عملوند اول قرار میگیرد. از این دستور معمولا برای حاصل جمع بایتهای پارازشتر با CF که نتیجه حاصل جمع بایتهای کم ارزشتر است استفاده میشود.

SUB

REG MEMORY

MEMORY REG

REG REG

REG IMMEDIATE

MEMORY IMMEDIATE

دو عملوندی هست و عملوند دوم را از عملوند اول کم می کند و حاصل درون عملوند اول قرار میگیرد.

$$\text{OP1} = \text{OP1} - \text{OP2}$$

SBB

REG MEMORY

MEMORY REG

REG REG

REG IMMEDIATE

MEMORY IMMEDIATE

دو عملوندی است

$$\text{OP1} = \text{OP1} - \text{OP2} - \text{CF}$$

INC

REG

MEMORY

تک عملوندی است و یک واحد به عملوند اضافه خواهد شد.

DEC

REG

MEMORY

تک عملوندی است و یک واحد از عملوند کم می کند.

NEG

REG

MEMORY

تک عملوندی است و مانند مکمل دو عمل می کند.

CMP

REG MEMORY

MEMORY REG

REG REG

REG IMMEDIATE

MEMORY IMMEDIATE

دو عملوندی است OP1-OP2 محاسبه می کند حاصل را در جایی نمیریزد و فقط برخی از ثباتهای FLAG را تحت تاثیر قرار می دهد.

MUL

REG

MEMORY

تک عملوندی است و برای حاصل ضرب اعداد بدون علامت استفاده میشود.

برای عملوند ۸ بیتی

$AX=AL*OP$

برای عملوند ۱۶ بیتی یا یک WORD

$DX\ AX=AX*OP$

IMUL

REG

MEMORY

تک عملوندی است و برای حاصل ضرب علامتدار استفاده میشود.

برای عملوند ۸ بیتی

$AX=AL*OP$

برای عملوند ۱۶ بیتی یا یک WORD

$DX\ AX=AX*OP$

DIV

REG

MEMORY

تک عملوندی است و برای حاصل تقسیم بدون علامت استفاده میشود

برای عملوند ۸ بیتی

$AL=AX/OP$

$AH=REMAINDER$

برای عملوند ۱۶ بیتی یا یک WORD

$AX=DX\ AX /OP$

$DX=RAMAINDER$

IDIV

REG

MEMORY

تک عملوندی است و حاصل تقسیم علامت دار را محاسبه می کند

برای عملوند ۸ بیتی

AL=AX/OP

AH=REMAINDER

برای عملوند ۱۶ بیتی یا یک WORD

AX=DX AX /OP

DX=RAMAINDER

دستورات انتقالی کنترل

JMP LABEL

دستور پرش بدون شرط است و کنترل برنامه را به آدرس برچسب می برد

JCXZ LABEL

این دستور یک پرش کوتاه است و اگر مقدار ثبات CX برابر با صفر بود به آدرس برچسب پرش انجام میشود.

جلسه ۲۰

JG LABEL

یک دستور پرش کوتاه است که اگر عملوند اول از عملوند دوم بزرگتر باشد (این دستور تحت تاثیر دستور CMP) است عمل پرش به برچسب انجام میشود یا

اگر ZF=0 و SF=OF عمل پرش کوتاه به برچسب انجام میشود.

**JGE LABEL**

یک دستور پرش کوتاه است اگر عملوند اول از عملوند دوم بزرگتر یا مساوی باشد(تحت تاثیر دستور **CMP**) عمل پرش به برچسب مورد نظر انجام میشود یا اگر  $SF=OF$  عمل پرش به برچسب انجام میشود.

**JL LABEL**

یک دستور پرش کوتاه است اگر عملوند اول از عملوند دوم کوچکتر باشد(تحت تاثیر دستور **CMP**) عمل پرش به برچسب مورد نظر انجام میشود یا اگر  $SF!=OF$  عمل پرش به برچسب انجام میشود.

**JLE LABEL**

یک دستور پرش کوتاه است اگر عملوند اول از عملوند دوم کوچکتر یا مساوی باشد(تحت تاثیر دستور **CMP**) عمل پرش به برچسب مورد نظر انجام میشود یا اگر  $ZF=1$  یا  $SF=OF$  عمل پرش به برچسب انجام میشود.

**JO LABEL**

یک دستور پرش کوتاه است اگر مقدار **OF** برابر یک باشد عمل پرش به برچسب مورد نظر انجام میشود.

**JNO LABEL**

یک دستور پرش کوتاه است اگر مقدار **OF** برابر صفر باشد عمل پرش به برچسب مورد نظر انجام میشود

**JS LABEL**

یک دستور پرش کوتاه است اگر مقدار SF برابر یک باشد عمل پرش به برچسب مورد نظر انجام میشود این دستور تحت تاثیر دستوراتی مانند ADD SUB TEST AND OR XOR CMP می باشد.

### JNS LABEL

یک دستور پرش کوتاه است اگر مقدار SF برابر صفر باشد عمل پرش به برچسب مورد نظر انجام میشود.

دستورات کنترل ریزپردازنده

### STC

مقدار CF را برابر ۱ می کند.

### CLC

مقدار CF را پاک کرده و صفر می کند.

### CMC

مقدار CF را مکمل می کند اگر CF=0 باشد ان را یک می کند و اگر CF=1 باشد ان را صفر می کند.

### STD

مقدار DF را برابر یک میکند دو ثبات SI , DI کاهش پیدا میکنند.

### CLD

مقدار DF را برابر صفر می کند. دو ثبات SI و DI را افزایش می دهد.

### STI

مقدار IF فلگ وقفه را برابر یک می کند در واقع وقفه ریزپردازنده را فعال می کند.

### CLI

مقدار IF یا فلگ وقفه را برابر صفر می کند.

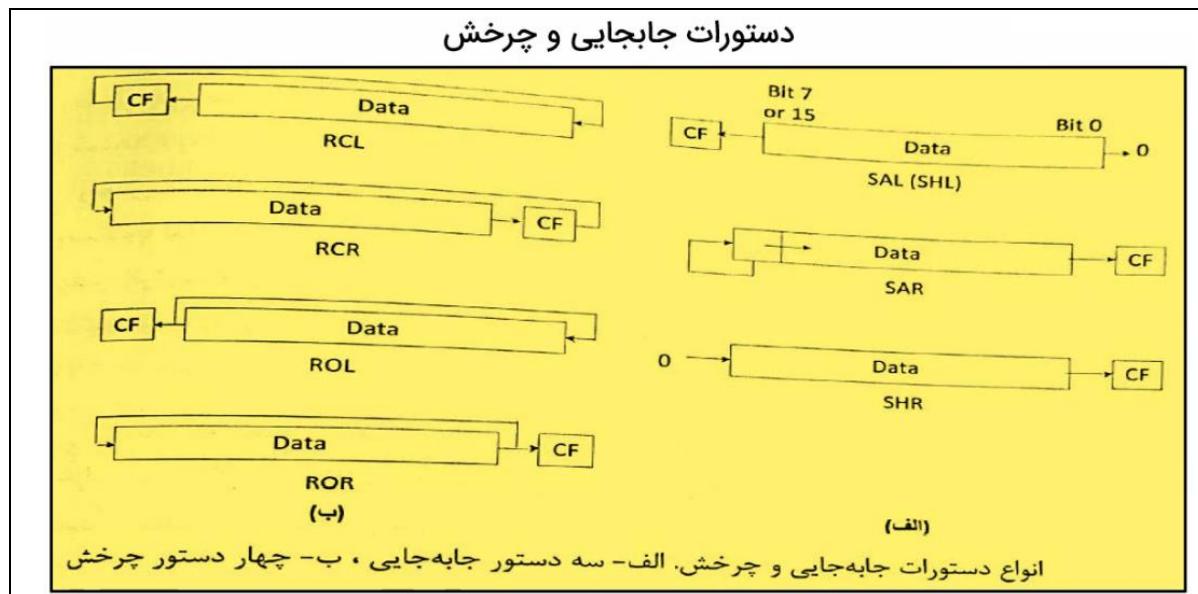
**HLT**

این دستور سیستم را متوقف میکند و کار به اتمام می‌رسد.

**NOP**

این دستور NO OPERATION است و هیچ عملی صورت نمی‌گیرد.

دستورات جابه جایی و چرخش



دستورات جابه جایی

**SAL(SHL)****REG IMMEDIATE****MEMORY IMMEDIATE****MEMORY CL****REG CL**

این دستور دو عملوند است عملوند اول مقداری که میخواهید جایه جایی روی ان انجام شود و عملوند دوم تعداد دفعات جایه جایی را بیان می کند.

در این دستور بیتها به سمت چپ جایه جا میشود بیت خارج شده از سمت چپ وارد CF میشود و بیت صفر از سمت راست وارد عدد میشود.

SAR

REG IMMEDIATE

MEMORY IMMEDIATE

MEMORY CL

REG CL

در این دستور بیتها به سمت راست جایه جا میشود بیت خارج شده از سمت راست وارد CF میشود و بیت علامت از سمت چپ وارد عدد میشود.

SHR

REG IMMEDIATE

MEMORY IMMEDIATE

MEMORY CL

REG CL

در این دستور بیتها به سمت راست جایه جا میشود بیت خارج شده از سمت راست وارد CF میشود و بیت صفر از سمت چپ وارد عدد میشود.

دستورات چرخشی

RCL

REG IMMEDIATE

MEMORY IMMEDIATE

MEMORY CL

REG CL

در این دستور بیتها به سمت چپ جایه جا میشود بیت خارج شده از سمت چپ وارد CF میشود و مقدار قبلی درون CF به سمت راست عدد وارد میشود.

RCR

REG IMMEDIATE

MEMORY IMMEDIATE

MEMORY CL

REG CL

در این دستور بیتها به سمت راست جایه جا میشود بیت خارج شده از سمت راست وارد CF میشود و مقدار قبلی درون CF به سمت چپ عدد وارد میشود.

ROL

REG IMMEDIATE

MEMORY IMMEDIATE

MEMORY CL

REG CL

در این دستور بیتها به سمت چپ جایه جا میشود بیت خارج شده از سمت چپ وارد CF میشود و بیت خارج شده به سمت راست عدد وارد میشود.

ROR

REG IMMEDIATE

MEMORY IMMEDIATE

**MEMORY CL****REG CL**

در این دستور بیتها به سمت راست جابه جا میشود بیت خارج شده از سمت راست وارد CF میشود و بیت خارج شده به سمت چپ عدد وارد میشود.

**:۲۱ جلسه:**

با ۸۰۸۶ برنامه ای بنویسید که N عدد متوالی را با هم جمع کنید

ORG 100H

MOV AX,05H ;

MOV DX,00H ;

LOP:ADD DX,AX ; DX=AX+DX

SUB AX,01H ; DEC AX : AX=AX-1

CMP AX,01H ; AX>=<1

JGE LOP ; Jump Greater Or Equal

Ret

برنامه ای که دو عدد را مقایسه کند و به صورت صعودی در حافظه ذخیره کند.

org 100h

MOV AX,05H

MOV BX,04H

CMP AX,BX

JG Label1 ; Jump Greater

MOV [1000H],AX

MOV [1001H],BX

JMP E1

Label1:MOV [1000H],BX

MOV [1001H],AX

E1: MOV DL,[1000H]

MOV DH,[1001H]

Ret

برنامه ای که ۱۰ عدد را از ورودی دریافت کنید و حاصل جمع این ۱۰ عدد را محاسبه و در حافظه ذخیره کنید

org 100h

Num DW 12H,10H,09H,25H,32H,5FH,0AAH,07H,23H,6AH

SUM DW 00H

MOV CX,0AH

MOV BP,00H

LOP:MOV AX,NUM[BP]

ADD SUM,AX

DEC CX

JCXZ END

INC BP

INC BP

JMP LOP

END:MOV DX,SUM

HLT

Ret

نکته: INC BP فقط به اندازه یک بایت اضافه میشود ولی در اینجا کلمات از نوع WORD هستند پس باید دوبار عمل اضافه کردن BP انجام شود.

## جلسه ۲۲

دستورات میکروکنترلر: ۸۰۵۱

دستورات محاسباتی:

۱ - دستور ADD

 $OP1 = \text{مقدار}$  $OP2 = \text{مبدا}$ ADD OP1,OP2 ; $OP1 = OP1 + OP2$ 

مثال:

ADD A,7FH      ادرس دهی مستقیم

ADD A , @R0      ادرس دهی غیر مستقیم

ADD A ,R7      ادرس دهی ثبات

ADD A,#35H      ادرس دهی بلا فاصله

حاصل درون عملوند اول ریخته میشود.

۲ - دستور SUB

SUB OP1,OP2 ; $OP1 = OP1 - OP2$ 

۳ - دستور ADDC

دستور جمع با نقلی یا CARRY

ADDC OP1,OP2 ; $OP1 = OP1 + OP2 + C$ 

برای جمع دو بایت بالارزش دو عدد ۱۶ بیتی مورد استفاده قرار میگیرد.

۴ - دستور SUBB

## دستور تفریق با قرض

SUBB OP1,OP2 ; OP1=OP1-OP2-C

نکته مهم: برای دستورات جمع و تفریق یکی از عملوندها باید اکومولاتور A باشد.

## ۵- دستور افزایش INC

INC A ; A=A+1  
MOV R3,#05H  
INC R3 ; R3=06H

## ۶- دستور کاهش DEC

DEC A ; A=A-1  
MOV R3,#05H  
DEC R3 ; R3=04H

## ۷- دستور ضرب MUL

MUL AB

عدد ثبات A در عدد ثبات B ضرب شده یک عدد ۱۶ بیتی به دست می آید و بایت پرارزش در ثبات B و بایت کم ارزش در ثبات A ذخیره میشود.

## ۸- دستور تقسیم DIV

DIV AB

محتوای اکومولاتور A بر محتوای ثبات یا اکومولاتور B تقسیم میشود و خارج قسمت در ثبات A و باقی مانده در ثبات B ذخیره میشود.

مثال:

محتوای ثبات A برابر  $55H$ ، ثبات B برابر  $22H$  و ثبات وضعیت برابر  $00H$  است محتوای ثباتها بعد از اجرای

دستور **MUL AB** چیست

حل:

$$A=55H=85$$

$$B=22H=34$$

$$85 \times 34 = 2890 = 0B4AH$$

$$B=0BH$$

$$A=4AH$$

چون در جواب، ۴ بیت ۱ وجود دارد بیت توازن یک میشود چون توازن زوج است.

چون نتیجه محاسبات بزرگتر از ۲۵۵ است سرریز رخ داده و بیت OV هم یک میشود.

ثبات وضعیت (PSW):

جدول (۳-۲) خلاصه‌ای از بیت‌های ثبات وضعیت (PSW)

بیت	سمبل	ادرس	توضیح بیت
PSW.7	CY یا C	D7H	بیت پرچم نقلی <sup>۱</sup>
PSW.6	AC	D6H	بیت پرچم نقلی کمکی <sup>۲</sup>
PSW.5	F0	D5H	پرچم <sup>۳</sup>
PSW.4	RS1	D4H	انتخاب 1 بانک ثبات
PSW.3	RS0	D3H	انتخاب 0 بانک ثبات
			بانک 0 = 00: از آدرس 00H تا 07H
			بانک 1 = 01: از آدرس 08H تا 0FH
			بانک 2 = 10: از آدرس 10H تا 17H
			بانک 3 = 11: از آدرس 18H تا 1FH
PSW.2	OV	D2H	بیت پرچم سرریز
PSW.1	--	D1H	رزرو شده
PSW.0	P	D0H	بیت پرچم توازن زوج <sup>۴</sup>

پنجمین بیت وضعیت PSW برابر مقدار  $00000101B$  میشود. فقط بیت توازن و سرریز مقدار ۱ دارند.

**۹ - دستور جمع برای اعداد BCD**

چون اعداد BCD در محدوده ۰ تا ۹ قرار دارند در جمع دو عدد BCD حاصل باید عددی در همین محدوده ۰-۹ باقی بماند برای همین بعد از دستور ADDC یا ADD باید از دستور DA استفاده کرد تا عدد حاصل در محدوده ۰-۹ قرار گیرد.

MOV A,#37H

MOV B,#46H

ADD A,B ; A=A+B=7DH

DA A ; A=83H              7DH+06H=83H

نحوه عملکرد دستور DA

**روش دیگر در جمع اعداد BCD**

اگر حاصل جمع کوچکتر از ۱۰ باشد حاصل برابر خود جمع می شود

$$02 + 04 = 06 \quad 0000\ 0010 + 0000\ 0100 = 0000\ 0110$$

اگر حاصل جمع بین ۱۰ تا ۱۹ باشد حاصل با عدد ۶ جمع می شود :

$$09 + 04 = D \quad 0000\ 1001 + 0000\ 0100 = 0000\ 1101$$

$$D + 6 = 13H \quad 0000\ 1101 + 0000\ 0110 = 0001\ 0011$$

اگر حاصل جمع بین ۲۰ تا ۲۹ باشد حاصل با عدد ۱۲ جمع می شود

اگر حاصل جمع بین ۳۰ تا ۳۹ باشد حاصل با عدد ۱۸ جمع می شود و ...

-۱۰

دستور صفر کردن بیت یا پرچم نقلی C

CLR C

-۱۱

دستور یک کردن بیت یا پرچم نقلی

SETB C

-۱۲

دستور مکمل کردن بیت یا پرچم نقلی

CPL C

**دستورات منطقی****۱ - دستور AND منطقی**

**ANL OP1,OP2 ; OP1=OP1 AND OP2**

**ANL A,7FH**      ادرس دهی مستقیم

**ANL A , @R0**      ادرس دهی غیر مستقیم

**ANL A ,R7**      ادرس دهی ثابت

**ANL A,#35H**      ادرس دهی بلافضله

در مبدا یا OP2 می توان عدد ثابت یا ثبات یا خانه حافظه بکار برد.

در مقصد ثبات یا خانه حافظه قرار می گیرد.

**MOV A,#00001110B**

**MOV B,#10100111B**

**ANL A,B**

**A=00000110B**

**۲ - دستور OR منطقی:**

**ORL OP1,OP2**

نکته: از دستور ANL برای صفر کردن بیت‌های دلخواه بدون تغییر بقیه بیت‌ها استفاده می‌شود.

نکته: از دستور ORL برای یک کردن بیت‌های دلخواه بدون تغییر بقیه بیت‌ها استفاده می‌شود.

**۳ - دستور XOR**

**XRL OP1,OP2**

**نکته:** یکی از کاربردهای XOR معکوس کردن بیتها است. از این دستور برای مکمل کردن بیتها دلخواه بدون تغییر بقیه بیتها استفاده میشود.

**نکته:**

XRL A, #0000000B ; A=A

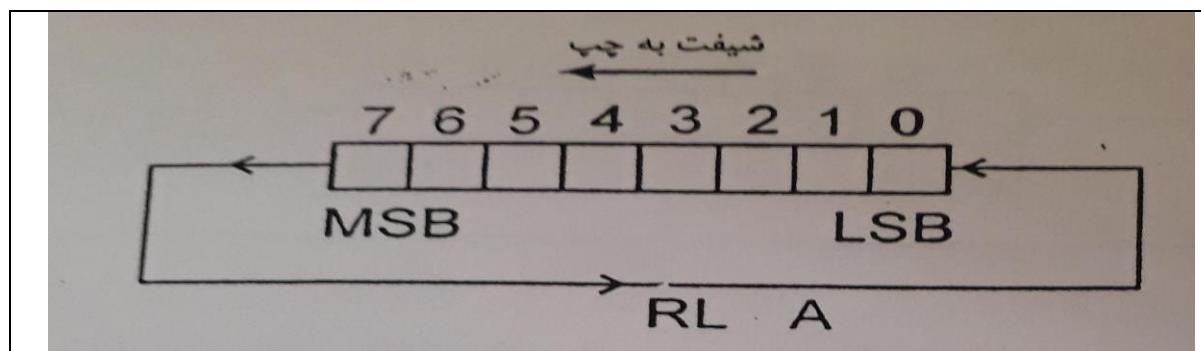
XRL A, #1111111B ;A=A'

دستورات چرخش:

۱- دستور چرخش ثبات یا اکومولاتور A به چپ

RL A

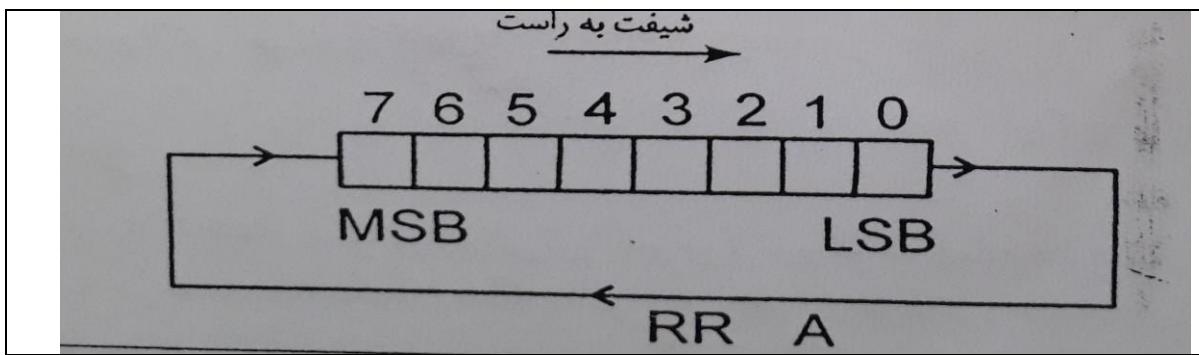
این دستور محتوای ثبات A را یک بیت به چپ چرخش می دهد، یعنی محتوای ثبات A را یک بیت به چپ شیفت می دهد و پارزشترین بیت MSB را به کم ارزشترین بیت LSB منتقل می کند.



۲- دستور چرخش ثبات A به راست

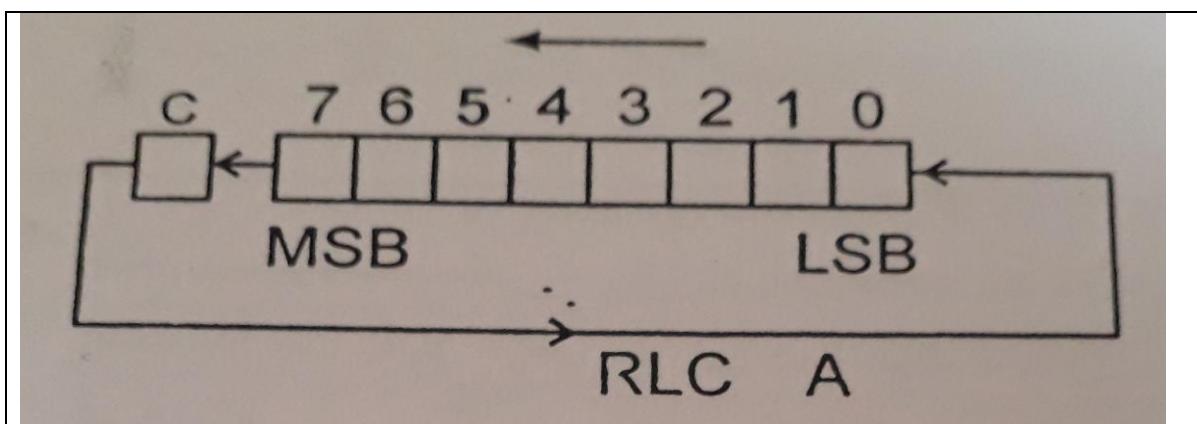
RR A

این دستور محتوای ثبات A را یک بیت به راست چرخش می دهد، یعنی محتوای ثبات A را یک بیت به راست شیفت می دهد و کم ارزشترین بیت را به پر ارزشترین بیت منتقل می کند.



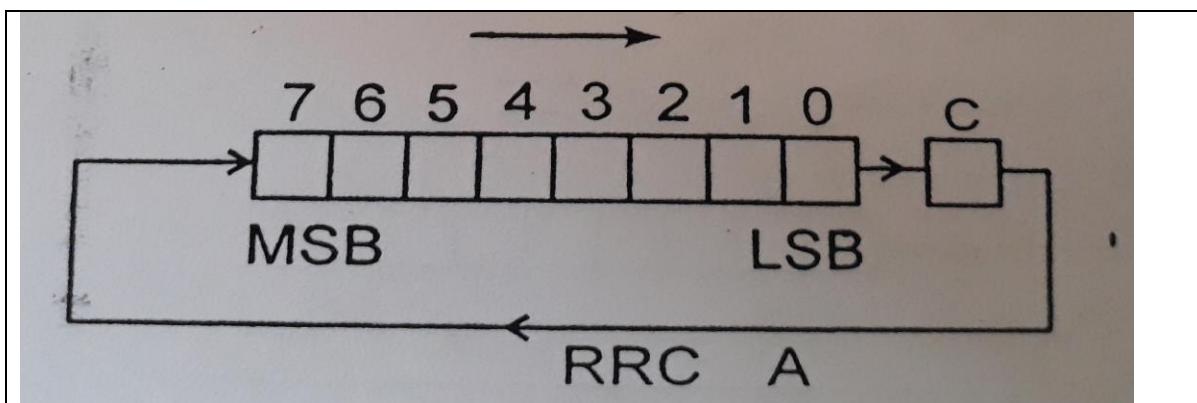
۳- دستور چرخش ثبات A به چپ از طریق بیت نقلی C

RCL A



۴- دستور چرخش ثبات A به راست از طریق بیت نقلی C

RRCA



## ۵- دستور SWAP A

چهار بیت کم ارزشتر را با چهار بیت پر ارزشتر ثبات A جایه جا می کند.

نکته: دستورات چرخش و دستور SWAP فقط روی ثبات یا اکومولاتور A کار می کند

سوال: برای چرخش محتوای ثبات A سه بیت به چه کار کنیم؟ روش را بگویید و مزایا و معایب هر کدام را بگویید؟

روش اول:

RL A

RL A

RL A

روش دوم:

SWAP A

RR A

دستورات انتقال اطلاعات:

۱- دستورات انتقال اطلاعات حافظه داده داخلی RAM

مبدا، مقصد MOV

محتوای مبدا را درون مقصد قرار می دهد و مقصد می تواند ثبات یا حافظه باشد.

مبدا می تواند حافظه، ثبات یا عدد ثابت باشد.

نکته: ثباتهای موجود در میکروکنترلر ۸۰۵۱

## ثبات A و B

ثبات R0 تا R7 از بانک ثباتها

## ۲- دستور ۱۶ بیتی انتقال اطلاعات MOV

برای مقدار اولیه دادن به اشاره گر DPTR به صورت زیر انجام میشود

MOV DPTR, #DATA 16

MOV DPTR ,#1234H

ثبات اشاره گر داده DPTR برای دسترسی به حافظه داده خارجی یا جداول حافظه استفاده میشود.

## ۳- دستور XCH

XCH A , SOURCE

این دستور محتویات SOURCE و ثبات A را جایه جا می کند

XCH A,20H

محتوای خانه 20H را با محتوای ثبات A جایه جا می کند.

## ۴- دستور XCHD

XCHD A, @Ri

این دستور باعث میشود که ۴ بیت کم ارزشتر خانه حافظه ای که با Ri نشان داده شده است، با چهار بیت کم ارزشتر ثبات A جایه جا شود.

نکته: چون دستور MOV برای ثبات B وجود ندارد. هشت استفاده از B می توان از آدرس حافظه این ثبات

استفاده کرد

MOV B, #12H      غلط است

MOV OFOH , #12H

## ۵- دستور حافظه داده خارجی RAM

دستور انتقال اطلاعات که داده را با حافظه خارجی مبادله می کند. مانند **MOVX** از آدرس دهی غیر مستقیم استفاده می کند.

آدرس دهی غیر مستقیم:

یا یک بایت ادرس از طریق **Ri** برابر **R0** یا **R1** است) استفاده می کند.

و یا دو بایت ادرس اشاره گر داده را به صورت **@DPTR** به کار می برد.

فقط با دستورات **MOVX** می توان به حافظه داده خارجی **RAM** دسترسی پیدا کردو در این صورت یکی از عملوندها حتما ثبات A است.

**MOVX A, @DPTR**

مثال : دستوراتی بنویسید که محتوای خانه های حافظه خارجی به ادرس های **10F4H** و **10F5H** را بخواند و آنها را به ترتیب در ثباتهای **R6** و **R7** قرار دهد؟

حل:

**MOV DPTR ,#10F4H**

**MOVX A,@DPTR**

**MOV R6,A**

**INC DPTR**

**MOVX A,@DPTR**

**MOV R7,A**

## ۶- دستورات جستجوی جداول در حافظه

جداول حافظه مانند یک رشته داده در حافظه برنامه می باشند آنها فقط خوانده میشوند و نوشته نمیشوند یا تغییر نمی کنند. دو فرم دستور انتقال برای خواندن جداول و رشته در حافظه وجود دارد

الف) دستور **MOVC** با اشاره گر داده **DPTR**

این دستور اشاره گر داده D PTR را به عنوان ثبات پایه استفاده می کند و ثبات A به عنوان افست یا فاصله نسبی به کارمی رو د.

**MOVC A, @A+D PTR**

=ابتدای جدول یا رشته اطلاعات

ب) دستور MOVC با کنتور برنامه PC

در این دستور از PC به عنوان ثبات پایه و ثبات A به عنوان افست استفاده می شود.

**MOVC A , @A+PC**

نکته: با دستور MOVC می توان به هر جدول یا رشته اطلاعات در حافظه ROM داخل تراشه میکروکنترلر دسترسی پیدا کرد.

دستورات روی بیت:

حافظه داخلی RAM دارای ۱۲۸ بیت ادرس پذیر(بیتهاي با ادرس ۰۰H تا ۷F H) است. ثباتهاي کاربرد خاص نيز داراي ۱۲۸ بیت حافظه ادرس پذير هستند(بیتهاي با ادرس ۸۰H تا F7H). تمام خطهاي پورتها نيز ادرس پذير هستند و هر يك می توانند به عنوان يك بیت مجزا تلقی می شوند.

۱- يك کردن بیت

**SETB P1.7**

**SETB C**

**SETB 06H**

۲- صفر کردن بیت

**CLR P1.7**

۳- دستور MOV یا انتقال روی بیت

**MOV DES, SOR**

**MOV C , FLAG**

**MOV P1.0 , C/CY**

نکته: C همان CY است و در عملیات انتقال بیت با دستور MOV یکی از عملوندها حتما باید بیت نقلی یا C باشد.

عملیات منطقی روی بیتها:

۴- دستور AND روی بیت

**AND C , SOURCE**

۵- دستور OR روی بیت

**ORL C , SOURCE****ORL C , P2.2**

می توانید آدرس بیت P2.2 را هم بگذارید.

در این دستور بیت ۲ پورت ۲ با بیت نقلی OR میشود و نتیجه در بیت نقلی C قرار میگیرد.

نکته: عملیات AND و OR بیت حتما باید با بیت نقلی یا C انجام شوند و نتیجه در بیت نقلی یا C قرار میگیرد.

نکته: میکروکنترلر ۸۰۵۱ فقط دستورات AND و OR منطقی روی بیت دارد که یکی از عملوندها بیت نقلی یا C است و نتیجه نیز در C قرار میگیرد. بهقیه دستورات منطقی روی بیت را می توان با برنامه تولید کرد.

دستورات تست بیت:

۱- دستور تست بیت نقلی JC

**JC ADDRESS/LABLE**

اگر بیت نقلی C برابر ۱ باشد به آدرس مقصد پرش کن در غیر این صورت دستور بعدی را اجرا کن

**JC SKIP****CPL C** دستور بعدی

SKIP: ادامه برنامه

## ۲- دستور تست بیت نقلی JNC

JNC ADDRESS/LABLE

اگر بیت نقلی یا C برابر ۱ نیست و ۰ است به ادرس مقصد پرش میکند در غیر این صورت دستور بعدی اجرا میشود.

## ۳- دستور تست بیت JB

JB BIT , ADDRESS/LABLE

اگر بیت مورد نظر یک بود به ادرس مقصد پرش می کند در غیراین صورت دستور بعدی را اجرا می کند.

JB P1.2 SKIP

CLP C

SKIP: ادامه برنامه

## ۴- دستور تست بیت JNB

JNB BIT , ADDRESS/LABLE

اگر بیت مورد نظر برابر یک نباشد و ۰ باشد به ادرس مقصد پرش می کند و گرنه دستور بعدی اجرا میشود.

## ۵- دستور تست بیت JBC

JBC BIT , ADDRESS/LABLE

اگر بیت مورد نظر یک باشد آن را صفر میکند و به ادرس مقصد پرش می کند. در غیر این صورت دستور بعدی را اجرا می کند.

JBC ACC.3 , SKIP

CPL C

SKIP: ادامه برنامه

بیت سوم از ثبات یا اکومولاتور A را تست می کند اگر یک باشد ان را صفر میکند و به ادرس مقصد پرش می کند در غیر این صورت دستور بعدی اجرا می شود.

دستورات پرسهای شرطی:

### ۱ - دستور پرش شرطی JZ

#### JZ ADDRESS/LABLE

اگر محتوای ثبات A برابر صفر است به ادرس مقصد پرش می کند در غیر این صورت دستور بعدی اجرا میشود.

### ۲ - دستور پرش شرطی JNZ

#### JNZ ADDRESS/LABLE

اگر محتوای ثبات A برابر ۰ نباشد به ادرس مقصد پرش می کند و گرنه دستور بعدی اجرا میشود.

### ۳ - دستور پرش شرطی DJNZ

#### DJNZ اپراند ADDRESS/LABLE

این دستور برای عملیات حلقه استفاده میشود.

اپراند می تواند ادرس یک محل حافظه یا یکی از ثباتها باشد. این دستور ابتدا از اپراند یک واحد کم می کند اگر نتیجه ۰ نباشد به ادرس مقصد پرش می کند. بدیهی است اگر از ۰۰ یک واحد کم شود، عدد FFH حاصل میشود. ادرس مقصد با اضافه کردن ادرس نسبی به محتوای کنتور برنامه PC دستور بعدی حاصل میشود.

MOV R7,#10

LOOP: CPL P1.7

.

.

.

## DJNZ R7, LOOP

این برنامه یک حلقه با ۱۰ بار تکرار است.

## ۴- دستور پرش شرطی CJNE

CJNE , اپراند ۲ , اپراند ۱ ADDRESS/LABLE

این دستور هم برای عملیات حلقه بکار میرود.

این دستور اپراند ۱ را با اپراند ۲ مقایسه می کند. اگر مساوی نباشد به ادرس مقصد پرش می کند در غیر این صورت دستور بعدی اجرا میشود.

CJNE A , #03H , SKIP

CPL C

SKIP : ادامه برنامه

## دستورات مریبوط به تابع فرعی:

ما باید در برنامه برای افزایش سرعت و کاهش پیچیدگی ها تا حد امکان برنامه را به قسمت های کوچکتری به نام تابع تبدیل کنیم و از تابع فرعی استفاده کنیم .

## ساختار تابع فرعی:

FUN\_NAME:

INS1

INS2

.

.

.

RET

دستور RET از اين دستور برای بازگشت از تابع فرعی استفاده می شود. آخرین دستور که در تابع استفاده میشود اين دستور است.

دستور CALL برای فراخوانی تابع فرعی بکار می رود.

call FUNC\_NAME

جلسه ۲۳

شبۀ دستور ORG:

ORG 20H

اين دستور تعیین می کند که دستورات در چه ادرسی از حافظه قرار گیرند

دستور پایان END:

پایان دستورات اسمبلي را مشخص می کند و اخرين خط دستورات می باشد.

شبۀ دستور DB

DATA DB 10

X DB -15

STR1 DB "WELCOME"

**دستور NOP:**

این دستور هیچ عملی را انجام نمی دهد. و فقط برای ایجاد وقفه در انجام دستورات استفاده میشود.

**چند مثال:**

مثال ۱ - کاربرد دستور ADDC  
 برنامه‌ای بنویسید که دو عدد ۱۶-بیتی  $2A46H$  و  $3687H$  را جمع و نتیجه را در ثبات‌های R1 و R2 قرار دهد (بایت کم‌ارزش‌تر حاصل جمع در ثبات R1 قرار گیرد)



برای این منظور برنامه زیر را می‌نویسیم:

```
;example of 16 bits addition for
;chapter 3
;add16.asm
;
ORG 0
CLR C ;(1)
MOV A, #46H ;(2)
ADD A, #87H ;(3)
MOV R1, A ;(4)
MOV A, #2AH ;(5)
ADDC A, #36H ;(6)
MOV R2, A ;(7)
END
```

**مثال ۲- کاربرد دستور CJNE**

برنامه‌ای بنویسید که اگر محتوای اکومولاتور (A)، مساوی 50H باشد، ثبات B را 1 کند، در غیر این صورت ثبات B را 0 کند.



برای این منظور برنامه زیر را می‌نویسیم:

```
; example of instruction CJNE
; chapter 3
; cjne.asm

        MOV A, #50H      ;(1)
        MOV B, #0         ;(2)
        CJNE A, #50H, NEXT ;(3)
        MOV B, #01H       ;(4)
NEXT:   NOP             ;(5)
        END             ;(6)
```

**مثال ۳- کاربرد آدرس دهی غیرمستقیم**

برنامه‌ای بنویسید که از خانه‌های حافظه RAM به آدرس 10H تا 15H مقدار 25H را بنویسد.



برای این منظور برنامه زیر را می‌نویسیم:

```
; Indirect addressing
; Example for chapter 3
; Indirect.asm

        MOV A, #25H      ;(1)
        MOV R0, #10H       ;(2)
        MOV R2, #5         ;(3)
AGAIN:  MOV @R0, A        ;(4)
        INC R0             ;(5)
        DJNZ R2, AGAIN    ;(6)
        END               ;(7)
```

**مثال ۴- کاربرد دستور DJNZ در حلقه**

برنامه‌ای بنویسید که اکومولاتور (A) را پاک کند و 12 بار عدد 5 را به آن، اضافه کند و نتیجه را در ثبات R5 قرار دهد.



برای این منظور برنامه زیر را می‌نویسیم:

```
;example of DJNZ instruction for
;looping
;LOOP1.ASM
;
    MOV A,#0          ;(1)
    MOV R4,#12         ;(2)
BACK:   ADD A,#05        ;(3)
        DJNZ R4,BACK      ;(4)
        MOV R5,A          ;(5)
        END                ;(6)
```

**مثال ۵- کاربرد دستور DJNZ در حلقه تودرتو**

برنامه‌ای بنویسید که اکومولاتور (A) را، با عدد 0AAH بار کند و 500 بار اکومولاتور (A) را مکمل کند.



برای این منظور در ثبات‌های R6 و B به ترتیب اعداد 50 و 10 را، قرار می‌دهیم که تعداد تکرار مکمل اکومولاتور (A)، مساوی  $50 \times 10 = 500$  شود.  
لذا برنامه مذکور مطابق زیر می‌شود:

```
;example of loop inside the loop
;loop2.asm
;
    MOV A,#0AAH       ;(1)
    MOV B,#10          ;(2)
BACK1:  MOV R6,#50        ;(3)
BACK2:  CPL A          ;(4)
        DJNZ R6,BACK2      ;(5)
        DJNZ B,BACK1        ;(6)
        END                ;(7)
```

**مثال ۶- استفاده از سابر و تین و ایجاد تأخیر نرم افزاری**

برنامه‌ای بنویسید که مقدار OAAH را در اکومولاتور (A) قرار دهد و آن را به پورت 0 (PO) ارسال کند. بعد از یک تأخیر (DELAY1)، اکومولاتور (A) را مکمل و مجدداً بعد از همان تأخیر، اکومولاتور (A) را، به پورت 0 (PO) ارسال کند و عملیات را تکرار نماید.

```
;example of delay
;delay1.asm
;
BACK1: MOV A,#0AAH      ;(1)
        MOV P0,A       ;(2)
        LCALL DELAY1   ;(3)
        CPL A          ;(4)
        SJMP BACK1    ;(5)
;
;----delay subroutine----
;
DELAY1: MOV R0,#0FFH     ;(6)
HERE:   DJNZ R0,HERE    ;(7)
        RET             ;(8)
;
END
```

**مثال ۷- با استفاده از حلقه تودرتو، تأخیر سابر و تین مثال (۶) را تا حد ممکن افزایش دهید.**



برای این مسظور برنامه زیر را می‌نویسیم:

```
;example of long delay by loop
;inside loop
;delay2.asm
;
DELAY2:
        MOV R6,#0FFH    ;(1)
BACK1:  MOV R7,#0FFH    ;(2)
HERE:   DJNZ R7,HERE  ;(3)
        DJNZ R6,BACK1  ;(4)
        RET            ;(5)
        END
```

مثال ۸ - برنامه‌ای بنویسید که با استفاده از دو حلقه تودرتو، و دستور NOP، حداقل تأخیر را ایجاد کند.

```
; example of delay
;delay3.asm
;
        MOV  R5 ,#0FFH      ;(1)
BACK1:  MOV  R6 ,#0FFH      ;(2)
BACK2:  NOP              ;(3)
        NOP              ;(4)
        NOP              ;(5)
DJNZ  R6 ,BACK2          ;(6)
DJNZ  R5 ,BACK1          ;(7)
        END              ;(8)
```

### مثال ۹ - نوشتن در پورت

برنامه‌ای بنویسید که مقدار AAH را به پورت 1 (P1) بفرستد، بعد از یک تأخیر (DELAY)، آن را مکمل کند و عملیات را تکرار کند.



برای این منظور برنامه زیر را می‌نویسیم:

```
;example of writing on a port
;
;write_po
;
        MOV A,#0AAH      ;(1)
BACK1: MOV P1,A       ;(2)
        CALL DELAY     ;(3)
        CPL A          ;(4)
        SJMP BACK1    ;(5)
;
; DELAY subroutine
DELAY:
        MOV R0,#0FFH    ;(6)
HERE:  DJNZ R0,HERE   ;(7)
        RET             ;(8)
;
        END             ;(9)
```

### مثال ۱۰ - خواندن از پورت ورودی

برنامه‌ای بنویسید که اطلاعات را از پورت 0 (PO) بخواند و در پورت 1 (P1) بنویسد و این عمل را تکرار کند.



برای این منظور برنامه زیر را می‌نویسیم:

```
;example of reading from port
;read_po
;---- making port 0 as input---
        MOV A,#0FFH      ;(1)
        MOV P0,A         ;(2)
;----reading from P0,writng on P1
BACK:  MOV A,P0        ;(3)
        MOV P1,A         ;(4)
        SJMP BACK       ;(5)
        END             ;(6)
```

### مثال ۱۱- کاربرد دستور رشته اطلاعات



MOVC A@A+DPTR

برنامه‌ای بنویسید که رشته حروف TEHRAN که در حافظه برنامه ROM در آدرس 18H به بعد قرار دارد را، به حافظه داده (RAM) از آدرس 10H به بعد، انتقال دهد.

برای این کار برنامه زیر را می‌نویسیم:

```
;example of string for chapter 3
;string.asm
;
        ORG 0          ;(1)
        MOV DPTR,#STRING ;(2)
        MOV R0,#10H      ;(3)
LOOP1: CLR A           ;(4)
        MOVC A,@A+DPTR ;(5)
        JZ STOP          ;(6)
        MOV @R0,A         ;(7)
        INC DPTR         ;(8)
        INC R0           ;(9)
        SJMP LOOP1       ;(10)
STOP:   SJMP STOP       ;(11)
;
;---on-chip code memory used for
;      string
;        ORG 18H
STRING: DB 'TEHRAN 0' ;(12)
        END
```

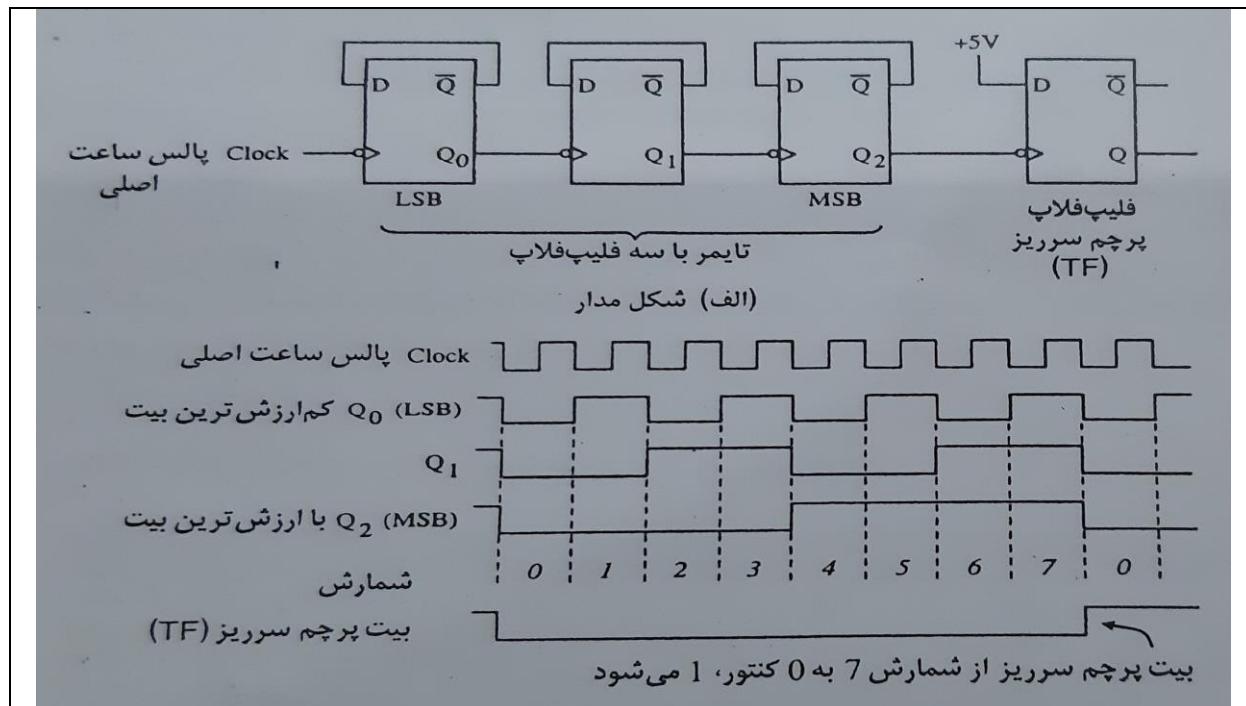
نحوه اجرا

### تایمرها:

تایمرها برای اندازه گیری زمان بین دو لحظه یا شمارش رویدادها و هم چنین برای تولید پالس انتقال اطلاعات در پورت سری استفاده می‌شود.

اصولاً تایمر یک شمارنده یا کنتور است که از یک سری فلیپ فلاب تشکیل می‌شود، که هر فلیپ فلاب پالس ساعت ورودی خود را بر ۲ تقسیم می‌کند.

در میکروکنترلر ۸۰۵۱ دو تایمر ۱۶ بیتی به نام تایمر ۰ و تایمر ۱ وجود دارد. در میکروکنترلر ۸۰۵۲ یک تایمر ۱۶ بیتی اضافی به نام تایمر ۲ وجود دارد.



برای دسترسی و کار با تایمرهای ۸۰۵۱ از ۶ ثبات کاربرد خاص SFR استفاده می شود. برای دسترسی به تایمر ۲ که در میکروکنترلر ۸۰۵۲ وجود دارد از ۵ ثبات دیگر کاربرد خاص SFR استفاده میشود.

جدول (۱-۴) ثبات‌های کاربرد خاص (SFR) تایمرها

ثبات‌های SFR تایمر	کاربرد	آدرس ثبات	پست آدرس پذیرا
TCON	کنترل تایمر	88H	آری
TMOD	تعیین حالت یا مد تایمر	89H	خیر
TL0	بایت کوچکتر تایمر 0	8AH	خیر
TL1	بایت کوچکتر تایمر 1	8BH	خیر
TH0	بایت بزرگتر تایمر 0	8CH	خیر
TH1	بایت بزرگتر تایمر 1	8DH	خیر
T2CON*	کنترل تایمر 2	C8H	آری
RCAP2L*	بایت کوچکتر ثبات گرفتن تایمر 2	CAH	خیر
RCAP2H*	بایت بزرگتر ثبات گرفتن تایمر 2	CBH	خیر
TL2*	بایت کوچکتر تایمر 2	CCH	خیر
TH2*	بایت بزرگتر تایمر 2	CDH	خیر

\* فقط در میکروکنترلر 8052 وجود دارد

## ثبات حالت تایمر (TMOD)

ثبت مد یا ثبات حالت TMOD یک ثبت 8 بیتی است که بیتهای ۰ تا ۳ آن برای تایمر ۰ و بیتهای ۴ تا ۷ آن تایمر ۱ را کنترل می کند. ثبات TMOD مشخص می کند که هر یک از تایمرها به صورت تایمر برای اندازه گیری زمان بکاربرده شوند یا به عنوان کنتور ۸ بیتی و یا ۱۶ بیتی برای شمارش رویداد استفاده شوند.

جدول (۲-۴) خلاصه ای از طرز کار ثبات حالت تایمر (TMOD)

		نام	شماره بیت	عملکرد
7	GATE	1		بیت گیت زمانی که ۱ می شود، تایمر به شرطی کار می کند که $\overline{INT1}$ برابر ۱ باشد.
6	$C/\bar{T}$	1		بیت انتخاب $\bar{T}$ که تایمر ۱ را به عنوان کنتور یا تایمر فعال می کند (اگر $C/\bar{T} = 0$ باشد به عنوان تایمر برای اندازه گیری زمان و اگر $C/\bar{T} = 1$ باشد به عنوان کنتور برای شمارش رویداد)
5	M1	1		بیت ۱ حالت تایمر ۱ (رجوع به جدول (۳-۴))
4	M0	1		بیت ۰ حالت تایمر ۱ (رجوع به جدول (۳-۴))
3	GATE	0		بیت گیت زمانی که ۰ شود تایمر ۰ به شرطی کار می کند که $\overline{INT0}$ برابر ۱ باشد
2	$C/\bar{T}$	0		بیت انتخاب $\bar{T}$ که تایمر ۰ را به عنوان کنتور یا تایمر فعال می کند (مانند فوق)
1	M1	0		بیت ۱ حالت تایمر ۰ (رجوع به جدول (۳-۴))
0	M0	0		بیت ۰ حالت تایمر ۰ (رجوع به جدول (۳-۴))

باید توجه کرد که بیتهای ثبات TMOD را نمیتوان بطور مجزا آدرس دهی کرد. معمولا در ابتدای برنامه یکبار مقدار اولیه به ثبات حالت TMOD داده میشود که طرز کار تایمرها مشخص شود. سپس در ادامه با مقدار دادن به دیگر ثباتهای کلربرد خاص SFR به خصوص ثبات کنترل TCON، تایمر راه اندازی یا در صورت لروم متوقف میشود.

جدول (۳-۴) حالت های تایمر ۰ و تایمر ۱

M1	M0	حالات تایمر	عملکرد تایمر
0	0	0	حالات تایمر ۱۳ بیتی (حالت 8048)
0	1	1	حالات تایمر ۱۶ بیتی
1	0	2	حالات تایمر ۸ بیتی با بار شدن اتوماتیک
1	1	3	حالات تایمر دو قسمتی
■ تایمر ۰ دو قسمت می شود یعنی دو تایمر مجزای ۸ بیتی TL0 و TH0 (شکل ۲-۴-۵)			■ تایمر ۱ متوقف می شود

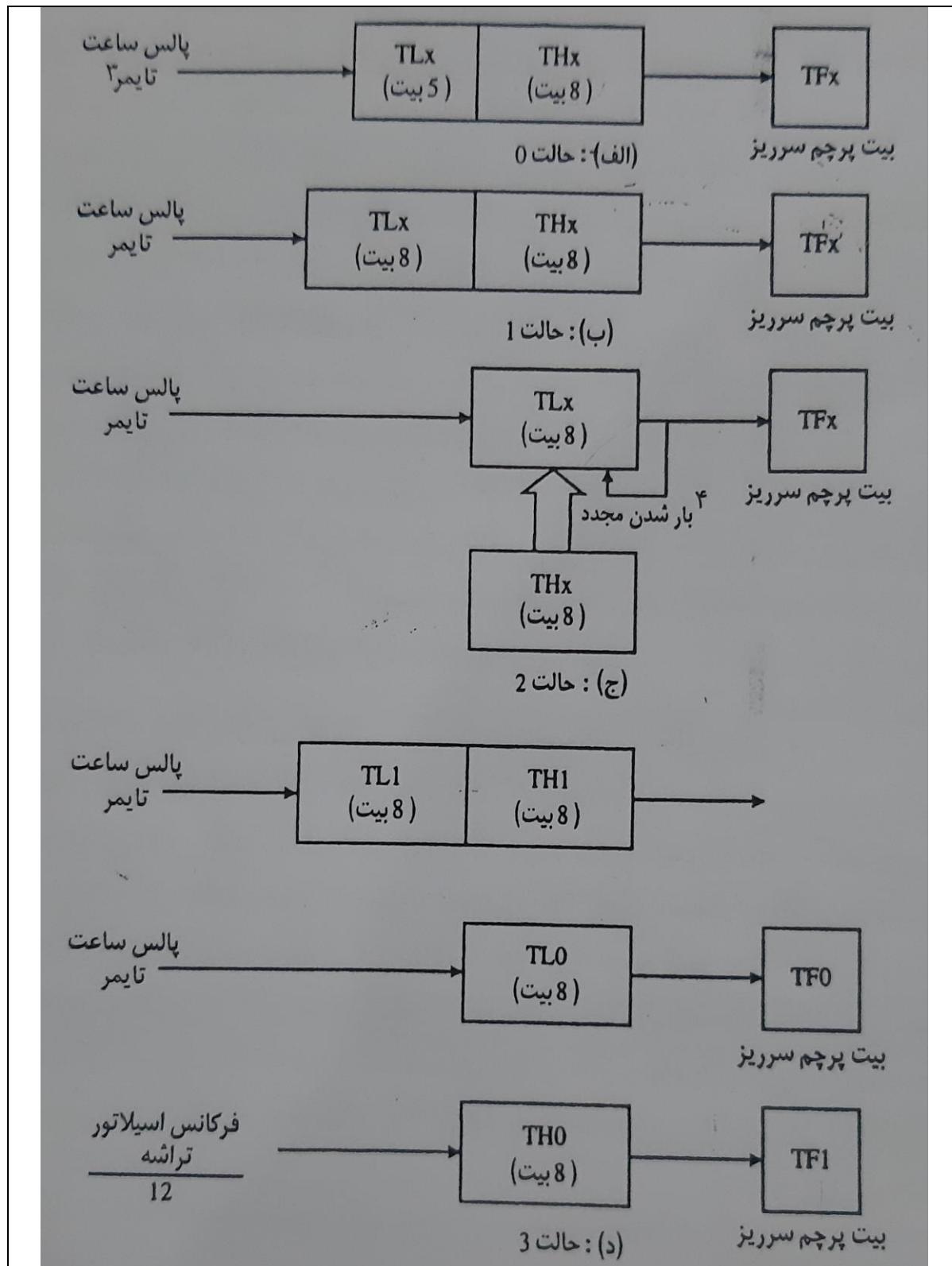
## ثبات کنترل تایمر TCON

با ثبات کنترل TCON می‌توان وضعیت تایمرهای ۰ و ۱ را بررسی نمود و آنها را راه اندازی یا متوقف کرد. چهار بیت بالای ثبات کنترل TCON یعنی بیتهاي ۴ تا ۷ برای راه اندازی یا توقف تایمر ۰ (TR0) یا تایمر ۱ (TR1)، یا برای اعلام سرریز در تایمر ۰ (توسط بیت TFO) یا تایمر ۱ (توسط بیت TF1) مورد استفاده قرار می‌گیرند. چهار بیت پایین(بیتهاي ۰ تا ۳) برای تایمرها نیستند، بلکه برای تشخیص و ایجاد وقفه خارجی هستند.

جدول (۴-۴) خلاصه‌ای از طرز کار ثبات کنترل تایمر (TCON)

				عملکرد
	شماره بیت	نمیبول بیت	ادمین بیت	
TCON.7	TF1	8FH		<ul style="list-style-type: none"> <li>موقعی که تایمر پُر شد بیت سرریز (TF1) تایمر ۱، برابر ۱ می‌شود که با برنامه میکروکنترلر می‌توان آن را ۰ نمود * به صورت سخت‌افزاری زمانی که پروسسور، روتین وقفه *** را اجرا می‌کند ۰ می‌شود.</li> </ul>
TCON.6	TR1	8EH		<ul style="list-style-type: none"> <li>بیت کنترل راهاندازی تایمر ۱ (TR1) اگر با برنامه ۱ شود **، تایمر ۱ راهاندازی می‌شود و در صورتی که با برنامه ۰ گردد *** تایمر ۱ متوقف می‌شود.</li> </ul>
TCON.5	TFO	8DH		<ul style="list-style-type: none"> <li>بیت سرریز (TFO) تایمر ۰، مانند (TF1) عمل می‌کند ولی برای تایمر ۰</li> </ul>
TCON.4	TR0	8CH	0	<ul style="list-style-type: none"> <li>بیت کنترل راهاندازی تایمر ۰ (TR0) مانند (TR1) عمل می‌کند ولی برای تایمر ۰</li> </ul>
TCON.3	IE1	8BH		<ul style="list-style-type: none"> <li>بیت پرچم حساس به لبه وقفه ۱ خارجی ۲ (IE1): موقعی که لبه پایین رونده پالس وقفه، در ورودی INT1 ظاهر شود، این بیت (IE1) برابر ۱ می‌شود. با برنامه میکروکنترلر می‌توان آن را ۰ نمود و یا موقعی که روتین وقفه را اجرا نمود، ۰ می‌شود.</li> </ul>
TCON.2	IT1	8AH		<ul style="list-style-type: none"> <li>بیت پرچم نوع وقفه ۱ خارجی (IT1): اگر این بیت با برنامه ۱ شود، وقفه به لبه پایین رونده INT1 حساس می‌شود *** و اگر این بیت با برنامه ۰ شود وقفه با سطح پایین INT1 حساس می‌شود ***.</li> </ul>
TCON.1	IE0	89H		<ul style="list-style-type: none"> <li>بیت پرچم حساس به لبه وقفه ۰ خارجی (IE0)، مانند (IE1) عمل می‌کند ولی برای تایمر ۰</li> </ul>
TCON.0	IT0	88H		<ul style="list-style-type: none"> <li>بیت پرچم نوع وقفه ۰ خارجی (IT0): مانند IT1 عمل می‌کند ولی برای تایمر ۰</li> </ul>

حالت یا مدهای مختلف تایمر و بیت پرچم سرریز



### دسترسی و مقدار اولیه دادن به ثباتهای تایمر:

در ابتدای برنامه با مقدار دهی ثبات TMOD تایمر مورد نظر و حالت مورد نظر و اینکه تایمر یا شمارنده میخواهیم مشخص میشود

در صورتی که تایمر از مقدار اولیه ای باید شروع شود در این صورت ثباتهای TLx و THx تایمر مورد نظر باید مقدار دهی اولیه شوند. x دو مقدار ۰ و ۱ می توانند باشد . مربوط به تایمر ۰ و ۱ مربوط به تایمر ۱ است

نکته: بخاطر داشته باشید که تایمرها به صورت صعودی میشمارند و موقعی که از 0000H به FFFFH تغییر یابند بیت پرچم سرریز برابر ۱ میشود.

مثال : اگر زمان ۱۰۰ میکرو ثانیه مورد نیاز باشد، مقدار اولیه ثباتهای TL و TH تعداد ۱۰۰ شماره کمتر از 0000H یعنی -۱۰۰ (منفی ۱۰۰) یا FF9CH قرار داده میشود.

```
Mov TMOD,#10H ;select timer1 in mode 16 bit
```

```
MOV TL1 ,#9CH
```

```
MOV TH1,#0FFH
```

مثال:

## مثال ۱ : تایمر ۰ در حالت یا مُد ۱

برنامه‌ای بنویسید که بیت ۲ پورت ۲ (P2.2) را با استفاده از تایمر ۰ به مدت ۵۰۰ میکروثانیه برابر ۱ کند (مانند ).



برای این منظور باید:

- ۱- ثبات حالت (TMOD) تایمر را، برای تایمر ۰، در مُد ۱ (16 بیتی) قرار دهیم.
- ۲- عدد ۵۰۰ - را در ثبات‌های TLO و TH0 تایмер بار کنیم.
- ۳- تایمر ۰ را، راهاندازی کنیم و بیت ۲ پورت ۲ (P2.2) را ۱ کنیم.
- ۴- بالاخره بعد از ۵۰۰ پالس، تایمر را متوقف و بیت ۲ پورت ۲ (P2.2) را ۰ کنیم.

برای این کار برنامه زیر را می‌نویسیم:

```
;example of timer
```

```
;tim_md11.asm
```

```

        CLR P2.2          ;(1)
        MOV TMOD, #01H    ;(2)
        MOV TH0, #0FEH    ;(3)
        MOV TL0, #0CH     ;(4)
        SETB P2.2          ;(5)
        SETB TR0           ;(6)
HERE:   JNB TF0, HERE      ;(7)
        CLR TR0           ;(8)
        CLR P2.2          ;(9)
        CLR TF0           ;(10)
        END                ;(11)
    
```

## مثال ۲ : تایمر ۰ در مُد ۱

برنامه‌ای بنویسید که بیت ۲ پورت ۱ (P1.2) را، ابتدا ۱ و سپس ۰ کند (مانند ). حداقل این زمان چقدر می‌تواند باشد.



**حل:** حداقل زمان موقعی است که تایمر ۰ در مُد ۱ قرار گیرد و مقدار ۰ در ثبات‌های TLO و TH0 بار شود. در این صورت بیت پرچم سرریز (TFO)، بعد از ۶۵۵۳۶ پالس یا میکروثانیه برابر ۱ می‌شود. بنابراین زمان ۱ بودن بیت ۲ پورت ۱ (P1.2) برابر حداقل ۶۵۵۳۶ میکروثانیه می‌باشد\*. لذا برنامه زیر را می‌نویسیم:

```
;example of timer
```

```
;tim_md12
```

```

        CLR P1.2          ;(1)
        MOV TMOD, #01H    ;(2)
        MOV TL0, #00       ;(3)
        MOV TH0, #00       ;(4)
        SETB P1.2          ;(5)
        SETB TR0           ;(6)
HERE:   JNB TF0, HERE      ;(7)
        CLR TR0           ;(8)
        CLR TF0           ;(9)
        CLR P1.2          ;(10)
        END                ;(11)
    
```

مثال ۳- برای زمان‌های بیشتر، می‌توان تایمر ۱ در مُد ۱ را،  
چند بار راهاندازی کرد

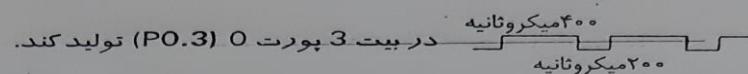
برنامه‌ای بنویسید که تایمر ۱ را در مُد ۱، ۲۵۰ بار از حالت ۰۰۰۰ راهاندازی  
کند، حداقل تأخیر ایجاد شده چقدر است؟



```
;example of timer for longer time
;tim_md13.asm
;
    MOV TMOD, #10H ;(1)
    MOV R0, #250 ;(2)
- BACK1:   MOV TH1, #0 ;(3)
            MOV TL1, #0 ;(4)
            SETB TR1 ;(5)
BACK2:   JNB TF1, BACK2 ;(6)
            CLR TR1 ;(7)
            CLR TF1 ;(8)
            DJNZ R0, BACK1 ;(9)
END ;(10)
```

حداقل تأخیر : 250\*65535

مثال ۴- تایمر ۰ در مُد ۲  
برنامه‌ای بنویسید که تایمر ۰ را در مُد ۲ قرار دهد و عدد هگزادسیمال معادل  
۲۰۰ - را در ثبات TH0 بارگذارد تا موج مربعی به صورت



که حل : برای این منظور برنامه زیر را می‌نویسیم:



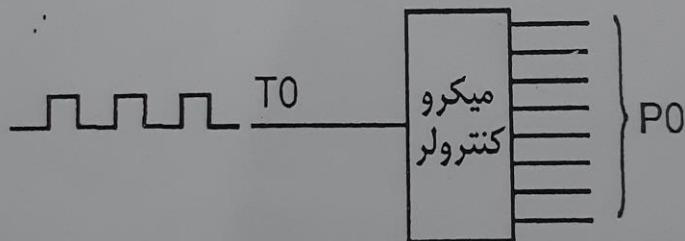
```
;example of mode 2 timer
;tim_md21.asm
;
    MOV TMOD, #02H ;(1)
    MOV TH0, #38H ;(2)
AGAIN1: SETB P0.3 ;(3)
        ACALL DELAY1 ;(4)
        ACALL DELAY1 ;(5)
        CLR P0.3 ;(6)
        ACALL DELAY1 ;(7)
        SJMP AGAIN1 ;(8)
;
DELAY1:
    SETB TRO ;(9)
AGAIN2: JNB TFO, AGAIN2 ;(10)
        CLR TRO ;(11)
        CLR TFO ;(12)
        RET ;(13)
END ;(14)
```

چون میخواهد هر بار ۲۰۰ تا شمارش شود از مد ۲ استفاده شده و عدد منفی ۲۰۰ در ثبات TH0 برای بار شدن خودکار تایمر ۰ با این مقدار قرار گرفته است



### مثال ۵- تایمر ۰ به عنوان کنتور مُد ۲

فرض می‌کنیم پالس ساعتی با فرکانس ۱ Hz به ورودی T0 میکروکنترلر متصل است، برنامه‌ای بنویسید که تایمر ۰ را به عنوان کنتور، در مُد ۲ قرار دهد، پالس ورودی (T0) را به شمارد و مقدار TL0 را، به پورت ۰ (PO) برای نمایش روی ارسال کند (شکل ذیل).



**حل :** برای این منظور باید تایمر ۰ در حالت کنتور و T0 در حالت ورودی قرار گیرد. لذا برنامه زیر را می‌نویسم:

```
; example of timer 0 as a counter
;coun_md2.asm
;
MOV TMOD, #00000110B      ;(1)
MOV TH0, #0                 ;(2)
;---making T0 as input-----
SETB P3.4                  ;(3)
BACK1: SETB TR0             ;(4)
BACK2: MOV A, TL0            ;(5)
      MOV PO, A              ;(6)
      JNB TF0, BACK2          ;(7)
      CLR TR0                ;(8)
      CLR TF0                ;(9)
      SJMP BACK1              ;(10)
END
```



