معماری کامپیوتر

مهر ۰۵–۰۴

جلسه اول

اِلمان‌های منطقی

کَار با زبان ماشین

در این ترم قرار است چه یاد بگیریم؟

- یادآوری المان‌های منطقی که برای فهم این درس لازم است آنها را از قبل بشناسید.
- آشنایی با ساختار و نیز زبان ماشین PC (که یک ماشین پردستور است).
- مقدمات و آشنایی با معماری کامپیوتر. مفاهیم پایه معماری. گذری بر تاریخچه و انواع معماری کامپیوتر.
- انتخاب یک کامپیوتر ساده کم دستور (کامپیوتر پایه مانو در مقایسه با کامپیوتر MIPS پترسون- هنسی) برای موشکافی معماری در این درس. طراحی کامل زبان اسمبلی و ساختار سخت‌افزاری این کامپیوتر و نحوه استفاده از زبان اسمبلی آن.
- طراحی کامل CPU آن کامپیوتر.
- مباحث تکمیلی مانند خط لوله و سلسله مراتب حافظه.

منابع درسی

- در صورتی که علاوه بر اسلایدها (جزوه این درس) خواستید مطالعه بیشتری برای فهم مطالب این درس داشته باشید، منابع درسی به ترتیب اهمیت عبارتند از:
- یک. کتاب معماری کامپیوتر موریس مانو ترجمه دکتر سپیدنام (خلاصه این مباحث در کتاب معماری هست اما در صورتی که بر مباحث مدارهای منطقی اصلا مسلط نیستید مطالعه کتاب مدارهای منطقی همین نویسنده و مترجم نیز توصیه می‌شود)
- دو. کتابی در مورد زبان ماشین و اسمبلی کامپیوتر IBM PC (۸۰۸۶). در میان کتاب‌های موجود کتاب ریچارد دتمر ترجمه هاشمی اصل (دانشگاه علم و صنعت) یا کتاب دکتر سیدرضی بیشتر توصیه می‌شود.
- سه. کتاب معماری کامپیوتر پترسون. ما در این درس کمتر به مباحث این کتاب خواهیم پرداخت اما برای کسانی که بصورت حرفه‌ای بدنبال یادگیری سخت‌افزار هستند مفید است.

معماری کامپیوتر چیست؟

- معماری (در همه رشته‌ها) یعنی چه؟
- معماری کامپیوتر یعنی چه؟
- تاریخچه معماری کامپیوتر چیست (با اشاره به معماری‌های کم‌دستور و پردستور)؟
- دانستن معماری کامپیوتر برای کسی که نمیخواهد معمار کامپیوتر شود چه اهمیتی دارد؟
- زبان ماشین و اسمبلی چیست؟
- ورود به مقدمات این درس و پاسخ این سوالات (البته غیر از سوال آخر) را پس از دو سه جلسه آشنایی با کامپیوتر PC خواهد بود.

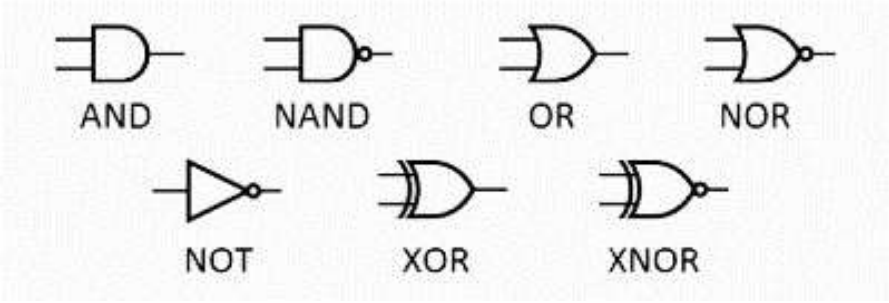
پیش‌نیاز درک معماری کامپیوتر

تا پیش از این درس، دروس نرم‌افزاری (از قبیل برنامه‌نویسی مقدماتی و پیشرفته و ساختمان داده‌ها) و دروس سخت‌افزاری (مدارهای الکتریکی، الکترونیکی و منطقی) جداگانه دنبال می‌شدند .اما در درس معماری کامپیوتر این دو شاخه با هم تلاقی می‌کنند.

به بیان دیگر دانستن در مورد المان‌های مدارهای منطقی و نیز دانستن برنامه نویسی (هم سطح بالا مانند C و هم سطح پایین یعنی زبان ماشین و اسمبلی) کلید ورود به معماری کامپیوتر است.

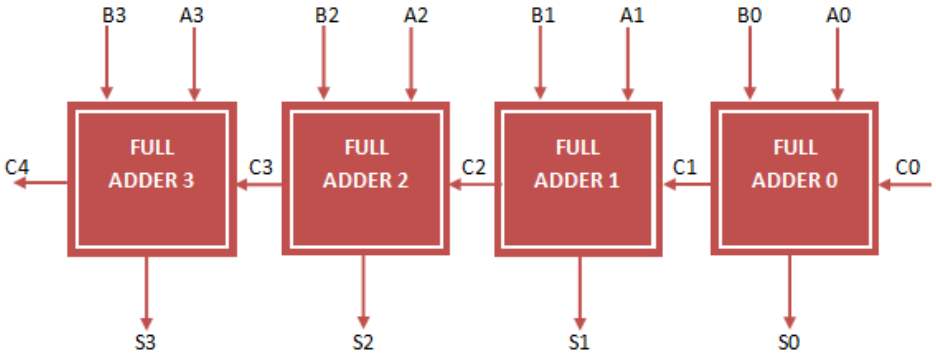
مروری سریع بر المان‌های منطقی

- در درس مدارهای منطقی گیت‌های پایه و المان‌هایی که با استفاده از آن ساخته می‌شوند، معرفی شده‌اند.
- در آنجا تاکید بر طرز ساخت (مدارهای داخلی) المان‌ها و نیز آشنایی با طرز کار آنها بود. اما در این درس نیازمند به استفاده حرفه‌ای از آنها هستیم.
- در این درس تنها فرصتی برای مرور آنها داریم. **اگر با مطالعه ادامه درس امروز متوجه شدید که بر آنها مسلط نیستید، حتما جزوه درس مدارهای منطقی‌تان را مرور کنید. بدون اینها چیزی از معماری کامپیوتر نخواهید فهمید.**
- درس معماری کامپیوتر ساده، اما بسیار پیوسته است. اگر بخاطر ندانستن مقدمات یا از دست دادن یکی دوجلسه، از سیر کلاس عقب بمانید جبران آن بسیار دشوار است.



گیت‌های پایه

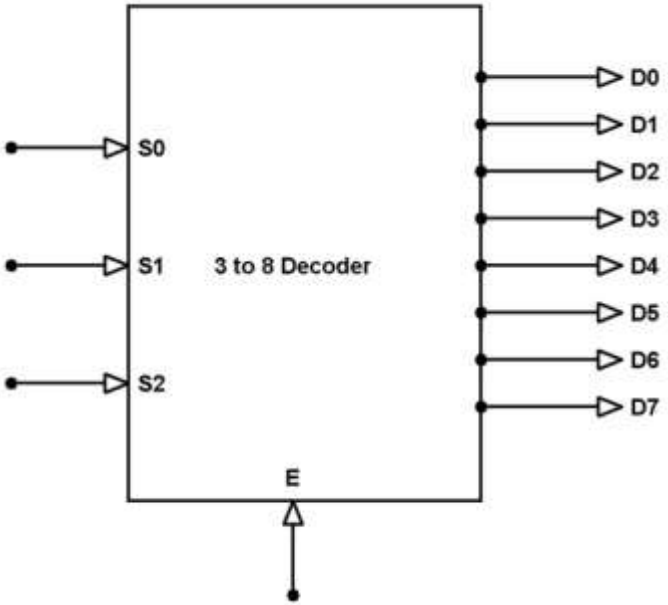
- گیت‌های and و or و not عملگرهای پایه جبر بولی را شبیه‌سازی می‌کنند. با استفاده از گیت‌های پایه می‌توان فورمول‌های بولی را تبدیل به مدارهای عملی کرد.
- طرز کار مدار با جدول درستی نشان داده می‌شود. جدول درستی مستقیما به یک فورمول کانونی قابل تبدیل است و همچنین با استفاده از جدول کارنو می‌شود آن را تبدیل به یک فورمول استاندارد ساده شده کرد.



جمع‌کننده

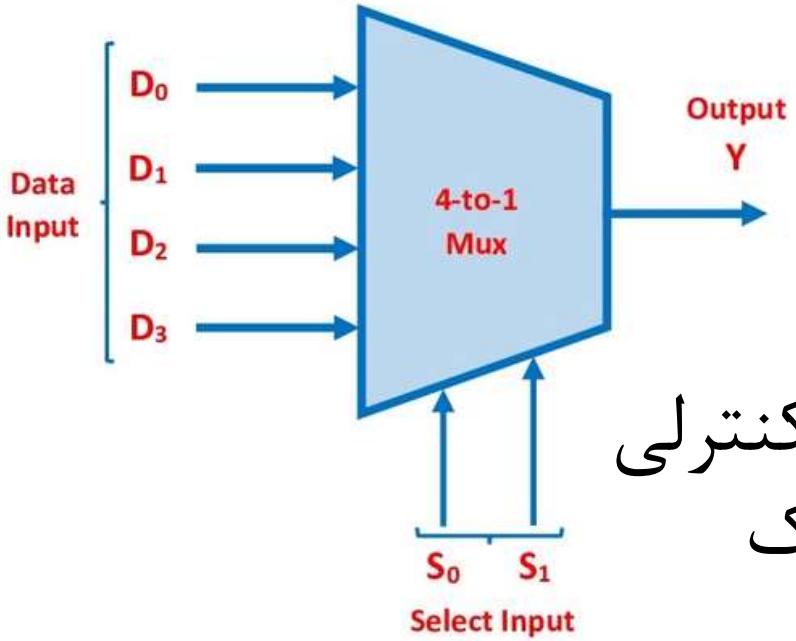
- جمع‌کننده المانی است که دو عدد n بیتی بعنوان ورودی دریافت می‌گَند و حاصل جمع (عددی و نه منطقی) که آنهم n بیتی است را به همراه یک carry یا ov تحویل می‌دهد (n+1 بیت). دو عدد ورودی و خروجی هم طول‌اند.
- جمع‌کننده (مثل اکثر المان‌ها) قابل گسترش است (یعنی مثلا با دو تا چهاربیتی یک هشت بیتی می‌شود ساخت).
- برای تفریق و نیز جمع اعداد علامت‌دار سیستم عددی تغییر می‌کند و در سیستم مکمل دو عمل می‌شود. نحوه محاسبه سرریزنیز فرق دارد (یکسانی کری آخر و قبل آخر).

دیکدر



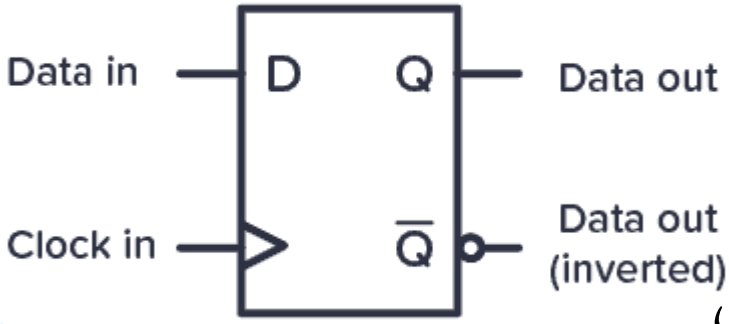
- قطعه‌ای است با n ورودی و 2^n خروجی.
- بسته به کد باینری که روی ورودی است، یکی از خروجی ها 1 شده و بقیه 0 می‌مانند (اگر ورودی Enable یا فعال‌ساز 0 باشد همه 0 می‌مانند).

مالتی‌پلکسر



- قطعه‌ای است با n ورودی کنترلی و 2^n ورودی داده و تنها یک خروجی.
- بر اساس مقداری که به ورودی‌های کنترلی داده می‌شود مقدار یکی از ورودی‌های داده عینا به خروجی منتقل می‌شود.

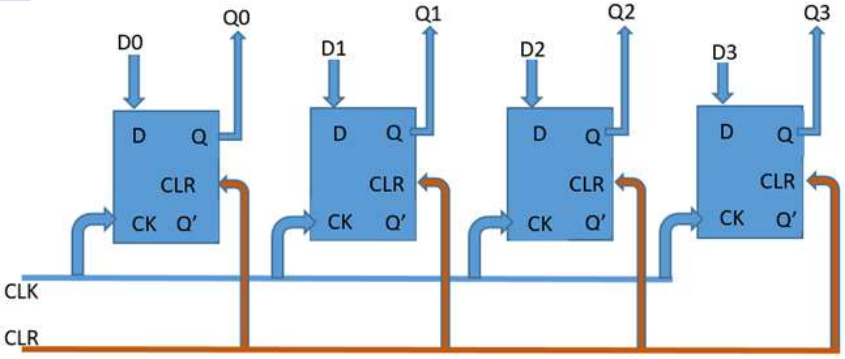
قطعات ترتیبی ساعتدار: فلیپ‌فلاپ‌ها

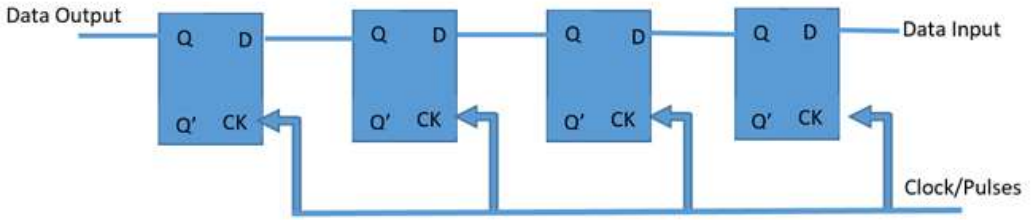


- فلیپ‌فلاپ قطعه‌ای است ترتیبی که خروجی آن غیر از ورودی به حالت داخلی خود قطعه هم بستگی دارد. این قطعه سنکرون است یعنی مقدار خروجی فقط با رسیدن "لبه ساعت" تغییر می‌کند.
- ساده‌ترین فلیپ‌فلاپ نوع D است که یک دیتای یک بیتی را از ورودی گرفته و ذخیره می‌کند. انواع دیگر خصوصا JK هم وجود دارد که به آن هم در این درس نیاز است.

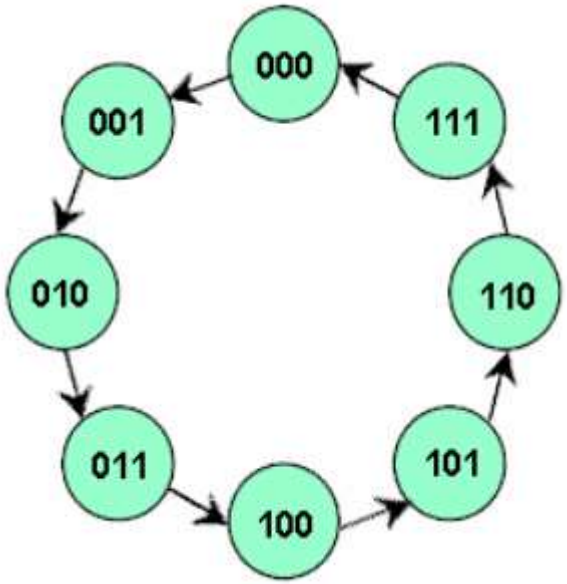
رجیسترها

- رجیسترها محل‌هایی هستند که در دستگاه دیجیتال یک داده چندبیتی را در خود ذخیره می‌کنند. آنها با حافظه RAM (که قابل آدرس‌دهی است) فرق دارند.
- بارگیری داده در رجیسترها می‌تواند موازی باشد (چند خط همزمان) یا متوالی (یک خط که داده را در طی سیکل‌های متوالی ساعت یک بیت یک بیت می‌فرستد.)





شمارنده‌ها



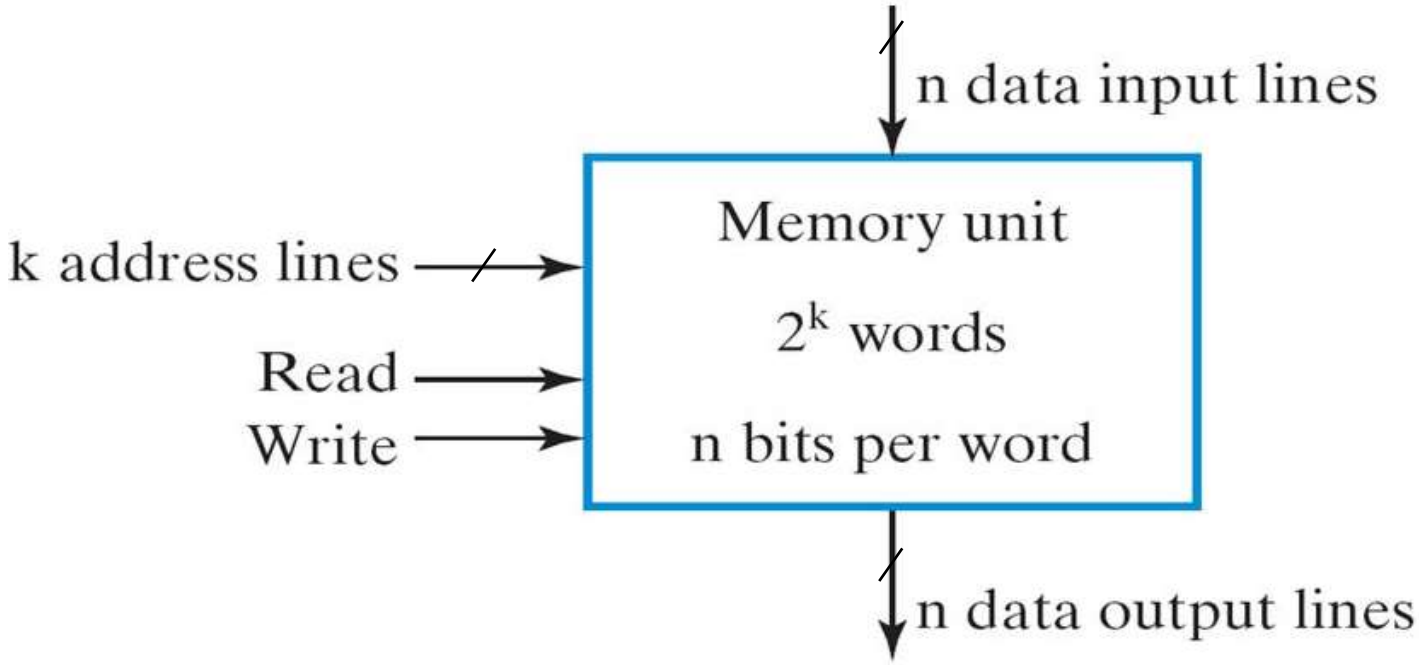
- شمارنده قطعه‌ای است که با رسیدن پالس یک واحد به خروجی آن اضافه می‌شود. وقتی به حداکثر مقدار خود برسد در مرحله بعدی صفر می شود. همچنین آنها معمولا یک ورودی ریست (صفر کننده) دارند.

واحد حافظه RAM

- هر بخش از یک سیستم دیجیتال که بتواند داده‌ای را ذخیره کند، دارای حافظه است. یک فلیپ‌فلاپ می‌تواند یک بیت و یک رجیستر چند (مثلا ۸ یا ۱۶) بیت را ذخیره نماید.
- با این حال وقتی صحبت از واحد حافظه می‌شود منظور واحدی است که آدرس‌پذیر است و در هر آدرس از خود یک داده چند بیتی را ذخیره دارد.
- بسیاری از سیستم‌های ساده دیجیتالی (مثل ماشین‌حساب) فقط دارای رجیستر هستند و واحد حافظه آدرس‌پذیر ندارند. برعکس سیستم‌های کامپیوتر همگی دارای واحد حافظه هستند.

واحد حافظه RAM

- تعداد خطوط آدرس (k) نشان می‌دهد که حافظه چند خانه دارد. مثلا اگر ۴ خط آدرس وجود داشته باشد یعنی حافظه 16 خانه دارد. n هم اندازه هر خانه (کلمه) است و طبیعتا معادل تعداد خطوط دیتا ورودی و خروجی. مثلا ممکن است اینجا n=8 باشد.
- وجود خط مورب نشاندهنده این است که چند خط وجود دارد.
- فعال بودن خط خواندن یا خط نوشتن تعیین می‌کند که ما می‌خواهیم چه کاری با حافظه انجام دهیم.



زبان ماشین و اسمبلی

- زبان ماشین، زبان برنامه‌نویسی است که در آن بطور مستقیم دستورات توسط سخت افزار شناسایی و اجرا می‌شوند. این دستورات در مبنای دو بصورت صفر و یک در حافظه ذخیره و به ترتیب اجرا می‌شوند (صفر و یک ها مدارات منطقی را فعال می‌کنند). گاهی برای سادگی این دستورات (بجای مبنای۲) در مبنای شانزده نوشته می‌شوند.
- در زبان اسمبلی هر دستور زبان ماشین بجای یک عدد با یک نماد حرفی نوشته می‌شود. ولی بر خلاف زبان‌های سطح بالا که یک دستور ممکن است به چندین دستور زبان ماشین ترجمه شود، هر دستور اسمبلی دقیقا معادل یک دستور زبان ماشین است.

زبان ماشین و اسمبلی کامپیوتر PC

- شناخت ساختار ماشین و برنامه نویسی اسمبلی را جلسه آینده با کامپیوتر PC کار خواهیم کرد.
- بر روی کامپیوتر می‌توان با استفاده از برخی برنامه‌های تحت داس مثل MASM و LINK و DEBUG برنامه‌های اسمبلی نوشت و اجرا کرد و تست نمود.
- اما راه آسانتر برنامه امولیتور تحت ویندوز بنام emu8086 است. ۸۰۸۶ جد همه کامپیوترهای PC است که امروزه در نسل پنتیوم است. تمامی کامپیوترهای PC با نسل‌های قبل از خود سازگارند، یعنی تمام PCهای امروزی می‌توانند برنامه‌های 8086 را اجرا کنند.

پیش ارائه جلسات درس

- مقدمه هر جلسه از درس اهمیت زیادی دارد. در این مقدمه گفته می‌شود که قرار است امروز چه مطالبی ارائه شود و این مطالب به چه کاری می‌آید.
- پیش ارائه به صورت فیلمی کوتاه (چند دقیقه‌ای) هر هفته در روز قبل از درس ارسال می‌گردد. توصیه می‌شود این فیلم را در زمانی که فرصت و تمرکز ذهنی دارید مشاهده کنید تا موضوع در ذهن‌تان جای گیرد و برای درک درسی که فردا در کلاس ارائه می‌شود آمادگی خوبی پیدا نمایید.

نمره پایان ترم

- **ارزیابی مستمر در طول کلاس (۵ نمره):**

✓ تمرین‌های هفتگی ۵ نمره
توجه: نمره تمریناتی که اصالت نداشته باشند در پایان ترم حذف خواهد شد. فرقی بین دهنده و گیرنده تمرین نیست. تقلب در تمرین و امتحان در نهایت مرتکبین و کل کلاس را متضرر خواهد کرد. توضیح بیشتر در وب سایت موجود است.

✓ کوئیز (در ترم معمولا ۳ یا ۴ کوییز خواهد بود که کلا) ۱ نمره دارد

✓ امتحان پایان ترم ۱۴ نمره

✓ نمره ارفاقی به همه بطور مساوی (مقدار آن آخر ترم معلوم می‌شود)

✓ در صورتی که امتحان پایان ترم به هر دلیلی مجازی شود، نمره کلاسی از ۶ به ۱۰ نمره افزایش خواهد یافت.

کوییزهای انفرادی و گروهی

- برخی کوییزها انفرادی هستند و برخی بصورت گروهی برگزار می‌شوند.
- قوانین: دانشجویان بطور تصادفی در گروه‌هایی چند نفره قرار خواهند گرفت. می‌توانید با کمک هم به حل مساله بپردازید و در نهایت یک برگه با ذکر شماره گروه و نام همه اعضا گروه تحویل داده خواهد شد و به همه نمره یکسان تعلق می‌گیرد.
- در موارد زیر نمره صفر به همه اعضای گروه تعلق خواهد گرفت:
  ۱- عدم تحویل پاسخ.
  ۲- ارسال بیش از یک پاسخنامه از یک گروه.
  ۳- اگر نام گروه و اعضای آن با آنچه هنگام تعیین گروه‌ها اعلام شده متفاوت باشد (کم یا زیاد باشد).
- در کلاس‌های حضوری گروه‌بندی توسط من در کلاس انجام می‌شود و گروه‌ها دور هم نشسته و سوال را با مشورت هم حل می‌کنند.

تمرین‌ها

- تمرین‌ها: هر هفته به صورت مرتب. زمان تحویل تا فقط هفته آینده شش صبح روز کلاس. تمرین تاریخ گذشته ارزشی ندارد.
- برای اطلاع از جزییات شرایط تحویل تمرین‌ها و نحوه استفاده از سیستم گوگل کلاس به سایت من مراجعه شود whiteboard.ir
- پلتفرم ارتباطی کلاس برای دریافت اسلایدها و ارسال تمرین‌ها گوگل کلاس و برای پرسش‌های کلاسی گوگل چت است. لازم است این دو اپلیکیشن (خصوصا اولی) را روی گوشی نصب داشته باشید و راهنمای استفاده از آنها در سایت وایت‌برد موجود است.
- کد پیوستن به کلاس معماری:
کنتکت ارتباطی گوگل چت:
- دریافت تمرین فقط از طریق سیستم گوگل کلاس و گفتگوی درسی فقط از طریق گوگل چت به آدرس فوق می‌باشد. تمرین دریافتی از طریق ایمیل قابل تصحیح نیست و مباحث درسی از طریق سایر ایمیل آدرس‌ها ممکن نمی‌باشد.

معماری کامپیوتر

جلسه دوم

شروع زبان ماشین PC

زبان ماشین و اسمبلی

- زبان ماشین، زبان برنامه‌نویسی است که در آن بطور مستقیم دستورات توسط سخت افزار شناسایی و اجرا می‌شوند. این دستورات در مبنای دو بصورت صفر و یک در حافظه ذخیره و به ترتیب فراخوانی و سپس اجرا می‌گردند (صفر و یک ها مدارات منطقی را فعال می‌کنند). معمولا برای سادگی این دستورات روی کاغذ در مبنای شانزده نوشته می‌شوند.
- در زبان اسمبلی هر دستور زبان ماشین بجای یک عدد با یک نماد حرفی نوشته می‌شود. خانه‌های مورد استفاده حافظه هم نام‌گذاری می‌شوند.
- بر خلاف زبان‌های سطح بالا که یک دستور ممکن است به چندین دستور زبان ماشین ترجمه شود، هر دستور اسمبلی تقریبا همیشه معادل یک دستور زبان ماشین است.

وابستگی برنامه‌نویسی به سخت‌افزار

- زمانی که به زبان‌های سطح بالا مثل جاوا و C و پایتون برنامه می‌نویسید، سخت افزار مهم نیست (مثلا مهم نیست روی PC هستید یا مکینتاش).
- زمانی که به زبان ماشین یا اسمبلی برنامه می‌نویسید کاملا وابسته به سخت افزار هستید. یعنی یک برنامه اسمبلی کامپیوتر PC با برنامه اسمبلی کامپیوتر دیگر زمین تا آسمان متفاوت است. ضمن آنکه تا زمانی که سخت‌افزار آن کامپیوتر را نشناسید اصلا نمی‌توانید بر روی آن برنامه‌نویسی سطح پایین انجام دهید.
- یادگیری زبان ماشین و اسمبلی یک کامپیوتر (مثل PC) به ما این توانایی را می‌دهد که طرز کار کامپیوتر را بهتر درک کنیم و آمادگی درک معماری کامپیوتر را داشته باشیم.

مقایسه زبان‌ها

High-level Language

```
temp    = v[k];
v[k]    = v[k+1];
v[k+1]  = temp;
```

```
TEMP  = V(K)
V(K)  = V(K+1)
V(K+1) = TEMP
```

C/Java Compiler

Fortran Compiler

Assembly Language

```
lw  $to,   0($2)
lw  $t1,   4($2)
sw  $t1,   0($2)
sw  $t0,   4($2)
```

MIPS Assembler

Machine Language

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

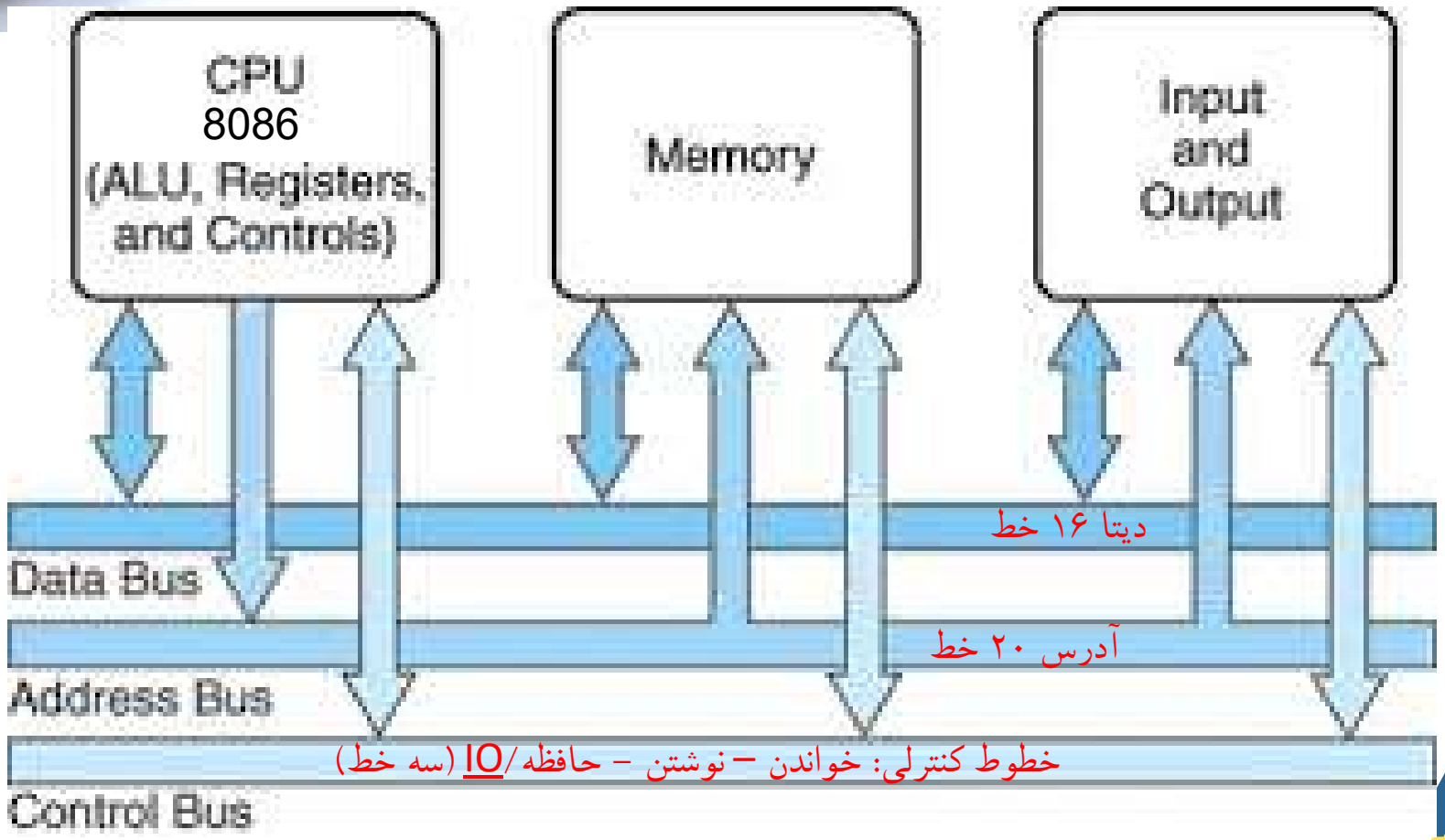
پردازنده ۸۰۸۶ و کامپیوتر PC

- PC XT با پردازنده (8) 8086 جد تمام کامپیوترهای PC است و همچنان همه PC ها با آن **سازگار**ند (بعدا به تاریخچه بصورت کامل‌تر خواهیم پرداخت).
- در زمانه‌ای که مفهوم ماشین مجازی مطرح نبود، سازگاری کامپیوترها بسیار اهمیت داشت چون اگر سازگاری نبود برنامه‌ای که برای یک کامپیوتر نوشته شده به درد اجرا روی کامپیوترهای دیگر اجرا نمی‌خورد. (با ماشین مجازی می‌توان روی سخت افزارهای ناسازگار سیستم واحدی مانند اندروید را شبیه‌سازی کرد)
- ما در این جا می‌خواهیم با برنامه‌نویسی به زبان اسمبلی بر روی کامپیوتر PC آشنا شویم.

ساختار ارتباطی ۸۰۸۶

- هر کامپیوتر (از جمله کامپیوتر PC) تشکیل شده است از یک پردازنده مرکزی CPU، یک واحد حافظه اصلی و تعدادی دستگاه‌های ورودی و خروجی.
- از آنجا که کامپیوتر همه منظوره است باید بتواند به دستگاه‌های جانبی متنوعی وصل شود و انواع حافظه را پشتیبانی کند.
- به همین خاطر برای ارتباط پردازنده و حافظه و دستگاه‌های جانبی از سیستمی بنام کانال مشترک یا باس استفاده می‌شود.
- مثلا موقع اجرای دستور a = 6 هنگام اجرای برنامه باید مقدار ۶ (که داخل پردازنده است) به خانه‌ای از حافظه (که بیرون پردازنده است) ارسال شود.

باس‌ها در ۸۰۸۶



این ساختار باس به چه معناست؟

- همانطور که در شکل می‌بینید باس داده ۱۶ خط است. بنابراین داده در بسته‌های دوبایتی بین پردازنده و حافظه و دستگاه‌های جانبی جابجا می‌شود. همه می‌توانند با استفاده از باس داده (کانال مشترک داده)، فرستنده یا گیرنده باشند.
- حافظه واحدی است که درش تعدادی مکان دارد. در هر مکان داده‌ای (۱۶ بیتی) قابل ذخیره و یا خواندن است. هر مکان دارای یک آدرس است.
- باس آدرس ۲۰ خط است. یعنی این کامپیوتر می‌تواند حداکثر از یک مگابایت حافظه RAM استفاده کند. فقط CPU است که روی این باس ارسال کننده است و بقیه فقط از آن می‌خوانند.
- باس کنترل حاوی بیت‌هایی است. اینکه منظور خواندن است یا نوشتن؟ یا حافظه منظور است یا دستگاه‌های جانبی؟

سگمنت‌ها

- هر برنامه در حال اجرا (دست‌کم) به داشتن سه قسمت در حافظه اصلی کامپیوتر نیاز دارد که به این قسمت‌ها سگمنت می‌گویند.
- ۱. سگمنت کد که برنامه در حال اجرا (به زبان ماشین) در آن مستقر می‌شود و کامپیوتر دستورات را یک به یک اجرا می‌کند.
- ۲. سگمنت دیتا که متغیرها و آرایه‌های تعریف شده در برنامه اصلی در آن قرار می‌گیرند. هر متغیر (که معمولا در ابتدای برنامه تعریف می‌شود) جایی متناسب با اندازه خود در این سگمنت خواهد داشت.
- ۳. سگمنت پشته که به آن جداگانه خواهیم پرداخت.

کامپیوتر چند منظوره:

کامپیوتر ساده تک منظوره:

سگمنت کد

سگمنت داده

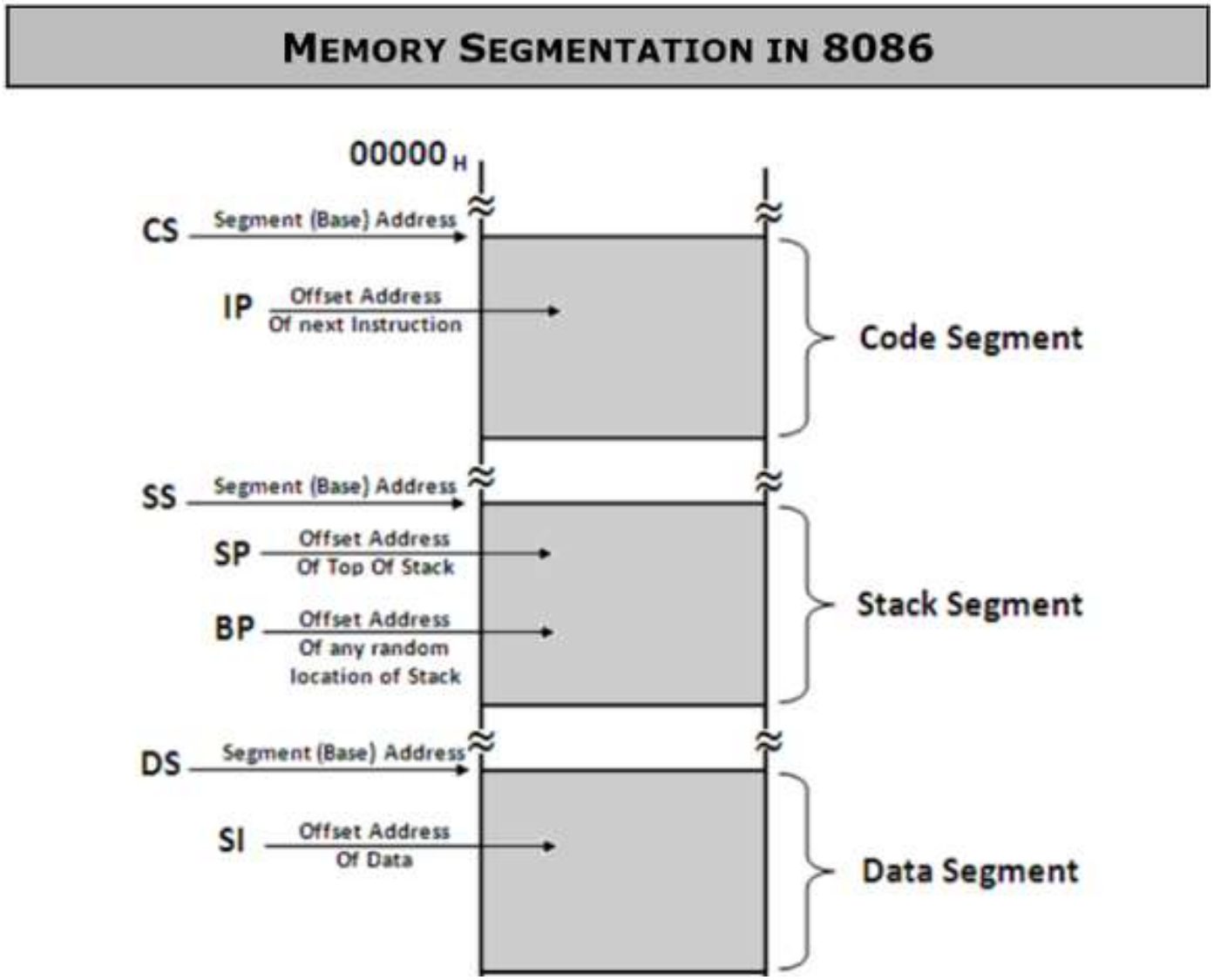
سگمنت پشته

مربوط سیستم‌عامل یا برنامه‌های دیگری که بطور موازی در حال اجرا هستند.

سگمنت‌بندی حافظه در کامپیوتر PC

- می‌دانیم PC یک کامپیوتر چند منظوره است. اندازه هر کدام از سگمنت‌ها در این کامپیوتر ۶۴ کیلو بایت است (این درحالی است که کل حافظه حداکثر ۱ مگابایت است).
- همانطور که در اسلاید قبل می‌بینید، سگمنت‌ها جای مشخصی ندارند و هر بار که برنامه اجرا می‌شود سیستم عامل جایی که می‌داند خالی است را به آنها اختصاص می‌دهد. به این علت که PC کامپیوتری است همه منظوره، ممکن است بخش‌هایی از حافظه از قبل اشغال باشد (توسط برنامه‌های دیگر یا توسط خود سیستم عامل).
- در کامپیوترهای ساده و تک منظوره می‌توان حافظه را از قبل بین سگمنت‌ها تقسیم کرد به گونه‌ای که همیشه جای‌شان ثابت باشد. علت آن است که در چنین کامپیوتری نه سیستم عامل هست نه اجرای چند برنامه همزمان. کل حافظه بصورت دربست در اختیار یک برنامه است و سیستم هم با همان برنامه بالا می‌آید (بوت می‌شود).

سگمنت‌بندی حافظه در کامپیوتر PC



محدودیت‌های کامپیوتر PC

- کامپیوتر PC XT که در آن از پردازنده 8086 استفاده شده کامپیوتری است ۱۶ بیتی. یعنی اندازه تمام رجیسترها ۱۶ بیتی است و عملیات حافظه و عملیات ریاضی (مانند جمع) در همین اندازه صورت می‌گیرد.
- با توجه به محدود بودن اندازه سگمنت‌ها به ۶۴ کیلوبایت (در حالت معمولی) کد اجرایی هیچ برنامه‌ای نمی‌تواند بیش از ۶۴ کیلوبایت باشد و همینطور اندازه کل متغیرهای برنامه.
- البته برای غلبه بر این محدودیت راه حل‌هایی وجود دارد و یک سگمنت بنام سگمنت اضافه (extra segment) هم در ۸۰۸۶ هست که می‌تواند برای این منظور استفاده شود که ما در این درس با این مسایل کاری نداریم.

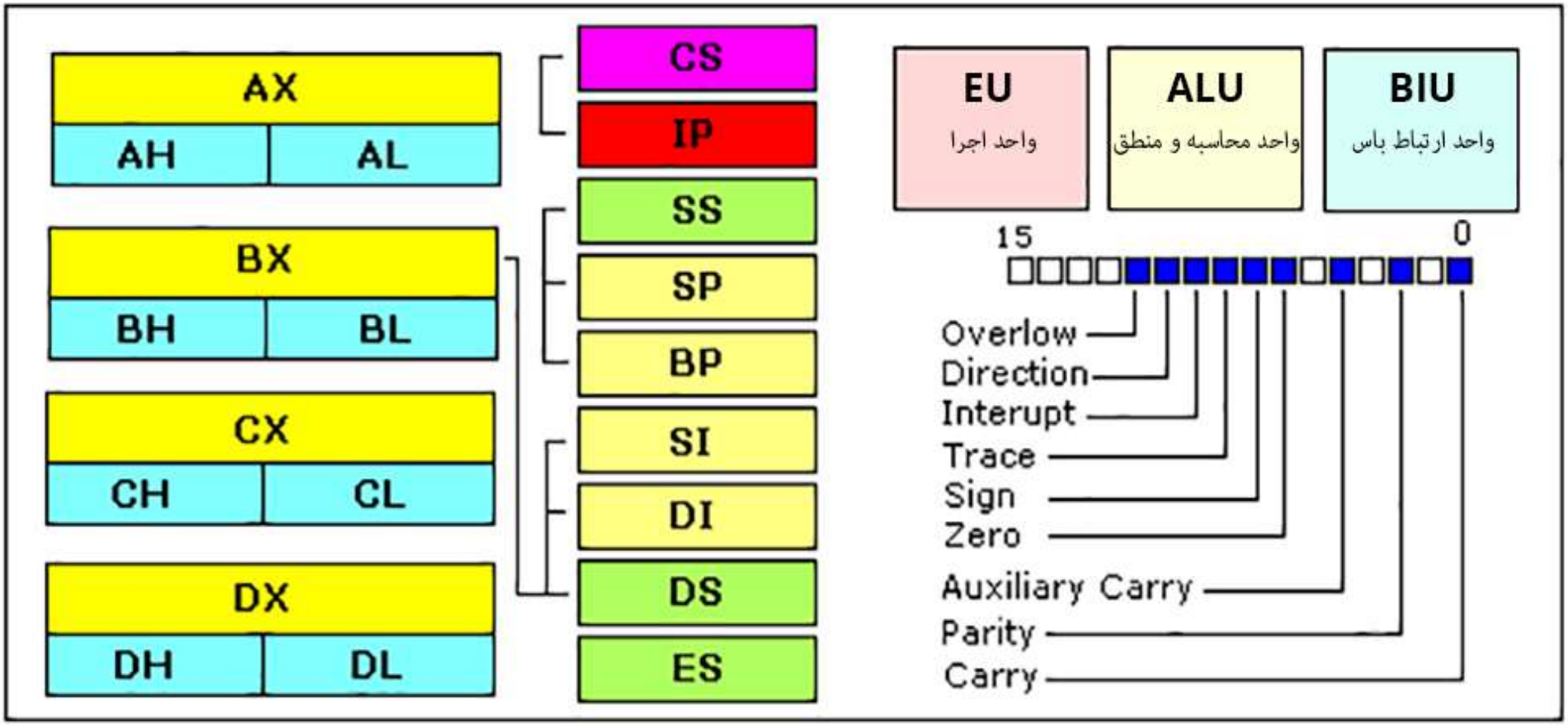
ماشین حساب حداقل چند رجیستر دارد؟

رجیسترها

- برخی سیستم‌های دیجیتال برای ذخیره مقادیر دم دستی رجیستر دارند. مثلا ماشین‌حساب ساده حداقل دو رجیستر دارد. یکی برای مقداری که الان از کیبورد وارد شده و یکی عددی که قبلا محاسبه گردیده (با احتساب M سه تا).
- در کامپیوتر، محل رجیسترها داخل CPU است. آنها با نام شناخته می‌شوند و مانند خانه‌های حافظه آدرس ندارند اما داده‌های مورد نیاز پردازنده را در خود نگه می‌دارند.
- رجیستر (که در کتاب‌های فارسی ثبات ترجمه شده!!) در هر پردازنده به تعداد اندکی (مثلا ده تا) موجود است.
- تعداد و نوع رجیسترهای هر کامپیوتر با رجیسترهای کامپیوتری که از نوع دیگر است متفاوت است. این تفاوت یکی از اصلی‌ترین دلایل تفاوت کامپیوترهاست.
- با دستورات برنامه زبان ماشین می‌توانیم محتوای رجیسترها را به همدیگر و نیز به حافظه منتقل کنیم و نیز روی آنها اعمال حسابی (مثل جمع و تفریق) انجام دهیم.

رجیسترها و واحدهای ۸۰۸۶

- چهار رجیستر داده (سمت چپ)، نُه رجیستر آدرس (پوینتر و اشاره‌گر) که همگی ۱۶ بیتی هستند و نه بیت پرچم.



رجیسترهای عمومی

- چهارتایند به نام‌های AX و BX و CX و DX. مانند همه رجیسترها ۱۶ بیتی‌اند اما اگر لازم باشد بصورت نصفه هم قابل استفاده‌اند. مثلا نیمه بالایی (سمت چپ) رجیستر AX به نام AH خوانده می‌شود.
- از این رجیسترها آزادانه برای ذخیره داده و انجام عملیات ریاضی استفاده می‌شود.

رجیسترهای سگمنت

- چهارتایند که عبارتند از CS ، DS ، SS و ES که آدرس شروع کد سگمنت، دیتا سگمنت، استک سگمنت و اکسترا سگمنت در آنها قرار می‌گیرد.
- همانطور که گفتیم سیستم‌عامل است که محل سگمنت‌ها را تعیین می‌کند و در نتیجه این رجیسترها را مقداردهی می‌کند و ما موقع برنامه‌نویسی با آنها کاری نداریم.
- همانطور که می‌دانیم حافظه یک مگابایتی است و در واقع هر خانه واقعی حافظه یک آدرس ۲۰ بیتی دارد. اما رجیسترها ۱۶ بیتی هستند و نمی‌تواند یک آدرس ۲۰ بیتی را در خود نگه دارند. برای اینکه مشکل حل شود یک 0 در مبنای ۱۶ (یا چهار 0 در مبنای ۲) به سمت راست آدرسی که رجستر سگمنت هست افزوده می‌شود. مثلا اگر محتوای رجیستر CS باشد 09A1 H آنگاه محل واقعی شروع سگمنت کد می شود 09A10 H. به عبارت دیگر آدرس محل شروع سگمنت‌ها همیشه روند و آخرش صفر (بر ۱۶ بخشپذیر) است.

رجیسترهای اشاره‌گر

- این رجیسترها به خانه‌های داخل سگمنت‌ها اشاره می‌کنند و دیتای داخل آنها را در اختیار سیستم قرار می‌دهند.
- اشاره‌گر دستورالعمل ip یا instruction pointer که برخی به آن program counter (pc) هم می‌گویند نشان می‌دهد که برنامه تا کجایش اجرا شده و دستورالعمل زبان ماشین بعدی که قرار است اجرا شود کدام است.
- برای دسترسی به یک متغیر از آدرس آن (یا نامی که توسط اسمبلر با آدرس جایگزین می‌شود) استفاده می‌کنند، اما رجیسترهای si و di در مواردی چون تفاوت مکان آرایه‌ها کاربرد دارند.
- اشاره‌گرهای پشته استک پوینتر sp و بیس پوینتر bp.

پرچم‌ها

- تعدادی بیت هستند که اجرای دستورات برنامه بصورت خودکار روی آنها اثر می گذارد.
- CF اگر آخرین بیت محاسبه ایجاد نقلی (کری) کرده باشد ست وگرنه ریست.
- ZF اگر نتیجه محاسبه صفر (زیرو) باشد ست وگرنه ریست.
- SF اگر نتیجه محاسبه منفی (ساین) باشد ست وگرنه ریست.
- OF اگر محاسبه ایجاد سرریز (اورفلو) کرده باشد ست وگرنه ریست.
- بقیه فلگ‌ها چندان مورد نظر ما نیستند مثلا پریتی فلگ با تعداد فرد یک در نتیجه و پرچم کری اضافه با کری بیت وسطی و محاسبه ست می‌شود و....

زبان ماشین و اسمبلی کامپیوتر PC

- بر روی کامپیوتر می‌توان با استفاده از برخی برنامه‌های تحت داس مثل MASM و LINK و DEBUG برنامه‌های اسمبلی نوشت و اجرا کرد و تست نمود.
- اما راه آسانتر برنامه امولیتور تحت ویندوز بنام emu8086 است. (همه کامپیوترهای PC که امروزه در نسل پنتیوم است به ۸۰۸۶ سازگارند.)
- بعد از نصب این برنامه کار با آن را شروع می‌کنیم.
- فیلمی کوتاه در مورد شروع کار با emu8086 را می‌توانید در اینجا ببینید:

https://www.aparat.com/v/hSbGE

شروع کار با emu8086

- پس از اجرای برنامه new و بعد از آن EXE template را انتخاب کنید. یک برنامه نمونه به سه سگمنت ظاهر می‌شود که یک متغیر بنام pkey، یک پشته به اندازه ۱۲۸ ورد و ده خط برنامه اسمبلی در آن وجود دارد که همه اینها را می‌توانید تغییر دهید.
- توصیه می‌شود اندازه پشته را عوض نکنید. همچنین دو خط اول یعنی

```
mov ax, data
mov ds, ax
```

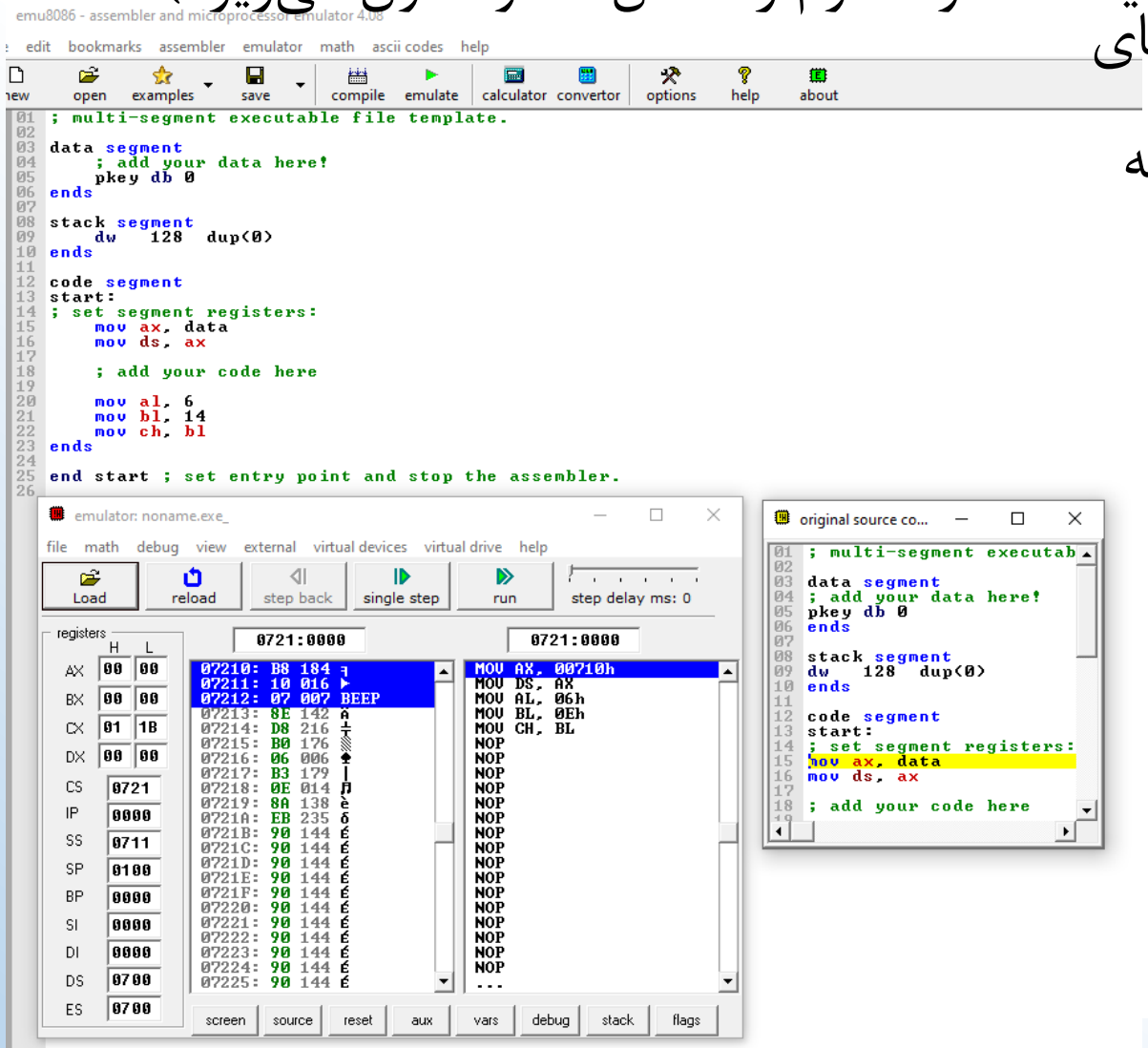
در همه برنامه‌هایی که سگمنت دیتا دارند، لازم است. چون سیستم عامل این سگمنت را (بر خلاف پشته و کد) بطور خودکار مقداردهی نمی‌کند.

- با زدن فلش سبز رنگ emulate حالت اجرا آغاز می‌شود و با هر بار زدن روی single step اجرا یک خط جلو می‌رود.
- محتوای تمام لحظه‌ای رجیسترها را در سمت چپ مشاهده می‌کنید همچنین با زدن vars و stack و flags می‌توانید محتوای متغیرها، پشته و پرچم‌ها را ببینید.

مثال

- برنامه‌ای بنویسید که عدد ۶ را داخل al و عدد ۱۴ را داخل bl بریزد. سپس محتوای bl را به ch کپی کند. (دستور mov در زبان اسمبلی دستوری است که همیشه عملوند دوم را داخل عملوند اول می‌ریزد)

به محتوای رجیسترهای عمومی ax ,bx,cx و همچنین ip الان که ابتدای برنامه هستیم دقت کنید.



پس از اجرای تمام دستورات

emu8086 - assembler and microprocessor emulator 4.08

edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
01 ; multi-segment executable file template.
02
03 data segment
04     ; add your data here!
05     pkey db 0
06 ends
07
08 stack segment
09     dw   128  dup(0)
10 ends
11
12 code segment
13 start:
14 ; set segment registers:
15     mov ax, data
16     mov ds, ax
17
18     ; add your code here
19
20     mov al, 6
21     mov bl, 14
22     mov ch, bl
23 ends
24
25 end start ; set entry point and stop the assembler.
26
```

emulator: noname.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	07	06
BX	00	0E
CX	0E	1B
DX	00	00

CS	0721
IP	000B
SS	0711
SP	0100
BP	0000
SI	0000
DI	0000
DS	0710
ES	0700

0721:000B

```
07210: B8 184 ╕
07211: 10 016 ►
07212: 07 007 BEEP
07213: 8E 142 Ä
07214: D8 216 ╪
07215: B0 176 ░
07216: 06 006 ♠
07217: B3 179 │
07218: 0E 014 ♫
07219: 8A 138 è
0721A: EB 235 δ
0721B: 90 144 É
0721C: 90 144 É
0721D: 90 144 É
0721E: 90 144 É
0721F: 90 144 É
07220: 90 144 É
07221: 90 144 É
07222: 90 144 É
07223: 90 144 É
07224: 90 144 É
07225: 90 144 É
```

0721:000B

```
MOV AX, 00710h
MOV DS, AX
MOV AL, 06h
MOV BL, 0Eh
MOV CH, BL
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

screen source reset aux vars debug stack flags

original source co...

```
10 ends
11
12 code segment
13 start:
14 ; set segment registers:
15 mov ax, data
16 mov ds, ax
17
18 ; add your code here
19
20 mov al, 6
21 mov bl, 14
22 mov ch, bl
23 ends
24
25 end start ; set entry po
26
27
```

شرح اجرای برنامه

- اول کار محتوای رجیسترهای داده اعداد تصادفی و بی‌معنا هستند. مقدار IP هم 0 است یعنی در حال اجرای اولین دستور از سگمنت کد هستید که طبعا در خانه صفر این سگمنت قرار دارد. داخل سگمنت داده هم یک متغیر از نوع بایت (اندازه ۸ بیتی) با مقدار اولیه 0 تعریف شده که در برنامه از آن استفاده نشده است.
- پس از پنج بار زدن فلش سبزرنگ single step هر پنج خط برنامه اجرا شده است. al، bl و ch همانطور که صورت سوال از ما خواسته مقداردهی شده‌اند و همینطور IP دارای مقدار 11 (000B H) است که معنایش این است که این پنج دستور در کل 11 بایت طول داشته‌اند.

تمرین

- اول: برنامه ای بنویسید که 7 را داخل AX و aaaa H را داخل BX قرار دهد و سپس با واسطه قرار دادن یکی از رجیسترها محتویات AX و BX را با هم عوض نماید.
(قسمت اول تمرین را همانند مثال با ارسال دو تصویر از اسکرین انجام دهید، نوشتار دستی پذیرفته نیست)
- دوم: منظور از سگمنت و رجیستر چیست؟ در برنامه قسمت اول کدام سگمنت‌ها و کدام رجیسترها مورد استفاده بوده‌اند.
(در سه الی پنج سطر قسمت دوم را پاسخ دهید).

معماری کامپیوتر

جلسه سوم

محتوای سگمنت‌ها
دستورالعمل‌های انتقال

سگمنت کد

- دستورات اسمبلی PC بعد از اینکه به سادگی اسمبل شده و تبدیل به زبان ماشین گردیدند در این سگمنت جای گرفته و یک به یک اجرا می‌شوند.
- در 8086 اندازه دستورات متفاوت بوده و یک دستور زبان ماشین بین یک تا شش بایت طول دارد.
- رجیستر IP (اشاره‌گر دستورالعمل) همیشه به دستوری که قرار است بعد از این اجرا شود اشاره می‌کند. مقدار این اشاره‌گر معمولا به هنگام اجرای دستورات یکی یکی (یک دستور یک دستور) جلو می‌رود مگر آنکه پرش jump پیش آید.

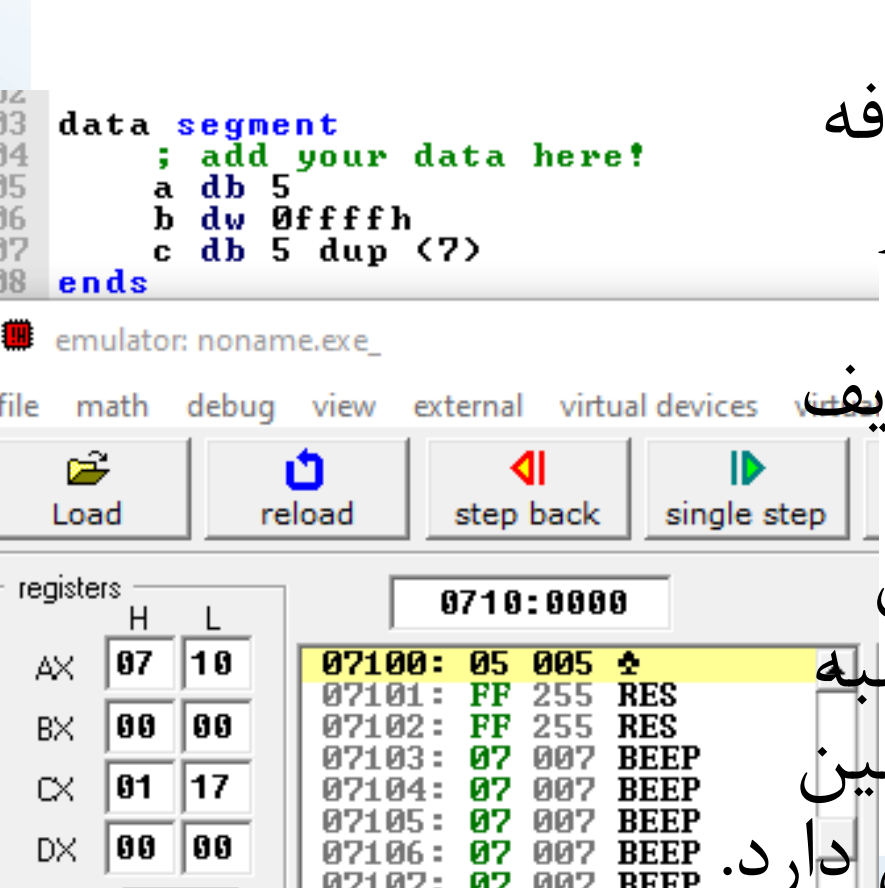
سگمنت دیتا

- متغیرهایی که قرار است در برنامه استفاده شوند در این سگمنت جای دارند. در زبان ماشین متغیرها فقط یک آدرس (آفست) دارند اما در اسمبلی می توان برای آنها نام داشت و در زمان اسمبل آدرس جایگزین نام شود.
- به ظاهر با تغییر مقدار سگمنت‌ها و اشاره‌گرها امکان دارد برنامه به همه حافظه دسترسی داشته باشد. اما در عمل سیستم عامل مانع آن می‌شود که برنامه‌ای به سگمنت‌های برنامه دیگر دست درازی کند (وقتی چند برنامه بصورت موازی اجرا می‌شوند پردازنده و رجیسترها در یک زمان مشخص به هر برنامه داده می‌شوند اما محتوای سگمنت‌ها در حافظه تا پایان اجرای برنامه باقی می‌ماند.)
- در زبان ماشین و اسمبلی متغیرها نوع خاصی ندارند. آنها محتوی تعدادی صفر و یک هستند و خود برنامه نویس می‌داند آنها چه هستند و چه هدفی را برآورده می‌کنند.
- هر چند آنها نوع ندارند (int, float, char, ... نیستند) اما اندازه دارند. B برای بایت (۸ بیتی) W برای ورد (۱۶ بیتی) و D برای دابل ورد (۳۲ بیتی) است.

سگمنت دیتا

- به مثال زیر توجه کنید. یک متغیر بنام a از نوع بایت با مقدار اولیه 5 سپس یک متغیر بنام b از نوع ورد با مقدار اولیه تماما 1 (ffff) (برای اینکه مقدار عددی با کاراکتر شروع نشود یک 0 اول آن اضافه شده) و سپس یک متغیر بنام c از نوع بایت با پنج مقدار متوالی 7 (آرایه) تعریف شده که محتوای حافظه را می‌بینید و بدون دیدن هم بایستی بتوانید محاسبه کنید که مثلا آدرس دومین عنصر آرایه c چه آدرسی دارد.

```
data segment
    ; add your data here!
    a db 5
    b dw 0ffffh
    c db 5 dup (7)
ends
```

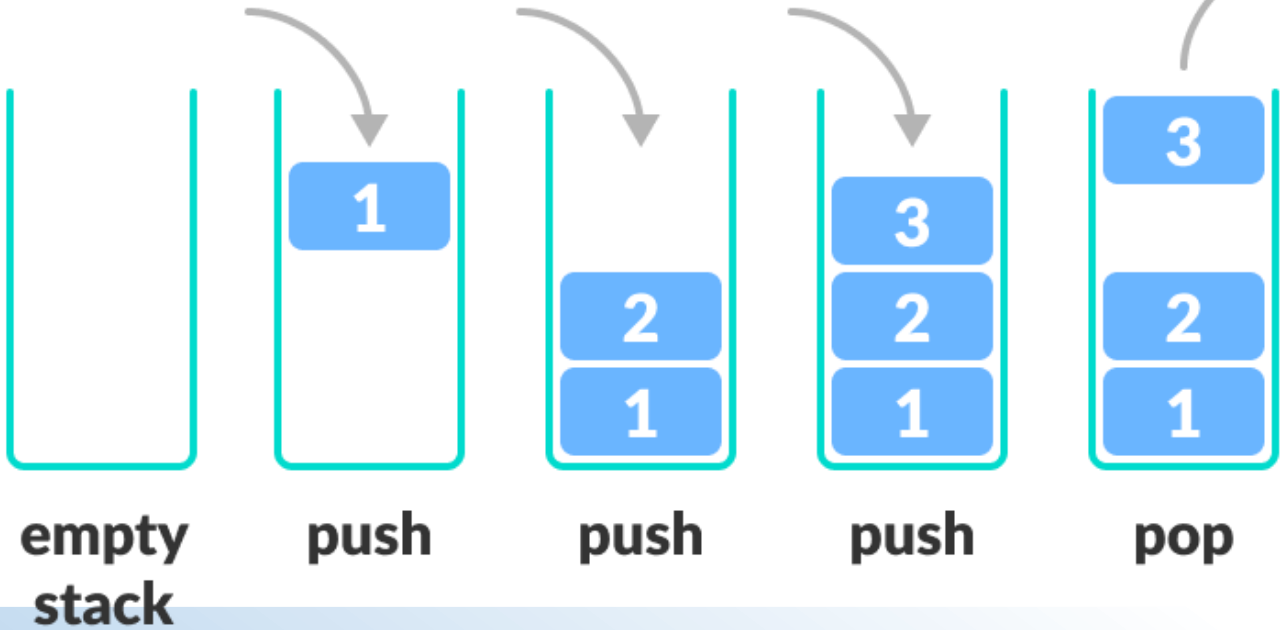


پشته

- پشته یا استک stack یک ساختمان داده است. در یک ساختمان داده بجای اینکه داده‌ها را بصورت ساده و با یک آدرس در حافظه ذخیره کنیم، به آنها شکل منظم‌تری می‌دهیم. از جمله ساختمان‌های داده می‌توان آرایه، صف، پشته، لیست پیوندی و درخت را نام برد.
- بر خلاف آرایه که ساختاری با اندازه ثابت است، طول پشته متغیر است و هر چقدر داده‌ای که ما در آن وارد می‌کنیم بیشتر شود طول پشته هم افزایش خواهد یافت (البته از آنجا که کامپیوتر ماشینی محدود است پشته هم لاجرم زمانی پر می‌شود و دیگر نمی‌تواند رشد کند.)

پشته

- با عمل push می‌توان مقداری به پشته اضافه کرد و با عمل pop می‌توان مقداری از پشته حذف کرد و آن را خواند. البته می‌توان مقدارهای میانی پشته را هم خواند اما حذف کردن فقط روی خانه آخر انجام می‌شود.
- دسترسی به اطلاعات پشته LIFO است یعنی هر کس که آخر همه وارد شده اول همه خارج می‌شود.
- اما چرا این ساختار داده‌ای، پشته، این‌قدر مهم شده که برای هر برنامه در حال اجرا یک پشته آماده می‌شود؟



مساله صدازدن پروسجرها

```
5   int main()
6   {
7       int  i;
8       i = 0;
9   a:  .
10      .
11      .
12      .
13      i++;
14      if (i<=10) goto a;
```

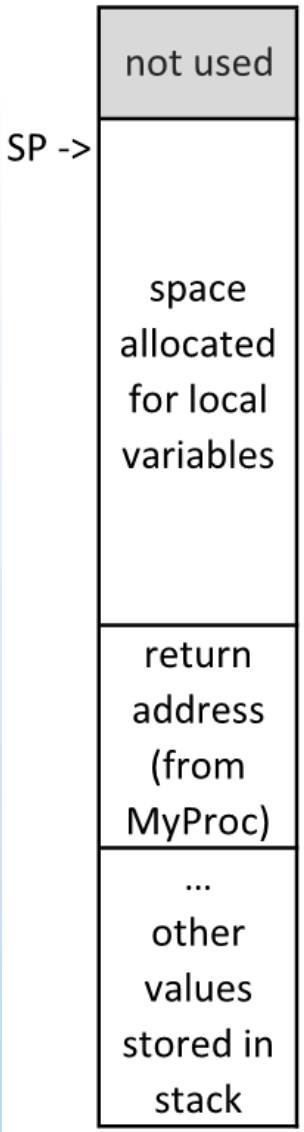
- در زبان ماشین دستورات ساخت یافته انتقال مانند for, while, .. چندان مطرح نیست و انتقال‌ها بصورت jump یا همان goto خودمان صورت می‌گیرد. در مثال بالا می‌بینید انجام این کار بسیار ساده است.
- اما اگر بخواهیم صدا زدن پروسیجرها را هم به همین صورت پیاده کنیم چه؟؟ یعنی بجای دستور call با یک jump محل اجرای برنامه به اول پروسجر منتقل شود. آیا امکان‌پذیر است یا مشکلی به وجود می‌آید؟

مساله صدازدن پروسجرها

```
5   int main()
6   {
7       .
8       .
9       call a (goto a)
10  x:  .
11      .
12      call a (goto a)
13  y : .
14      .
15      .
16  a:  . (procedure start)
17      .
18      .
19  return (goto ??? x? y?)
20  }
```

- هنگام call مشکلی نیست و به راحتی با jump به ابتدای پروسیجر منتقل می‌شویم. مشکل وقتی است که می‌خواهیم آخر کار پروسیجر برگردیم (return) در این زمان سیستم نمی‌داند که باید به کجا برگردد.
- اگر جایی باشد که زمان رفتن به محل پروسیجر، آدرس برگشت را در آن ذخیره کنیم مشکل تقریبا حل می‌شود.
- اما ذخیره یک آدرس مشکل را کاملا حل نمی‌کند. چرا که فراخوانی‌های تو در تو و حتی خود فراخوانی (ریکرسیو) در برنامه‌ها ممکن است اتفاق بیافتد. بنابراین شاید لازم باشد چندین آدرس برگشت ذخیره شوند.
- برای این منظور پشته انتخاب مناسبی است چون آدرس‌های برگشت باید بشکل LIFO بازیابی شوند. یعنی آخرین آدرسی که ذخیره شده اولین آدرسی است که بازیابی می‌شود.

سگمنت پشته و پروسیجرها



- استفاده از پشته فقط به ذخیره آدرس بازگشت از پروسیجر محدود نمی‌شود.
- زمانی که پروسیجری صدا زده می‌شود مقادیری بعنوان پارامتر به آن ارسال می‌شود و در پایان کار نیز ممکن است مقداری بعنوان نتیجه تابع بازگردانده شود. تمام این مقادیر داخل پشته قرار داده می‌شوند و بواسطه آن پاس داده می‌شوند (البته اگر کم تعداد باشند رجیسترهای عمومی هم می‌توانند برای پاس دادن مقدار به کار آیند) .
- هر پروسیجر ممکن است دارای متغیرهای محلی باشد که پس از پایان کار پروسیجر لازم است فضای مربوط به آنها در حافظه آزاد شود. این متغیرها هم درون پشته جای داده می‌شوند.

سگمنت پشته

- دستور push یک مقدار را به انتهای پشته اضافه می‌کند (طول پشته را افزایش می‌دهد) و دستور pop مقداری را از ته پشته حذف می‌کند و این مقدار را بر می‌گرداند. مقدارهایی که در پشته ذخیره می‌شوند ورد (۱۶ بیتی) هستند.
- با پایان اجرای پروسیجر آدرس برگشت، متغیرهای محلی و مقدارهای پاس داده شده همگی از پشته پاپ می‌شوند و حافظه آزاد می‌گردد.
- رجیستر sp به آخرین خانه پشته اشاره دارد. اگر بخواهیم مقداری را از پشته بخوانیم یا تغییر دهیم بدون آنکه آن را از پشته حذف کنیم (معمولا متغیرهای محلی) از رجیستر bp برای اشاره به آن استفاده می‌کنیم.
- رشد پشته نامحدود نیست و بالاخره پر می‌شود. این اتفاق معمولا زمانی می‌افتد که تعداد زیادی فراخوانی تو در تو داشته باشیم. مثل یک تابع خودفراخوان بدون شرط پایان.

دستورالعمل‌های PC

- دستورالعمل به یک دستور زبان ماشین گفته می‌شود. این دستورات معمولا کارهای پایه (مثل انتقال) را انجام می‌دهند.
- دستورالعمل‌های کامپیوتر PC شامل دستورات:
  ۱. انتقال (انتساب) ۲. محاسبات ریاضی و منطقی
  ۳. پرش‌های شرطی و غیرشرطی
  ۴. تعریف و فراخوانی پروسیجرها ۵. وقفه‌ها می‌باشد.
- ما به برخی از دستورات موارد ۱ تا ۴ خواهیم پرداخت و همینطور سخت‌افزاری که چنین دستوراتی را انجام می‌دهد (به طور کلی، وارد طراحی کامپیوتر PC نمی‌شویم) مورد بحث خواهند بود. از آنجا که به وقفه‌های PC نمی‌پردازیم برنامه‌هایی که می‌نویسیم بدون ورودی و خروجی (کیبورد و مونیتور) خواهند بود و فقط بر رجیسترها و حافظه اثر می‌گذارند.

انواع عملوندها

- فرض کنید دستوری داریم:
mov ax, XXX
در اینجا عملوند اول رجیستر ax است اما بجای XXX که عملوند دوم است چه می‌توان داشت؟
- ۱. عملوند بلافصل (بی‌واسطه). یعنی خود مقدار را درون کد زبان ماشین بنویسیم مثلا 67
- ۲. عملوند رجیستر. یک رجیستر دیگر باشد. مثلا bx
- ۳. عملوند حافظه مستقیم. یک آدرس (آفست) در سگمنت داده که داخل کروشه نوشته شود مثل [0000h]. همچنین در اسمبلی می‌شود نام متغیری که در دیتا سگمنت تعریف شده را نوشت مثل a.
- ۴. عملوند حافظه غیرمستقیم. آدرس در کد برنامه نوشته نمی‌شود. این آدرس در جای دیگری (پوینتر) است. در PC (فقط) از رجیستر BX (بیس پوینتر) برای این کار استفاده (آدرس حافظه قبلا در آن قرار داده شده) و نوشته می‌شود [bx]
- ۵. عملوند حافظه غیر مستقیم با ایندکس. برای آرایه‌هاست و ازرجیسترهای si و di استفاده می‌کند. به آن نمی‌پردازیم.

دستورالعمل‌های انتقال

- دستور انتقال مقداری را از جایی (مبدا) به جایی دیگر (مقصد) کپی می‌نماید. برای این کار در دستور mov به این صورت استفاده می‌شود:

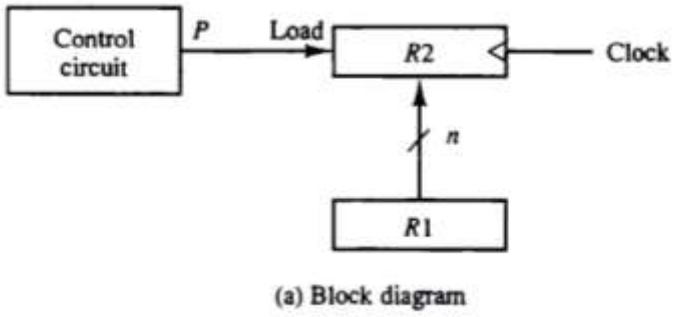
mov انتقال مقصد , انتقال مبدا

- به طور کلی انواع عملوند ها یعنی بی‌واسطه، رجیستر، حافظه مستقیم و حافظه غیر مستقیم می‌توانند مبدا و همه آنها (بدیهی است غیر از بی‌واسطه) می‌توانند مقصد باشد.
- در زبان‌های سطح بالا هیچ محدودیتی برای انتساب و محاسبات پیچیده نیست. اما ماشین هر کامپیوتری محدودیت‌هایی دارد و شاید همه انتقالات را پشتیبانی نکند (ممکن است مدارات آن اصلا پیاده سازی نشده باشد). البته کامپیوتر ۸۰۸۶ بسیار پیشرفته است و تقریبا همه انواع ممکن انتقال را انجام می‌دهد غیر از کارهای غیر ممکن از جمله اینکه مبدا و مقصد نمی‌توانند هر دو حافظه باشند.

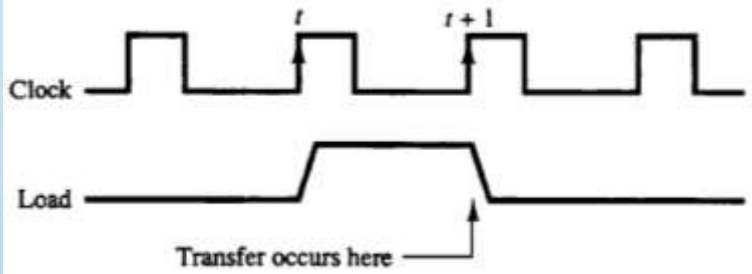
نحوه انتقال بین رجیسترها

- در این قسمت به نحوه پیاده‌سازی سخت‌افزاری انتقال می‌پردازیم. این بحث کلی در مورد همه کامپیوترها صادق است، نه فقط در مورد PC.
- در درس مدارهای منطقی طراحی سخت‌افزاری رجیسترها را آموختیم و دیدیم هر رجیستر n بیتی (با بارگیری موازی) دارای n خط ورودی داده و یک ورودی load است که در صورت یک (فعال) بودن load و رسیدن پالس ساعت مقدار خطوط ورودی بارگیری می‌شود. با این حساب اگر ورودی یک رجیستر به خروجی رجیستر دیگر وصل شود و خط load فعال باشد با رسیدن پالس ساعت انتقال انجام می‌شود.

mov R2, R1



(a) Block diagram



(b) Timing diagram

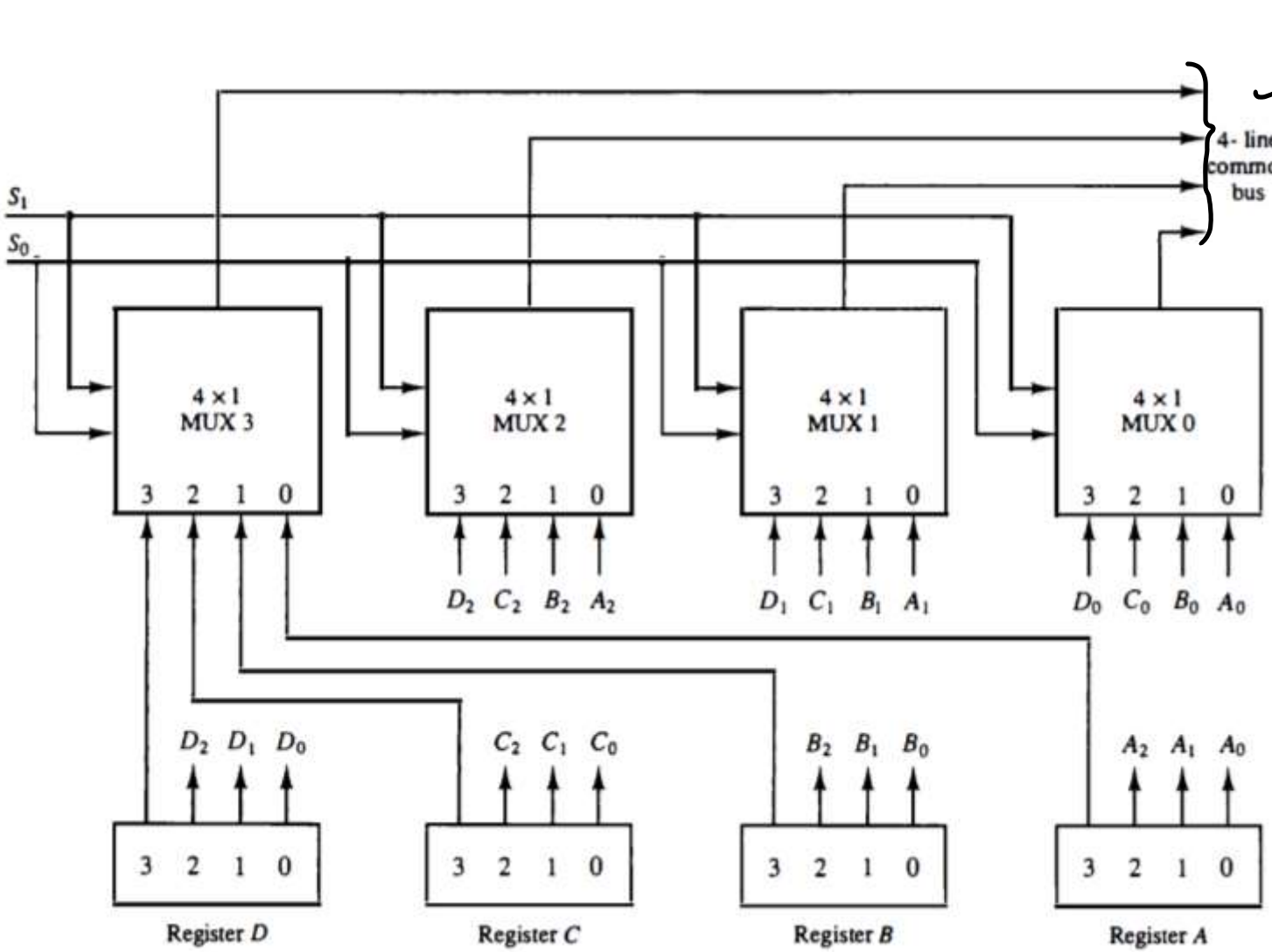
نحوه انتقال بین رجیسترها

- از آنجا که پردازنده تعدادی رجیستر دارد، سیم‌کشی مستقیم بین همه رجیسترها بسیار پیچیده و زیاد خواهد شد. راه حل ساده و کاراتر ایجاد یک گذرگاه مشترک (باس) بین رجیسترهاست (مشابه سیستمی که برای ارتباط پردازنده با واحدهای حافظه و خروجی شناختیم).
- در چنین شرایطی دو مساله وجود دارد که باید حل شود: ۱. چگونه فرستنده (رجیستر مبدا) اطلاعات را روی گذرگاه مشترک قرار دهد. ۲. چگونه گیرنده (رجیستر مقصد) اطلاعات را از روی باس بردارد.
- مورد دوم آسان است. در حالی که خطوط داده باس به ورودی همه رجیسترها وصل است، دستورالعمل انتقال فقط خط Load رجیستر گیرنده را فعال می‌کند.

نحوه انتقال بین رجیسترها

- می‌دانیم که در مدارهای منطقی امکان وصل کردن چند خروجی به هم وجود ندارد.
- با استفاده از مالتی‌پلکسر می‌توان چند ورودی مختلف داشت ولی انتخاب کرد که کدام ورودی روی خروجی (در واقع روی باس) قرار گیرد.
- در شکل اسلاید بعد می‌بینیم که با استفاده از ۴ عدد مالتی‌پلکسر ۴ بیتی می‌توان محتوای یکی از چهار رجیستر را روی باس منتقل کرد.
- برای مثال اگر قرار باشد که محتوای رجیستر B روی باس قرار گیرد (یعنی در دستور mov مبدا B باشد) آنگاه اجرای دستور باعث می‌شود که خط S0 =1 و خط S1=0 شود تا مالتی‌پلکسرها چنین کاری رانجام دهند.

نحوه انتقال بین رجیسترها



سایر انواع انتقال

- در انتقال بی‌واسطه مقداری که در کد دستور در حال اجراست بر روی گذرگاه قرار می‌گیرد.
- انتقال‌های حافظه و IO هم چه از نوع مستقیم باشند (یعنی آدرس حافظه در کد برنامه باشد) یا غیر مستقیم (یعنی آدرس حافظه در رجیسترها باشد)، تقریبا به همین صورت انجام می‌شود. با این تفاوت که خطوط انتقال باس در اینجا خارج از ریزپردازنده است.

انتقال در کامپیوتر PC

- به طور کلی معمار و طراح کامپیوتر الزامی ندارد که همه دستورات ممکن (از جمله همه انواع انتقال‌ها) را پیاده‌سازی کند، اما چون 8086 پردازنده پیچیده و قدرتمندی است تقریبا همه انواع انتقال در آن ممکن است بجز مواردی که استثنا و ممنوع شده است:
۱. زمانی که هر دو عملوند حافظه باشند (که از نظر فنی اصلا ممکن نیست). ۲. انتقال بی‌واسطه به رجیسترهای سگمنت یا انتقال بین رجیسترهای سگمنت (کاری بیهوده و مخرب. این کار را سیستم عامل باید انجام دهد.)
۳. انتقال مقدار به مقصد پرچم‌ها و IP (jump دستورات خودش را دارد با mov نباید انجام داد).
۴. زمانی که عملوندها هم اندازه نباشند.

مثال

- برنامه‌ای بنویسید که بدون ایجاد تغییر در رجیسترهای عمومی (ax...dx) ، مقدارهای a و b که دو خانه ۱۶ بیتی تعریف شده در حافظه هستند را با هم تعویض کند (سگمنت داده فقط شامل همین دو متغیر با مقدار دلخواه است).

مثال

- برنامه‌ای بنویسید که بدون ایجاد تغییر در رجسترهای عمومی (ax...dx) ، مقدارهای a و b که دو خانه ۱۶ بیتی تعریف شده در حافظه هستند را با هم تعویض کند (سگمنت داده فقط شامل همین دو متغیر با مقدار دلخواه است). راهنمایی: در اینجا تغییر ندادن رجیسترها به طور مطلق ناممکن است اما می‌توان مقدار آنها را جایی (کجا؟؟) ذخیره کرد و هنگام پایان برنامه سر جایش گذاشت.

```
03 data segment
04     ; add your data here!
05     a dw 0h
06     b dw 0ffffh
07 ends
08
09 stack segment
10     dw   128  dup(0)
11 ends
12
13 code segment
14 start:
15 ; set segment registers:
16     mov ax, data
17     mov ds, ax
18     mov es, ax
19
20     ; add your code here
21
22     push ax
23     push bx
24
25     mov ax,a
26     mov bx,b
27     mov b,ax
28     mov a,bx
29
30     pop bx
31     pop ax
32
```

تمرین ۲

- ۱. توضیح دهید اگر برنامه‌ای الف. سگمنت کد ب. سگمنت داده ج. سگمنت پشته نداشته باشد چه امکاناتی را از دست می‌دهد (برای هر کدام جداگانه. در مجموع حداکثر ۳ سطر)
- ۲. برنامه اسلاید قبل را بازنویسی کنید. به این صورت که برنامه همین کار را انجام دهد، اما از اسم متغیر استفاده نشود و آدرس حافظه را مستقیما در کد بنویسد (آدرس‌ها (آفست) را حدس بزنید و پیدا کنید و داخل برنامه بنویسید و برنامه را اجرا کنید و مطمئن شوید که درست کار می کند).

معماری کامپیوتر

جلسه چهارم

ساخت یک ALU

(واحد محاسبه و منطق)

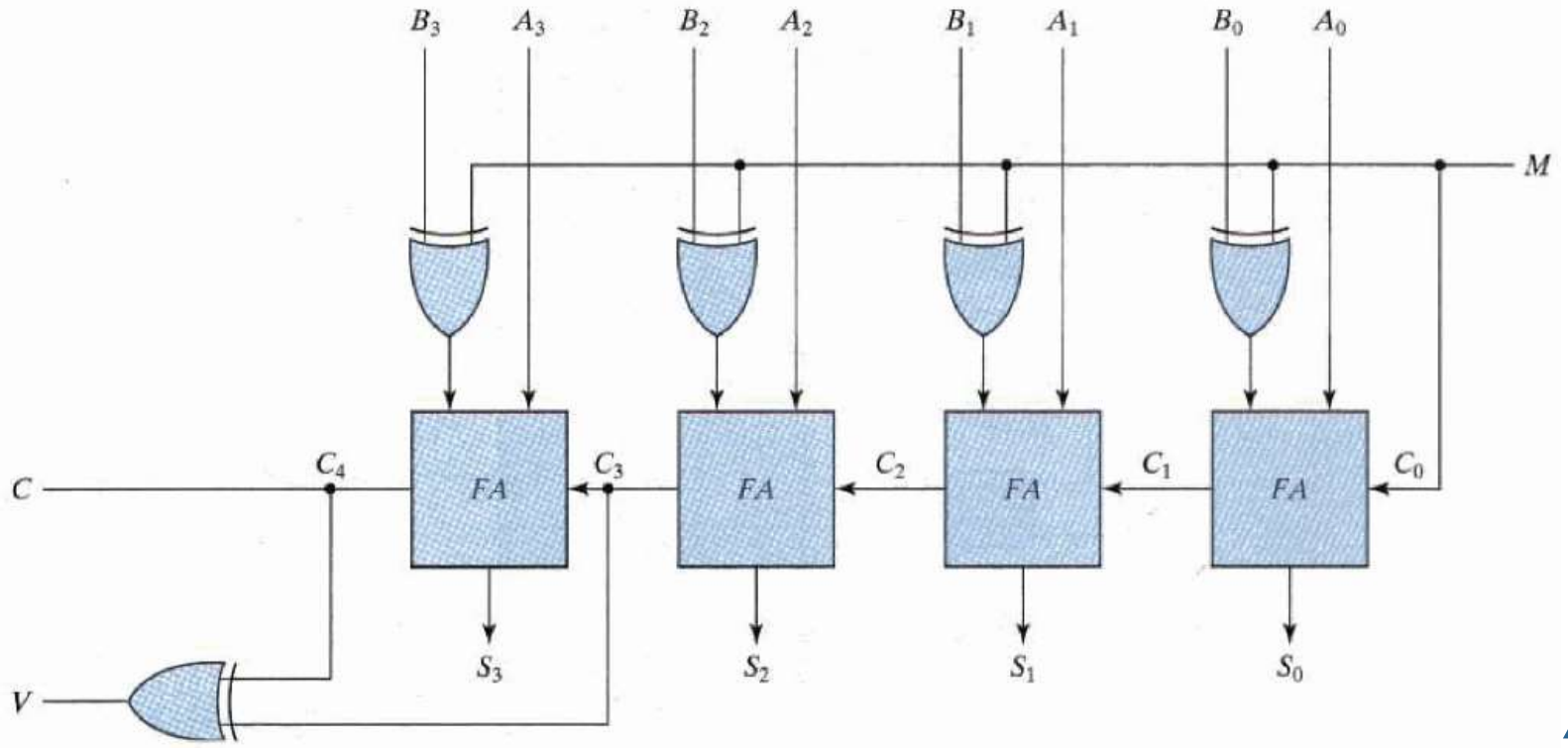
دستورات محاسباتی و پرش

دستورالعمل‌های محاسباتی

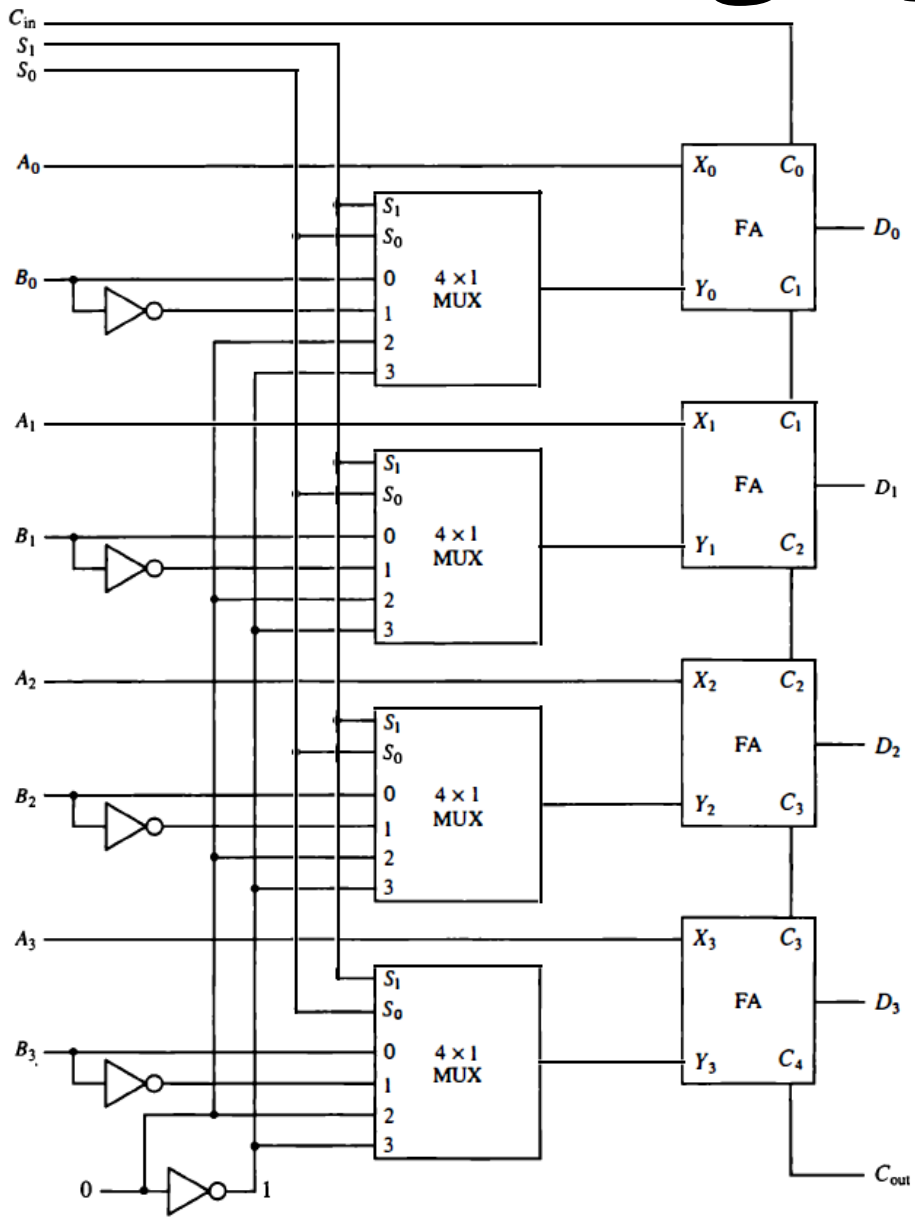
- کامپیوتر یعنی ماشین محاسب.
- کامپیوتری که فقط بتواند داده‌ها را در خود ذخیره کند و از جایی به جایی انتقال دهد بی‌فایده است. کامپیوتر باید قادر به انجام محاسبه روی داده‌ها باشد.
- عملیات محاسباتی شامل ۱: دستورالعمل‌های حسابی
۲: دستورالعمل‌های منطقی (دستکاری بیت‌ها)
۳: دستورالعمل‌های شیفت (جابجایی بیت‌ها) می‌شود.
- دستورالعمل‌های محاسباتی تا حدودی شبیه دستورالعمل‌های انتقال هستند، با این تفاوت که هنگام انتقال بیت‌ها مستقیما از مبدا روی گذرگاه قرار می‌گرفت اما در اینجا ابتدا از واحد محاسبه و منطق می‌گذرد و سپس روی گذرگاه مشترک قرار می‌گیرد.

جمع و تفریق‌کننده چهاربیتی

- این مدار را در درس مدارهای منطقی بررسی کردیم. دو عدد چهاربیتی وارد آن می‌شوند و یک عدد چهاربیتی خروجی اصلی آن است. ورودی M تعیین می‌کند که جمع صورت گیرد یا تفریق و خروجی‌های C و V کری خروجی و سرریزند.



مدار حساب چهاربیتی



- مدار روبرو مشابه همان مدار جمع و تفریق کننده است اما قدری کاملتر شده تا کارهای بیشتری انجام دهد.
- همانطور که می‌بینید اینجا بجای xor از mux استفاده شده است و بجای یک خط (کنترل عملکرد M) در این مدار دو خط (S1,S2) وجود دارد.

مدار حساب چهاربیتی

- همانطور که می‌بینید بجز عمل جمع (add) و تفریق (subtract) در اینجا اعمال افزودن یک (incerement همان ++) و کاهش یک (decrement همان --) و انتقال بی تغییر (transfer همان mov) توسط این مدار انجام می‌گیرد.
- با نگاه کردن به شکل مدار و می‌توانید نحوه انجام این اعمال را درک کنید. مثلا برای عمل ++ عدد با 0001 جمع می‌شود و برای -- عدد با 1111 که معادل 1- است جمع می‌گردد.

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

سایر اعمال حسابی

- کامپیوترهای ساده فقط دارای پیاده‌سازی سخت‌افزاری جمع و تفریق تقریبا به همین شکل هستند. در این کامپیوترها برای بقیه اعمال محاسباتی بایستی برنامه‌نویسی کرد.
- کامپیوترهای قوی (از جمله PC) عملگرهای محاسباتی دیگر مثل ضرب و تقسیم را نیز به صورت سخت‌افزاری پیاده‌سازی کرده‌اند و نیز ممکن است محاسبات اعشاری را هم انجام دهند. همچنین پیاده‌سازی جمع و تفریق در آنها به گونه‌ای است که سریعتر عمل کنند (در این پیاده‌سازی ساده باید به اندازه تاخیر انتشار همه جمع‌کننده‌ها صبر کرد).

محاسبات منطقی

- دستورالعمل‌های منطقی دو رشته چند بیتی را دریافت می‌کنند و بر روی هر دو بیت متناظر بطور مستقل یک عمل منطقی انجام می‌دهند. مثلا اگر دو رجیستر چهار بیتی داشته باشیم می‌توان با دستور زیر (دستوری که با 1 شدن خط کنترلی P اجرا می‌شود) آنها را xor کرد:

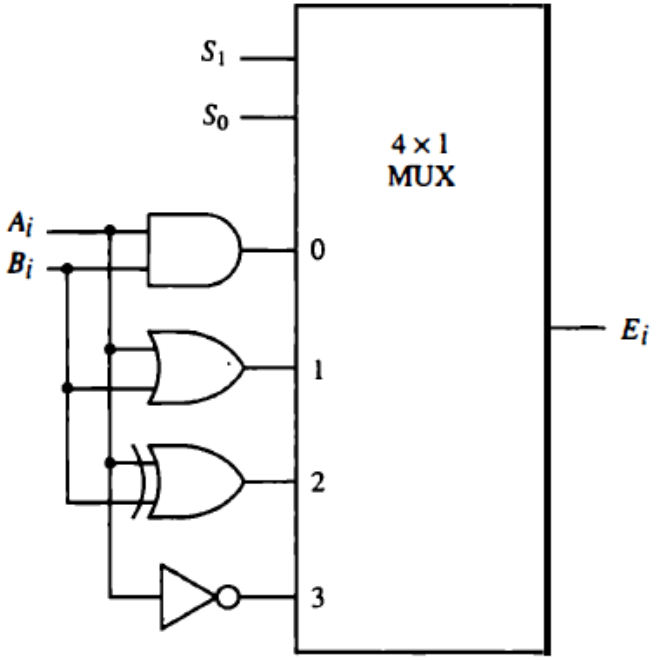
$$P: \quad R1 \leftarrow R1 \oplus R2$$

- اگر قبل از عمل محتوای R1 و R2 بصورت زیر باشد پس از عمل R1 چنین مقداری خواهد گرفت:

1010	Content of $R1$
1100	Content of $R2$
0110	Content of $R1$ after $P = 1$

مدار محاسبات منطقی

- محاسبات منطقی کاربردهای مختلفی دارند. مثل صفر یا یک کردن تمام یا بخشی از بیت‌های یک رجیستر و نیز اجرای شرط‌های چندگانه و غیره.
- مدار زیر با دو ورودی کنترلی می‌تواند اعمال منطقی مورد نظر را انجام دهد:



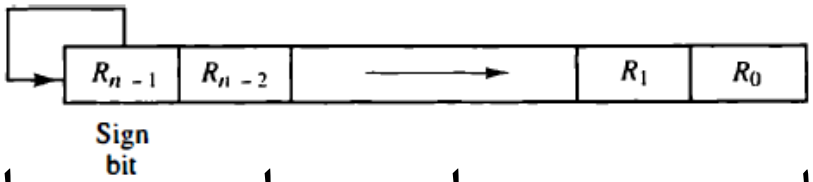
(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

شیفت (جابجایی)

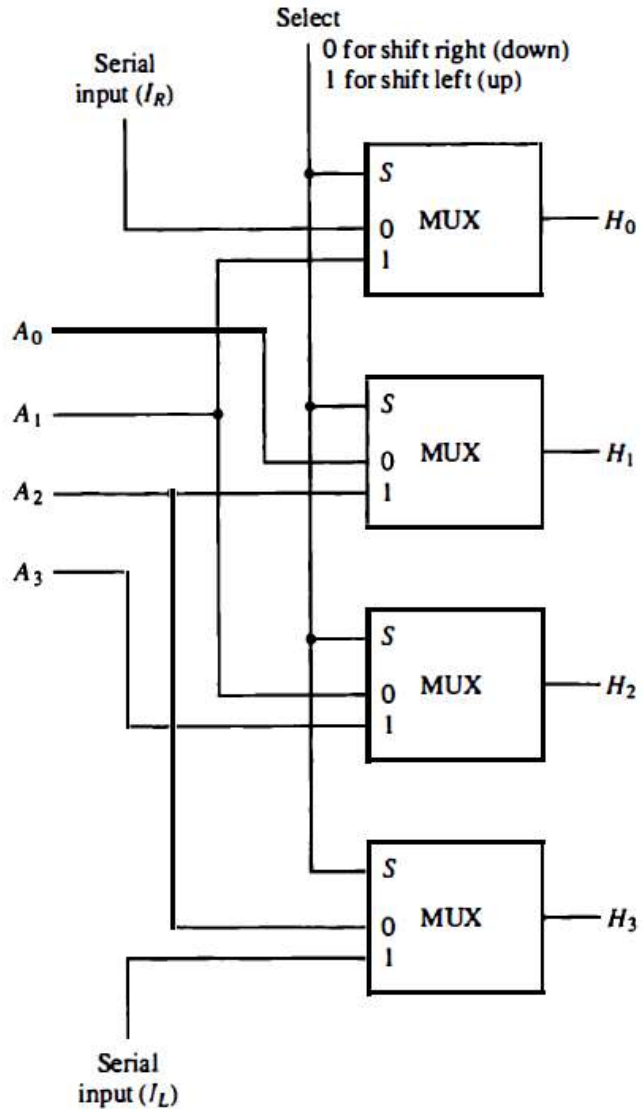
- دستورالعمل‌های شیفت بیت‌ها را درون یک رجیستر جابجا می‌کنند. سه نوع شیفت وجود دارد.
- **شیفت منطقی** که در آن 0 از یک طرف (چپ یا راست) وارد رجیستر شده و بقیه بیت‌ها به یکی آن طرف‌تر انتقال می‌یابند. هر بار انجام این نوع شیفت در اعداد مثبت بی‌علامت به چپ آن را دوبرابر و به راست آن را نصف می‌کند.
- **شیفت حسابی** آخرین بیت سمت چپ (بیت علامت) را سر جای خود نگه داشته و بقیه بیت‌ها را شیفت منطقی می‌دهد.



- **شیفت چرخشی** بیت‌های رجیستر را بدون از دست دادن هیچ بیتی می‌چرخاند (از یک سمت بیتی را خارج و همان را از سمت دیگر داخل می‌کند).

مدار شیفت

- در درس مدارهای منطقی ساخت رجیسترهایی را دیدیم که دارای امکان شیفت بودند.
- در اینجا نیز مداری رسم شده که با خط کنترلی امکان شیفت به راست و چپ را فراهم می‌کند.



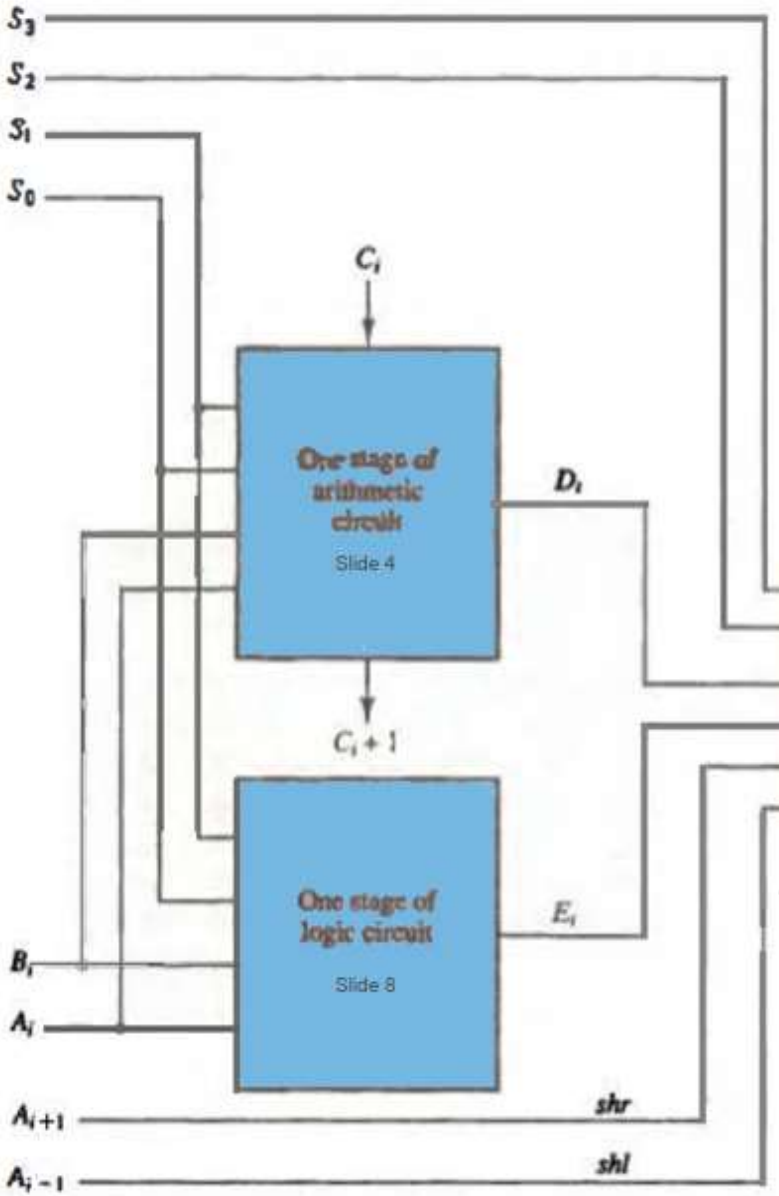
Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

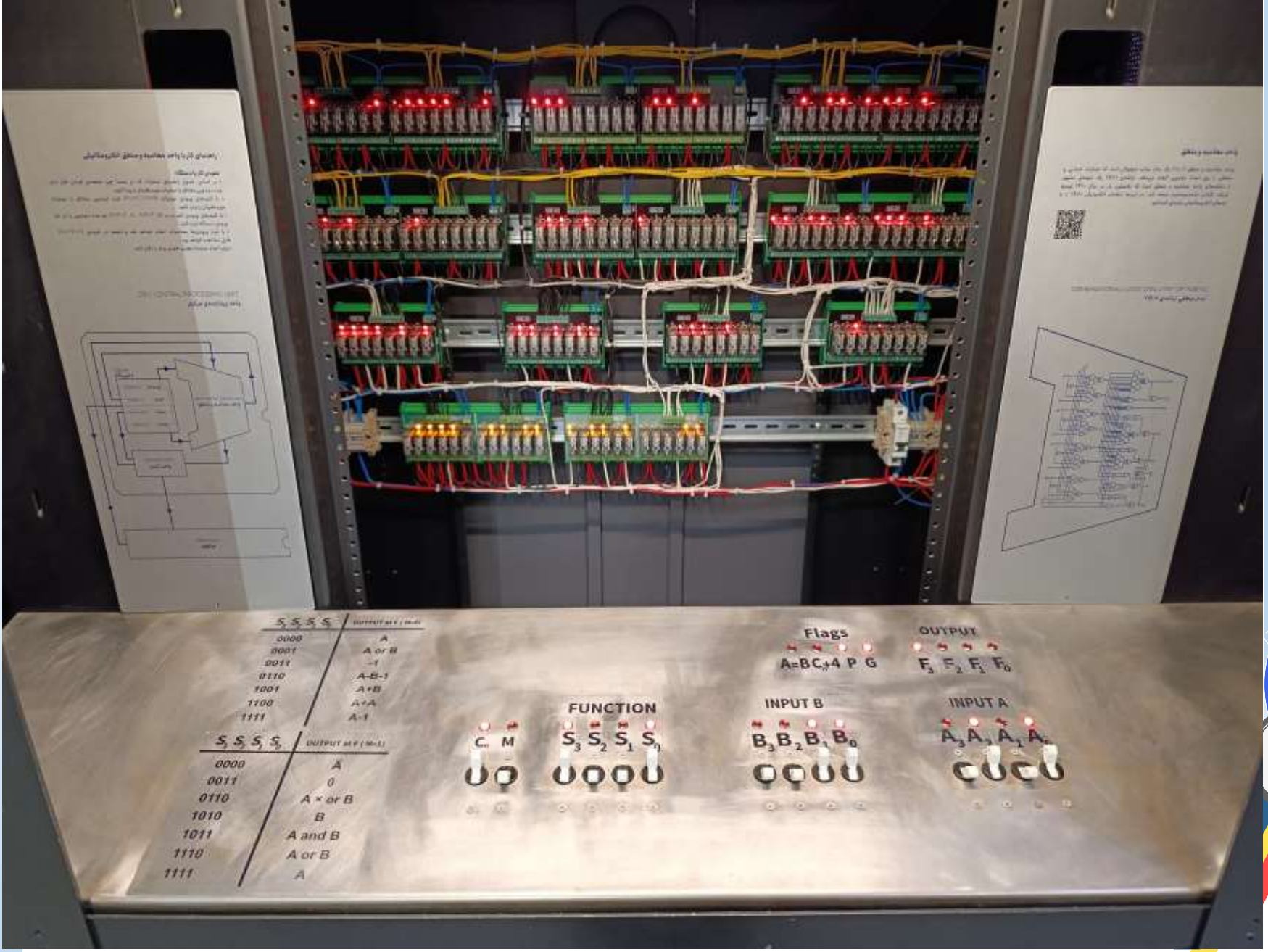
واحد محاسبه و منطق

- در کامپیوترها، رجسیترهای مختلف به یک واحد محاسبه و منطق ALU بطور مشترک متصل هستند و همه از آن استفاده می‌کنند.
- واحد محاسبه (شکل اسلاید ۴) و واحد منطق (شکل اسلاید ۸) در این مدار جای گذاری میشوند و تا یک واحد کامل محاسبه و منطق را شکل دهند.
- این شکل یک طبقه یک بیتی از مدار است و می‌توان آن را به اندازه دلخواه گسترش داد و مثلا با گذاشتن ۱۶ تا از آن در کنار هم یک ALU ۱۶ بیتی ساخت.

مدار ALU



S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A + B$	Addition
0	0	0	0	1	$F = A + B + 1$	Add with carry
0	0	0	1	0	$F = A + \overline{B}$	Subtract with borrow
0	0	0	1	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	0	0	$F = A$	Transfer A
0	0	1	0	1	$F = A + 1$	Increment A
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F



- یک ALU چهاربیتی پیاده سازی شده با رله‌های برقی – محل موزه کامپیوتر ایران
(اگر به این محل رفتید یادتان باشد که Cin دستگاه معکوس عمل می‌کند!)

دستورالعمل‌های محاسباتی PC

- در پردازنده 8086 عملوندهای محاسباتی (مثل انتقال) می‌توانند از انواع مختلف باشند: بی‌واسطه، رجیستر، حافظه مستقیم و حافظه غیر مستقیم.
- دستورات حسابی یک عملوندی عبارتند از :
neg عملوند را منفی می‌کند (مکمل ۲ جایگزین می‌شود)
inc به عملوند یکی می‌افزاید.
dec از عملوند یکی کم می‌کند.
- مثلا دستور
dec ax
یک واحد از مقدار موجود در رجیستر ax کم می‌کند.

دستورات حسابی دو عملوندی

- این دستورات به صورت

دستورالعمل عملوند۱ ، عملوند۲

نوشته می‌شوند و همیشه نتیجه محاسبه در عملوند ۱ ذخیره می‌گردد.

- دستورات پایه حسابی عبارتند از add و sub (جمع و تفریق) مثلا:

```
add ax,[0005]
```

محتوای رجیستر ax را با محتوای خانه شماره ۵ از دیتا سگمنت جمع می‌کند و نتیجه را هم در ax می‌ریزد.

دستورالعمل‌های ضرب، تقسیم و اعشاری

- کامپیوتر PC کامپیوتری است قوی و نسبتا پیچیده که دستورات ضرب و تقسیم را بصورت سخت‌افزاری انجام می‌دهد (بر خلاف ALU ساده‌ای که ساختیم).
- در اینجا وارد جزئیات دستورات ضرب و تقسیم PC نمی‌شویم فقط این نکته را باید دانست که در ضرب طول عملوند مقصد دو برابر عملوندهای مبدا می‌باشد و در تقسیم نیز مقسوم و مقسوم علیه و خارج قسمت و باقیمانده هم طول نیستند.
- جمع، تفریق ضرب و تقسیمی که دیدیم از نوع عدد صحیح است. بطور کلی در نسل x86 تا قبل از 486 در صورت نیاز به محاسبات اعشاری توسط سخت افزار بایستی یک کمک پردازنده کنار پردازنده اصلی روی مادربرد نصب می‌شد.

طول و زمان اجرای دستورالعمل‌ها

- از آنجا که ۸۰۸۶ پردازنده‌ای پیچیده است طول و زمان اجرای دستورالعمل‌ها متفاوت است. مثلا یک دستور کد ماشین ممکن است ۲ بایت طول داشته باشد و اجرای آن تنها ۲ سیکل ساعت طول بکشد و دستور دیگر ممکن است ۶ بایت طول داشته باشد و ۱۰ سیکل ساعت طول بکشد.
- برای مثال در اینجا (برای مقایسه) چند دستور مختلف همراه **طول** (بایت) و **زمان اجرا** (سیکل)شان ذکر می‌شود:

mov رجیستر به رجیستر	2	2
mov کلمه بی‌واسطه به رجیستر	2-3	4
mov رجیستر و حافظه مستقیم	4	15
add رجیستر با رجیستر	2	3
mul (ضرب) رجیستر ۸ (۱۶) بیتی	2	77 (133)

دستورالعمل‌های منطقی و شیفت

- دستورالعمل تک عملوندی not و سه دستور دو عملوندی یعنی and, or, xor در اسمبلی PC قابل استفاده هستند که عملکرد آنها همانست که دیدیم.
- دستورات شیفت می‌توانند تکی یا چند تایی باشند. مثلا برای چهار بیت شیفت دادن یک رجیستر بجای چهار بار نوشتن دستور شیفت می‌توان درخواست شیفت ۴ بیتی نمود. دستور بصورت دو عملوندی و به فرمت زیر نوشته می‌شود:

  عملوند شیفت، تعداد شیفت sXX

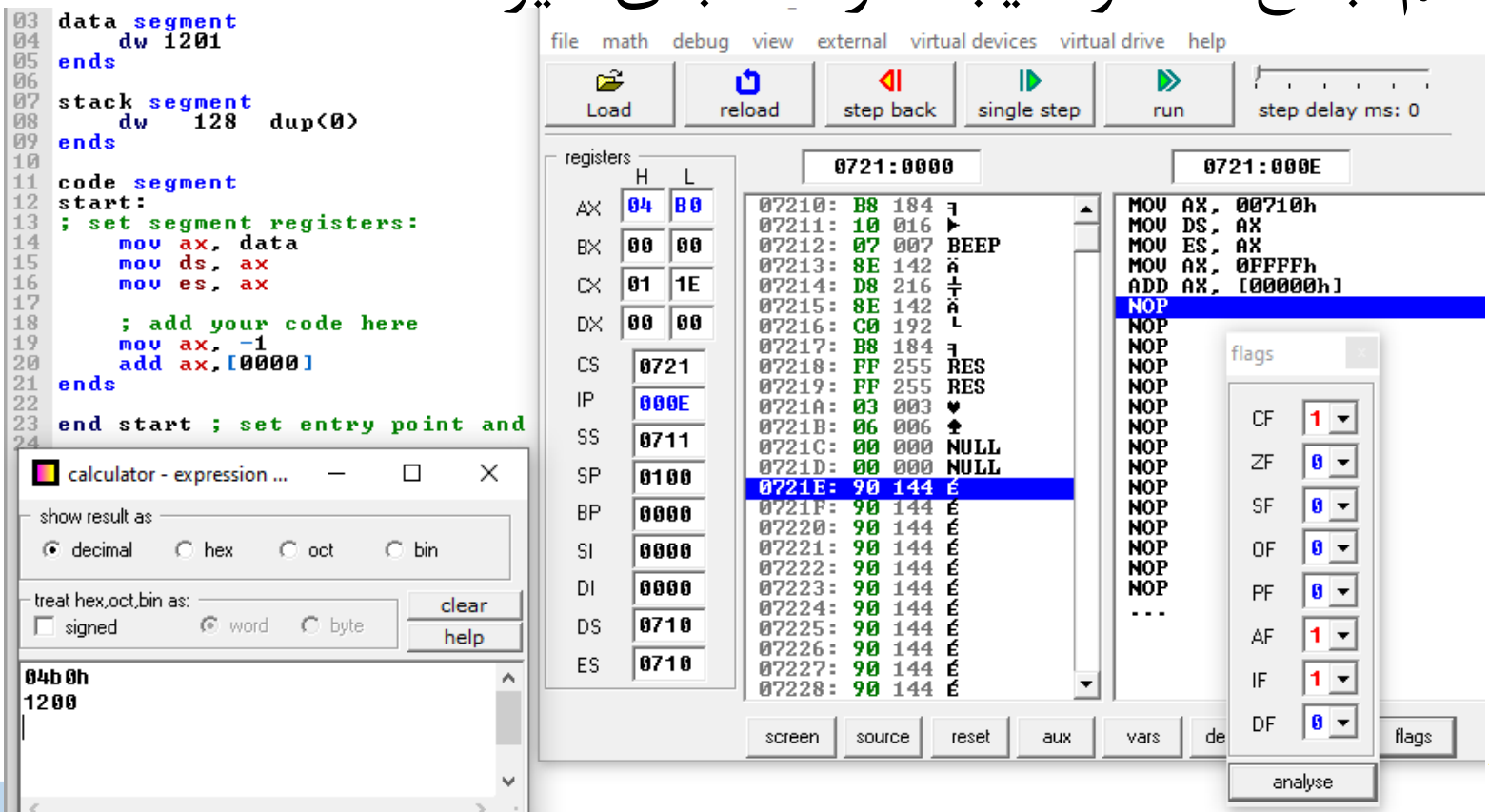
  تعداد شیفت فقط می‌تواند دو چیز باشد: 1 یا cl (یا یکی یا به تعداد عدد موجود در cl)
- دستورات شیفت که در اینجا ذکر می‌شوند شش تایند:

shr شیفت منطقی به راست	shl منطقی به چپ
sar ریاضی به راست	sal ریاضی به چپ
ror گردش به راست	rol گردش به چپ

مثال

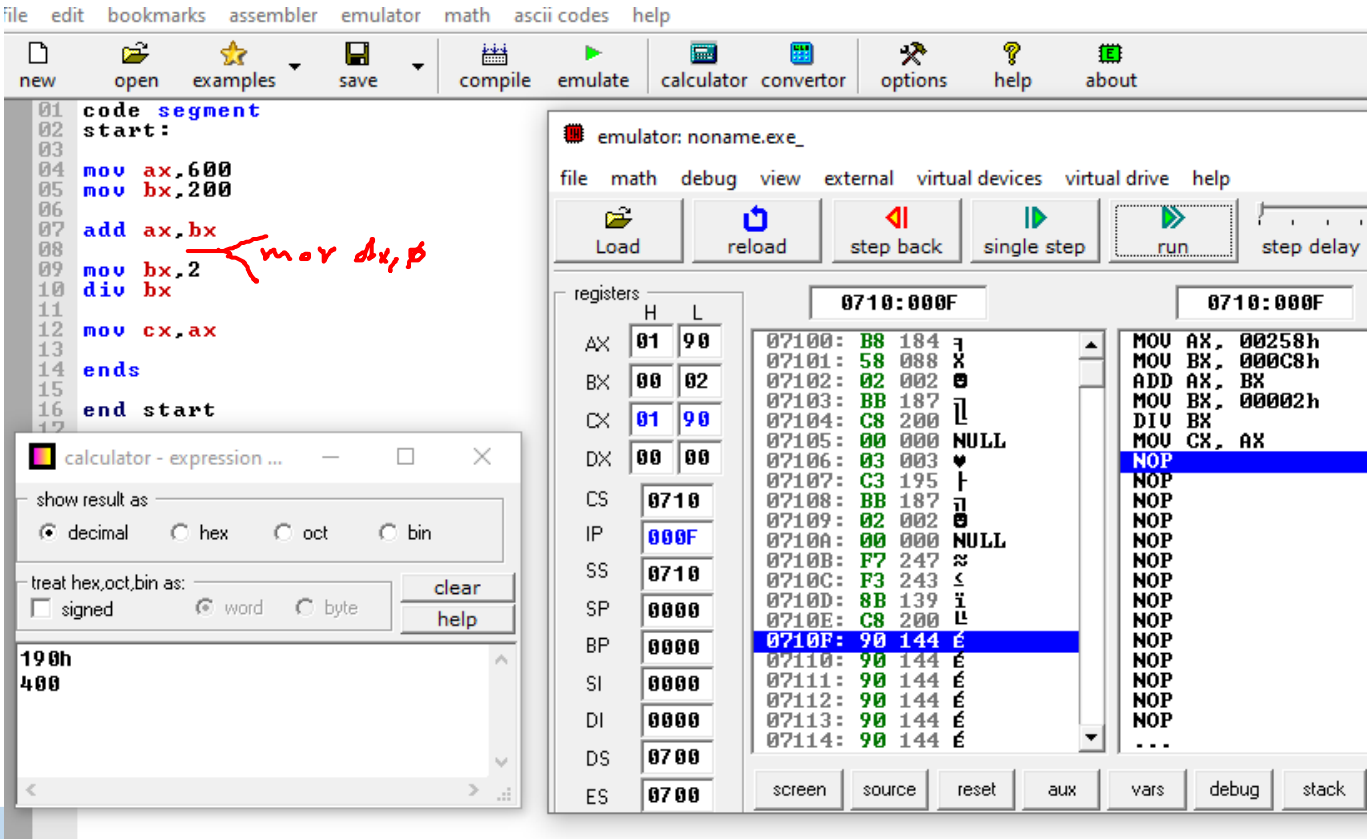
- برنامه‌ای بنویسید که یک خانه به اندازه ورد در حافظه که محتوی مقدار 1201 است در حافظه تعریف کند. سپس مقدار 1- را در رجیستر ax قرار داده سپس این دو را با هم جمع کند و نتیجه در ax جای گیرد.

```
data segment
    dw 1201
ends

stack segment
    dw   128  dup(0)
ends

code segment
start:
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax

    ; add your code here
    mov ax, -1
    add ax,[0000]
ends

end start ; set entry point and
```



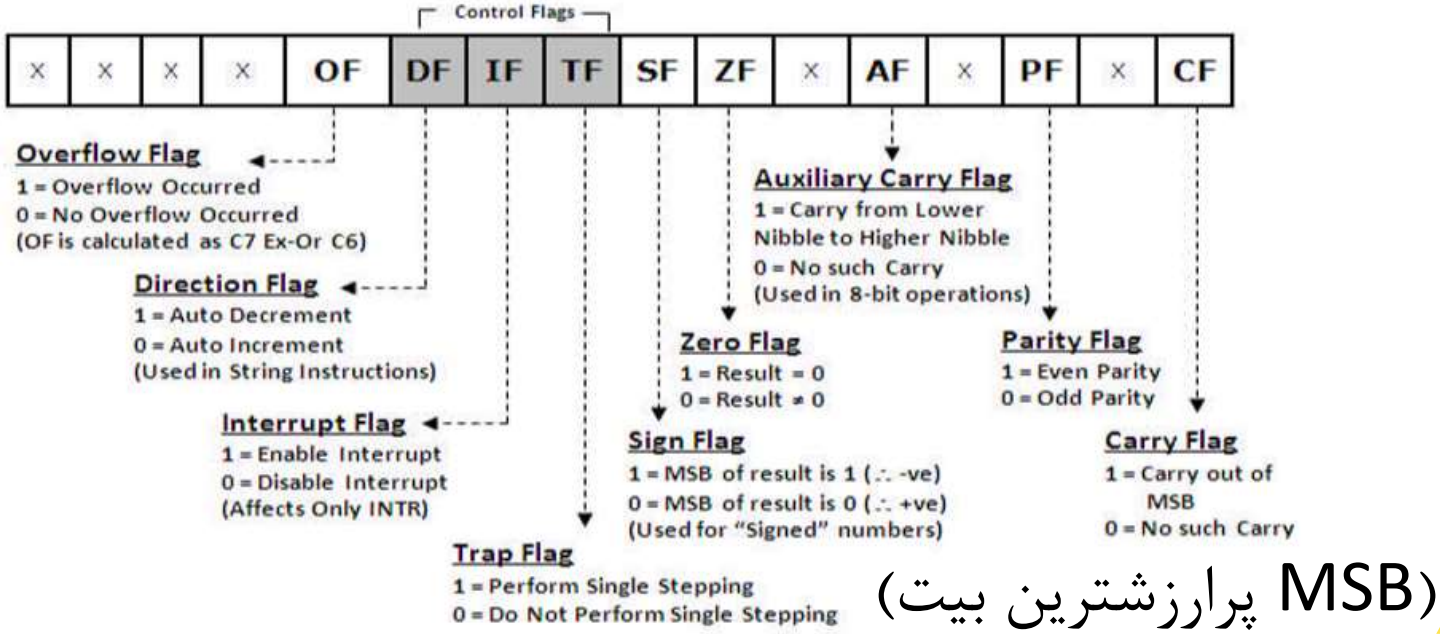
مثال (اختیاری برای مشاهده تقسیم)

- برنامه‌ای بنویسید که دو عدد مثبت ۶۰۰ و ۲۰۰ را بترتیب در ax و bx قرار دهد. سپس معدل محتوای این دو رجیستر را محاسبه نموده و نتیجه را در cx قرار دهد.



رجیستر پرچم

- گرچه رجیسترها همه ۱۶ بیتی هستند اما تنها ۹ بیت از آنها در رجیستر پرچم 8086 مورد استفاده قرار گرفته‌اند.
- تمامی دستورات محاسباتی برخی از بیت‌های رجیستر پرچم را تحت تاثیر قرار می‌دهند.



(MSB پرارزشترین بیت)

بیت‌های رجیستر پرچم

- CF پرچم کری در صورتی که از پرارزشترین بیت کری خروجی داشته باشیم 1 وگرنه 0 می‌شود. بیت خارج شده از شیفت هم در آن قرار می‌گیرد.
- ZF اگر نتیجه محاسبه صفر شود 1 وگرنه 0 می‌شود.
- SF در صورتی که نتیجه محاسبه صحیح منفی (پرارزشترین بیت آن 1) شود 1 وگرنه 0 می‌شود.
- OF اگر نتیجه محاسبه صحیح سرریز شود 1 وگرنه 0 می‌شود.
- از ۹ بیت موجود، ما بیشتر با همین ۴ بیت سروکار داریم.

تمرین

- شکل مدار حساب ۴ بیتی (اسلاید صفحه ۴) را درنظر بگیرید. آن را چاپ کنید یا روی کاغذ بکشید یا با نرم‌افزارهای گرافیکی ادیت کنید. دو رقم کم ارزش شماره دانشجویی تان را روی صفحه بنویسید. مدار قرار است این رو رقم را با هم جمع کند. تمام خصوص hi (دارای مقدار 1) را با رنگ قرمز و low ها را با آبی رنگ کنید.
- روی ۸۰۸۶ برنامه ای بنویسید که دو رقم فوق را معدل بگیرد و نتیجه را در ax بگذارد. نتیجه برنامه و فلگ‌ها در تصویری که می‌فرستید مشخص باشد. (از دستور تقسیم استفاده نکنید. برای معدل گیری اول دو عدد با هم جمع شوند و سپس با یک شیفت حاصل نصف شود).

معماری کامپیوتر

جلسه پنجم

کنترل برنامه و واحد اجرا
تاریخچه معماری کامپیوتر
(مفاهیم، کامپیوتر پایه)

واحد اجرا

- تاکنون به وظایف و طراحی سخت‌افزار واحد محاسبه و منطق پرداختیم و با کنترل گذرگاه مشترک آشنا شدیم.
- واحد اجرا EU قسمت دیگری از پردازنده است. این واحد دستورات زبان ماشین را به ترتیب از حافظه به داخل پردازنده می‌آورد و موجبات اجرای آنها را فراهم می‌کند.
- در اینجا (درس امروز) نمی‌خواهیم به پیاده‌سازی سخت‌افزاری واحد اجرا بپردازیم، فقط به مرور نحوه کنترل اجرای برنامه و نیز نحوه انجام دستورات پرش خواهیم پرداخت.

دستورات پرش

- واحد اجرا دستورات زبان ماشین را از ابتدای کد برنامه شروع کرده و یک به یک آنها را اجرا می‌کند. آدرس دستوری که باید بعد از این اجرا شود در رجیستری بنام شمارنده برنامه PC (یا اشاره‌گر دستورالعمل IP) قرار دارد.
- اگر دستور پرش پیش آید PC به روزرسانی شده و دستور بعدی در جایی غیر از بعد از دستور فعلی خواهد بود.
- پرش‌ها یا شرطی هستند یا غیر شرطی. در دستورات پرش شرطی با توجه به وضعیت پرچم‌ها معلوم می‌شود که پرش انجام بگیرد یا نگیرد.
- پرش‌های غیرشرطی به عقب موجب لوپ بی‌پایان و به جلو موجب بلااستفاده شدن بخشی از کد می‌شوند، مگر اینکه به همراه پرش‌های شرطی بکار گرفته شوند.

دستورات پرش در کامپیوتر PC

- در نسل 8086 دستورات پرش معمولا از نوع کوتاه هستند. در پرش کوتاه پریدن به ۱۲۷ بایت جلو یا عقب ممکن است (آدرس بین + و - ۱۲۷ یعنی آدرس یک بایتی). البته پرش غیرشرطی با نوع دیگری از دستور به فواصل دورتر هم ممکن است که ما اینجا به آن نمی‌پردازیم.
- در اسمبلی می‌توان بجای آدرس مقصد، برای آن لیبل (یک نام دلخواه) قرار داد که در این صورت اسمبلر خودش آدرس‌ها را محاسبه و جایگذاری می‌کند.
- اسمبل کردن معمولا در دو گذر pass انجام می‌شود چرا که در پرش‌های آینده چون هنوز دستورات بعدی به زبان ماشین ترجمه نشده، آدرس مقصد در گذر اول آنها نامشخص است، پس آدرس خالی گذاشته می‌شود و در گذر دوم پر می‌شود.

دستورات پرش در کامپیوتر PC

- پرش غیر شرطی به این صورت نوشته می‌شود:

  jmp label

- پرش‌های شرطی می‌توانند مستقیما بر اساس پرچم‌ها باشند (یک یا صفر بودن):

  js/jns بسته به پرچم علامت jc/jnc بسته به پرچم کری

  jo/jno بسته به پرچم سرریز jz/jnz بسته به پرچم صفر

- دستور cmp دو عملوندی است و دقیقا همانند دستور تفریق sub عمل می‌کند، اما نتیجه تفریق را نگه نمی‌دارد و فقط روی پرچم‌ها اثر می‌گذارد.

- دستورات پرشی که بعد از دستور cmp a,b با توجه به محتوای یک یا چند پرچم تصمیم می‌گیرد: above/below برای عدد بی علامت و grater/less علامتدار.

  ja/jg وقتی عملوند اول (a) بزرگتر از عملوند دوم (b) باشد

  jae/jge وقتی عملوند اول بزرگتر یا مساوی عملوند دوم باشد

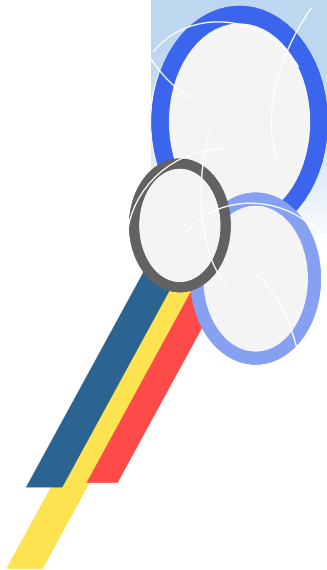
  jb/jl وقتی عملوند اول کوچکتر از عملوند دوم باشد

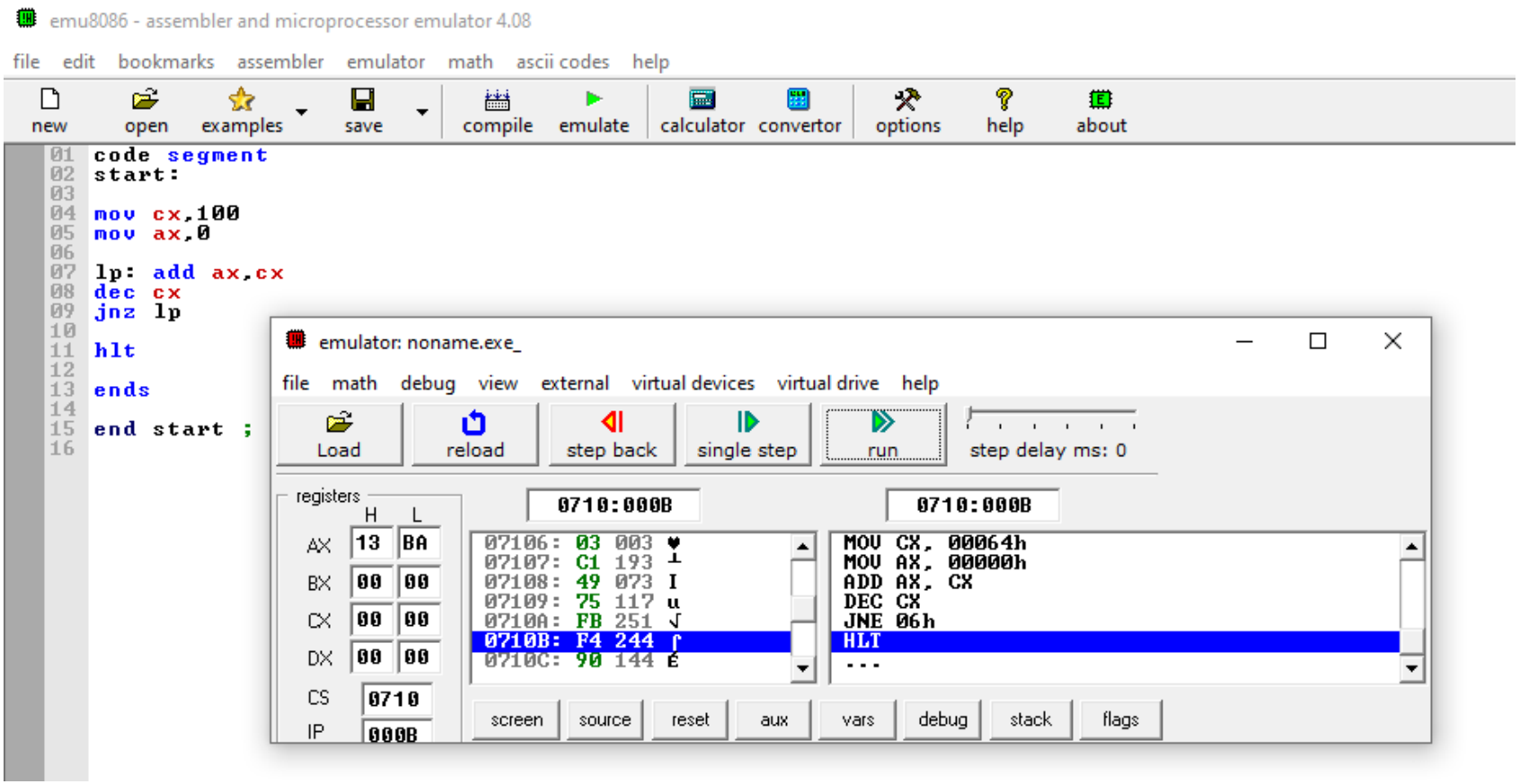
  jbe/jle وقتی عملوند اول کوچکتر یا مساوی عملوند دوم باشد

دستورات پرش در کامپیوتر PC

- به این ترتیب می‌توان انواع شرط‌های مقایسه‌ای را بزبان ماشین یا اسمبلی در کامپیوتر PC پیاده‌سازی نمود. همچنین ایجاد حلقه‌ها را. (هر چند دستورات ساده‌تری برای حلقه نیز وجود دارد.)

- مثال: برنامه‌ای بنویسید که حاصل جمع اعداد یک تا صد را محاسبه و در ax قرار دهد.
راهنمایی: می‌توان در ابتدا 100 را در رجیستری مثلا cx قرار داد و بعد یکی یکی آن را کم کرد.

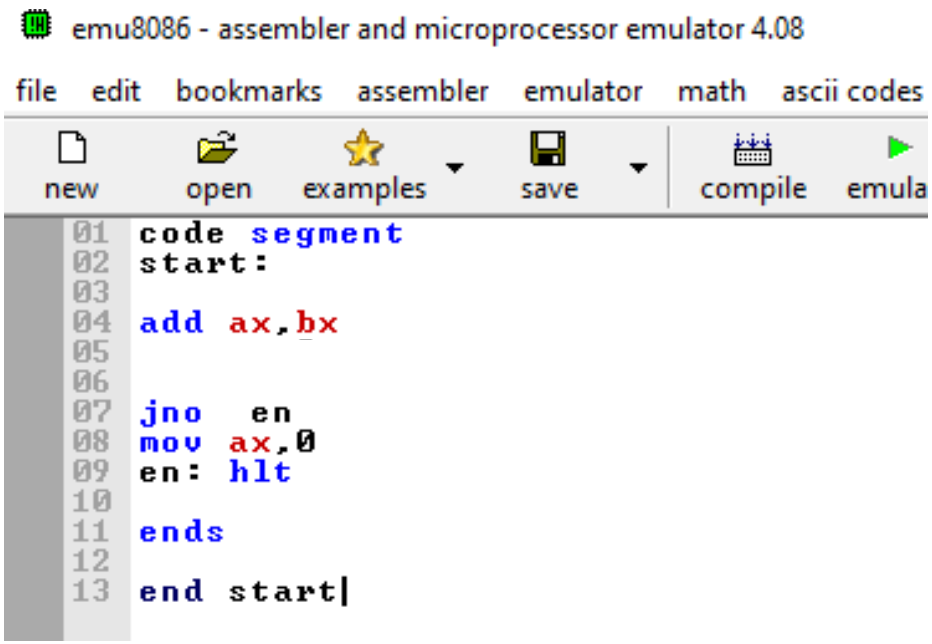




- hlt دستوری است که پایان برنامه را با سیستم اعلان می‌کند و بعد از آن دیگر سیستم به دنبال آوردن دستور بعد نخواهد رفت.

مثال

- برنامه‌ای بنویسید که مقدار ax و bx که دو عدد علامتدار در آنهاست را جمع کند. اگر نتیجه سر ریز نشد آن را در ax بگذارد و اگر سرریز شد 0 را در آن بگذارد.



معماری کامپیوتر چیست؟

- سوالات زیر مقدمه معماری کامپیوترند. اکنون با دانشی که بدست آوریم پاسخ‌های بهتری برای این آنها خواهیم داشت:
- معماری (در همه رشته‌ها) یعنی چه؟
- معماری کامپیوتر یعنی چه؟
- تاریخچه معماری کامپیوتر چیست (همراه با آشنایی با معماری‌های کم دستور و پردستور)؟
- کامپیوتر پایه چیست و چگونه با استفاده از آن معماری کامپیوتر می‌آموزیم؟
- دانستن معماری کامپیوتر برای کسی که نمی‌خواهد معمار کامپیوتر شود چه اهمیتی دارد؟

معماری چیست

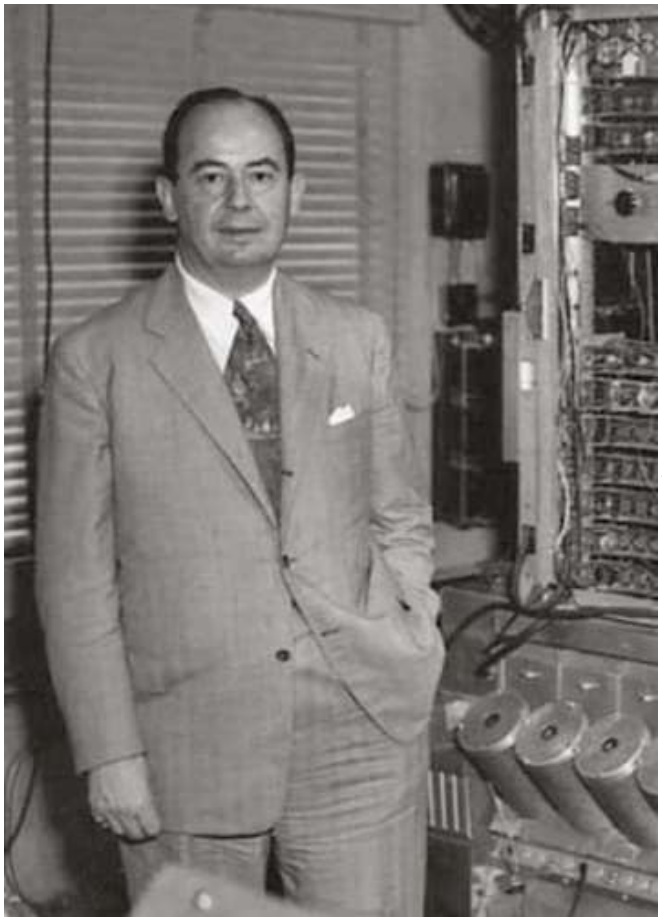
- معماری ایجاد نقشه ساخت چیزی است که تا به حال مشابه آن وجود نداشته است. معماری با طراحی متفاوت است. طراحی به معنای تعیین جزئیات پیاده‌سازی چیزی است که ساختار کلی آن معلوم است.
- برای مثال اتومبیل را کسی معماری نمی‌کند. این کار یکبار انجام شده و کلیات (جای چراغ‌ها، جای موتور، تعداد چرخ‌ها، جای و شکل ابزار کنترلی مثل فرمان و پدال‌ها و ...) معلوم است و در تمام خودروها یکیست.
- ساختمان را معماری می‌کنند چرا که هیچ چیز (تعداد اتاق‌ها، محل‌آنها، محل و تعداد آسانسور و پله‌ها و ...) مشخص نیست و این معمار است که همه چیز را از صفر خلق می‌کند.

معماری کامپیوتر

- فرآیند ساخت یک کامپیوتر جدید دست کم شامل موارد زیر است:
۱. معماری مجموعه دستورالعمل‌ها: ایجاد یک مدل برای تعیین: a)رجیسترها، b)اندازه و نوع داده‌ها و c)مجموعه دستورالعمل‌هایی که قرار است کامپیوتری آنها را اجرا کند. (تقریبا آنچه در مورد PC آموختیم)
۲. معماری و سازماندهی کامپیوتر یا ریزمعماری: شامل ایجاد یک طرح منسجم برای مدارات سخت‌افزاری با هدف اجرای آن مجموعه دستورالعمل خاص است.
- بنابراین ساخت کامپیوتر نه تنها یک فرآیند معماری است، بلکه از چند فرآیند معماری در کنار هم تشکیل شده است.

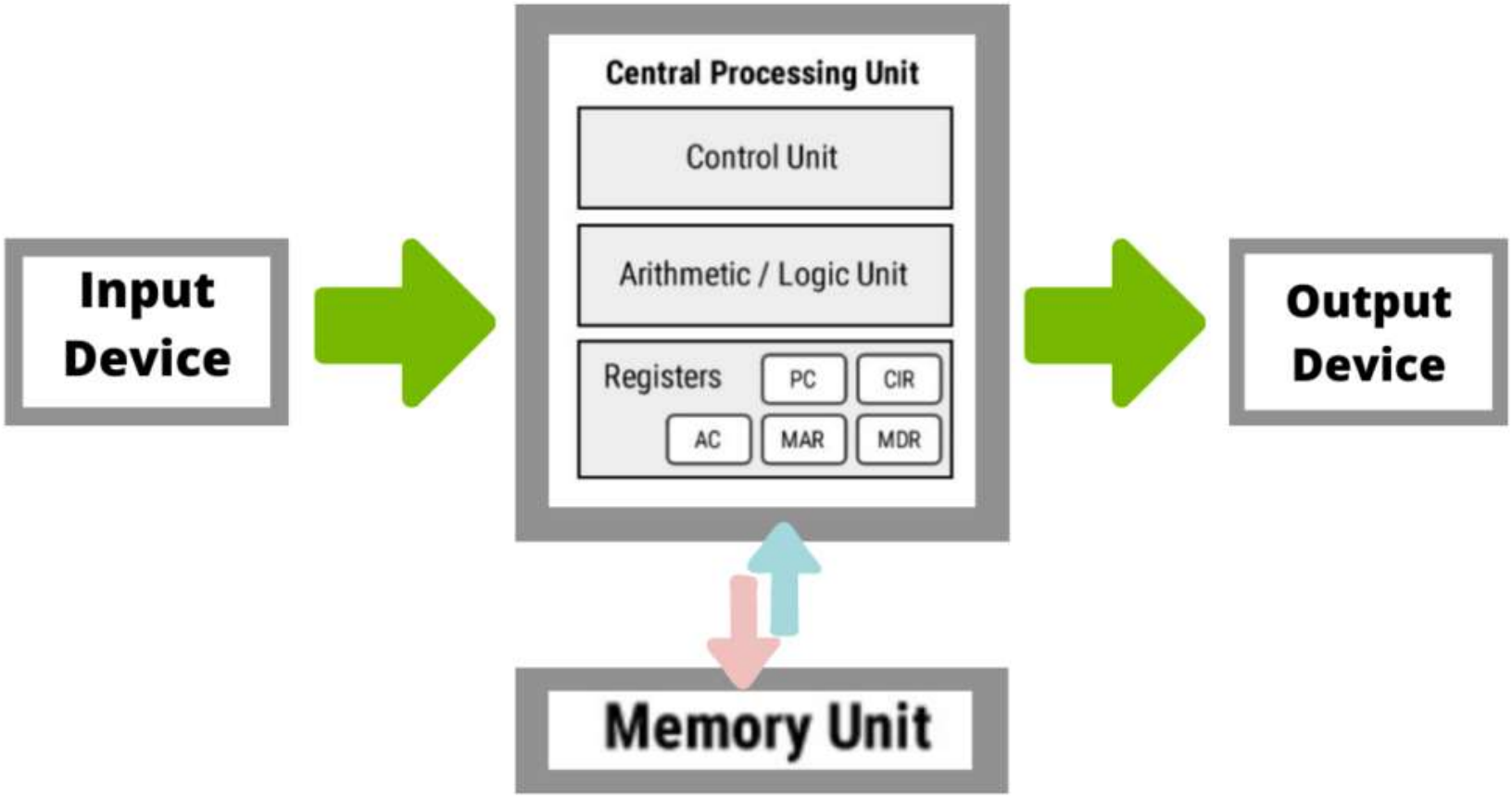
معماری کامپیوتر

- کامپیوترهای الکترونیکی اکثرا از معماری فُن نویمان استفاده می‌کنند.
- جان فن نویمان (۱۹۰۳-۱۹۵۷) علامه (دانشمند همه کاره) ای بود که در فیزیک شاخه‌ای از مکانیک کوانتومی، در اقتصاد نظریه بازی‌ها، در محاسبات طرح معماری کامپیوتر از ابداعات اوست. او در ساخت نخستین کامپیوتر (انیاک) و نیز ساخت نخستین بمب اتمی جزء دست‌اندرکاران اصلی پروژه بود.



معماری فن نویمان

- ماشین فن نویمان دارای یک CPU، یک حافظه اصلی و یک سیستم IO (ورودی خروجی) است.
- این ماشین یک کامپیوتر با برنامه ذخیره شده است و این برنامه ذخیره شده درون حافظه اصلی قرار دارد.
- دستورات بصورت ترتیبی اجرا می‌شوند و کامپیوتر در هر لحظه فقط مشغول اجرای یک دستور است.
- یک مسیر مشترک برای انتقال آدرس و یک مسیر برای انتقال داده بین حافظه و CPU وجود دارد.
- هر دستور دارای یک کد عملیاتی (opcode) است که نشان می‌دهد چه کاری باید انجام شود و غیر از آن بخش‌هایی در دستور هست که می‌گوید کدام رجیسترها، مقدارهای بلافصل و یا خانه حافظه مورد نظر هستند.
- برنامه دنباله‌ای از دستورات است که به همان ترتیب نوشته شده اند اجرا می‌شوند مگر آنکه دستوری درخواست کند که ترتیبی غیر از این مد نظر است.


Central Processing Unit
Control Unit
Arithmetic / Logic Unit
Registers
PC
CIR
AC
MAR
MDR
Input Device
Output Device
Memory Unit

تاریخچه معماری کامپیوتر

- معماری‌های انتزاعی کامپیوتر از دهه ۱۹۳۰ میلادی بعنوان طرح‌های آرمانی آغاز شد. پیاده‌سازی و ساخت یک کامپیوتر واقعی اما تا اواسط دهه ۱۹۴۰ انجام نشد.
- انیاک نخستین کامپیوتری بود که به صورت واقعی ساخته شد. این کامپیوتر حافظه اصلی نداشت و همه چیز را در رجیسترهایش ذخیره می‌کرد. مشخصات آن در مقایسه با یک PC امروزی:

	desktop PC	ENIAC	difference
CPU clock	4.2 GHz	5 kHz	840,000 x faster
memory	32 GB	90 bytes*	382,000,000 x larger
mass storage	4 TB	none**	
transistors (or tubes)	~3 billion	18,000	167,000 x more
weight	30 lbs.	54,000 lbs.	1800 x smaller
size	25"x12.5"x21"	9'x3'x98'	700 x smaller
power	800W	150kW	188 x smaller
cost	$2000	$487,000 (6.9 million in 2018)	3450 x cheaper

lbs تقریبا ۴۵۰ گرم

چرا کامپیوترهای قدیمی بزرگ بودند

- یک ALU ساده (مانند آنچه جلسه قبل طراحی کردیم) اگر ۱۶ بیتی باشد نیاز به حدود ۶۰۰ گیت منطقی برای پیاده سازی دارد (با احتساب مدارات داخلی مالتی پلکسرها و جمع‌کننده‌ها و ...).
- کامپیوتر همچنین برای رجیسترها (هر رجیستر ۱۶ بیتی ساده حدود ۲۰۰ گیت می‌خواهد) و واحد اجرا نیاز به مدارات منطقی مستقلی دارد. بنابراین ساده‌ترین کامپیوترها با هزاران گیت پیاده سازی می‌شوند.
- برای ساخت هر گیت منطقی به دو تا پنج ترانزیستور (یا لامپ خلا) نیاز است. بنابراین رقم ۱۸،۰۰۰ که در انیاک استفاده شده یک حداقل است و در پردازنده PC های امروزی به چند میلیارد می‌رسد.

چرا کامپیوترهای قدیمی بزرگ بودند

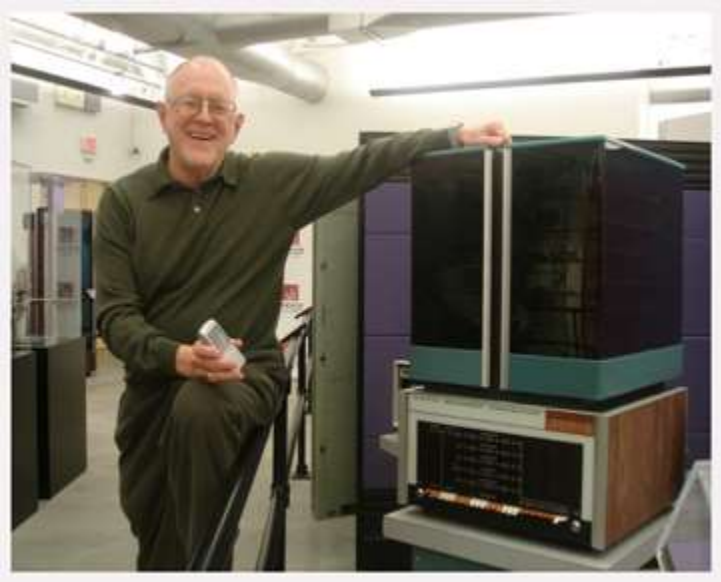
- کامپیوتر انیاک با لامپ‌ها خلا ساخته که بسیار پرحجم بودند کامپیوترهای دهه چهل و پنجاه میلادی همه این گونه بودند. کامپیوترهای ترانزیستوری که در دهه شصت میلادی آمدند کوچکتر و کم مصرف‌تر و ارزانتر بودند (مقایسه یک رادیوی لامپی و یک رادیو ترانزیستوری).
- هر لامپ خلا به اندازه انگشت شصت و هر ترانزیستور به اندازه یک عدس است. با این حال نصب صدها هزار ترانزیستور در یک کامپیوتر همچنان آن را سنگین و بزرگ می‌کرد تا اینکه کم‌کم مدارهای مجتمع IC که هرکدام محتوی چندین ترانزیستور بودند در صنعت الکترونیک، همه دستگاه‌ها را کوچک و کوچکتر کردند.

یک کامپیوتر لامپی دهه ۵۰
IBM 702

یک کامپیوتر ترانزیستوری دهه ۶۰
IBM 360

معماری‌های پردستور و کم‌دستور

- کامپیوترها در حالت عادی سعی می‌کنند برای راحتی برنامه‌نویس انواع دستورات زبان ماشین را پیشیبانی کنند. مثلا در کامپیوتر PC دیدیم چندین و چند نوع روش برای انتقال mov وجود دارد. انواع محاسبات را پردازنده می‌تواند انجام دهد و دیگر دستورات نیز تنوع بالایی دارد.
- چنین «معماری مجموعه دستورالعمل‌ها» نیاز به یک «ریزمعماری» سنگین و پیچیده دارد. این یعنی تعداد گیت‌ها و درنتیجه ترانزیستورها بسیار زیاد است (پردستور).
- برای نخستین بار زمانی که سازندگان کامپیوتر تصمیم به ساخت کامپیوترهای mini گرفتند (که از mainframe ها کوچکتر اما هنوز در حد میکروکامپیوترهای شخصی کوچک نبودند) ایده کامپیوتر کم‌دستور مطرح شد.



PDP-8 یک کامپیوتر مینی دهه شصت میلادی: با استفاده از معماری کم‌دستور کامپیوتری نسبتا کم‌حجم ساخته شد؛ توسط شرکت DEC (که بعدا توسط کامپک خریداری شد). در این درس توجه ویژه‌ای به این کامپیوتر خواهیم داشت.

تفاوت RISC و CISC

- کامپیوترهای کم دستور (مجموعه دستورالعمل‌های کاهش یافته RISC) با کامپیوترهای پردستور (مجموعه دستورالعمل‌های پیچیده CISC) تفاوت‌های اساسی دارند. البته کامپیوترهای مدرن دارای عنوان RISC ، فقط کم دستور نیستند، بلکه ملاحظات دیگری مانند اجرای هر دستور در یک سیکل و دسترسی کم به حافظه هم در آنها اعمال شده است، بر خلاف مینی‌کامپیوترهای کم‌دستور قدیمی.

پر دستور CISC	کم دستور RISC
تعداد دستورالعمل‌ها زیاد (بیشتر از ۱۰۰) و دست برنامه‌نویس باز است.	تعداد دستورالعمل‌ها کم (حدود ۳۰) و برنامه‌نویس باید آن را جبران کند.
زمان اجرای دستورات مختلف و در چند سیکل ساعت است. طول دستورات نیز متغیر است.	زمان اجرای دستورات یکسان و معمولا یک سیکل ساعت است. همه دستورالعمل‌ها هم طولند.
اکثر دستورات امکان دسترسی به حافظه را دارند.	دستوراتی که امکان دسترسی به حافظه را دارند حداقل ممکنه است (احتمالا فقط لود و استور).
رجیسترها متنوع و متعدد هستند.	فقط یک رجیستر داده وجود دارد.
مزایا: کدنویسی ساده‌تر، طول کد کوتاه‌تر و پیاده‌سازی کامپایلر ساده‌تر است.	مزایا: اولا زمان اجرای یک دستور کمتر است. ثانیا طرح سخت‌افزاری ساده‌تر بنابراین تعداد ترانزیستورها کمتر است.

CISC در دنیای امروز

- با پیشرفت تکنولوژی ساخت IC پردازنده‌های پردستور با حجم بسیار کم ساخته شدند که 8086 یکی از نخستین نمونه‌های آن است. بنابراین نیاز به ساخت کامپیوترهای مینی کم دستور برای مدتی کم شد. اما این پایان کار RISC نبود.
- در همان زمان پردازنده‌های عرضه شدند که میکروکنترلر نام گرفتند و 8085 یکی از آنها بود. این پردازنده‌های کم‌دستور برای پیاده‌سازی دستگاه‌های ساده (مثل مدار کنترل آسانسور) به کار می‌روند و نیاز به اجرای برنامه‌های پیچیده (همانند کامپیوترهای همه منظوره) ندارند.

RISC در دنیای امروز

- اما در زمان کنونی نقش CISC بسیار پررنگ شده است. هر چند تکنولوژی امکان جا دادن چند میلیارد ترانزیستور را در یک IC فراهم کرده اما چنین قطعه ابعاد چندین سانتی‌متری داشته و برای خنک کردن آن نیاز به فن و رادیاتور هست. این تجهیزات نمی‌توانند در یک گوشی موبایل یا تبلت نصب شوند.RISC در اینجا تنها چاره است.
- مشکل سختی نوشتن برنامه‌های پیچیده با پردازنده‌های RISC را (و همچنین مشکل ناسازگاری سخت افزارها را) «ماشین مجازی» حل کرده است.

ماشین مجازی

- کامپیوترهایی بودند که امروز دیگر نیستند. مثلا آتاری ۲۶۰۰ کنسول بازی محبوب بچه‌های دهه شصت شمسی بود. اما بازی‌ها (برنامه‌ها)ی آنها همچنان موجود است. با نوشتن یک ماشین مجازی بر روی کامپیوتر PC می‌توان کاری کرد که PC خود را بصورت آتاری شبیه سازی و همه آن بازی‌ها را اجرا کند.
- اما ماشین مجازی پروسه‌ای یک کامپیوتر غیر واقعی را شبیه‌سازی می‌کند (مثل ماشین مجازی .net و java روی انواع کامپیوتر) در این صورت مشکل عدم سازگاری سخت‌افزاری بین کامپیوترها به کلی حل می‌شود.
- آنچه نوآوری در سخت‌افزار را محدود کرد مساله سازگاری همه کامپیوترهای با پردازنده intel با هم بود. چرا که در صورت تغییر اساسی در سخت‌افزار همه نرم‌افزارها بایستی از نو نوشته می‌شدند. اما همه سیستم‌های اندروید امروزی دارای ماشین مجازی ART هستند و هر چند سخت‌افزارها متفاوت و ناسازگارند اما همگی می‌توانند برنامه‌های اندرویدی را اجرا کنند. مساله سختی کار با کامپیوترهای کم‌دستور هم به کلی حل شده است چون برنامه‌نویسان برنامه‌های قابل اجرا با ART را می‌نویسند.

کامپیوتر پایه

- کامپیوترهای کم دستور با طرح سخت‌افزاری ساده (علاوه بر کم بودن تعداد ترانزیستورها) مزیت دیگری نیز دارند.
- ساختار داخلی آنها را می‌توان به راحتی مطالعه و درک کرد. از همین رو برای درس معماری کامپیوتر مناسبند.
- آموختن معماری یک کامپیوتر ساده (و در عین حال کامل) ما را موفق به شناخت طرز کار دستگاهی بنام کامپیوتر می‌کند. در ادامه این درس یک کامپیوتر را بعنوان کامپیوتر پایه Basic انتخاب و طراحی آن را از ابتدا تا انتها انجام خواهیم داد.

کدام کامپیوتر پایه

- انتخاب (در دانشگاه‌های دنیا) بین دو کامپیوتر است: کامپیوتر پایه مانو و کامپیوتر پایه پترسون- هنسی
- کامپیوتر پایه مانو به کامپیوترPDP-8 که چند اسلاید قبل با آن آشنا شدیم شباهت زیادی دارد، هر چند با آن یکی نیست. رجیسترهای داده ۱۶ بیتی و رجیسترهای آدرس ۱۲ بیتی هستند (در PDP-8 همه ۱۲ بیتی‌اند).
- کامپیوتر پایه پترسون- هنسی به کامپیوترهای طراحی شده واقعی MIPS توسط این دو تن شباهت دارد هر چند بسیار ساده‌تر شده (دستورات فقط شامل ذخیره و بازیابی از حافظه، جمع، تفریق، و ، یا، شیفت به چپ، پرش غیر شرطی و پرش در صورت تساوی است).


معماری کامپیوتر
مولف: موریس مانو
مترجم: دکتر قدرت سپیدنام
طراحی، معماری
و
سازمان کامپیوتر
واسط سخت افزار / نرم افزار
ویراست
چهارم
دکتر احسان ملکیان
دکتر علی ذاکرالحسینی
پروفسور دیوید پترسون
پروفسور جان هنسی
چاپ سوم

کدام کامپیوتر پایه

- مزایای کتاب مانو: شخص موریس مانو معلمی فوق‌العاده است. این کتاب و همچنین کتاب مدارهای منطقی او بسیار ساده فهم نوشته شده که البته ترجمه دکتر سپیدنام هم در این امر موثر است.
- مزایای کتاب پَترسون-هنسی: نویسندگان در این زمینه واقعا دانشمندند. آنها سالها عضو (و رئیس) معتبرترین دانشگاه‌های دنیا بوده‌اند و نیز به خاطر طراحی کامپیوترهای RISC در سال ۲۰۱۷ جایزه تورینگ (معادل نوبل علم کامپیوتر) را گرفته‌اند. کامپیوتر معرفی شده در این کتاب جزئیاتی مشابه پردازنده‌های امروزی و مدرن RISC دارد.
- معایب کتاب پترسون-هنسی: کتاب به اندازه کتاب مانو خوب درک نمی‌شود. این امر هم به دلیل توضیح خوب مانو و هم به دلیل پیچیدگی زیادتر کامپیوتر پترسون-هنسی است. به طور کلی مطالعه این کامپیوتر پایه (اگر تدریس درست و کامل انجام شود) زمان بسیار بیشتری می‌گیرد. ترجمه این کتاب خوب ولی از ترجمه سپیدنام ضعیف‌تر است. همچنین نویسندگان کتاب که خودشان از بنیان‌گذاران MIPS هستند نگاهی ناعادلانه و تحقیرآمیز به محصولات کمپانی‌های دیگر (از جمله اینتل) دارند و این نگاه در تمام کتاب منعکس شده است.

شروع کار آشنایی با کامپیوتر مانو

- کامپیوتر مانو انتخاب ما برای این درس است. کامپیوتری که هرگز به صورت تجاری ساخته نشده ولی هر ترم دانشجویان جهان در درس آزمایشگاه معماری کامپیوتر صدها هزار نمونه از آن را می‌سازند و خراب می‌کنند.
- در این کامپیوتر هدف اصلی ساخت کامپیوتری است که قادر به اجرای همه برنامه‌ها باشد. هر چند که به برخی مباحث کارایی (سرعت اجرا) نیز پرداخته شده است.
- با شناخت کامل معماری یک کامپیوتر ساده (از جمله کامپیوتر مانو) ما طرز کار کامپیوتر را بطور واقعی درک خواهیم کرد.

تمرین

- برنامه‌ای بنویسید که دو عدد دلخواه (منفی یا مثبت) در دو رجیستر ax و bx قرار دهد. سپس آنها را مرتب کند به گونه ای که عدد کوچکتر در ax و بزرگتر در bx بماند.
- مینی کامپیوتر چیست و چه فرقی با مین‌فریم داشته است؟ معماری آنها (از نظر RISC و CISC) به هم چه تفاوتی دارد؟

معماری کامپیوتر

جلسه ششم

معماری مجموعه دستورالعمل‌های کامپیوتر مانو

درس امروز

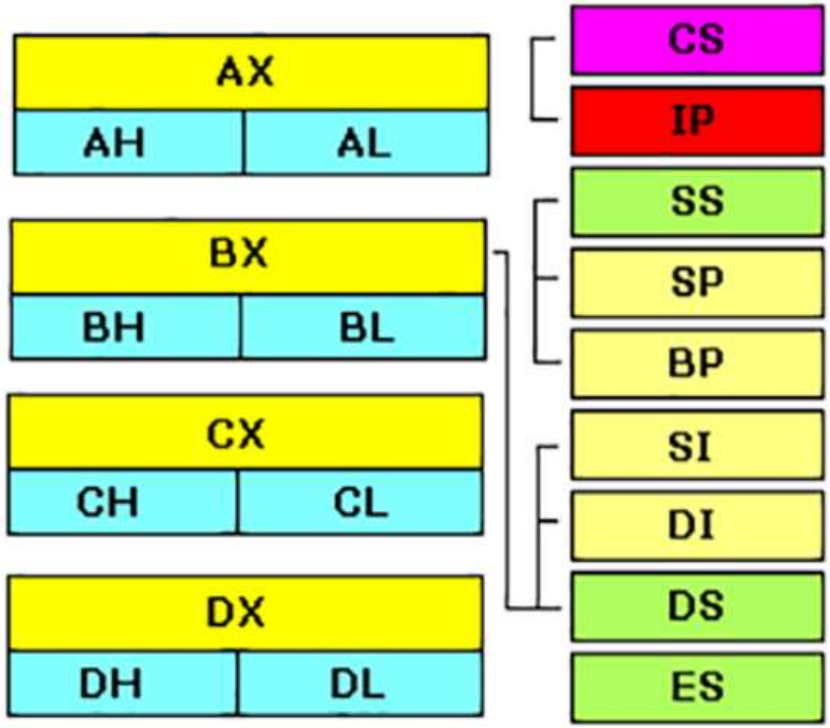
- به معماری دستورالعمل‌های کامپیوتر پایه خواهیم پرداخت. یعنی در ابتدا در مورد مشخصات کلی یعنی اندازه حافظه، اندازه داده‌ها (تعداد خطوط باس‌ها) و تعداد رجیسترها تصمیم می‌گیریم و سپس به طراحی مجموعه دستورالعمل می‌پردازیم.
- در این جلسه به طراحی سخت‌افزار نخواهیم پرداخت (بجز مختصری در طراحی باس داده). طراحی سخت‌افزاری که قرار است این دستورالعمل‌ها را اجرا کند کار جلسات آینده است.

فرم کلی کامپیوتر مانو

- قصد داریم کامپیوتری طراحی کنیم که:

1. آدرس آن ۱۲ بیتی و باس داده آن ۱۶ بیتی باشد. با چنین گذرگاه‌هایی کامپیوتر اولا ۴ کیلوبایت حافظه را آدرس‌دهی می‌کند (2^{12}). ثانیا اندازه داده‌ها و نیز دستورالعمل‌ها بصورت فیکس ۱۶ بیت خواهد بود.
2. تعداد رجیسترها حداقل یعنی تا جایی که ممکن است کم باشد.
3. تعداد دستورالعمل‌ها هم اندک باشد.
4. کامپیوتر ساده و تک منظوره، بنابراین تک برنامه در حال اجرا و بدون سیستم‌عامل باشد.
5. گرچه در کامپیوترهای RISC هر کدام از دستورالعمل‌ها قرار است در یک سیکل ساعت اجرا شوند اما از آنجا که فعلا قصد معرفی خط لوله را نداریم، محدودیتی در تعداد سیکل ساعت نداریم.

یادآوری رجیسترهای PC

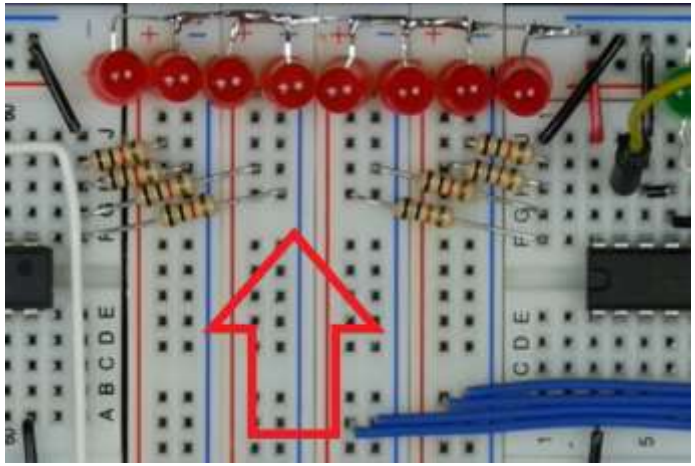


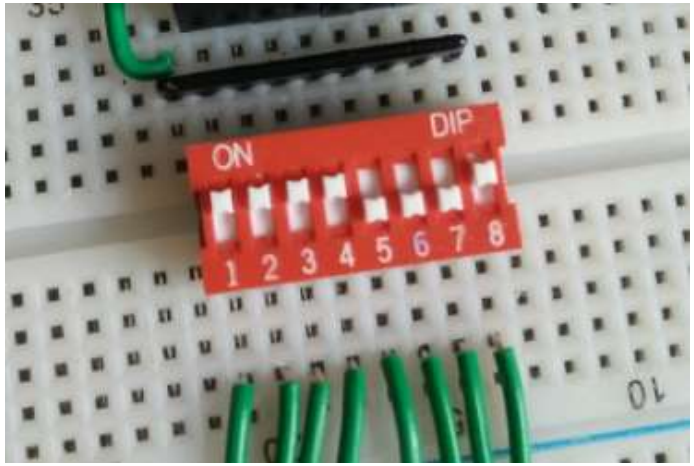
رجیسترهای در دسترس برنامه‌نویس

- از آنجا که کامپیوتر ساده و بدون سیستم‌عامل است حافظه سگمنت‌بندی نمی‌شود و آدرس حقیقی (۱۲ بیتی) خانه‌های حافظه مورد استفاده قرار می‌گیرد. بنابراین نه سگمنتی داریم و نه رجیستری برای سگمنت‌ها.
- پشته هم نداریم (خواهیم دید برگشت از زیرروال‌ها چگونه انجام می‌شود).
- فقط یک رجیستر داده عمومی برای مقدارهایی که برنامه زبان ماشین با آنها کار می‌کند داریم (بر خلاف ۸۰۸۶ که چهار رجیستر دارد). نام این تک رجیستر را اکومولیتر یا AC یا انباشتگر می‌گذاریم.
- رجیستری برای نگهداری آدرس دستورالعملی که قرار است بعدا اجرا شود داریم که نام آن شمارنده برنامه PC است (در ۸۰۸۶ دقیقا چنین رجیستری داشتیم فقط نامش IP بود). این رجیستر در برنامه مستقیما قابل مقداردهی نیست اما با دستورات پرش می‌توان مقدارش را تغییر داد.

رجیسترهای ورودی - خروجی

- در کامپیوترهای پیچیده از طریق وقفه‌ها و پورت و گذرگاه با دستگاه‌های ورودی و خروجی مختلف و متعدد (مثل مانیتور) ارتباط برقرار می‌شود.
- در کامپیوترهای ساده ورودی و خروجی بصورت رجیستر طراحی می‌شود. ورودی خروجی این رجیسترها در واقع سیم‌هایی هستند که از کامپیوتر خارج شده و به ابزار ورودی خروجی ساده (مثل لامپ یا سویچ) متصل می‌گردند.
- برنامه زبان ماشین می‌تواند در رجیستر خروجی مقدار بنویسد و از رجیستر ورودی مقدار بخواند.
- در کامپیوتر پایه ما ارتباط با ورودی و خروجی ۸ بیتی است. ورودی و خروجی محدود است به دو رجیستر ۸ بیتی INPR و OUTR.

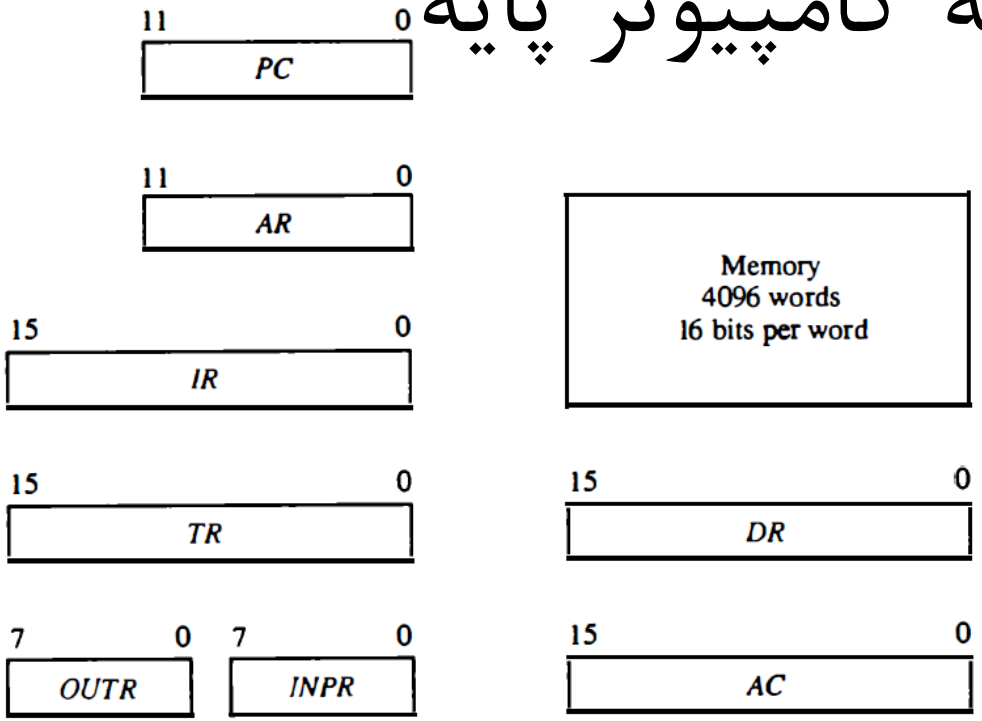




رجیسترهای خارج از دسترس برنامه‌نویس

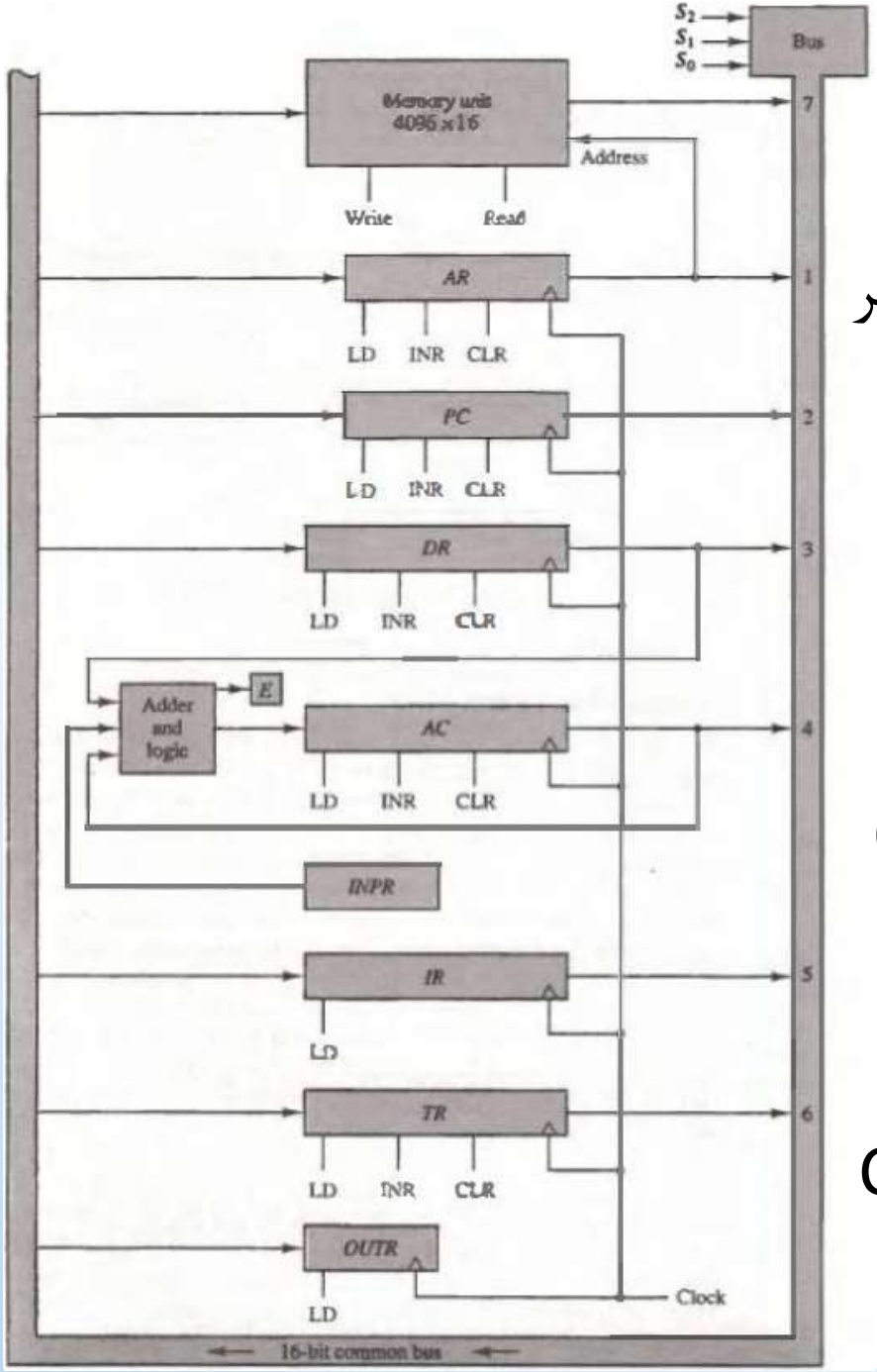
- در زمان آشنایی با زبان ماشین و اسمبلی ۸۰۸۶ فقط رجیسترهایی معرفی شد که توسط برنامه (زبان ماشین) مستقیما قابل دسترسی هستند. اما در اینجا (کامپیوتر پایه) چون هدف طراحی سخت‌افزاری نیز هست باید همه رجیسترها را بشناسیم:
- رجیستر دستورالعمل IR رجیستری است که دستورالعملی که می‌خواهد اجرا شود در آن قرار می‌گیرد.
- هرچند دسترسی به حافظه فقط با آدرس (مستقیم یا غیرمستقیم) است، اما پردازنده هنگام خواندن یا نوشتن در حافظه آدرس را در رجسیتر آدرس AR قرار داده و مقدار حافظه هم در رجیستر داده DR که حکم بافر حافظه دارد قرار می‌گیرد.
- بجز این ۷ رجیستر، رجیستر دیگری بنام رجیستر موقتی TR وجود دارد که کاربرد آن را خواهیم دید.
- رجیسترهایی که محتوی آدرس هستند (PC,AR) ۱۲ بیتی، رجیسترهای ورودی و خروجی ۸ بیتی و چهار رجیستر دیگر ۱۶ بیتی هستند.

رجیسترها و حافظه کامپیوتر پایه



Basic computer registers and memory.

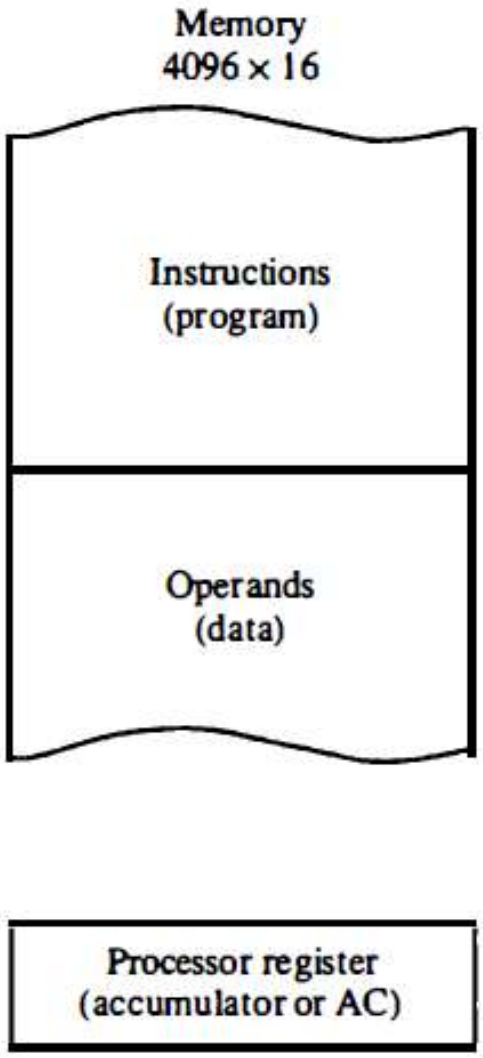
Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character



گذرگاه مشترک

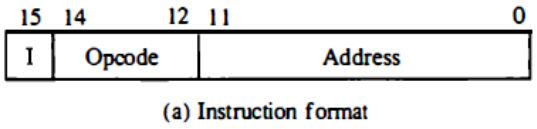
- سیستم گذرگاه مشترک یا باس ارتباط میان رجیسترها با یکدیگر و با حافظه را فراهم می‌کند.
- وقتی یکی از رجیسترهای ۱۲ بیتی به باس ۱۶ بیتی وارد می‌شود، چهار بیت پرارزش 0 قرار داده می‌شوند.
- مقدار ورودی $S_2S_1S_0$ تعیین می‌کند که کدام ورودی روی باس قرار گیرد. مثلا مقدار 011 مقدار DR را روی باس می‌فرستد.
- ورودی LD هر رجیستر معلوم می‌کند که آن رجیستری است که مقدار از باس برمی‌دارد. CLR آن را صفر می‌کند و INR یکی به آن می‌افزاید.

سازمان حافظه

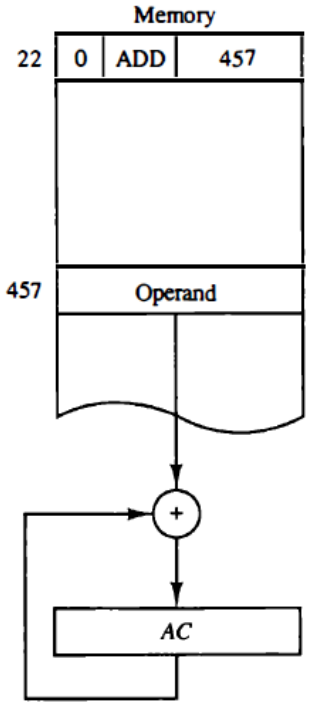


- در کامپیوتر پایه تمامی ۴۰۹۶ خانه حافظه در اختیار برنامه‌نویس است.
- بنابراین طبیعی است که برنامه اجرایی در خانه صفر به بعد قرار گیرد. وقتی کد برنامه به اتمام رسید بخش داده‌های برنامه شروع می‌شود.
- مثلا اگر برنامه ۱۰۰ دستور طول دارد در خانه ۰ تا ۹۹ مستقر شده و اولین متغیر برنامه نیز در آدرس ۱۰۰ حافظه جای خواهد داشت.
- جز این حافظه ۴کیلوبایتی و رجیستر AC، مکان دیگری برای دیتا نیست.

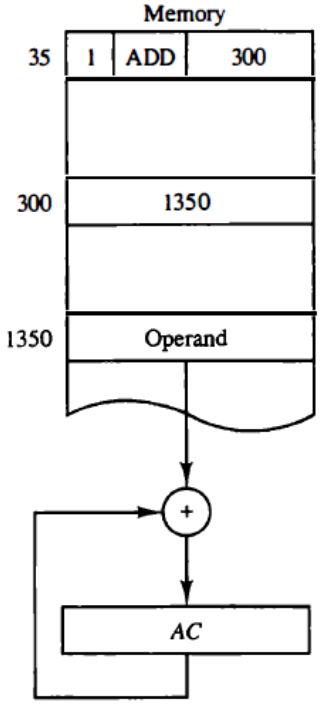
آدرس‌دهی حافظه



(a) Instruction format



(b) Direct address



(c) Indirect address

- مشابه ۸۰۸۶، در کامپیوتر پایه نیز حافظه با دو روش مستقیم (آدرس در کد برنامه) و غیر مستقیم (آدرس در یک مکان خاص) می‌توان به حافظه دسترسی پیدا کرد.
- در ۸۰۸۶ BX رجیستر پایه بود و آدرس حافظه غیر مستقیم را نگه می‌داشت. اما در اینجا محل نگهداری آدرس (**پوینتر**) خودش خانه‌ای از حافظه‌است که آدرس آن را داریم.
- در شکل روبرو I=0 نشانه حافظه مستقیم و I=1 غیرمستقیم است.

انواع دستورالعمل‌های انتقال

- کامپیوتر پایه از عملوند بلافصل (مثل mov ax,5) پشتیبانی نمی‌کند (چون طول دستورالعمل ثابت و فقط ۱۶ بیت است).
- در کامپیوتر پایه فقط یک رجیستر داده داریم (AC) بنابراین صحبت از انتقال بین ثبات‌ها بی‌معناست.
- انتقال حافظه به حافظه هم که هم در ۸۰۸۶ و هم در اینجا ممنوع و ناممکن است.
- بنابراین **تنها** نوع انتقالی که باقی می‌ماند بارکردن (load) رجیستر AC از حافظه و یا ذخیره کردن (store) رجیستر AC در خانه‌ای از حافظه است.

دستورالعمل‌های محاسباتی

- از آنجا که هدف حداقل بودن تعداد دستورات در کامپیوتر پایه بوده، فقط دو دستورالعمل دو عملوندی وجود دارند: ADD (جمع) و AND (و).
- به همان دلیلی که در مورد انتقال گفته شد، تنها حالت دو عملوندی ممکن یک طرف AC و یک طرف حافظه است.
- دستورالعمل‌های محاسباتی تک عملوندی (که فقط روی AC عمل می‌کنند)، تنوع بیشتری دارند:
  - ❊ پاک کردن AC (همه بیت ها صفر)
  - ❊ متمم کردن AC (بیت‌های صفر و یک برعکس)
  - ❊ افزایش AC (++)
  - ❊ شیفت چرخشی AC (با دخالت E)

فلیپ‌فلاپ E

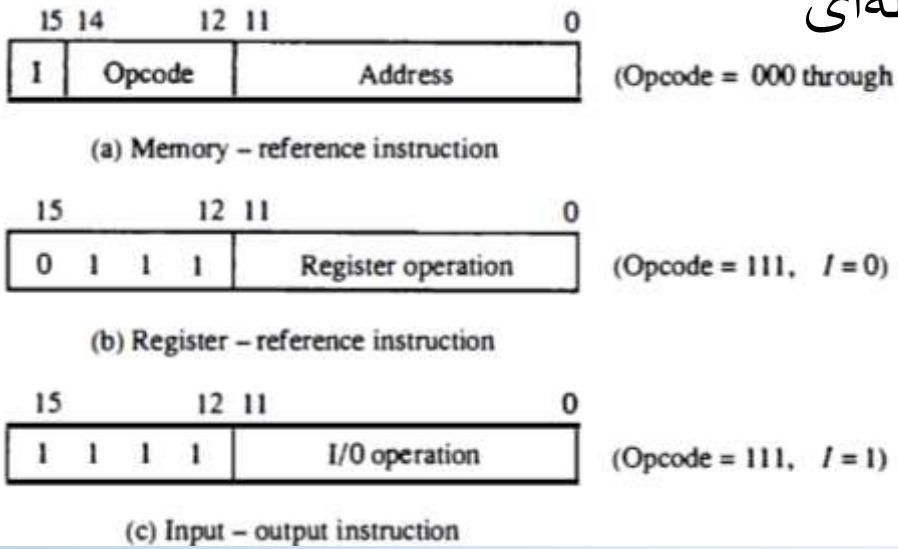
- در کامپیوتر پایه فقط یک پرچم محاسباتی هست و بخاطر آنکه می‌توان آن را گسترش AC دانست E نام دارد.
- رجیسترها مجموعه‌ای چندتایی از فلیپ‌فلاپ‌هایند، بنابراین یک بیت تکی را بجای پرچم، فلیپ‌فلاپ هم می‌توان نامید.
- E در صورتی که جمع نقلی (کری) داشته باشد ست می‌شود. هنگام شیفت چرخشی هم مقدار E از یک طرف وارد رجیستر شده و بیتی که از آن طرف خارج می‌شود بعد در E قرار می‌گیرد.
- دستورات دیگری هم هست که مستقیما بر E اثرگذارند:
  - پاک کردن E
  - متمم کردن E

دستورات پرش و انشعاب

- در کامپیوتر پایه دو نوع پرش وجود دارد:
۱. پرش Skip از روی دستور بعدی (مشروط)
۲. انشعاب Branch به یک آدرس در حافظه (نامشروط)
- پرش‌های مشروط در صورت برقراری شرطی (مثلا صفر بودن AC یا صفر بودن E) انجام می‌شوند.
- دستور ISZ هم هست که در پیاده کردن حلقه‌ها به درد می‌خورد. این دستور ابتدا یک واحد به محتوای خانه‌ای در حافظه می‌افزاید و سپس اگر مقدار آن خانه صفر شده باشد از دستور بعدی پرش می‌کند.
- پرش‌هایی که با ذخیره آدرس برگشت انجام می‌شود را هنگام برنامه‌نویسی بررسی خواهیم کرد.

قالب دستورالعمل‌ها

- همانطور که می‌دانیم دستورات زبان ماشین کامپیوتر پایه ۱۶ بیتی است.
- همانطور که در شکل زیر می‌بینید ۱۲ بیت کم ارزش (بیت‌های ۰ تا ۱۱) در دستورات حافظه‌ای به آدرس حافظه و در دستورات رجیستری و ورودی خروجی نوع عمل (فقط یک بیت 1) است.
- بیت‌های ۱۲و۱۳و۱۴ که opcode نامیده می‌شوند در دستورات حافظه‌ای (بخش بالای صفحه بعد) نوع عمل را نشان می‌دهند. در دستورات رجیستری (بخش وسط) و ورودی خروجی (بخش پایین) این سه بیت 111 هستند.
- بیت ۱۵ یا **I** در دستورات حافظه‌ای اگر 0 باشد دستور را حافظه مستقیم و اگر یک باشد حافظه غیر مستقیم می‌کند. در حالت غیر حافظه‌ای هم 0 بودن این بیت نشانه رجیستری و 1 بودن آن نشانه ورودی خروجی بودن دستور است.



Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description	
AND	0xxx	8xxx	AND memory word to *AC*	AND کردن کلمه حافظه با AC
ADD	1xxx	9xxx	Add memory word to *AC*	جمع کردن کلمه حافظه با AC
LDA	2xxx	Axxx	Load memory word to *AC*	بار کردن کلمه حافظه در AC
STA	3xxx	Bxxx	Store content of *AC* in memory	ذخیره محتوای AC در حافظه
BUN	4xxx	Cxxx	Branch unconditionally	انشعاب نامشروط
BSA	5xxx	Dxxx	Branch and save return address	انشعاب و ضبط آدرس بازگشت
ISZ	6xxx	Exxx	Increment and skip if zero	افزایش و گذر در صورت نتیجه صفر
CLA	7800		Clear *AC*	پاک کردن AC
CLE	7400		Clear *E*	پاک کردن E
CMA	7200		Complement *AC*	متمم کردن AC
CME	7100		Complement *E*	متمم کردن E
CIR	7080		Circulate right *AC* and *E*	چرخش AC و E به راست
CIL	7040		Circulate left *AC* and *E*	چرخش AC و E به چپ
INC	7020		Increment *AC*	افزایش AC
SPA	7010		Skip next instruction if *AC* positive	گذر از دستور بعدی اگر AC مثبت باشد
SNA	7008		Skip next instruction if *AC* negative	گذر از دستور بعدی اگر AC منفی باشد
SZA	7004		Skip next instruction if *AC* zero	گذر از دستور بعدی اگر AC صفر باشد
SZE	7002		Skip next instruction if *E* is 0	گذر از دستور بعدی اگر E صفر باشد
HLT	7001		Halt computer	توقف کامپیوتر
INP	F800		Input character to *AC*	دریافت کاراکتر و انتقال آن به AC
OUT	F400		Output character from *AC*	برداشتن کاراکتر از AC و انتقال آن به خر
SKI	F200		Skip on input flag	گذر مبتنی بر پرچم ورودی
SKO	F100		Skip on output flag	گذر مبتنی بر پرچم خروجی
ION	F080		Interrupt on	فعال‌کردن وقفه‌ها
IOF	F040		Interrupt off	غیرفعال کردن وقفه‌ها

مجموعه تمامی دستورالعمل‌ها

- مجموعه تمامی دستورالعمل‌های کامپیوتر پایه مانو را در صفحه قبل می‌بینید.
- این کامپیوتر ۷ دستور حافظه (۷ دستورالعمل حافظه مستقیم و ۷ دستورالعمل حافظه غیر مستقیم) دارد.
- ۱۲ دستور در مجموعه دستورات رجیستری است که یکی از آنها HLT موجب توقف اجرای برنامه می‌شود.
- ۶ دستور ورودی و خروجی و وقفه نیز وجود دارد که در آینده به آنها خواهیم پرداخت.
- بنابراین کل دستورات این کامپیوتر ۲۵ تاست.

ارزیابی کامپیوتر پایه

- این کامپیوتر دارای پیاده‌سازی سخت‌افزاری جمع است. با استفاده از جمع و مکمل می‌توان تفریق را برنامه نویسی کرد و نیر ضرب و تقسیم با شیفت و جمع قابل انجام است. همچنین با داشتن AND و NOT (مکمل) می‌توان به NAND رسید که اجرای تمام عملگرهای منطقی با آن ممکن است. بنابراین کامپیوتر پایه در واحد محاسبه و منطق کامل است و کمبودی ندارد.
- در قسمت کنترل برنامه نیز پرش‌های شرطی و نیز پرش‌هایی که امکان ایجاد حلقه و زیر روال را فراهم می‌کنند در آن تعبیه شده است، هر چند ضعیفتر از ۸۰۸۶ است: اجرای همزمان چند نسخه از یک زیرروال (که خودفراخوانی را فراهم می‌کند) ممکن نیست.
- هر چند این کامپیوتر تقریبا کامل است اما کارایی (سرعت اجرا) خوبی ندارد. وجود تنها یک رجیستر موجب مراجعه فراوان به حافظه می‌شود که وقت‌گیر است. کم بودن دستورات هم موجب می‌شود که مثلا برای یک تفریق نیاز به دو جمع و یک مکمل باشد در حالی که در کامپیوترهایی که پیاده سازی سخت‌افزاری تفریق را دارند برای جمع و تفریق زمان یکسانی صرف می‌شود.

یک مثال ساده برنامه‌نویسی

- برنامه‌ای بنویسید [هم به کد ماشین هم به اسمبلی] که دو عدد ۸۳ و ۲۳- را با هم جمع کند (مطابق برنامه C زیر) (موقع برنامه‌نویسی مجموعه دستورالعمل‌ها را در دسترس نگاه داشته و در صورت لزوم به آن مراجعه می‌کنیم)

```c
int main()
{
    short int a=83, b=-23, c;
    c = a + b;
}
```

Symbol	Hexadecimal code	Description
AND	0 or 8	AND *M* to *AC*
ADD	1 or 9	Add *M* to *AC*, carry to *E*
LDA	2 or A	Load *AC* from *M*
STA	3 or B	Store *AC* in *M*
BUN	4 or C	Branch unconditionally to *m*
BSA	5 or D	Save return address in *m* and branch to *m* + 1
ISZ	6 or E	Increment *M* and skip if zero
CLA	7800	Clear *AC*
CLE	7400	Clear *E*
CMA	7200	Complement *AC*
CME	7100	Complement *E*
CIR	7080	Circulate right *E* and *AC*
CIL	7040	Circulate left *E* and *AC*
INC	7020	Increment *AC*,
SPA	7010	Skip if *AC* is positive
SNA	7008	Skip if *AC* is negative
SZA	7004	Skip if *AC* is zero
SZE	7002	Skip if *E* is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

مثال

- اگر برنامه را کدهای سمبلیک بنویسیم و دستی آن را به زبان ماشین (کد شانزده‌شانزدهی یا دودویی) تبدیل کنید باید آدرس‌ها را هم خودمان محاسبه کنیم، اما اگر اسمبلی بنویسیم، اسمبلر خودش آدرس‌ها را محاسبه خواهد کرد.

Assembly Language Program to Add Two Numbers

```
        ORG 0       /Origin of program is location 0
        LDA A       /Load operand from location A
        ADD B       /Add operand from location B
        STA C       /Store sum in location C
        HLT         /Halt computer
A,      DEC 83      /Decimal operand
B,      DEC -23     /Decimal operand
C,      DEC 0       /Sum stored in location C
        END         /End of symbolic program
```

Hexadecimal Program to Add Two Numbers

Location	Instruction
000	2004
001	1005
002	3006
003	7001
004	0053
005	FFE9
006	0000

Program with Symbolic Operation Codes

Location	Instruction	Comments
000	LDA 004	Load first operand into *AC*
001	ADD 005	Add second operand to *AC*
002	STA 006	Store sum in location 006
003	HLT	Halt computer
004	0053	First operand
005	FFE9	Second operand (negative)
006	0000	Store sum here

Binary Program to Add Two Numbers

Location	Instruction code
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

تمرین

- (در حداکثر پنج سطر) شرح دهید که نظام حافظه کامپیوتر پایه مانو با ۸۰۸۶ چه تفاوت‌هایی دارد (از نظر آدرس‌دهی، سگمنت بندی، پشته و...)
- برای کامپیوتر پایه مانو برنامه‌ای بنویسید (هم به زبان اسمبلی و هم به کد ماشین شانزده شانزدهی) که معادل برنامه زیر را انجام دهد.

```
int main()
{
    short int a=83, b=-23, c=0;
    if (a > b) c = a - b; else c = 0;
}
```

معماری کامپیوتر

جلسه هفتم

سازماندهی کامپیوتر مانو
واحد اجرا EU:
زمانبندی و سیکل دستورالعمل، فچ

آنچه از کامپیوتر پایه می‌دانیم

- درک مطالب این جلسه نیازمند دانستن مطالب جلسه قبل است، شامل:

۱. مجموعه رجیسترهای کامپیوتر پایه مانو و سازمان حافظه این کامپیوتر و باس مشترکی که راه ارتباطی اینهاست.
۲. فرمت (بیت‌های) دستورالعمل‌های کامپیوتر پایه.
۳. مجموعه دستورالعمل‌های این کامپیوتر.

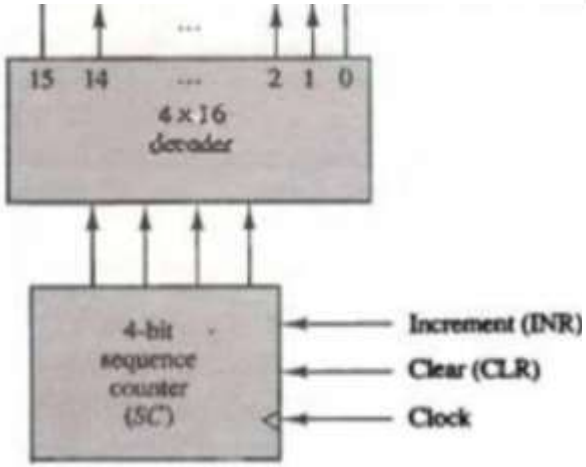
- در این جلسه به این موضوع خواهیم پرداخت:
  واحد کنترل کامپیوتر پایه
  نحوه زمانبندی و کنترل (سیکل دستورالعمل‌ها)

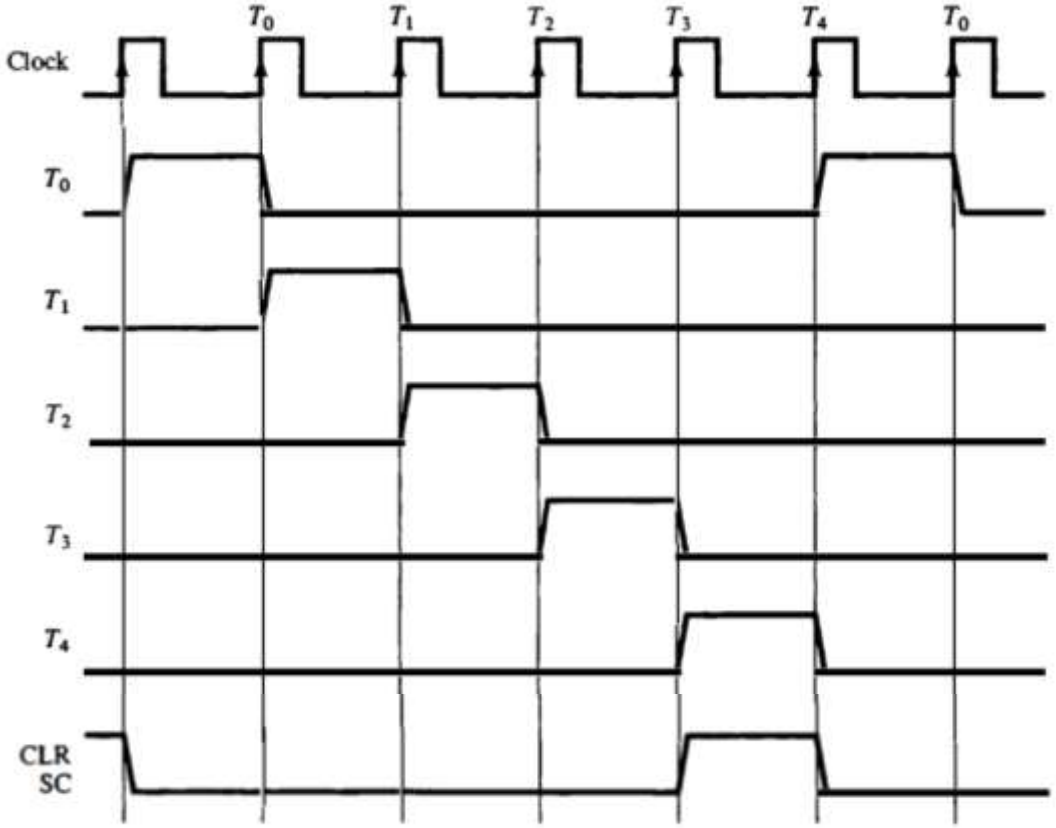
پالس ساعت و سیگنال‌های کنترل

- کامپیوتر یک ماشین سنکرون است و همه تغییرات در آن همگام با رسیدن لبه پالس ساعت انجام می‌شود.
- با این حال، پالس‌های ساعت مستقیما به خط کنترلی متصل نیستند و به خودی خود موجب تغییر هیچ مقداری در کامپیوتر نمی‌شوند. پالس‌های ساعت سیگنال‌های کنترل را تغییر می‌دهند و هر سیگنال کنترل موجب تغییراتی می‌شود.
- هر دستورالعمل کامپیوتر پایه در چند سیکل انجام می‌شود. شمارنده و سیگنال‌های کنترل (مثلا T3 در اسلاید بعد) مشخص می‌کنند که در چندمین سیکل از دستورالعمل فعلی هستیم و در نتیجه الان باید چه کاری انجام شود.
- در هر سیکل بخشی از کارهای انجام دستورالعمل انجام می‌شود که به آن ریز دستورالعمل می‌گویند. مثل یک دستورالعمل که در چهار سیکل ساعت انجام می‌شود شامل چهار ریز دستورالعمل است.

پالس ساعت و سیگنال‌های کنترل

- برای ساخت سیگنال‌های کنترل در بخشی از واحد کنترل یک شمارنده ۴ بیتی (SC) و یک دیکدر ۴ به ۱۶ قرار داده شده که این سیگنال‌ها را می‌سازند. با رسیدن هر پالس ساعت یک سیگنال کنترل جدید فعال می‌شود که نشان می‌دهد الان در سیکل چندم از دستورالعمل جاری هستیم.
- پاک‌کننده (CLR) در شروع هر دستورالعمل جدید شمارنده را صفر می‌کند.

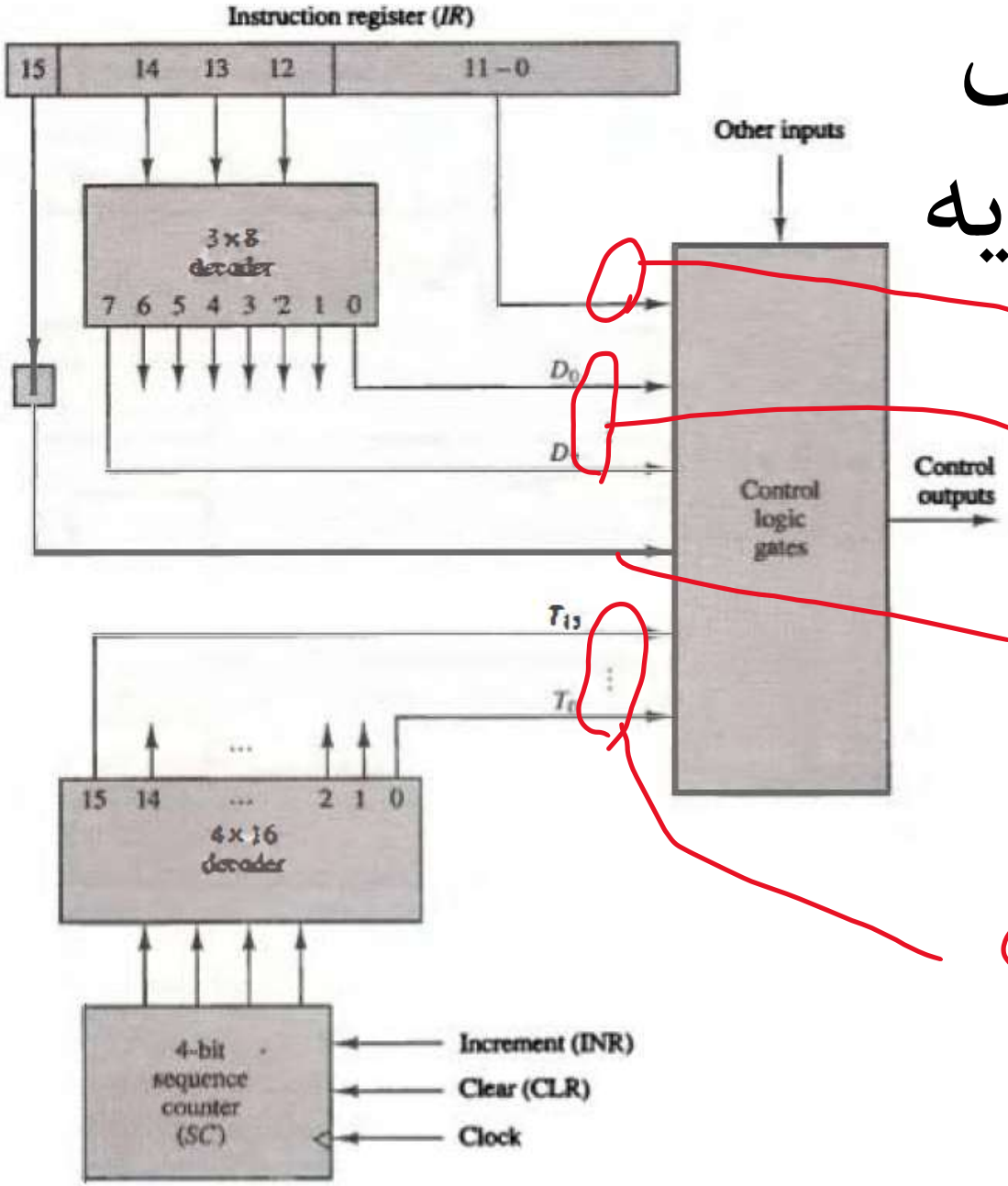




آوردن دستور به پردازنده

- (شکل اسلاید بعد) دستورات زبان ماشین داخل حافظه قرار دارند و می‌دانیم دستوری که قرار است الان اجرا شود بایستی ابتدا به درون رجسیتر دستورالعمل یعنی IR آورده شود.
- پس از ورود به IR، بخش آدرس حافظه (بیت‌های 0 تا 12) به رجیستر آدرس AR و بیت 15 به یک فلیپ‌فلاپ تکی بنام I منتقل می‌شود (این کار به درد زمانی می‌خورد که دستور حافظه‌ای است).
- بیت‌های opcode یعنی 12,13,14 نیز توسط یک دیکدر سه به هشت دیکد می‌شوند (همیشه بدرد می‌خورد).
- بیت‌های کم ارزش یعنی 0 تا 11 نیازی به دیکد ندارند چون اگر به مجموعه دستورالعمل‌های کامپیوتر رجوع کنیم می‌بینیم در هر دستور رجیستری یا IO فقط یکی از آنها 1 است (زمانی مورد استفاده قرار می‌گیرد که دستور رجیستری یا ورودی خروجی باشد).
- در نهایت ورودی واحد کنترل کامپیوتر پایه به این ترتیب شکل می‌گیرد: ۱۵ خط از قسمت سیکل دستورالعمل، I، ۸ خط از دیکد آپکد و ۱۲ خط از بیت‌های کم ارزش= ۳۶ خط.

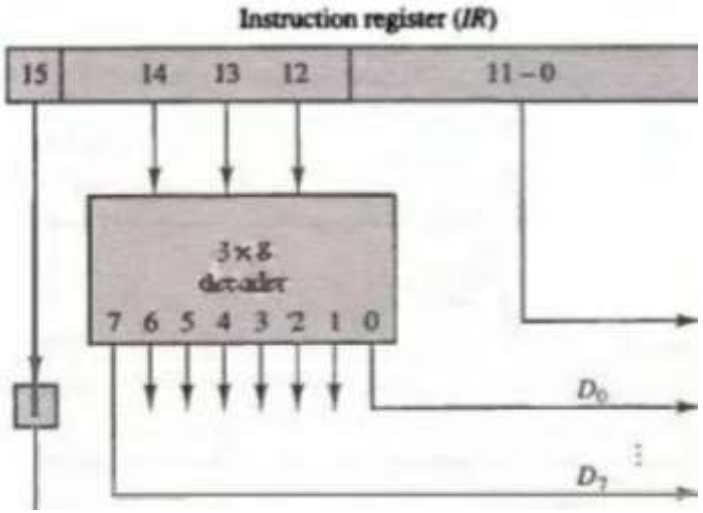
واحد کنترل کامپیوتر پایه

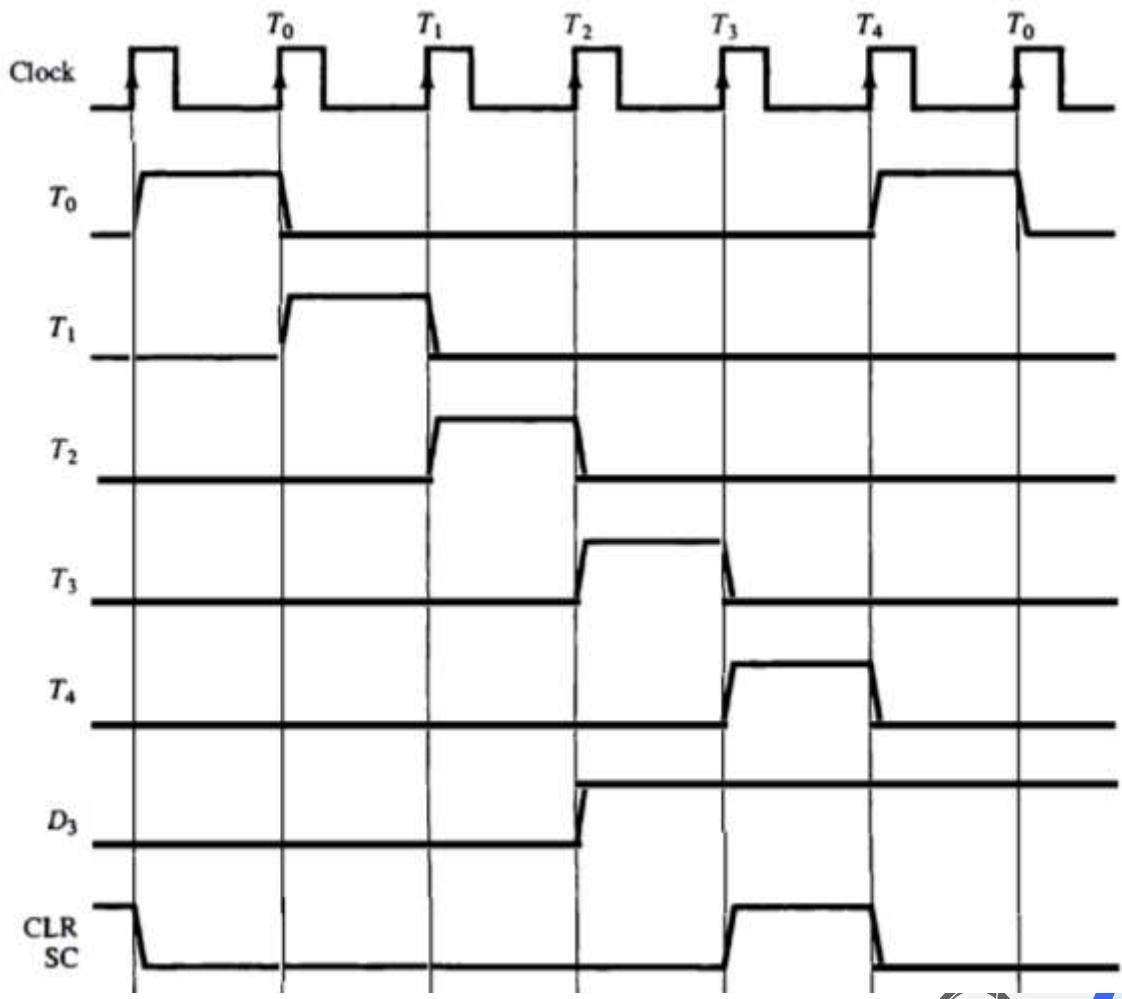


چه دستور رجیستری داریم؟

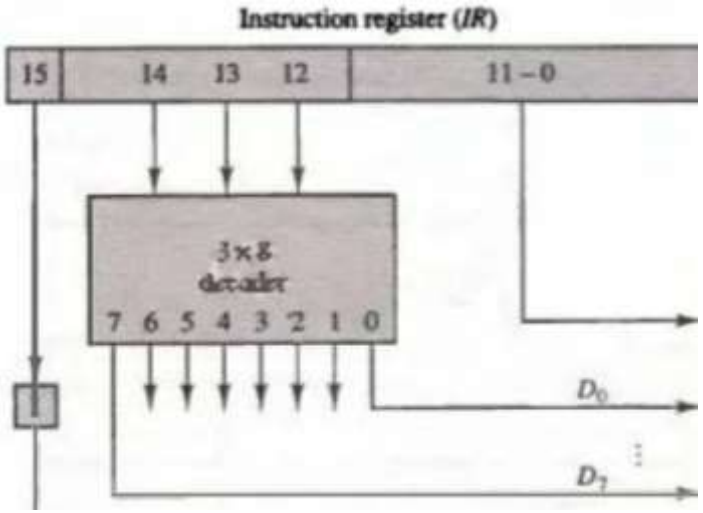
چه دستور حافظه‌ای؟

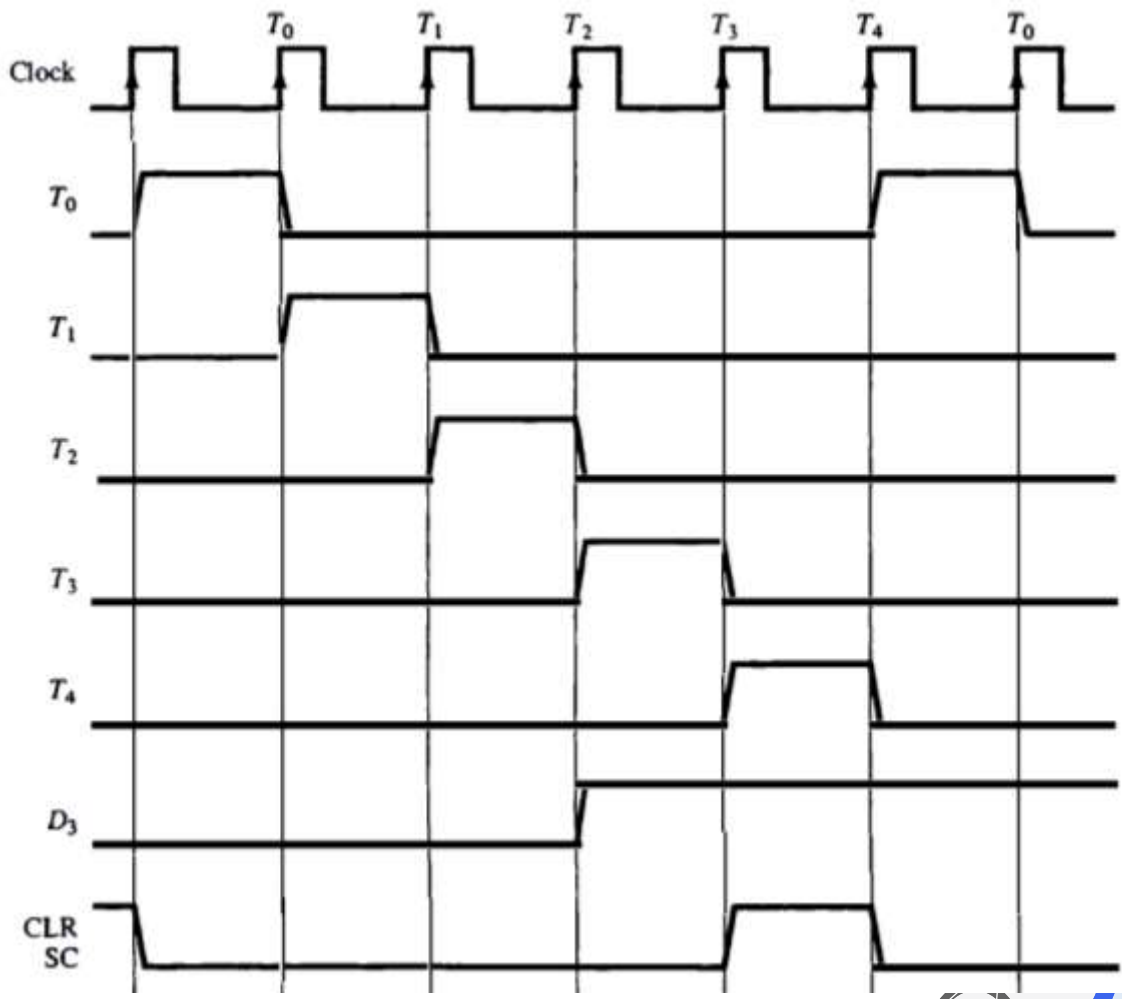
در کدام سیکل پیستم؟





- چرا D3 در سیکل T3 تغییر کرده ؟
- چرا شمارنده سیکل پس از T5 ریست شده؟





- همانطور که می‌بینید D3 در سیکل T3 تغییر کرده (یعنی تا قبل از آن اصلا دستورالعمل داخل IR قرار نگرفته بوده) پس متوجه می‌شویم که سه سیکل نخست (T0تاT2) صرف آوردن دستور به IR و دیکد شده است.
- همچنین می‌بینیم که دو سیکل دیگر (T3,T4) صرف اجرای دستور شده و پس از آن چون اجرای دستور تمام شده شمارنده صفر گردیده است. پس در مجموع برای اجرای این دستورالعمل خاص 5 سیکل زمان صرف شده است.

سیکل دستورالعمل

- هر برنامه از دنباله‌ای از دستورالعمل‌ها تشکیل شده که بترتیب یکی یکی به پردازنده آورده شده و اجرا می‌شوند.
- هر دوره کاری دستورالعمل شامل سیکل‌های جزیی (فازهایی) است.
- در کامپیوتر پایه هردوره دستورالعمل از این فازها تشکیل شده:
  ۱. فچ کردن دستور از حافظه
  ۲. دیکد کردن دستور
  ۳. خواندن آدرس موثر (در دستورهای حافظه – غیر مستقیم)
  ۴. اجرای دستور
- پس از پایان این چهار مرحله دوباره مرحله ۱ انجام می‌شود مگر آنکه آخرین دستور HLT باشد.
- کلمه فچ (Fetch) به معنای بازی‌ای است که در حضور سگ شی‌ایَ به فاصله دور پرتاب می‌شود و سگ به دنبال آن می‌رود و آن را باز می‌گرداند. این کلمه در برخی کتاب‌های فارسی ..... "واکشی" معنا شده است.

فچ و دیکد

- در آغاز اجرا برنامه، شمارنده برنامه PC با آدرس اولین دستور (در کامپیوتر پایه معمولا خانه 0) بارگذاری می‌شود. SC پاک می‌شود و در نتیجه سیگنال T0 ساخته می‌شود.
- پس از رسیدن هر پالس ساعت مقدار SC یکی افزایش یافته و T2,T1,T0 متوالیا تولید می‌شوند (مطابق شکل اسلاید ۴).
- ریزعمل‌های فازها برداشت و دیکد عبارت خواهند بود از:

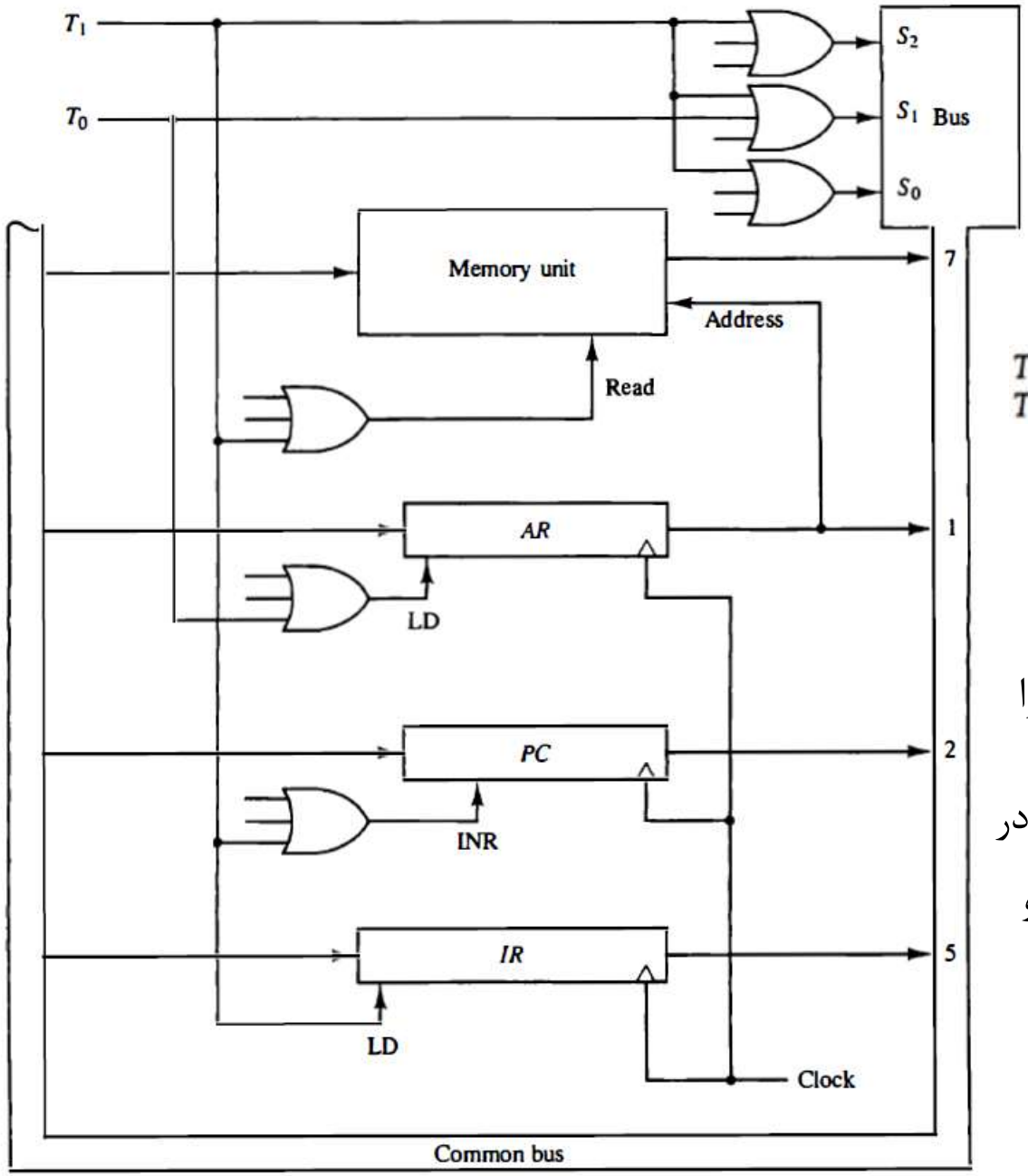
T_0: $AR \leftarrow PC$

T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$

T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$

- می‌دانیم AR تنها رجیستری است که مستقیما به ورودی آدرس حافظه وصل است (رجوع به اسلاید ۸ جلسه قبل). بنابراین آدرس دستورالعمل که در PC است باید ابتدا به AR منتقل شود. سپس خود دستور خوانده شده و به IR منتقل می‌شود و PC هم یکی افزایش می‌یابد تا احتمالا (اگر پرشی اتفاق نیافتد) آدرس دستور بعد را پیدا کند.

پیاده‌سازی انتقال رجیسترها در فاز فچ



T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$

در واحد حافظه، پایه Read آن را در وضعیت خواندن و Write در وضعیت نوشتن قرار می‌دهد. در مورد رجیسترها ورودی LD آن را در وضعیت بارگیری (از باس) قرار می‌دهد، CLR آن را صفر می‌کند و INR یکی به آن می‌افزاید.

تمرین

- در اسلاید صفحه ۱۱، فرض کنید قرار است:
T2: PC <- AR
مدار مربوط به T2 را رسم کنید (روی کاغذ شکل را کپی کنید و بعد T2 را به آن اضافه کنید یا با نرم‌افزارهای گرافیکی روی شکل کار کنید)
- شرح دهید که سیگنال‌های زمانبندی (مثل T1) چگونه تولید می‌شوند و چه استفاده‌ای دارند.

معماری کامپیوتر

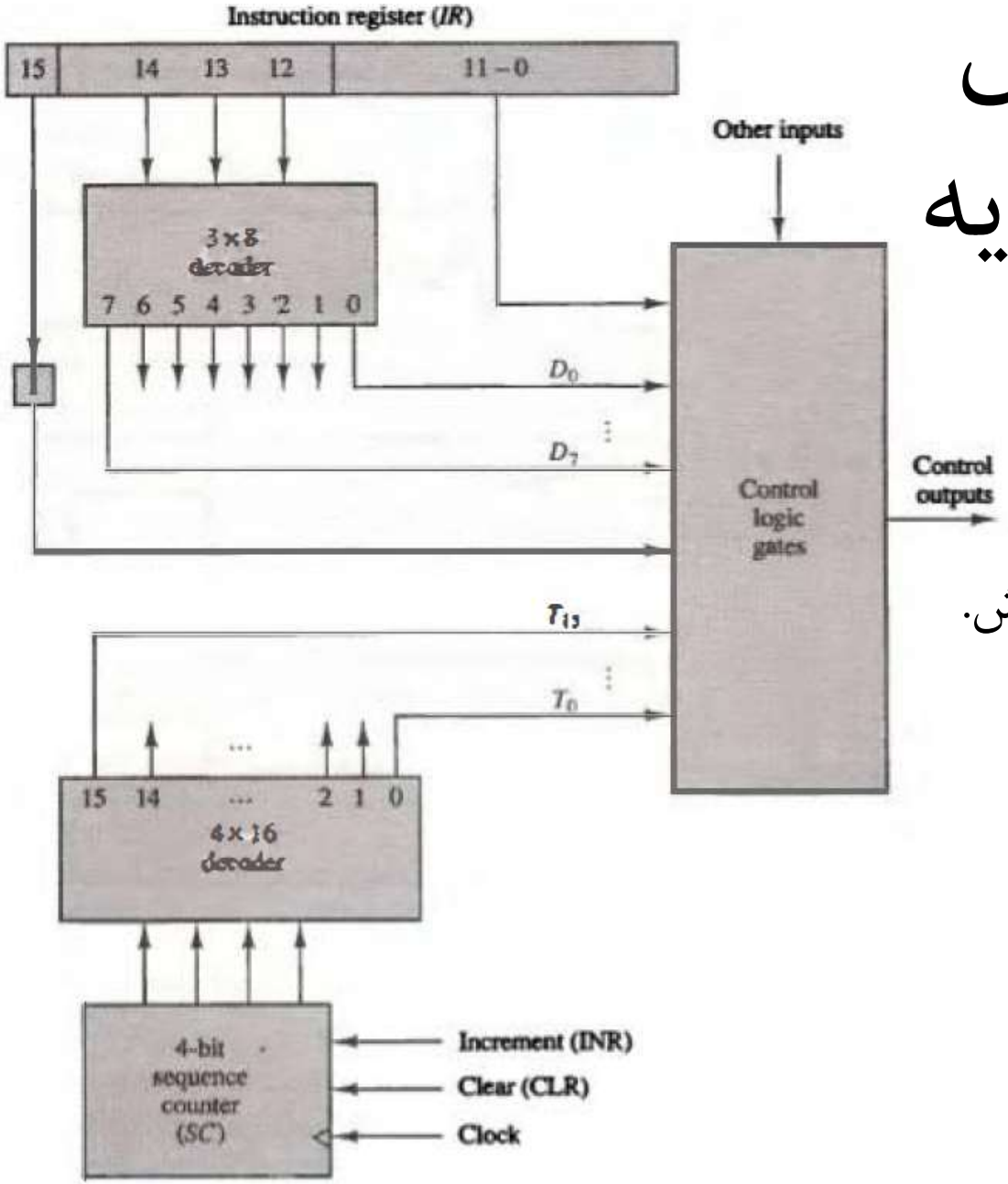
جلسه هشتم

سازماندهی کامپیوتر مانو:
اجرای دستورالعمل‌ها

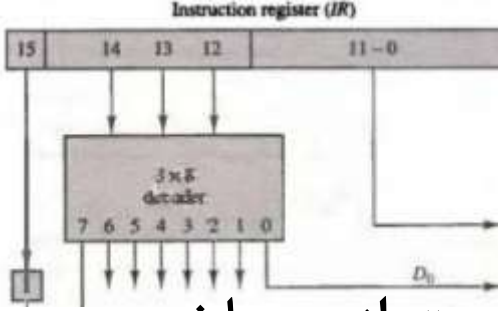
آنچه از کامپیوتر پایه آموخته‌ایم:

- در جلسه گذشته دیدیم که طرح کلی سخت‌افزاری کامپیوتر پایه مانو که قرار است مجموعه ۲۵ تایی دستورالعمل‌ها را اجرا کند، چگونه است. از آنجا که در این جلسه هنوز به این طراحی نیاز داریم، آن را در اسلاید بعد تکرار می‌کنیم.
- همچنین دیدیم که مرحله فچ (آوردن دستور از حافظه به پردازنده) که در تمام دستورالعمل‌ها مشترک است و در دو سیکل T0 و T1 انجام می‌شود چگونه است و چه طراحی سخت‌افزاری دارد.
- در سیکل T2 مرحله دیکد انجام می‌شود. بیت‌های IR12,13,14 بوسیله دیکدر تبدیل به D0 تا D7 می‌شوند، IR0 تا IR11 به AR (رجیستر آدرس) انتقال پیدا کرده و IR15 هم وارد فلیپ‌فلاپی بنام **I** می‌شود.

واحد کنترل کامپیوتر پایه



- ۳۶ خط شامل:
۱۶ خط نماینده سیکل دستورالعمل، I،
۸ خط از دیکد آپکد و
۱۲ خط از بیت‌های کم ارزش.

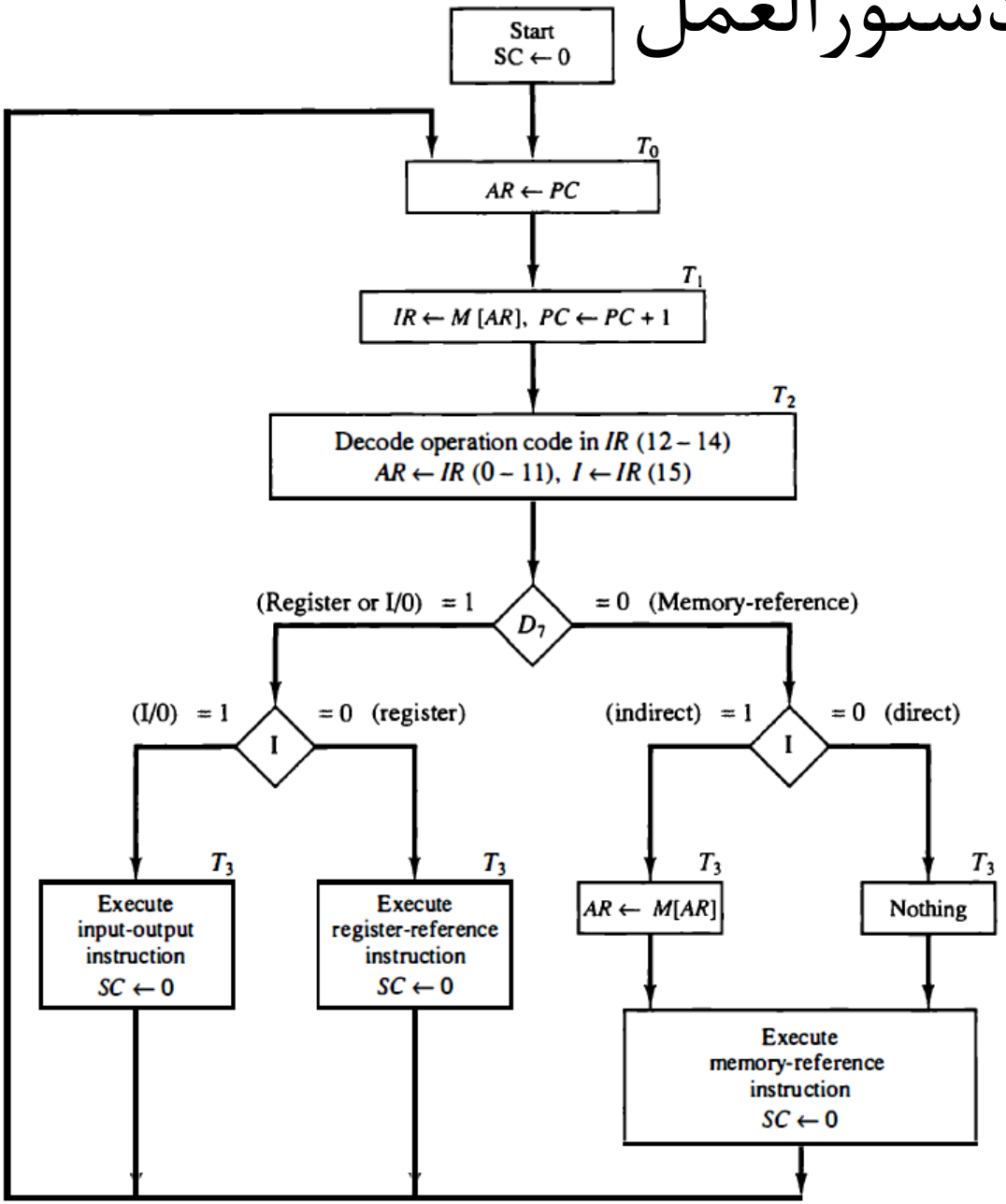


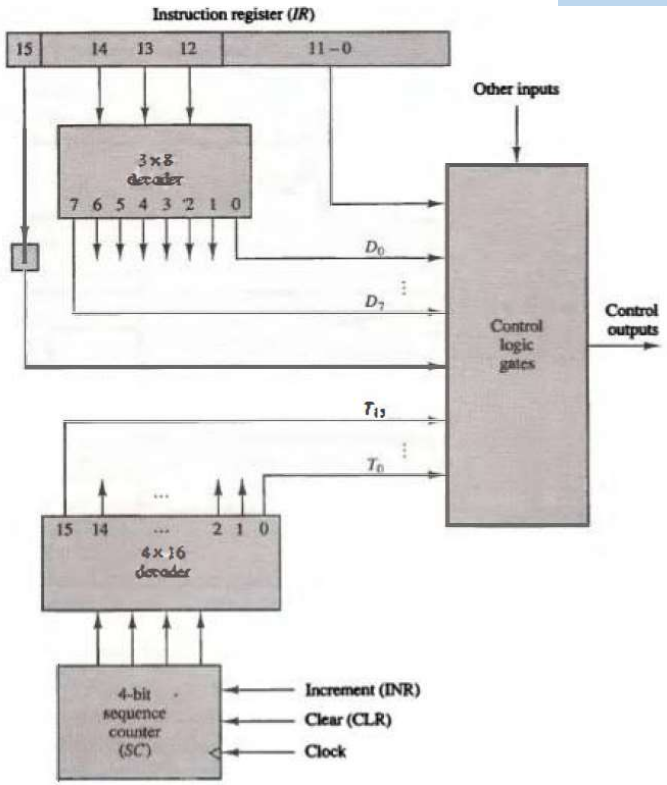
شناسایی نوع دستورالعمل

- دستورالعمل‌ها بطور کلی چهار گونه می‌توانند باشند:
  ۱. ورودی-خروجی (D7 و **I** هر دو 1 هستند)
  ۲. رجیستری (D7=1 ولی **I**=0 است)
  ۳. حافظه مستقیم (D7=0 است اما D0 تا D6 یکی‌شان 1 است. همچنین **I**=0 است)
  ۴. حافظه غیرمستقیم (همانند قبلی فقط **I**=1 است)
- از آنجا که در دستورات ورودی-خروجی و یا رجیستری فقط یکی از بیت‌های IR0 تا IR11 در دستورالعمل فعال است نیازی به دیکدر ندارند.

Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description	
AND	0xxx	8xxx	AND memory word to *AC*	AND کردن کلمه حافظه با AC
ADD	1xxx	9xxx	Add memory word to *AC*	جمع کردن کلمه حافظه با AC
LDA	2xxx	Axxx	Load memory word to *AC*	بار کردن کلمه حافظه در AC
STA	3xxx	Bxxx	Store content of *AC* in memory	ذخیره محتوای AC در حافظه
BUN	4xxx	Cxxx	Branch unconditionally	انشعاب نامشروط
BSA	5xxx	Dxxx	Branch and save return address	انشعاب و ضبط آدرس بازگشت
ISZ	6xxx	Exxx	Increment and skip if zero	افزایش و گذر در صورت نتیجه صفر
CLA	7800		Clear *AC*	پاک کردن AC
CLE	7400		Clear *E*	پاک کردن E
CMA	7200		Complement *AC*	متمم کردن AC
CME	7100		Complement *E*	متمم کردن E
CIR	7080		Circulate right *AC* and *E*	چرخش AC و E به راست
CIL	7040		Circulate left *AC* and *E*	چرخش AC و E به چپ
INC	7020		Increment *AC*	افزایش AC
SPA	7010		Skip next instruction if *AC* positive	گذر از دستور بعدی اگر AC مثبت باشد
SNA	7008		Skip next instruction if *AC* negative	گذر از دستور بعدی اگر AC منفی باشد
SZA	7004		Skip next instruction if *AC* zero	گذر از دستور بعدی اگر AC صفر باشد
SZE	7002		Skip next instruction if *E* is 0	گذر از دستور بعدی اگر E صفر باشد
HLT	7001		Halt computer	توقف کامپیوتر
INP	F800		Input character to *AC*	دریافت کاراکتر و انتقال آن به AC
OUT	F400		Output character from *AC*	برداشتن کاراکتر از AC و انتقال آن به خر
SKI	F200		Skip on input flag	گذر مبتنی بر پرچم ورودی
SKO	F100		Skip on output flag	گذر مبتنی بر پرچم خروجی
ION	F080		Interrupt on	فعال‌کردن وقفه‌ها
IOF	F040		Interrupt off	غیرفعال کردن وقفه‌ها

فلوچارت سیکل دستورالعمل

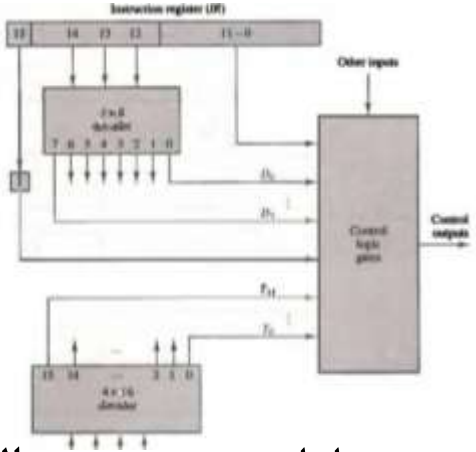




سیکل T3

- سیکل چهارم یعنی T3 نخستین فاز اجراییست که پس از فچ و دیکد آغاز می‌گردد.
- در دستورالعمل‌های رجیستری و یا ورودی خروجی، اجرا و تکمیل دستورالعمل در همین سیکل صورت می‌گیرد (در همین یک سیکل).
- در دستورالعمل‌های حافظه‌ای اگر حافظه غیر مستقیم باشد آدرس موثر در این سیکل بار می‌شود وگرنه (حافظه مستقیم) هیچ کاری در سیکل T3 انجام نمی‌شود.

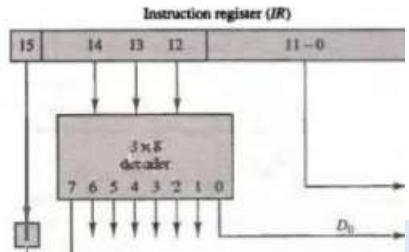
$D'_7 IT_3$: $AR \leftarrow M[AR]$
$D'_7 I'T_3$: Nothing
$D_7 I'T_3$: Execute a register-reference instruction
$D_7 IT_3$: Execute an input–output instruction



- در پایان هر دستورالعمل **باید شمارنده سیکل ریست** (SC←0) شود تا فچ دستورالعمل بعد شروع شود. افزایش SC بعد از هر سیکل چون بصورت خودکار و توسط شمارنده انجام می‌شود، در فلوچارت نمی‌آید.

دستورالعمل‌های با رجوع به رجیستر

- زمانی که D7=1 و **I** = 0 باشد یکی از بیت‌های 0 تا 11 موجود در IR برای تعیین یکی از دوازده دستورالعمل رجیستری (یعنی تنها ارجاع به رجیستر یا فلیپ‌فلاپ پرچم دارد) بکار می‌روند.
- همه این دستورالعمل‌ها در یک سیکل یعنی T3 انجام می‌گیرند. بنابراین همگی به یک خط کنترلی **D7I'T3** نیاز دارند. علاوه بر این تشخیص خود دستور به بیت کنترلی که ما آن را Bi می‌نامیم بستگی دارد. مثلا برای پاک کردن انباره (AC)، B11 (بیت یازدهم IR یا همان IR11) باید ست باشد و بقیه B ها طبیعتا خاموشند. پس 0000 0000 1000 0111 که معادل 7800 هگز است برای CLA خواهد بود.



لیست دستورالعمل‌های رجیستری

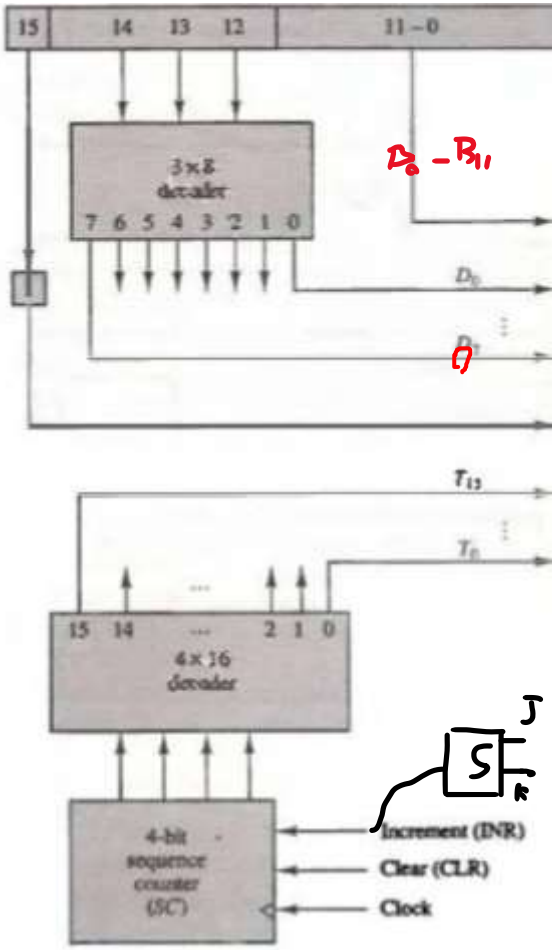
$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

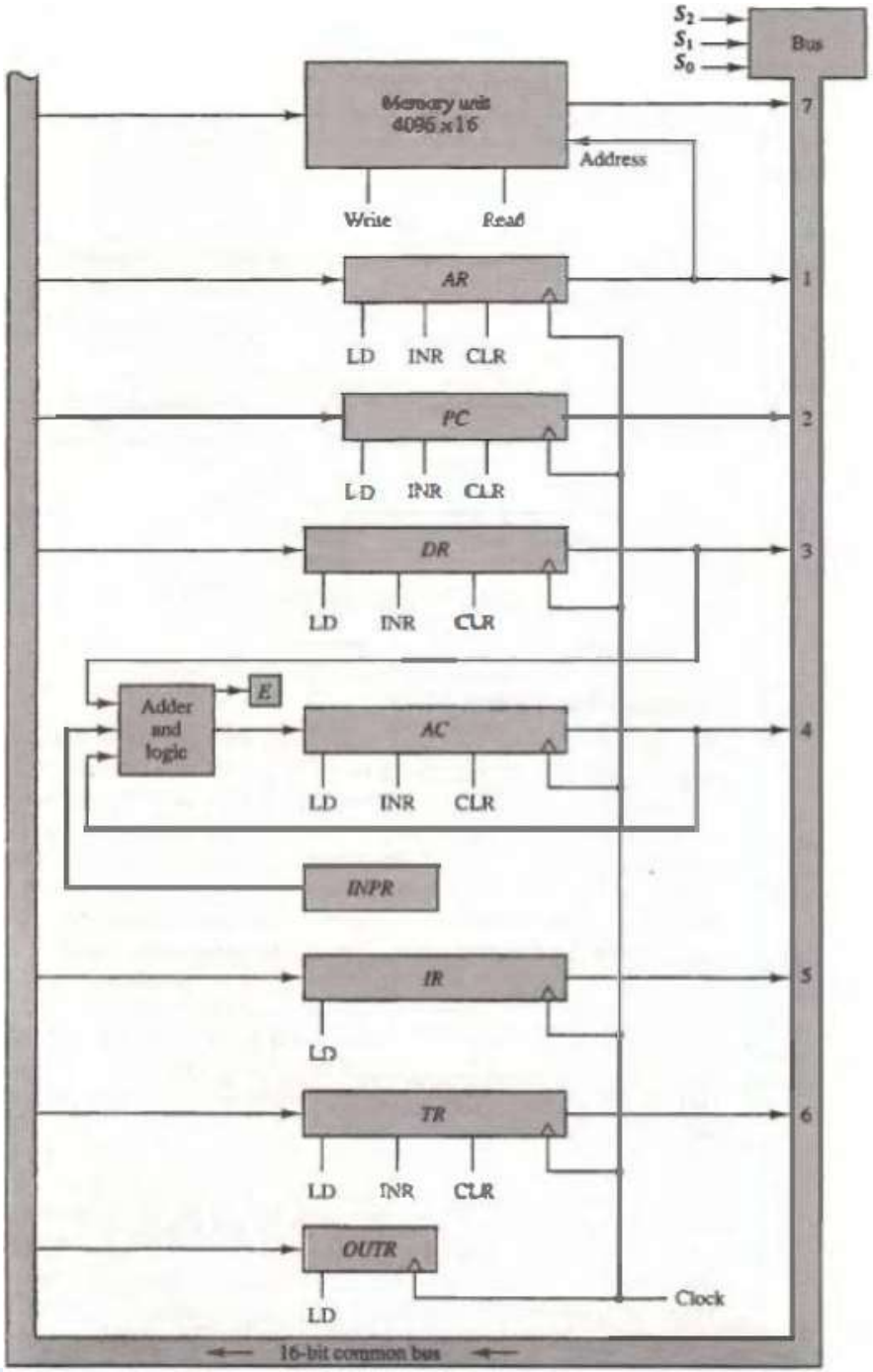
	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Clear پاک یا صفر کردن
Complement متمم کردن
Circulate شیفت چرخشی با دخالت E
Increment افزایش (++)
Skip پرش از دستور بعدی

فلیپ‌فلاپ‌های I و E و S

- تااینجا با سه فلیپ‌فلاپ (بیت‌های تکی که پرچم هم نامیده می‌شوند) سر و کار داریم:
- **I** که **IR15** به آن منتقل می‌شود و در تعیین نوع دستورالعمل بکار می‌آید.
- **E** که کری حاصل از عمل جمع در آن قرار می‌گیرد و در شیفت چرخشی هم واسطه بیت‌های سر و ته است.
- حال **S** که فقط در صورت **1** بودن آن است که شمارنده سیکل‌ها کار خواهد کرد. بنابراین وقتی دستور **HLT** آن را **0** می‌کند کار کامپیوتر متوقف می‌شود. برای ری‌استارت کردن برنامه می‌توان این فلیپ‌فلاپ را دستی (مثلا با یک دگمه فشاری) ست کرد.





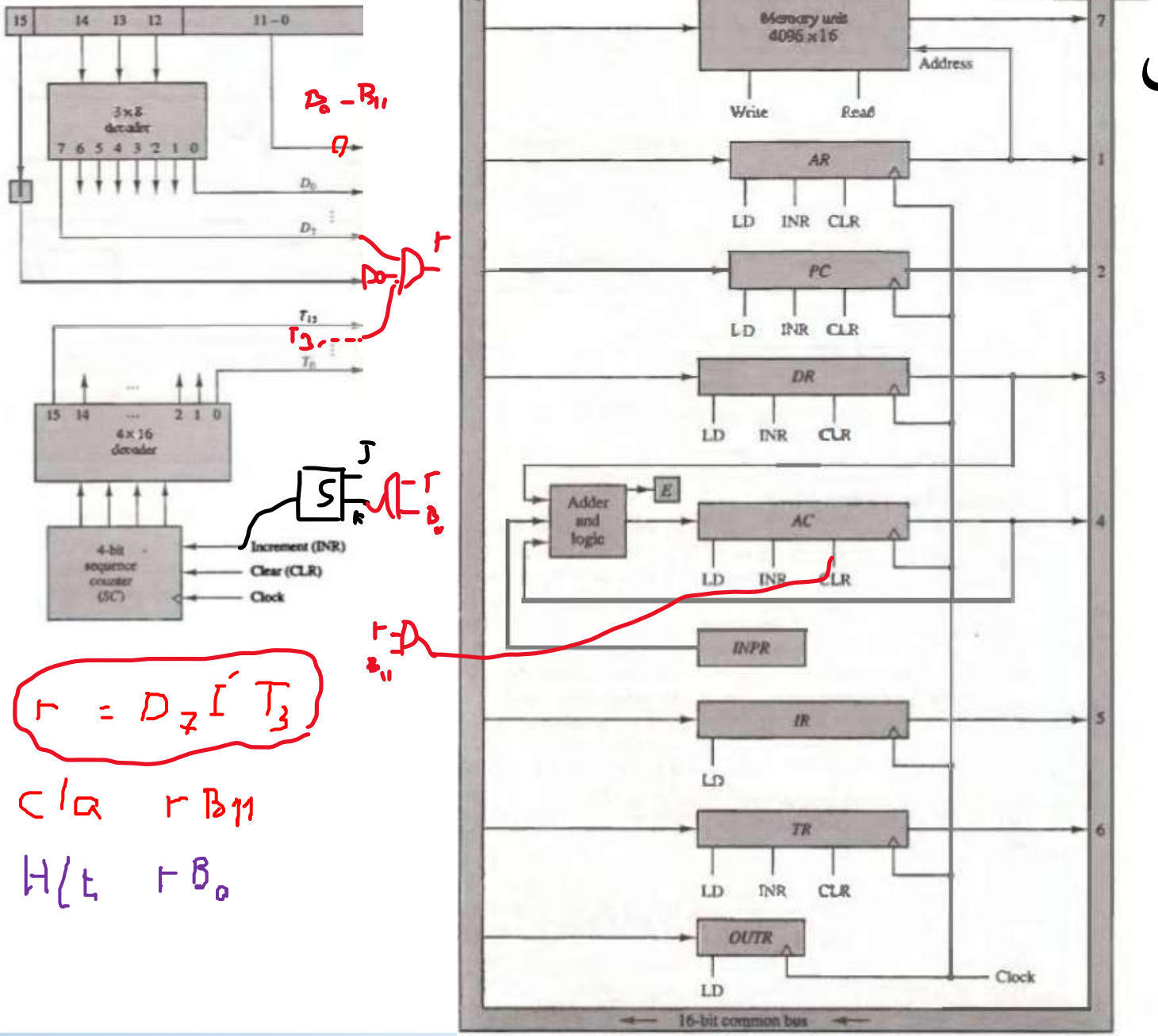
$r = D_7 I' T_3$

$CLR \quad r B_{11}$

$H/L \quad r B_0$

مثال

مثال



$r = D_7 I' T_3$

CLR $r B_{11}$

H/L $r B_0$

دستورالعمل‌های با رجوع به حافظه

- همانطور که قبلا گفته شد، دستورات حافظه‌ای از نظر نوع دسترسی به حافظه یا مستقیم هستند (یعنی آدرس عملوند حافظه داخل دستور است) یا غیر مستقیم (یعنی باید به آدرس داخل دستور برویم و از محتوای آن آدرس عملوند را پیدا کنیم. یعنی متغیری از نوع اشاره‌گر داریم).
- دستورات حافظه‌ای (چه مستقیم باشند چه غیر مستقیم) از نظر عملکرد دو نوعند. یا محاسباتی هستند. یعنی در واقع دو عملوند دارند که یکی خانه‌ای از حافظه و دیگری رجیستر AC (انباره) است. نتیجه محاسبه هم در انباره ذخیره می‌شود.
- یا این دستورات برای انشعاب هستند (در این صورت آدرس حافظه‌ای موجود در دستورالعمل محلی است که باید انشعاب به آن انجام شود). این دستورات به همراه پرش‌ها بعدا بررسی خواهند شد.
- در دستورات حافظه‌ای اجرای دستور از T4 شروع می‌شود و ممکن است بین یک تا سه سیکل طول بکشد (یعنی ممکن است در T4 یا T5 یا T6 اجرای دستور خاتمه یابد).

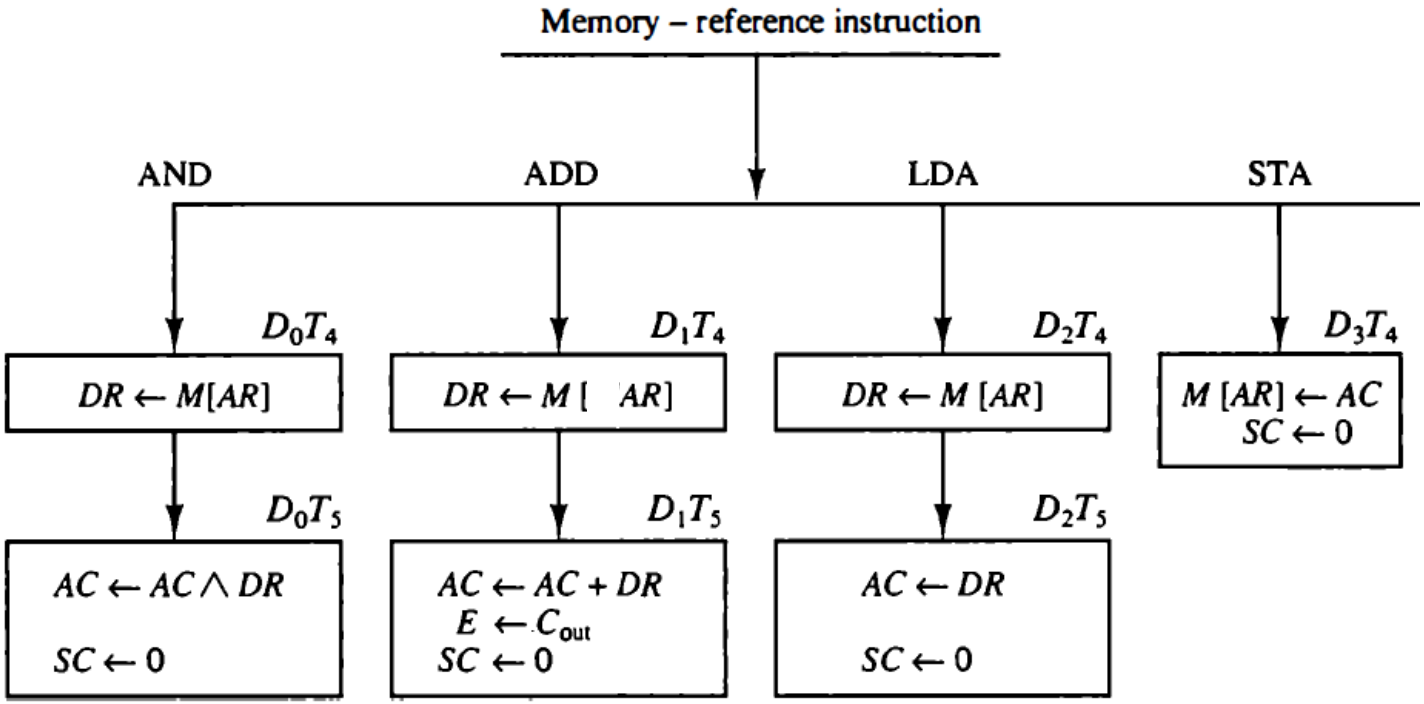
لیست دستورالعمل‌های محاسباتی حافظه‌ای

- همانطور که می‌دانیم دستورات محاسباتی کامپیوتر پایه مانو منحصرند به AND و ADD. همچنین دو دستور یکی برای بارگیری از حافظه (درون انباره) و ذخیره‌سازی در حافظه (از انباره) وجود دارد.

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$

- دیتایی که از حافظه خوانده می‌شود به رجیستر دیتا DR می‌رود (ورودی AC از باس نیست). بعد از آن در صورت لزوم می‌توان آن را به رجیسترهای دیگر (از جمله AC) کپی کرد.

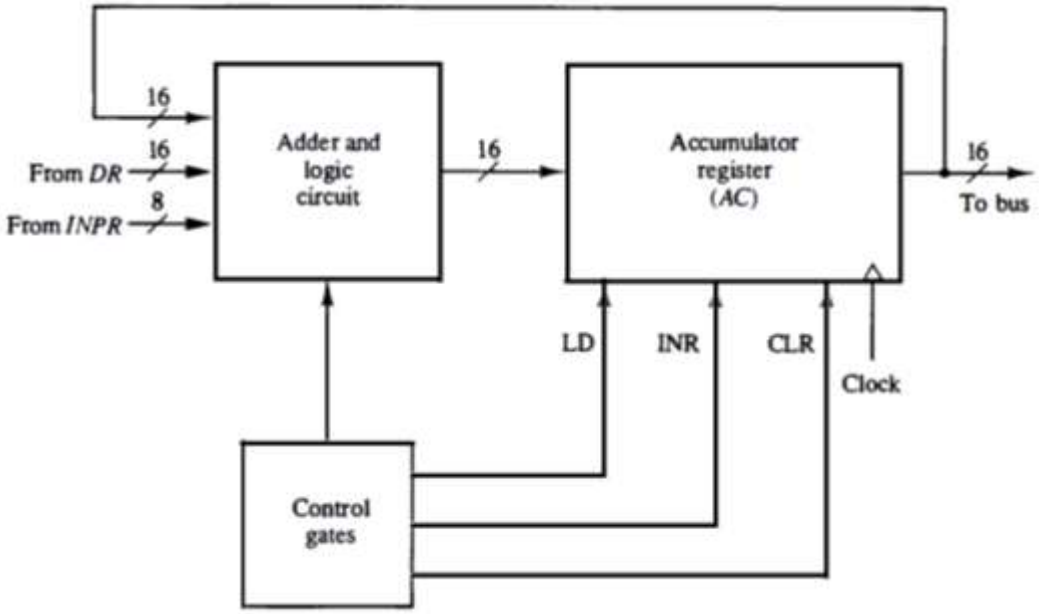
ریزعمل‌های دستورالعمل‌های حافظه



طراحی ALU و ورودی انباره

- کل ریزدستوراتی که ذخیره سازی در AC دارند عبارتند از:

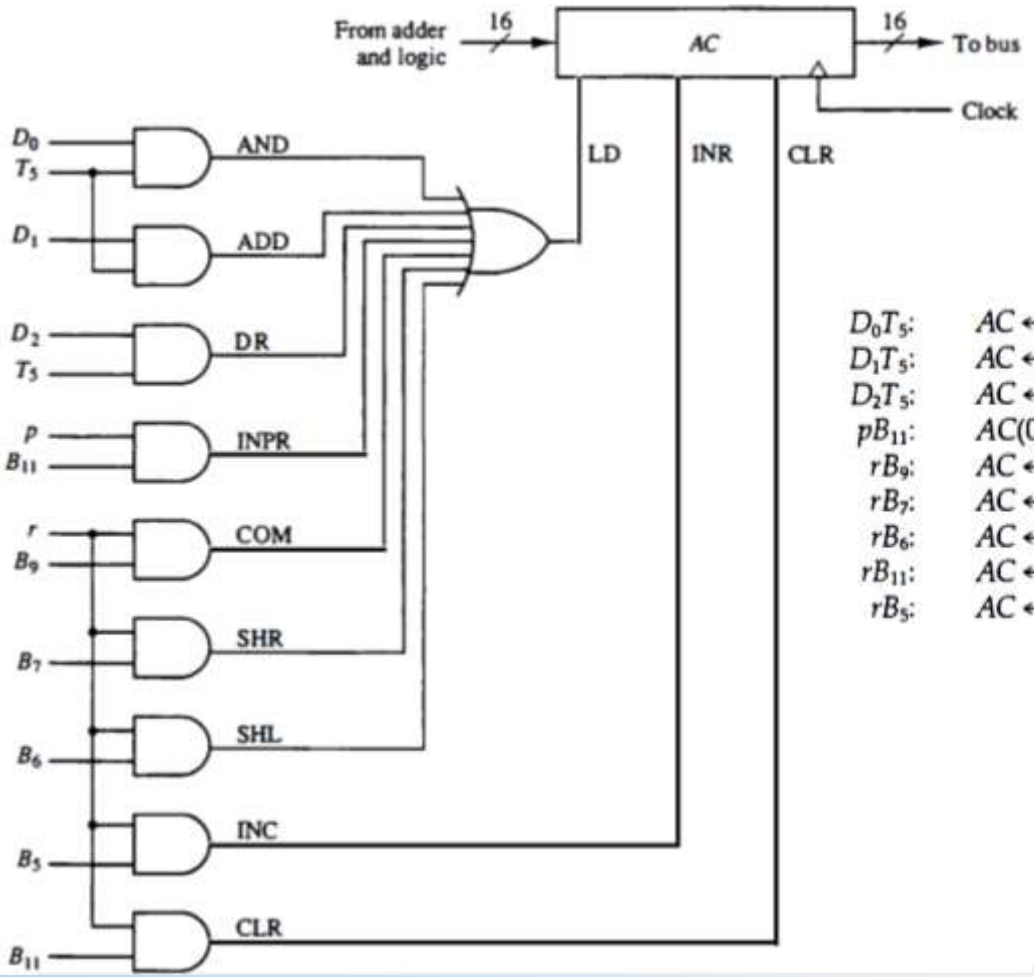
D_0T_5:	$AC \leftarrow AC \wedge DR$	AND with *DR*
D_1T_5:	$AC \leftarrow AC + DR$	Add with *DR*
D_2T_5:	$AC \leftarrow DR$	Transfer from *DR*
pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$	Transfer from *INPR*
rB_9:	$AC \leftarrow \overline{AC}$	Complement
rB_7:	$AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E$	Shift right
rB_6:	$AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E$	Shift left
rB_{11}:	$AC \leftarrow 0$	Clear
rB_5:	$AC \leftarrow AC + 1$	Increment



- بنابراین مدار AC می‌تواند طبق شکل روبرو باشد (خروجی به باس مشترک متصل است):

کنترل‌های CLR,INR,LD رجیستر AC

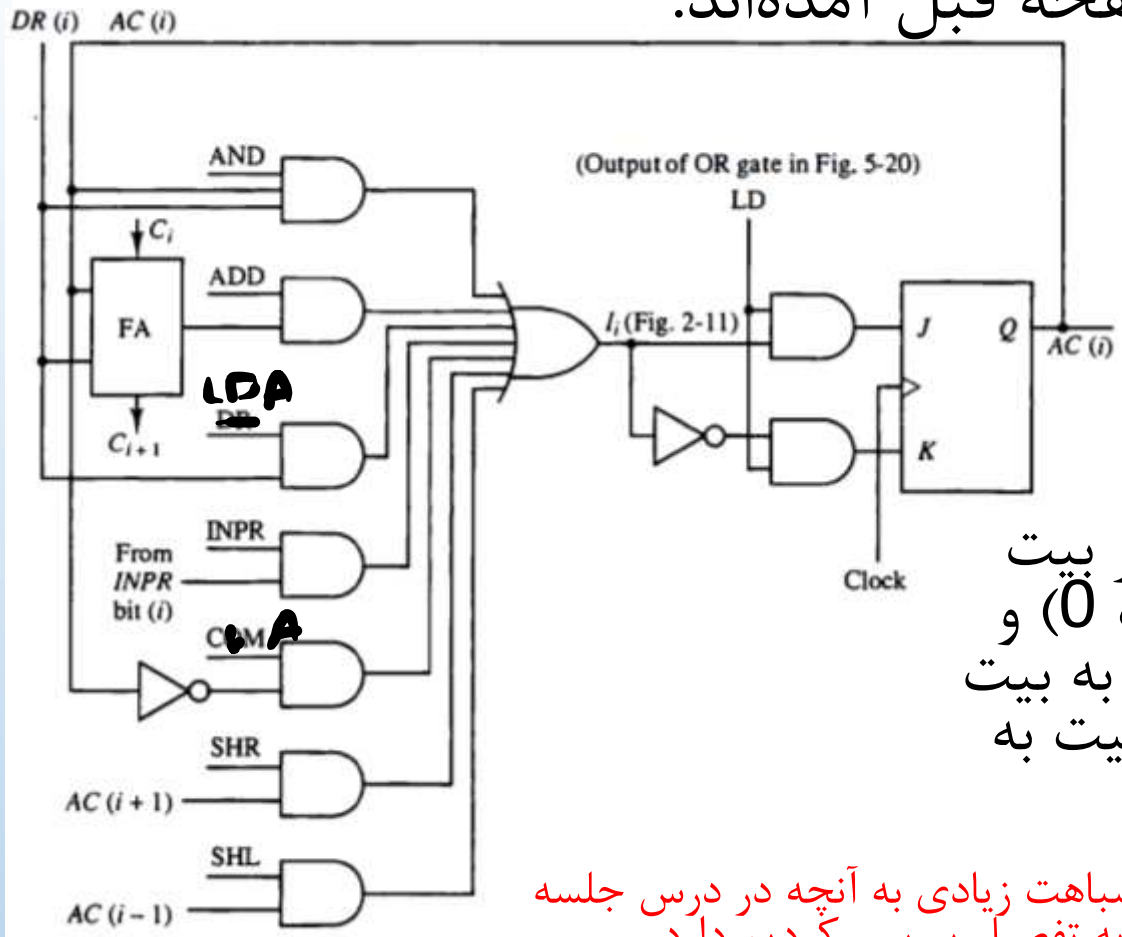
- LD خط بارگیری، INR خط افزایش (++) و CLR خط صفر کردن رجیستر AC است. از روی دستورالعمل‌ها می‌توانیم ورودی این سه خط را بسازیم.



D_0T_5:	$AC \leftarrow AC \wedge DR$	AND with *DR*
D_1T_5:	$AC \leftarrow AC + DR$	Add with *DR*
D_2T_5:	$AC \leftarrow DR$	Transfer from *DR*
pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$	Transfer from *INPR*
rB_9:	$AC \leftarrow \overline{AC}$	Complement
rB_7:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E$	Shift right
rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E$	Shift left
rB_{11}:	$AC \leftarrow 0$	Clear
rB_5:	$AC \leftarrow AC + 1$	Increment

مدار محاسبه و منطق و AC

- شکل زیر یک بخش (برای یک بیت) از شانزده بخش ALU و AC است. ورودی‌های AND,ADD, SHL, LD... و غیره از مدار صفحه قبل آمده‌اند.



- Ciکری‌ای است که از بیت قبل آمده (اولین بیت 0) و Ci+1 کری است که به بیت بعد می‌رود (آخرین بیت به فلیپ‌فلاپ E)

- طرح این ALU شباهت زیادی به آنچه در درس جلسه چهارم طراحی و به تفصیل بررسی کردیم دارد.

تمرین

- قصد داریم کامپیوتری طراحی کنیم با چهار رجیستر به نام‌های R1,R2,R3,R4. این رجیسترها با باس مشترک به هم متصل شده‌اند و فرستنده روی باس توسط مالتی‌پلکسری (که اندازه‌اش را شما تعیین می‌کنید) تعیین می‌شود. یک شمارنده سیکل و دیکدر مربوطه هم وجود دارد که حداکثر ۸ سیکل را می‌تواند بشمارد (از T0 تا T7). در سیکل T4 باید ریزدستورالعمل‌های زیر (بطور همزمان) اجرا شود:
  R3← R2 ; R1++ ; SC ← 0
  مدار کامپیوتر را (شامل رجیسترها، باس، شمارنده سیکل، خطوط کنترلی سیکل T4) طراحی کنید.

معماری کامپیوتر

جلسه نهم

سازماندهی کامپیوتر مانو:
پرش و انشعاب،
ورودی خروجی وقفه

پرش

- پرش (Skip) به معنای پریدن از روی دستور بعدی (اجرا نکردن یک دستور) است. این کار با یک واحد اضافه کردن PC انجام می‌شود (PC که محتوی آدرس دستور بعدی است اگر PC++ شود محتوی دستور دو خط بعد خواهد بود).
- چون دستورات هم طول (۱۶ بیتی) هستند، این کار ممکن است.
- پرش‌ها شرطی و رجیستری هستند. یعنی چیزی در رجیسترها چک می‌شود و در صورت برقراری شرط پرش صورت می‌گیرد (به حافظه کاری ندارد).
- پیاده‌سازی پرش‌های شرطی آسان است. عوامل دستورالعمل (مثلا r و B3) مانند جلسه قبل وارد گیت and می‌شوند و عامل شرط (مثلا خط 15 از AC) در گیت and به آنها می‌پیوندد.

SPA	rB_4:	If ($AC(15) = 0$) then ($PC \leftarrow PC + 1$)	Skip if positive
SNA	rB_3:	If ($AC(15) = 1$) then ($PC \leftarrow PC + 1$)	Skip if negative
SZA	rB_2:	If ($AC = 0$) then $PC \leftarrow PC + 1$)	Skip if AC zero
SZE	rB_1:	If ($E = 0$) then ($PC \leftarrow PC + 1$)	Skip if E zero

انشعاب

- دستورات انشعاب (نامشروط) Branch Unconditional محل اجرا (PC) را به یک خانه خاص از حافظه می‌برند. چون در این دستورات آدرس داخل دستورالعمل وجود دارد و طبعا چنین دستوری از نوع حافظه‌ای است.
- سه دستور از این نوع وجود دارد که ساده‌ترین آن انشعاب ساده یا BUN است. در این دستور مستقیما انشعاب به خانه‌ای از حافظه صورت می‌گیرد.

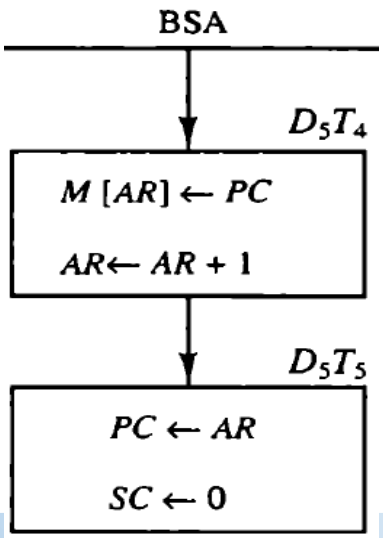
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

زیرروال‌ها

- می‌دانیم که کامپیوترهای پیشرفته برای ذخیره آدرس برگشت (و همچنین پاس دادن پارامترها و متغیرهای محلی) از پشته استفاده می‌کنند. همینطور می‌دانیم کامپیوتر پایه مانو به دلیل ساده بودن مجموعه دستورالعمل‌ها پشته ندارد.
- از سوی دیگر زیر روال‌ها (پروسیجرها) یکی از مهمترین پایه‌های برنامه‌نویسی هستند و کامپیوتری داشت که بصورت سخت‌افزاری از پروسیجر پشتیبانی نکند، به درد نمی‌خورد.
- کامپیوتر پایه مانو راهی ساده‌تر برای اجرای زیرروال‌ها دارد. برای پاس دادن مقدار یا بازگشت نتیجه از تنها رجیستر موجود یعنی AC می‌توان استفاده کرد و اگر کافی نباشد از خانه‌های حافظه (که در واقع متغیرهای سراسری هستند) استفاده می‌شود. عدم وجود پشته به معنای عدم امکان آزاد سازی حافظه و نداشتن متغیرهای محلی است.

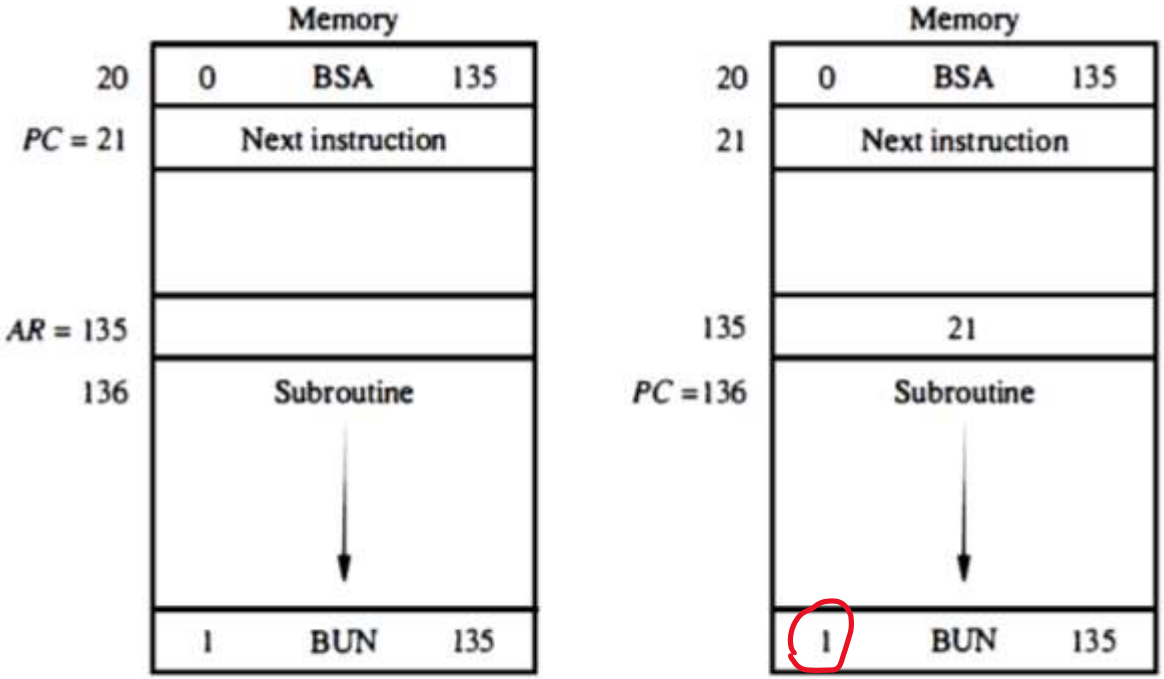
زیرروال‌ها

- اما برای اصلی‌ترین مساله یعنی ذخیره آدرس برگشت از زیرروال‌ها چه راه حلی وجود دارد؟
- در کامپیوتر پایه مانو در محل تعریف پروسیجر (زیرروال) نخستین خانه حافظه به ذخیره آدرس برگشت اختصاص یافته و کد پروسیجر بعد از آن شروع می‌شود.
- از آنجا که حافظه‌ای پویا به ذخیره آدرس برگشت و متغیرهای محلی اختصاص نیافته، تنها یک نسخه از پروسیجر در هر لحظه از زمان امکان اجرا دارد و خودفراخوانی در کامپیوتر مانو ممکن نیست.
- دستور BSA (انشعاب و ذخیره آدرس برگشت) ابتدا PC را در خانه‌ای از حافظه که آدرسش در دستورالعمل هست ذخیره کرده و سپس به خانه بعد از آن انشعاب می‌کند.



مثالی از یک زیرروال

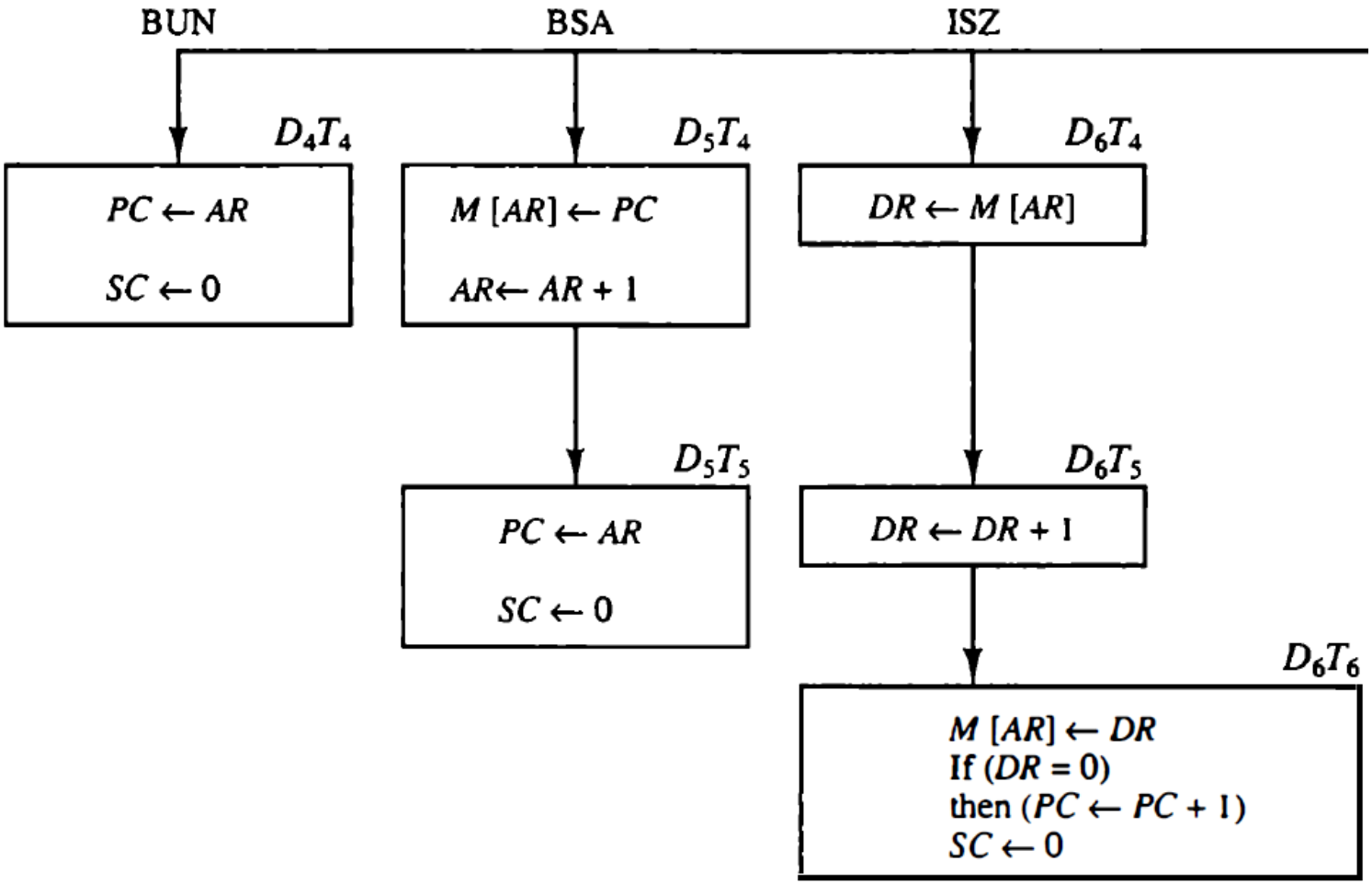
- همانطور که می‌بینید، دستور فراخوانی در خانه ۲۰ حافظه وجود دارد و خود زیرروال هم از خانه ۱۳۵ شروع می‌شود (شکل a). پس از رسیدن به BSA آدرس دستور بعدی یعنی ۲۱ در خانه ۱۳۵ ذخیره شده و اجرا از خانه ۱۳۶ ادامه می‌یابد (شکل b). در پایان پروسیجر (برای return) یک پرش غیر مستقیم به خانه ۱۳۵ (خانه اول پروسیجر) انجام می‌شود.



حلقه

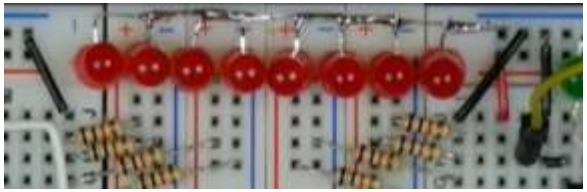
- ساختار دیگری که در برنامه‌نویسی بسیار مورد استفاده قرار می‌گیرد حلقه است. برای شمارنده حلقه نیاز به مکانی داریم. از آنجا که در کامپیوتر پایه تنها یک رجیستر برای داده هست که معمولا به کار دیگری گماشته شده است، نمی‌توان آن را برای این منظور استفاده کرد. بنابراین تنها چاره استفاده از خانه‌ای از حافظه بعنوان شمارنده حلقه (همان متغیر i) می‌باشد.
- همانطور که در اسلاید بعد (سیکل‌های دستورات پرش و انشعاب) می‌بینید، دستور ISZ (افزایش و سپس پرش در صورت صفر شدن) ابتدا به محتوای خانه‌ای از حافظه یکی اضافه می‌کند ++ و سپس اگر مقدار آن صفر شد از روی دستور بعد پرش می‌کند.
- در واقع متغیر حلقه عددی منفی است که آنقدر یکی‌یکی افزایش می‌یابد تا به صفر برسد. به محض اینکه این مقدار به صفر برسد از روی دستور بعد (که قاعدتا انشعاب به ابتدای حلقه است)، پرش صورت می‌گیرد.

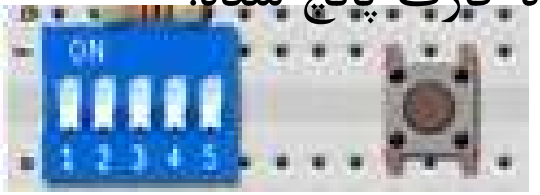
ریزعمل‌های دستورالعمل‌های انشعاب، فراخوانی، حلقه

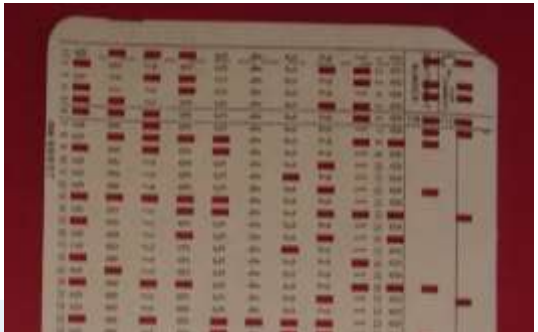


ورودی - خروجی IO

- کامپیوتر مانو تا اینجای کار می‌تواند انواع برنامه‌ها را اجرا کند، اما صرفا با دیتایی که در حافظه و رجیسترهای خود دارد کار می‌کند.
- در عمل کامپیوتری که نتواند با دنیای بیرون از خود ارتباط برقرار کند بلااستفاده است.
- کامپیوترها پیشرفته امروزی به دستگاه‌های ورودی و خروجی پیچیده‌ای مانند مانیتور و پرینتر و ماوس و کیبورد متصل می‌شوند. اما کامپیوترهای ساده ورودی خروجی‌های ساده‌ای دارند.
- ساده‌ترین ورودی ۸ بیتی (تکی) یک دیپ سویچ هشت‌تایی است. ساده‌ترین خروجی (تکی) ۸ بیتی هم هشت عدد لامپ است. کامپیوترهای قدیمی ورودی خروجی‌های ساده (پشت‌سرهم) اما کارایی داشتند. مثل پرینتر تله‌تایپ و خواننده کارت پانچ شده.







مساله همزمانی کامپیوتر با IO

- همه واحدهای داخل کامپیوتر توسط پالس ساعت سراسری با هم هماهنگ و همزمان (سنکرون) می‌شوند. چنین همزمانی بین کامپیوتر و ورودی و خروجی ناممکن است (چون نه یک ساعت مشترک دارند و نه سرعت پردازش‌شان لزوما با هم یکی است).
- بدون هماهنگی امکان ارتباط موثر کامپیوتر با ورودی خروجی ناممکن می‌شود. مثلا کامپیوتر دیتایی را روی خروجی (مثلا یک کاراکتر برای چاپ در تله تایپ) قرار می‌دهد. حال از کجا باید بفهمد که دیتا از روی خطوط خروجی برداشته و چاپ شده و حالا باید دیتای بعدی ارسال شود.
- در مورد ورودی هم همین وضع برقرار است. فرد دیتایی را روی دیپ سویچ قرار می‌دهد. حال کامپیوتر از کجا بداند دیتای جدیدی آماده شده (و دیتای قدیمی که یکبار خوانده شده هنوز روی ورودی نیست). این شرایط در مورد دستگاه‌های ورودی دیگر مثل کارت پانچ خوان و کیبورد برقرار است.

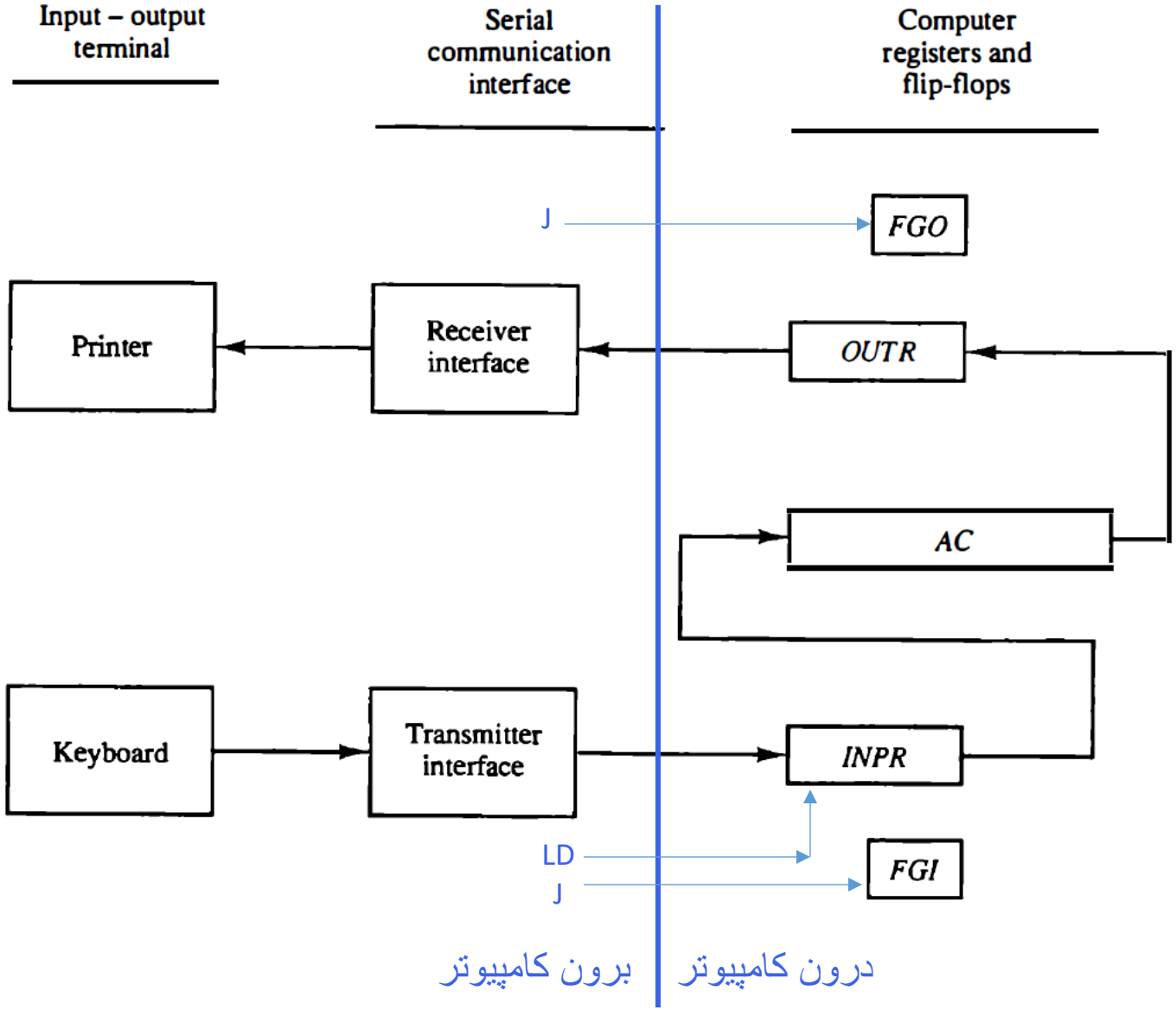
تحویل‌دار بانک و پیرمردها

- در شعبه بانک برخی اوقات پیرمردهایی که از نظر فعالیت فیزیکی کند هستند برای دریافت یا پرداخت پول به باجه مراجعه می‌کنند. پولی که قرار است مشتری دریافت یا پرداخت کند داخل باکس سیاه رنگ قرار می‌گیرد.
- مشکلی که ممکن است ایجاد شود این است که موقع پرداخت پیرمرد هنوز همه پول‌هایش را روی باجه قرار نداده و دارد کم‌کم پول اضافه می‌کند اما تحویل‌دار پول‌ها را بر می‌دارد. یا اینکه موقع دریافت پیرمرد دارد پول‌ها را کم‌کم داخل کیفش قرار می‌دهد، اما کارمند بانک (فرض کنید چون پایین نشسته باکس سیاه رنگ پول را نمی‌بیند) فکر می‌کند او همه پول‌ها را برداشته و سری بعدی پول را داخل باکس می‌دهد و پول‌ها قاطی می‌شود.
- راه حل: یک چراغ با سویچ روشن خاموش روی میز قرار داده شود. پیرمرد موقع پرداخت هر وقت همه پول‌ها را داخل باکس قرار داد چراغ را روشن می‌کند. تحویلدار هر وقت پول‌ها را برداشت چراغ را خاموش می‌کند. موقع دریافت هم وقتی پیرمرد تمام پول‌ها را برداشت چراغ را روشن می‌کند و کارمند وقتی پول سری بعد را داخل باکس گذاشت چراغ را خاموش می‌کند. در واقع همیشه پیرمرد با روشن کردن چراغ به کارمند علامت می‌دهد و کارمند با خاموش کردن آن.

پرچم‌های ورودی خروجی

- می‌دانیم کامپیوتر پایه دارای دو رجیستر برای ورودی و خروجی است که هر دو ۸ بیتی‌اند. رجیستر خروجی OUTR توسط کامپیوتر بارگذاری Load می‌شود و سپس واحدهای خارج از کامپیوتر می‌توانند مقدار را از روی آن بردارند.
رجیستر ورودی INPR که توسط فرد یا دستگاهی خارج از کامپیوتر بارگذاری می‌شود یعنی تنها رجیستری است که خط LD و ورودی داده آن داخل کامپیوتر نیست.
- علاوه بر این دو رجیستر دو فلیپ‌فلاپ (پرچم) هم وجود دارند: پرچم ورودی FGI و پرچم خروجی FGO. هر زمان که مقدار مربوطه داخل رجیستر ورودی گذاشته شد، این فلیپ‌فلاپ را هم روشن set خواهد کرد و هر وقت کامپیوتر مقدار را برداشت آن را reset می‌کند. پس خط ست (J) فلیپ‌فلاپ خارج کامپیوتر و خط ریست (K) داخل قرار دارد.
در هنگام عمل خروجی نیز هر زمان کامپیوتر مقدار را در رجیستر خروجی قرار داد FGO را ریست کرده و هر وقت دستگاه یا کاربر خارجی مقدار را برداشت آن را ست می‌نماید.

پیکربندی ورودی و خروجی



پیکربندی ورودی و خروجی

- در اسلاید صفحه قبل پیکربندی اتصال کامپیوتر پایه به دو دستگاه یکی ورودی (که صفحه کلیدی است که با یک مدار واسط کد کارکتر فشرده شده را در اختیار کامپیوتر می‌گذارد) و دیگری خروجی (یک پرینتر ساده تله‌تایپ که کد یک کاراکتر را از کامپیوتر دریافت و با یک ماشین شبیه ماشین تحریر مکانیکی روی کاغذ حک می‌کند) دیده می‌شود.
- دو فلیپ‌فلاپ FGI و FGO نیز پرچم‌های ورودی و خروجی هستند و همانند لامپ روی میز بانک پیرمردها عمل می‌کنند (اینجا لامپ پرداخت و دریافت جداست) و موجب می‌شوند که دستگاه‌های ورودی و خروجی (کیبورد و پرینتری که بسیار از کامپیوتر کندترند) هنگام ارسال و دریافت دیتای متوالی، با کامپیوتر هماهنگ شوند.

دستورات ورودی و خروجی

- این دستورات ساده‌اند. دستور INP که محتوای INPR را به AC کپی می‌کند و دستور OUT که AC را به OUTR کپی می‌کند. در هر دو مورد ۸ بیت کم ارزش AC مورد استفاده قرار می‌گیرند. هنگام اجرای دستور پرچم مربوطه نیز 0 می‌شود.
- دو دستور SKI و SKO پرچم‌های مربوطه را چک می‌کنند و اگر پرچم 1 بود از روی دستور بعدی پرش می‌نمایند. در واقع با این دو دستور می‌توان چک کرد که آیا ورودی رسیده و یا آیا خروجی برداشته شده یا خیر.

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag

مشکل سرکشی‌کردن Polling

- روشی که تاکنون برای ورودی و خروجی برشمردیم عملی است.
- یعنی هنگام ورودی سیستم آنقدر پرچم FGI را چک می‌کند و باز چک می‌کند و .... تا اینکه پرچم ست شود یعنی داده در ورودی آماده است.
- هنگام خروجی هم سیستم آنقدر پرچم FGO را چک می‌کند و باز چک می‌کند و .... تا اینکه پرچم ست شود یعنی داده از خروجی برداشته شده است.
- با وجود عملی بودن مشکلی وجود دارد؟؟!

مشکل سرکشی‌کردن Polling

- مشکل آن است که تمام (یا بخش عمده) وقت کامپیوتر صرف چک کردن حضور داده در ورودی و خروجی می‌شود در حالی که وقت پردازنده ارزشمند است و ممکن است کارهای دیگری هم برای انجام داشته باشد
- یک بازی کامپیوتری را در نظر بگیرید. گاهی گیمیر کلیدی را روی کیبورد فشار می‌دهد اما پردازنده باید همواره محاسباتی را در جریان بازی انجام دهد.

وقفه

- زیرروال‌ها Subroutines قطعه کدهایی هستند که در موقع لزوم توسط کد دیگری فراخوانی می‌شوند. در واقع صدازدن پروسیجرها امری از قبل برنامه‌ریزی شده و مشخص است.
- نوع دیگری از زیرروال‌ها وجود دارند. آنها همانند پروسجرهای معمولی در حافظه تعریف می‌شوند. اما صدازدن آنها از قبل در برنامه مشخص نشده است. وقتی کامپیوتر مشغول اجرای برنامه خودش است وقوع یک اتفاق باعث می‌شود پس از اتمام دستورالعمل فعلی، روال وقفه فراخوانی شود و طبیعتا پس از پایان این روال به ادامه برنامه بازگردد.
- این اتفاق وقفه سخت‌افزاری نام دارد و این روال نیز روال وقفه نامیده می‌شود.

وقفه سخت‌افزاری کامپیوتر پایه

- کامپیوتر پایه دارای تنها یک وقفه سخت‌افزاری است. این وقفه به هنگام ورودی و خروجی (یعنی ست شدن FGI یا FGO) فعال می‌شود (گویی کارمند بانک همواره مشغول حساب و کتاب خودش است تا زمانی که تابیدن شدن نور چراغی که پیرمرد روشن می‌کند در کارش وقفه بوجود آورد و به کار او روی آورد و پس از پایان کار پیرمرد دوباره به ادامه حساب و کتابش برگردد).
- فلیپ‌فلاپی وجود دارد بنام R (درخواست وقفه). ست شدن R (که بعلت یک شدن FGI یا FGO اتفاق می‌افتد) نشان دهنده این است که پس از پایان دستورالعمل جاری روال وقفه باید فراخوانی شود (جای آنکه دستورالعمل بعدی اجرا شود).
- فلیپ‌فلاپی وجود دارد بنام فعال‌کننده وقفه IEN. صفر بودن این فلیپ‌فلاپ نشان دهنده آن است که هرگز نباید روال وقفه اجرا شود. یعنی اگر هم ورودی یا خروجی‌ای در کار باشد R ست نشده و کار احتمالا به روش سرکشی انجام خواهد شد.

وقفه سخت‌افزاری کامپیوتر پایه

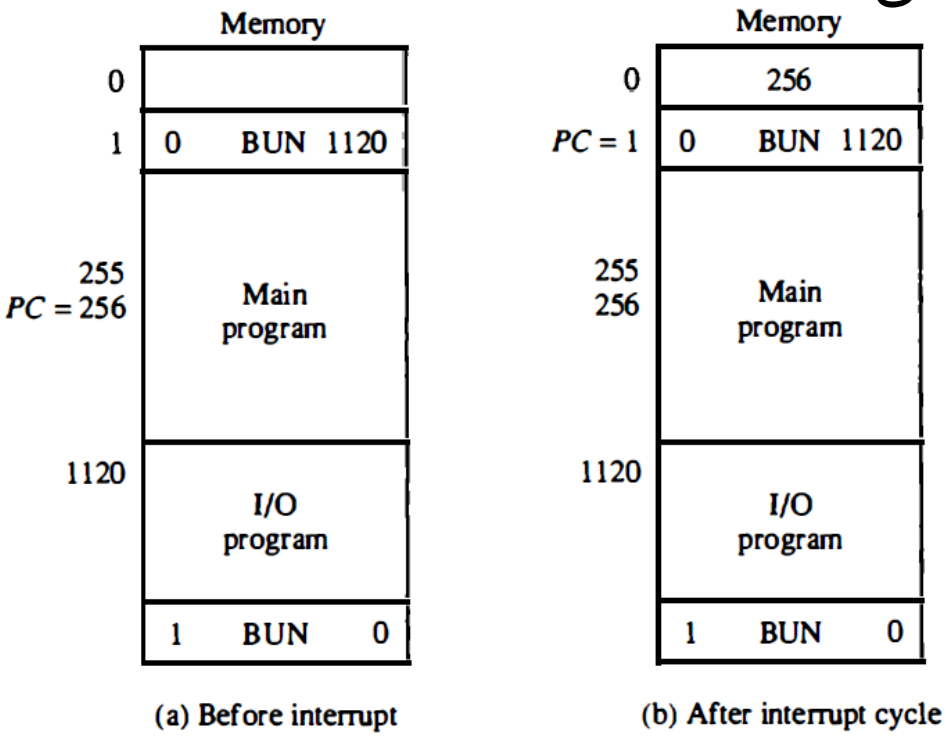
- دو دستور ورودی و خروجی برای ست و ریست کردن IEN وجود دارد:

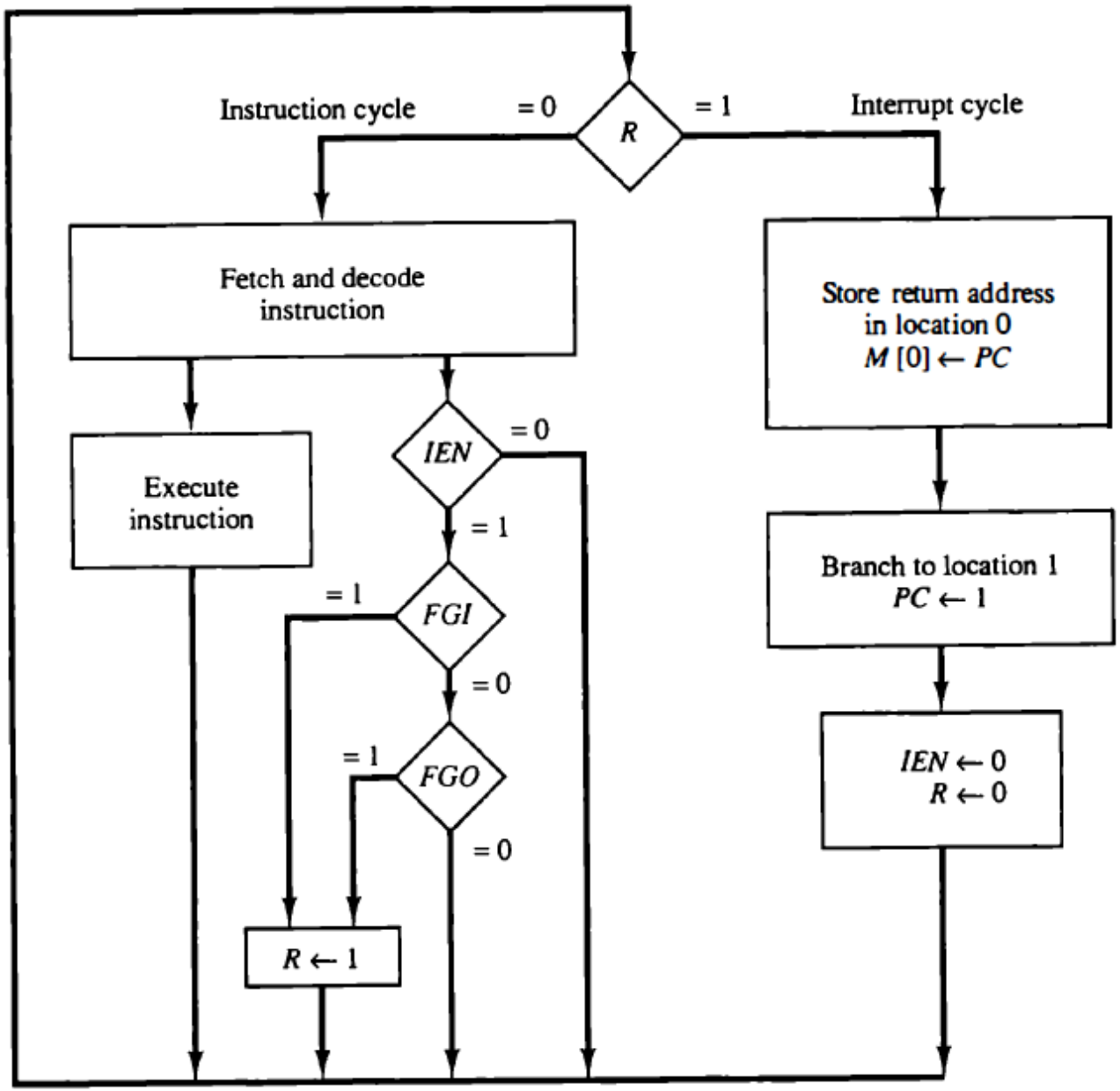
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

- آدرس شروع روال وقفه بطور پیش فرض خانه صفر حافظه است. یعنی پس از فراخوانی ابتدا آدرس برگشت در خانه 0 ذخیره شده و روال وقفه نیز از خانه 1 شروع می‌شود.
- وجود روال وقفه در خانه 0 مشکلی پیش نمی‌آورد. چرا که لزومی ندارد برنامه اصلی حتما از خانه صفر شروع شود. مدار restart کامپیوتر می‌تواند طوری طراحی شود که پس از روشن کردن (یا ریست کردن) کامپوتر، اجرا برنامه اصلی از محل خاصی (مثلا خانه 100H) آغاز شود و در آغاز این مقدار درونPC قرار داده شود. اگر روال وقفه در فضای ابتدای حافظه جا نشود، با انشعاب به ادامه آن می‌رویم (همانند مثال اسلاید بعد).

روال وقفه در حافظه

- در شکل زیر استقرار روال وقفه و وضعیت کامپیوتر قبل و بعد از فراخوانی آن را می بینید (کامپیوتر مشغول اجرای دستور خانه 255 حافظه بوده که وقفه حادث شده است).
- روال وقفه بصورت یک انشعاب BUN در ابتدای حافظه پیاده‌سازی شده و ادامه آن به بعد از برنامه اصلی (خانه 1120) موکول شده است.




Instruction cycle
= 0
R
= 1
Interrupt cycle
Fetch and decode instruction
Execute instruction
IEN
= 0
= 1
FGI
= 1
= 0
FGO
= 1
= 0
R ← 1
Store return address in location 0
M [0] ← PC
Branch to location 1
PC ← 1
IEN ← 0
R ← 0

سیکل وقفه

- ست شدن R در صورت وجود شرایط در هر زمانی ممکن است مگر در زمان فچ و دیکد یعنی T0,T1,T2.

$$T_0'T_1'T_2'(IEN)(FGI + FGO): \quad R \leftarrow 1$$

- اگر هنگام شروع دستور جدید R=1 باشد آنگاه بجای ورودی به سیکل فچ، وارد سیکل وقفه خواهیم شد.

$$\begin{aligned} RT_0&: \quad AR \leftarrow 0, \quad TR \leftarrow PC \\ RT_1&: \quad M[AR] \leftarrow TR, \quad PC \leftarrow 0 \\ RT_2&: \quad PC \leftarrow PC + 1, \quad IEN \leftarrow 0, \quad R \leftarrow 0, \quad SC \leftarrow 0 \end{aligned}$$

- در آغاز روال وقفه IEN صفر می‌شود تا زمان اجرای روتین وقفه، وقفه دیگری اتفاق نیافتد (تو در تو نشود).
- در اسلاید بعد مجموعه تمامی سیکل‌های دستورالعمل‌های کامپیوتر پایه را می‌بینید.

TABLE 5-6 Control Functions and Microoperations for the Basic Computer

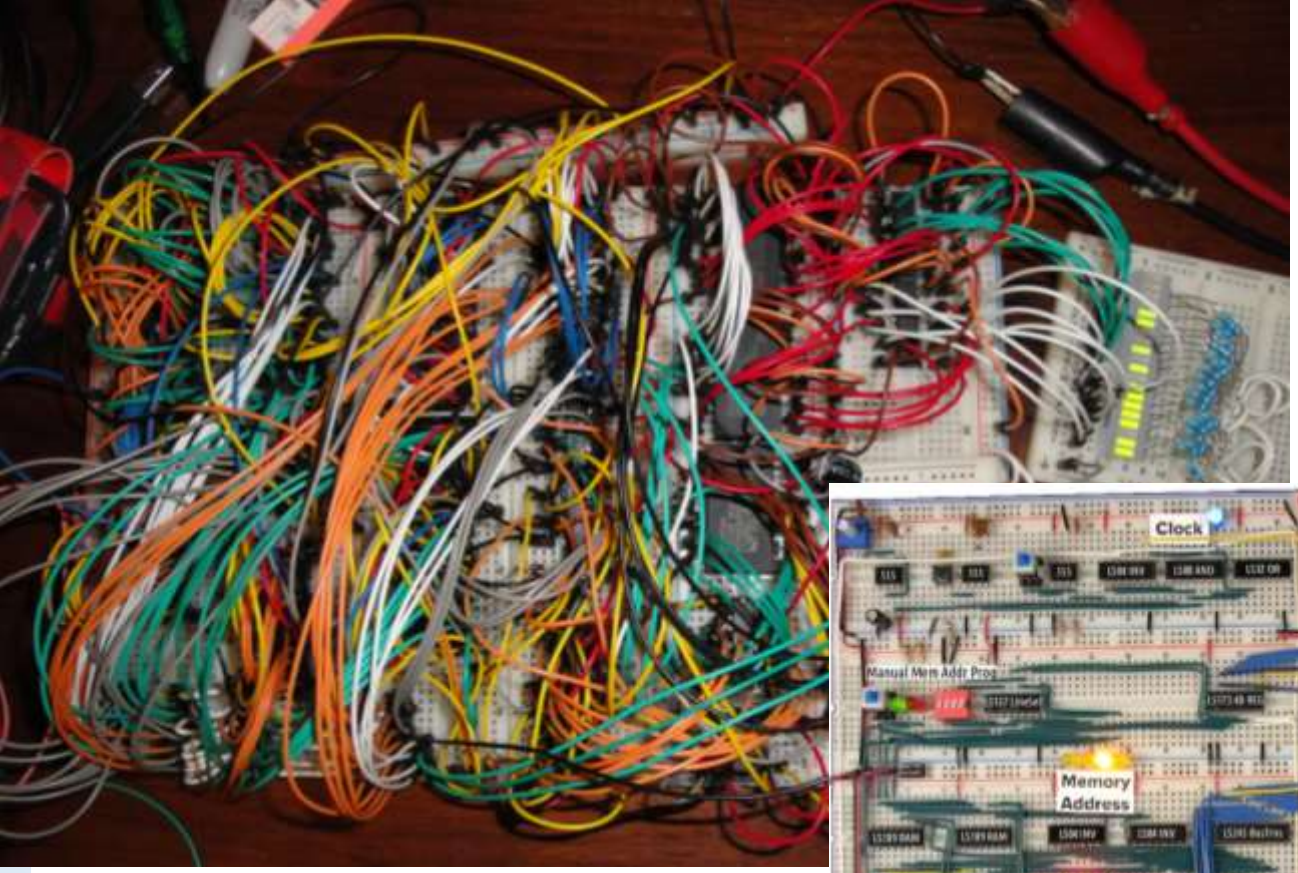
Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$
Indirect	D_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:		
	$T_0T_1T_2(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
Register-reference:		
		$D_7I'T_3 = r$ (common to all register-reference instructions)
		$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Input-output:		
		$D_7IT_3 = p$ (common to all input–output instructions)
		$IR(i) = B_i$ $(i = 6, 7, 8, 9, 10, 11)$
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

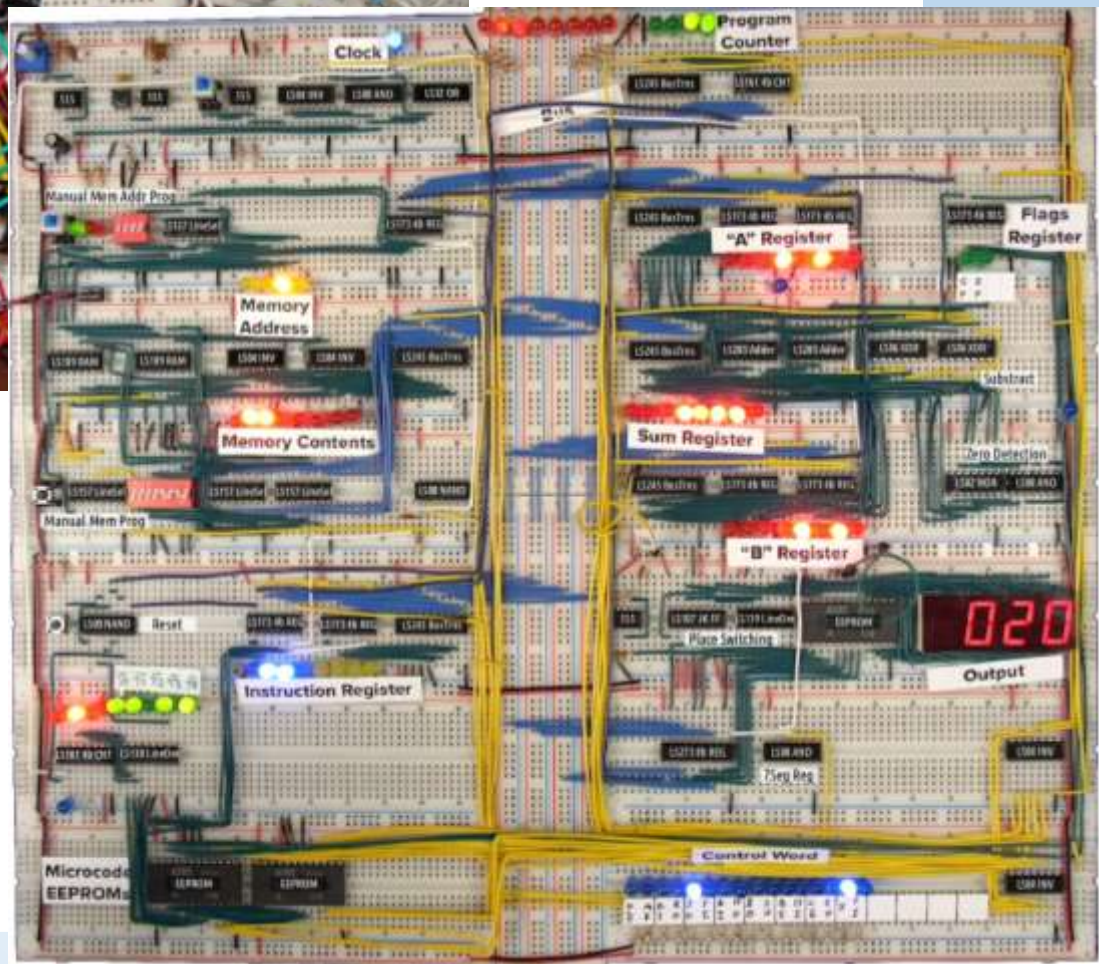
دستورات کامل شد

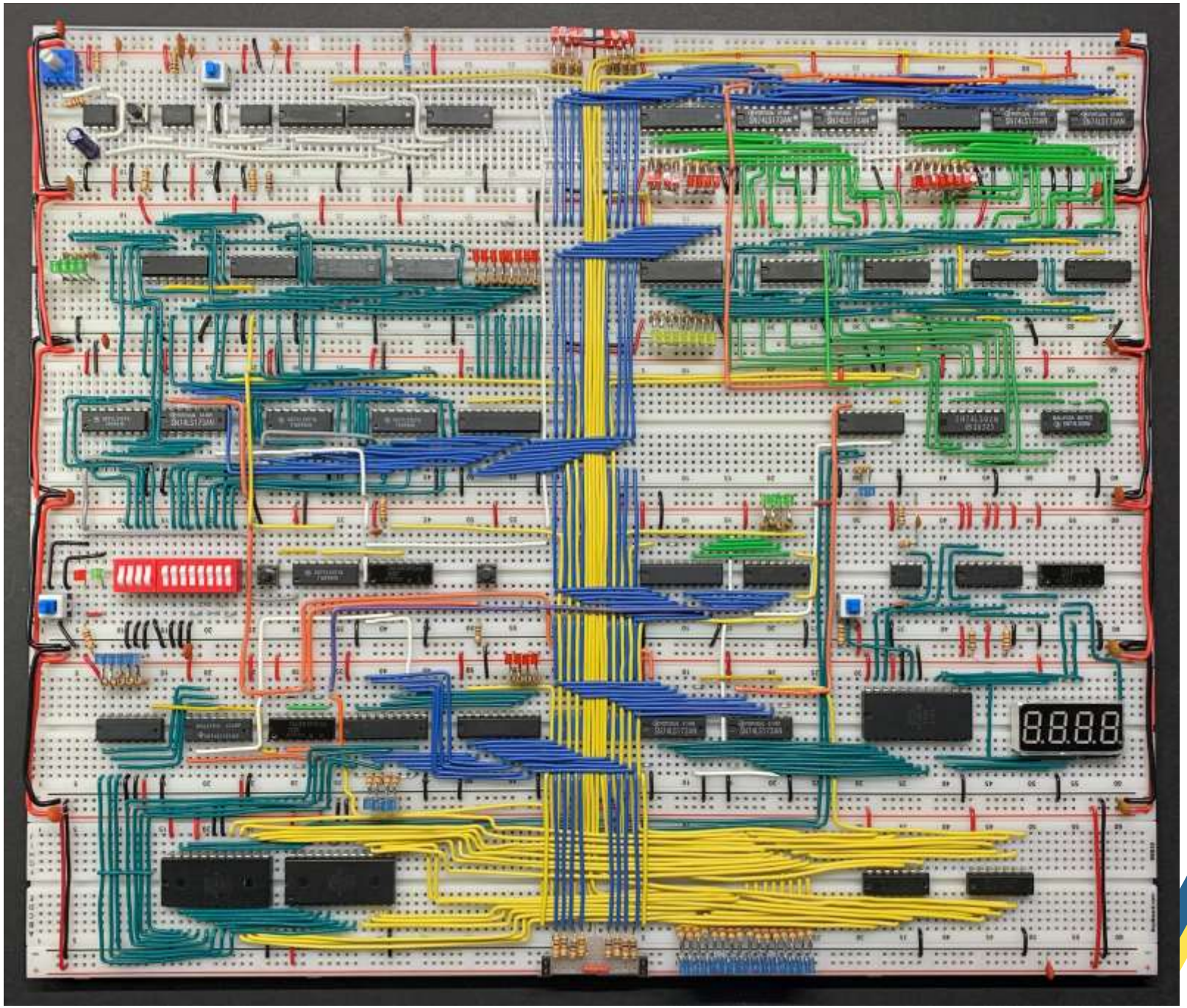
ساخت کامپیوتر پایه

- کامپیوتر پایه مانو هرگز بصورت تجاری تولید نشده است، اما هر سال صدها هزار دستگاه از آن توسط دانشجویان (در درس آزمایشگاه معماری کامپیوتر) ساخته و سپس باز می‌شود!
- برای ساخت این کامپیوتر به قطعات زیر نیاز داریم:

۱. یک واحد حافظه با ۴۰۹۶ کلمه ۱۶ بیتی.
۲. ۹ عدد رجیستر: SC, INPR, OUTR, TR, IR, AC, DR, PC, AR
۳. ۷ عدد فلیپ‌فلاپ FGO, FGI, IEN, R, E, S, I
۴. ۲ عدد دیکدر عملیاتی و ۱ دیکدر زمان‌گیر.
۵. یک باس مشترک که با مالتی‌پلکسر پیاده‌سازی می‌شود.
۶. واحد محاسبه و منطق متصل به AC
۷. تعدادی گیت منطقی.




Clock
Program Counter
Manual Mem Addr Prog
Memory Address
"A" Register
Flags Register
Memory Contents
Sum Register
Substract
Zero Detection
Manual Mem Prog
"B" Register
Reset
020
Place Switching
Output
Instruction Register
7Seg Reg
Control Word
EEPROMs



تمرین

- یک برنامه خروجی از آدرس ۲۱۰۰ نوشته شده است. این برنامه وقتی FGO=1 شود و تشخیص وقفه داده شود اجرا می‌گردد (IEN مساوی 1 است).
الف: چه دستوری باید در آدرس 1 حافظه قرار گیرد.
ب: دو دستور آخر برنامه خروجی چیست.
- مدار کنترل رجیستر PC را بدست آورید.
راهنمایی: برای این منظور به اسلاید قبلی بروید و همه دستوراتی که در آن PC تحت تاثیر قرار می‌گیرد را پیدا کنید. با استفاده از آنها برای خطوط ورودی LD و INR و CLR این رجیستر مدار طراحی کنید.

معماری کامپیوتر

جلسه دهم

برنامه‌نویسی پیشرفته
روی کامپیوتر مانو

برنامه‌نویسی کامپیوتر مانو

- همانطور که دیدیم، کامپیوتر پایه مانو کامپیوتری است با دستورات کم (۲۵ دستور) که با وجود ساده بودن، نوشتن (تقریبا) هر برنامه‌ای در آن ممکن است.
- از آنجا که کامپایلر مناسبی برای این کامپیوتر در دسترس نیست برنامه‌ها بایستی به زبان سطح پایین نوشته شوند: یعنی یا زبان ماشین یا زبان اسمبلی.
- از آنجا که ساختار دستورات ساده و ساختارهایی مثل حلقه و زیرروال و وقفه قبلا تشریح شده، ما در درس این جلسه فقط بر برنامه‌نویسی تمرکز می‌کنیم.
- برنامه زبان ماشین تشکیل شده از تعدادی 0 و 1 که هر دستورالعمل آن یک کلمه (۱۶ بیت) طول دارد. برنامه‌ها را برای سادگی و پرهیز از اشتباه بجای مبنای ۲ می‌توان در مبنای ۱۶ نوشت که در این صورت هر دستور ۴ رقم طول خواهد داشت.

مجموعه دستورالعمل‌ها

Symbol	Hexadecimal code	Description
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to $m + 1$
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC,
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

برنامه‌نویسی اسمبلی

- برای برنامه‌نویسی (چه به زبان اسمبلی چه به زبان ماشین) بایستی به مجموعه دستورالعمل‌های کامپیوتر (اسلاید قبل) بعنوان مرجع مراجعه کنیم.
- در زبان اسمبلی، یک خط دستور زبان اسمبلی جایگزین یک دستور کد ماشین می‌شود. در اسمبلی بجای کد شانزده‌شانزدهی (هگزادسیمال) دستورالعمل از **نماد** (سمبل) مربوطه استفاده می‌شود مثلا بجای 1xxx می‌نویسیم ADD. در اسمبلی می‌توان بجای نوشتن آدرس خانه‌های حافظه (برای متغیرها و محل‌هایی که انشعاب به آنها انجام می‌شود) از **نام** برای آنها استفاده کرد. مثلا بجای [025] نوشت a.
- برنامه اسمبلی را می‌توان به صورت دستی و روی کاغذ اسمبل کرد (تبدیل به زبان ماشین نمود) و یا از یک کامپیوتر دیگر استفاده کرد. مثلا اسمبلر کامپیوتر مانو بر روی کامپیوتر PC وجود دارد که این کار را انجام می‌دهد و دستورات زبان ماشین بصورت 0 و1 را تحویل ما می‌دهد.

برنامه‌نویسی اسمبلی

- شبه دستور org در اسمبلی تعیین می‌کند که دستورات یا متغیرهایی که در خطوط بعد از آن تعریف شده از کجای حافظه به بعد مستقر شوند.
- اسمبل کردن در یک یا دو گذر pass انجام می‌شود. در گذر اول دستورات نمادی با کد ماشین جایگزین می‌شوند. همینطور به تدریج جای خانه‌های حافظه که نامدار هستند مشخص می‌شوند و در جدولی توسط اسمبلر یادداشت می‌شود.
- در گذر اول اگر به نامی برخورد شود که آدرس آن در جدول موجود بود، آدرس در کد ماشین نوشته می‌شود. اما اگر هنوز معلوم نبود جای آن خالی گذاشته می‌شود. در گذر دوم که همه محل‌ها در جدول موجودند، جاهای خالی پر می‌شوند و کد ماشین تکمیل می‌گردد.

نوشتن برنامه اسمبلی تفریق دو عدد

- برنامه‌ای بنویسید که عدد ۲۳- را از عدد ۸۰ کم کند.

```
int main()
{
    short int min=83, sub=-23, dif=0;
    dif = min - sub; //106
}
```

Symbol	Hexadecimal code	Description
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to $m + 1$
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC,
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

نوشتن برنامه اسمبلی تفریق دو عدد

- برنامه‌ای بنویسید که عدد ۲۳- را از عدد ۸۰ کم کند.

```
int main()
{
    short int min=83, sub=-23, dif=0;
    dif = min - sub; //106
}
```

TABLE 6-8 Assembly Language Program to Subtract Two Numbers

```
        ORG 100      /Origin of program is location 100
        LDA SUB      /Load subtrahend to AC
        CMA          /Complement AC
        INC          /Increment AC
        ADD MIN      /Add minuend to AC
        STA DIF      /Store difference
        HLT          /Halt computer
MIN,    DEC 83       /Minuend
SUB,    DEC -23      /Subtrahend
DIF,    HEX 0        /Difference stored here
        END          /End of symbolic program
```

Symbol	Hexadecimal code	Description
AND	0 or 8	AND *M* to *AC*
ADD	1 or 9	Add *M* to *AC*, carry to *E*
LDA	2 or A	Load *AC* from *M*
STA	3 or B	Store *AC* in *M*
BUN	4 or C	Branch unconditionally to *m*
BSA	5 or D	Save return address in *m* and branch to *m* + 1
ISZ	6 or E	Increment *M* and skip if zero
CLA	7800	Clear *AC*
CLE	7400	Clear *E*
CMA	7200	Complement *AC*
CME	7100	Complement *E*
CIR	7080	Circulate right *E* and *AC*
CIL	7040	Circulate left *E* and *AC*
INC	7020	Increment *AC*,
SPA	7010	Skip if *AC* is positive
SNA	7008	Skip if *AC* is negative
SZA	7004	Skip if *AC* is zero
SZE	7002	Skip if *E* is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

ترجمه برنامه به باینری

- برنامه‌ای بنویسید که عدد ۲۳- را از عدد ۸۰ کم کند.

```
int main()
{
    short int min=83, sub=-23, dif=0;
    dif = min - sub; //106
}
```

```
        ORG 100
        LDA SUB
        CMA
        INC
        ADD MIN
        STA DIF
        HLT
MIN,    DEC 83
SUB,    DEC -23
DIF,    HEX 0
        END
```

Symbol	Hexadecimal code	Description
AND	0 or 8	AND *M* to *AC*
ADD	1 or 9	Add *M* to *AC*, carry to *E*
LDA	2 or A	Load *AC* from *M*
STA	3 or B	Store *AC* in *M*
BUN	4 or C	Branch unconditionally to *m*
BSA	5 or D	Save return address in *m* and branch to *m* + 1
ISZ	6 or E	Increment *M* and skip if zero
CLA	7800	Clear *AC*
CLE	7400	Clear *E*
CMA	7200	Complement *AC*
CME	7100	Complement *E*
CIR	7080	Circulate right *E* and *AC*
CIL	7040	Circulate left *E* and *AC*
INC	7020	Increment *AC*,
SPA	7010	Skip if *AC* is positive
SNA	7008	Skip if *AC* is negative
SZA	7004	Skip if *AC* is zero
SZE	7002	Skip if *E* is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

ترجمه برنامه به باینری

- برنامه‌ای بنویسید که عدد ۲۳- را از عدد ۸۰ کم کند.

```
int main()
{
    short int min=83, sub=-23, dif=0;
    dif = min - sub; //106
}
```

TABLE 6-9 Listing of Translated Program of Table 6-8

Hexadecimal code			
Location	Content		Symbolic program
			ORG 100
100	2107		LDA SUB
101	7200		CMA
102	7020		INC
103	1106		ADD MIN
104	3108		STA DIF
105	7001		HLT
106	0053	MIN,	DEC 83
107	FFE9	SUB,	DEC −23
108	0000	DIF,	HEX 0
			END

Symbol	Hexadecimal code	Description
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to $m + 1$
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC,
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

پیاده‌سازی عملیات حسابی

- از آنجا که کامپیوتر پایه تفریق سخت‌افزاری ندارد، بایستی از راه مکمل۲ (ابتدا مکمل کردن مفروق و سپس افزودن یکی به آن) انجام شود که در مثال قبل همینطور بود.
- از آنجا که در پرچم‌ها صرفه‌جویی شده پرچمی برای سرریز (علامتدار) نداریم اما در جمع بی علامت عملوندهای ۱۵ بیتی بیت شانزدهم و عملوندهای ۱۶ بیتی کری (فلیپ‌فلاپ E) نشانه سرریز است. در تفریق دو عدد مثبت از هم سرریز اتفاق نمی‌افتد اما اگر یکی منفی دیگری مثبت باشد ممکن است. برای جمع و تفریق اعداد مثبت و منفی از جدول قدرمطلقی زیر استفاده می‌شود.

Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

- انجام ضرب و تقسیم (با الگوریتمی مشابه انجام روی کاغذ) با استفاده از شیفت و جمع به راحتی ممکن است که در اینجا از تشریح آن صرفنظر می‌کنیم.

حلقه

- قصد داریم برنامه‌ای بنویسیم که صد عدد را که در یک آرایه (خانه‌های متوالی حافظه) قرار دارند را با هم جمع کند. مشابه این برنامه در C:

```
int main()
{
    short int A[100] = {  75,33,67,86,34,56,1,45,4,75,23,67,86,34,56,1,45,4,75,65,23,67,86,
    short int sum = 0, i;
    for (i = 0; i < 100; i++)
        sum = sum + A[i];
}
```

- اما با توجه به ویژگی‌های ماشین مانو (از جمله معکوس بودن شمارش حلقه تا رسیدن به صفر) قدری شیوه برنامه را تغییر می‌دهیم:

```
int main()
{
    short int A[100] = {  75,33,67,86,34,56,1,45,4,75,23,67,86,34,56,1,45,4,75,65,23,67,86,
    short int ptr, ctr,sum=0;

    ptr = 0;
    for (ctr = -100; ctr < 0; ctr++)
    {sum = sum + A[ptr];
    ptr++;}
}
```

```
ptr = 0;
for (ctr = -100; ctr < 0; ctr++)
{sum = sum + A[ptr];
ptr++;}
```

توضیح برنامه حلقه

```
Line
---------------------------------
   1            ORG 100
   2            LDA ADS
   3            STA PTR
   4            LDA NBR
   5            STA CTR
   6            CLA
   7    LOP,    ADD PTR I
   8            ISZ PTR
   9            ISZ CTR
  10            BUN LOP
  11            STA SUM
  12            HLT
  13    ADS,    HEX 150
  14    PTR,    HEX 0
  15    NBR,    DEC -100
  16    CTR,    HEX 0
  17    SUM,    HEX 0
  18            ORG 150
  19            DEC 75
   .
   .
   .
 118            DEC 23
 119            END
```

- سطر اول برنامه را از آدرس 100 شروع می‌کند. توجه داشته باشید که شماره سطر در اینجا کنار دستورات نوشته شده نه آدرس.
- سطر ۲ تا ۶ متغیرهای PTR و CTR و رجیستر AC را مقدار دهی اولیه می‌کند. چون در مقداردهی بلافصل نداریم باید از یک خانه دیگر حافظه برای مقدار اولیه استفاده شود.
- سطرهای ۷ تا ۱۰ حلقه اصلی ماست. سطر ۷ AC را با خانه PTRام (که در واقع خانه i آرایه است) جمع می‌کند. سطر ۸ فقط یکی به PTR می‌افزاید. سطر ۹ ضمن افزایش CTR شرط حلقه را هم کنترل می‌کند و اگر به صفر رسیده باشد با پرش از روی سطر ۱۰ عمله به گردش حلقه پایان می‌دهد.

حلقه برای جمع صد عدد

TABLE 6-13 Symbolic Program to Add 100 Numbers

Line			
1		ORG 100	/Origin of program is HEX 100
2		LDA ADS	/Load first address of operands
3		STA PTR	/Store in pointer
4		LDA NBR	/Load minus 100
5		STA CTR	/Store in counter
6		CLA	/Clear accumulator
7	LOP,	ADD PTR I	/Add an operand to *AC*
8		ISZ PTR	/Increment pointer
9		ISZ CTR	/Increment counter
10		BUN LOP	/Repeat loop again
11		STA SUM	/Store sum
12		HLT	/Halt
13	ADS,	HEX 150	/First address of operands
14	PTR,	HEX 0	/This location reserved for a pointer
15	NBR,	DEC −100	/Constant to initialized counter
16	CTR,	HEX 0	/This location reserved for a counter
17	SUM,	HEX 0	/Sum is stored here
18		ORG 150	/Origin of operands is HEX 150
19		DEC 75	/First operand
•			
•			
•			
118		DEC 23	/Last operand
119		END	/End of symbolic program

زیرروال

- می‌خواهیم که برنامه یک پارامتر ۱۶ بیتی (که در خانه‌ای از حافظه قرار دارد) را چهار بیت به چپ شیفت (منطقی-غیرگردشی) بدهد؛ یعنی از راست 0 وارد شود (به ظاهر مقدار ۱۶ برابر می‌شود).
- در این کامپیوتر چون دستور شیفت (غیر گردشی) نداریم، مقدار را شیفت حلقه‌ای می‌دهیم و سپس چهار بیت سمت راست حاصل را 0 می‌کنیم.
- برای کار از زیر روال استفاده می‌کنیم. مقدار را از حافظه خوانده، به روال ارسال کرده، سپس مقدار برگشتی را دوباره در حافظه ذخیره می‌کنیم. هم برای ارسال پارامتر و هم برای برگرداندن مقدار از انباره (رجیستر AC) استفاده می‌شود.
- برای آزمایش زیر روال، دو متغیر X با مقدار اولیه 1234 و Y با مقدار اولیه 4321 تعریف می‌کنیم. برنامه با استفاده از زیر روال مقدار این دو متغیر را آپدیت می‌کند.
- چنین زیرروالی برای پردازش دیتاهای ۴ بیتی مثل اعداد BCD مفید است.

زیرروال برای چهاربیت شیفت

TABLE 6-16 Program to Demonstrate the Use of Subroutines

Location			
		ORG 100	/Main program
100		LDA X	/Load X
101		BSA SH4	/Branch to subroutine
102		STA X	/Store shifted number
103		LDA Y	/Load Y
104		BSA SH4	/Branch to subroutine again
105		STA Y	/Store shifted number
106		HLT	
107	X,	HEX 1234	
108	Y,	HEX 4321	
			/Subroutine to shift left 4 times
109	SH4,	HEX 0	/Store return address here
10A		CIL	/Circulate left once
10B		CIL	
10C		CIL	
10D		CIL	/Circulate left fourth time
10E		AND MSK	/Set *AC*(13–16) to zero
10F		BUN SH4 I	/Return to main program
110	MSK,	HEX FFF0	/Mask operand
		END	

توجه: برخلاف مثال قبل ستون سمت چپ اینجا آدرس حافظه است نه شماره سطر.

تمرین نهم

```
        ORG 100
        CLE
        CLA
        STA CTR
        LDA WRD
        SZA
        BUN ROT
        BUN STP
ROT,    CIL
        SZE
        BUN AGN
        BUN ROT
AGN,    CLE
        ISZ CTR
        SZA
        BUN ROT
STP,    HLT
CTR,    HEX 0
WRD,    HEX 62C1
        END
```

- الف) توضیح دهید که برنامه روبرو چه کاری انجام می‌دهد. در پایان برنامه چه مقداری در محل CTR خواهد بود.
- ب) جدول آدرس‌های خانه‌های نامدار حافظه که در گذر اول بدست می‌آید را بنویسید.
- ج) کد شانزده شانزدهی برنامه به زبان ماشین را بنویسید.
- د) برنامه‌ای بنویسید که با استفاده از یک حلقه محتویات بین خانه‌های 500H و 5FFH را صفر کند.

معماری کامپیوتر

جلسه یازدهم

معماری کامپیوتر پیشرفته
افزایش کارایی
(حافظه کَش – خط لوله)

برنامه‌نویسی IO به روش سرکشی

- (a) می‌خواهیم برنامه‌ای بنویسیم که یک بایت را از ورودی (مثلا یک کاراکتر از کیبورد) دریافت کند، ورودی خوانده شده در خانه‌ای از حافظه بنام CHR ذخیره شود.
- (b) می‌خواهیم برنامه‌ای بنویسیم که دیتایی که در محلی از حافظه بنام CHR (در واقع ۸ بیت کم ارزش آن) واقع شده است را به خروجی (مثلا چاپگر) انتقال دهد. عدد 57 که مربوط به کاراکتر w است، در CHR باشد.
- برای انجام هر دو کار a و b ابتدا باید پرچم مربوطه چک شود. یعنی ورودی فقط در صورتی گرفته می‌شود که پرچم ورودی FGI روشن باشد. برای انجام عمل خروجی نیز FGO بایستی ست باشد.

Symbol	Hexadecimal code	Description
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to $m + 1$
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC,
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

برنامه‌نویسی IO به روش سرکشی

- برنامه a و b به این صورت است:

```
(a) Input a character:
CIF,    SKI       /Check input flag
        BUN CIF   /Flag=0, branch to check again
        INP       /Flag=1, input character
        STA CHR   /Store character
        HLT
CHR,    —         /Store character here
(b) Output one character:
        LDA CHR   /Load character into AC
COF,    SKO       /Check output flag
        BUN COF   /Flag=0, branch to check again
        OUT       /Flag=1, output character
        HLT
CHR,    HEX 0057  /Character is "W"
```

برنامه‌نویسی IO با وقفه

- همانطور که می‌دانید روال وقفه در هر زمانی در میانه اجرای برنامه ممکن است فراخوانی شود. پس از فراخوانی (مثل فراخوانی هر زیرروالی) آدرس برگشت در خانه اول آن ذخیره و اجرا از خانه دوم آغاز می‌شود.
- هنگام ورود به روتین وقفه کامپیوتر پایه حتما پرچم FGO یا FGI ست شده است. بنابراین کافیست در این روتین چک شود که کدامیک از دو پرچم ست است و بر آن اساس عمل شود.

نمونه‌ای از برنامه‌نویسی IO با وقفه

- در این مثال برنامه اصلی از خانه 100 شروع شده است. روتین اصلی وقفه نیز از خانه 200 که SRV نامیده شده شروع می‌شود. پس از فراخوانی و رفتن به خانه 1 حافظه فورا انشعاب به این خانه انجام می‌شود.
- این برنامه از خانه ۱۰۰ شروع کرده و مشغول کارهای معمولی است که در خانه ۱۰۳ ناگهان وقفه فرا می‌رسد.
- در روتین اصلی وقفه برای اینکه مقدار موجود در AC خراب نشود اول کار آن را جایی ذخیره می‌کنیم و پس از انجام کارهای وقفه آن را سر جایش می‌گذاریم، وقفه را فعال کرده به ادامه برنامه اصلی در خانه ۱۰۴ برمی‌گردیم.

```
Location
-------------------------------------------------------------------
    0      ZRO,    —          /Return address stored here
    1              BUN SRV    /Branch to service routine
  100              CLA        /Portion of running program
  101              ION        /Turn on interrupt facility
  102              LDA X
  103              ADD Y      /Interrupt occurs here
  104              STA Z      /Program returns here after interrupt
    .                .
    .                .
    .                         /Interrupt service routine
  200      SRV,    STA SAC    /Store content of AC
    .                .
    .                .
    .
                     LDA SAC    /Restore content of AC
                     ION        /Turn interrupt on
                     BUN ZRO I  /Return to running program
           SAC,      —          /AC is stored here
```

معماری کامپیوتر پیشرفته (افزایش کارایی)

- در این درس از ابتدا تاکنون ما کار انجام داده‌ایم: اول، "معماری مجموعه دستورالعمل‌ها"ی کامپیوتر PC با پردازنده 8086 را شناختیم. این کامپیوتر یک کامپیوتر مدرن و پردستور و با کارایی بسیار بالاست.
- دوم، معماری کامل ("مجموعه دستورالعمل‌ها" و هم "سازماندهی و ریزمعماری") کامپیوتر پایه مانو را شناختیم. این کامپیوتر با تعداد رجیستر و تعداد دستور حداقلی، ساده است.
- کامپیوتر مانو هرچند کامل است و تقریبا قدرت اجرا هر برنامه‌ای دارد، اما کارایی آن بسیار کم است (کامپیوترهای کم‌دستور مدرن باید هر دستورالعمل را در یک سیکل اجرا کنند نه چندین سیکل). از آنجا که هدف از طراحی آن کاربرد عملی (اجرا برنامه‌ها با سرعت بالا) نبوده است، هیچگونه تکنیک افزایش کارایی در طراحی سخت‌افزاری آن بکار نرفته است.

معماری کامپیوتر پیشرفته (افزایش کارایی)

- برای افزایش کارایی (افزایش سرعت اجرای برنامه‌ها) سه دسته بهبود در طراحی می‌توان ایجاد کرد:
  ۱. سازماندهی حافظه (حافظه کش)
  ۲. پردازش خط لوله‌ای
  ۳. بهینه‌سازی عملیات حسابی (چهار عمل اصلی + اعشاری)

  در اینجا ما با دو مورد اول آشنا می‌شویم. شناخت کامل‌تر روش‌ها و تکنیک‌های مربوط به بهینه‌سازی و افزایش کارایی سخت‌افزار در درس معماری کامپیوتر پیشرفته (کارشناسی ارشد) انجام می‌شود.

مشکل سرعت حافظه RAM

- در کامپیوترهای امروزی سرعت حافظه اصلی از سرعت پردازنده کمتر است. در نتیجه هنگام کار حافظه از CPU عقب می‌ماند (تعداد سیکل دستورات حافظه زیاد است).
- ساده‌ترین برخوردی که با این مشکل می‌توان داشت این است که پردازنده هنگام خواندن یا نوشتن در حافظه قدری معطل بماند (یعنی چند سیکل بیکاری داشته باشد) تا حافظه به او برسد.
- در این شرایط ساده‌ترین راهکار برای افزایش کارایی آن است که تعداد رجیسترها بیشتر شود تا بیشتر دستورات محاسباتی رجیستری باشد و مراجعه به حافظه کم باشد (دقیقا برعکس کامپیوتر پایه ما که یک سر محاسباتش همیشه حافظه است).

انواع حافظه

- برای پاسخ به این سوال که چرا حافظه کند است لازم است انواع حافظه RAM (خواندنی-نوشتنی) را بشناسیم.
- SRAM یا حافظه پایا (استاتیک) از فلیپ‌فلاپ‌های معمولی ساخته شده است و همان ساختار رجیسترها را دارد.
- DRAM یا حافظه پویا (داینامیک) از خازن‌های بسیار کوچک برای ذخیره داده استفاده می‌کند. این خازن‌ها پس از مدت کوتاهی خالی (discharge) می‌شوند. به همین خاطر باید در زمان‌های کوتاه خوانده و بازنویسی (refresh) شوند.

علت کندی حافظه

- حافظه SRAM با روشی شبیه خود CPU ساخته می‌شود و سرعت آن هم شبیه پردازنده است. اما مشکل آن است که هر ۱ بیت حافظه نیاز به ۲ فلیپ‌فلاپ و هر فلیپ‌فلاپ ۲ گیت و هر گیت نیاز به پنج ترانزیستور دارد. خلاصه هر بیت آن نیاز به دست‌کم بیست ترانزیستور دارد. چنین مداری برای حافظه‌های چند ده گیگابایتی (حتی با IC های امروزی) بسیار پرحجم و گران خواهد شود.
- حافظه DRAM کوچکتر و ارزانتر است. اما سرعت آن کم است. پردازنده برای اجرای دستورالعمل‌هایی که خواندن یا نوشتن در DRAM دارند بایستی سیکل‌های بیشتری صرف دستورالعمل‌های حافظه‌ای کند.

حافظه کَش (نهان)

- نوعی حافظه خواندنی نوشتنی است که حجم آن نسبتا کم و در درون CPU قرار دارد.
- سرعت دسترسی به این نوع حافظه بسیار زیاد است چرا که **اولا** از نوع SRAM است (چون حجمش کم است هزینه و فضای فیزیکی زیادی ندارد)، **ثانیا** فاصله فیزیکی آن با پردازنده کم است. توجه داشته باشید که در فرکانس‌های زیاد طول موج به چند سانتیمتر میرسد و چند سانتیمتر دوری و نزدیکی به پردازنده می‌تواند سرعت انتقال را کنْد کُنَد.

استفاده از کش

- به طور کلی فرض بر این است که وقتی خانه‌ای از حافظه مورد رجوع قرار گرفت، احتمال زیاد دارد که در آینده نزدیک باز هم به آن مراجعه شود (مثلا متغیری در برنامه) یا خانه‌های اطراف آن مورد مراجعه قرار گیرد (مثلا دستورات بعدی برنامه در حال اجرا).
- ایده کلی حافظه کش آن است که وقتی خانه‌ای از حافظه که مورد مراجعه قرار می‌گیرد (و اکنون در کش وجود ندارد)، آن خانه و چند خانه اطرافش به حافظه کش منتقل شود و تا قدری بعد از زمانی که مورد استفاده قرار گرفته است در آنجا باقی بماند.
- در کامپیوتر پایه نیازی به حافظه کش احساس نمی‌شود چون اولا حافظه اصلی بسیار کوچک (4KB) است و می‌تواند از نوع SRAM باشد، ثانیا کلاک پردازنده سرعت زیادی ندارد (کیلوهرتز) و حتی ممکن کلاکش پالس دستی باشد.

خط لوله

- در این کارخانه اتومبیل اگر مراحل مونتاژ (مثلا سوار کردن شاسی، اتصال قطعات بدنه، سوارکردن موتور، جاگذاری قطعات داخل اتاق و ...) پشت سر هم توسط یک گروه انجام شود (مثلا) طی شش روز می‌توانند یک اتومبیل را آماده کنند.
- اما اگر شش گروه مختلف مشغول بطور همزمان کارهای مراحل مختلف باشند و هر گروه پس از پایان کار خود، اتومبیل را تحویل گروه بعدی دهد هر روز یک ماشین جدید تولید می‌شود.

خط لوله در معماری کامپیوتر

- در کامپیوتر پایه مراحل مختلف اجرای دستور (فچ، دیکد، آوردن عملوند از حافظه، اجرا) برای هر دستور یک به یک انجام می‌شوند و در پایان کار این روند برای دستورالعمل بعد از نو شروع می‌شود. به هنگام انجام محاسبات نیز ALU به انجام تنها یک محاسبه می‌پردازد و پس از تکمیل آن سراغ محاسبه بعدی می‌رود.
- با استفاده از خط لوله کار مختلف در یک زمان واحد در حال اجرا خواهند بود. بدین صورت در زمان صرفه‌جویی می‌شود و کارایی افزایش می‌یابد.

خط لوله دستورالعمل

- در اجرای دستورالعمل کامپیوتری مانند کامپیوتر پایه مراحل زیر انجام می‌شود:
  FI فچ دستورالعمل
  DA دیکد کردن دستورالعمل و آوردن آدرس عملوند
  FO فچ عملوند (آوردن مقدار عملوند از حافظه)
  EX اجرا دستورالعمل
- می‌توان تمام این چهار مرحله را با استفاده از تکنیک خط لوله بطور همزمان به کارکردن واداشت.
- این کار کامپیوتر پایه را به کامپیوترهای کم دستور RISC واقعی نزدیک می‌کند که اگر خط لوله به بهترین صورت پیاده‌سازی شود، هر دستورالعمل در یک سیکل اجرا می‌شود.
- در اسلاید بعد می‌بینیم که این همزمانی چگونه خواهد بود.

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

- البته همیشه روند خط لوله طبق برنامه پیش نمی‌رود. مثلا در اینجا سه دستور اول بخوبی موازی سازی شده اند، چون مراحل آنها (مثلا فچ دستور دوم و دیکد دستور اول) کاری بهم ندارد.
- اما دستور چهارم به تعویق افتاده است. چرا که دستور سوم یک انشعاب دارد و تا اجرای دستور انشعاب به پایان نرسد، فچ دستور بعد از آن ناممکن است (چون مقصد انشعاب معلوم نیست).

مشکلات خط لوله دستورالعمل

- بطورکلی سه دسته مشکل خط لوله دستورالعمل را دچار اختلال و تعویق می‌کنند:
۱. مشکل دسترسی همزمان به حافظه. جدا کردن حافظه‌های کد و دیتا می‌تواند در این مورد کمک‌کننده باشد.
۲. مشکلی وابستگی داده‌های دستورالعمل‌ها. یعنی دستورالعمل فعلی نیاز به داده‌ای دارد که هنوز از دستورات قبلی بدست نیامده.
۳. مشکل انشعاب. دستوراتی که مسیر برنامه را عوض می‌کنند و تا دستور فعلی تکمیل نشود معلوم نمی‌شود که دستور بعدی باید از کجا اجرا شود (در اسلاید قبل این مشکل را دیدیم. برای شماره ۲ و ۳ راه حل تعویق است).

خط لوله حسابی

- می‌دانیم که اعمال حسابی مانند جمع و ضرب در واقع رشته‌ای از اعمال متوالی هستند که هر عمل وابسته نتیجه عمل قبلی است.
- به همین خاطر موازی‌سازی عملیات حسابی مشکل بوده اما تا حدودی ممکن است. طراحی سیستم به کمک کامپیوتر این کار را تسهیل می‌کند.
- برای پیاده‌سازی خط لوله (حسابی و دستورالعمل) به سخت‌افزار بیشتر نیاز داریم. اما نتیجه هزینه بیشتری که برای طراحی و ساخت سخت‌افزار می‌شود، کارایی (سرعت) بیشتر کامپیوتر خواهد بود.

تمرین

- یک روتین وقفه بنویسد که کار ورودی را انجام دهد (این برنامه اصلا خروجی ندارد و رفتن به روتین وقفه قطعا به معنای انجام ورودی است). هر بار که این روتین وقفه اجرا می شود مقدار دریافت شده از ورودی در یک خانه از آرایه صدتایی (که نام اولین خانه‌اش A است) نوشته می‌شود. اگر تعداد دریافت ورودی به ۱۰۰ برسد (یعنی آرایه پر شود) روتین وقفه غیر فعال شده و دیگر ورودی‌ای پذیرفته نمی‌شود.
- تفاوت بین SRAM و DRAM را به زبان خودتان و در حداکثر سه سطر بنویسید (پاسخ‌های کپی شده از جزوه و طولانی‌تر از سه سطر تصحیح نمی‌شود).

خسته نباشید

Q&A