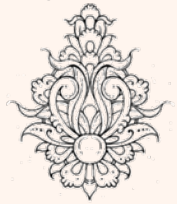
زبان ماشین و اسمبلی
(۰۰۵-۱۱-۱۳)
بخش دوم

آشنایی با مقدمات

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
زمستان ۱۳۹۳
احمد محمودی ازناوه

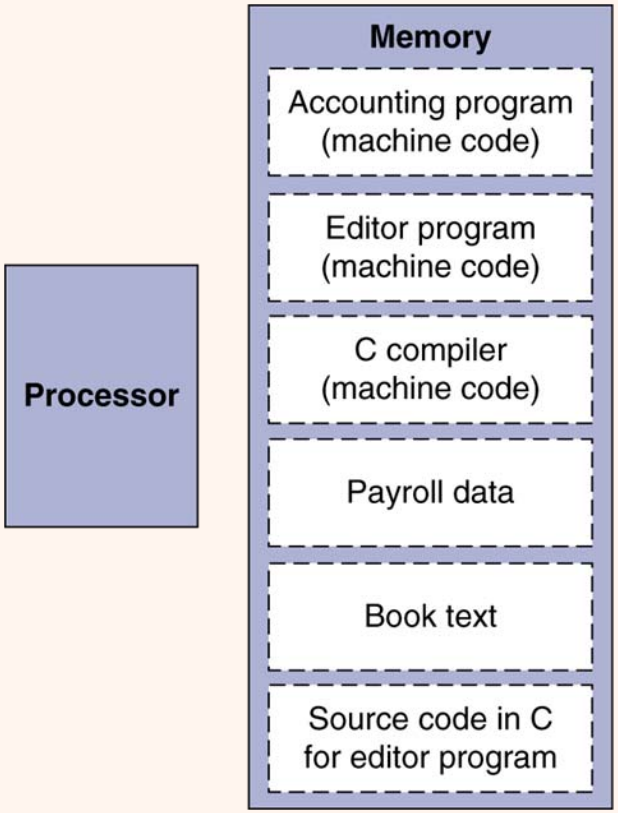
فهرست مطالب

- سلسله مراتب و تجرید
- سطوح مختلف اجرای برنامه
  - معماری مجموعه‌ی دستورالعمل
- اسمبلی چیست؟
- چرا اسمبلی؟
- ساختار کامپیوتر
  - مفهوم گذرگاه
  - حافظه و ثبات
  - انواع ثبات
  - مراحل اجرای دستورالعمل‌ها



ساختار کامپیوترها

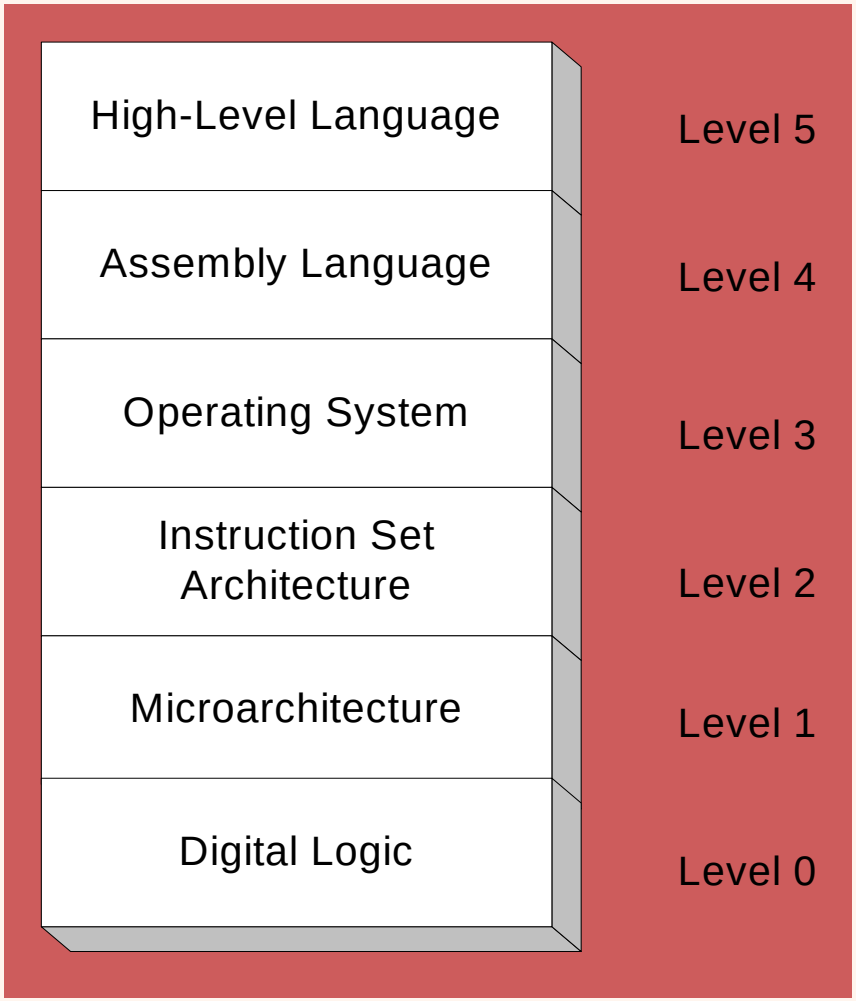
- برنامه‌ها، مانند داده‌ها به صورت دودویی نمایش داده می‌شوند.
- داده‌ها همراه با برنامه‌ها در حافظه ذخیره می‌شوند.
- در صورت استاندارد بودن دستورها، برنامه‌ها قابلیت اجرا روی کامپیوترها را خواهند داشت.



سلسله‌مراتب

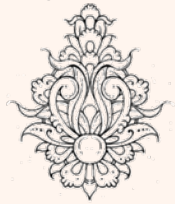
- «چه سخت افزار و چه نرم‌افزار، هر کدام از چندین لایه‌ی سلسله مراتبی تشکیل شده اند که هر لایه جزئیات درونی آن لایه را از لایه‌ی بالاتر پنهان نگاه می دارد. قاعده‌ی «تجرید» روشی است که طراحان سخت افزار و نرم افزار براساس آن از عهده‌ی پیچیدگی سیستم‌های کامپیوتری بر می‌آیند.
- تجرید کامپیوترها فرآیندی عقلی است که در آن طراح کامپیوتر از بعضی مشخصات یا صفات اختصاصیِ کامپیوترها صرف نظر می‌کند و ذهن خود را به یک خطّ فکری مشخص و مفاهیمی مشترک و عامّ منحصر می‌سازد».

سطوح مختلف اجرای برنامه



معماری مجموعه‌ی دستورالعمل

- یکی از کلیدی ترین واسط‌های بین سخت‌افزار و نرم‌افزار سطح پایین «معماری مجموعه‌ی دستورالعمل» (ISA) یا همان واسط است. سطوح تجرید، چنین واسط مجردی است که این امکان را فراهم آورده تا پیاده‌سازی‌های متعدد با قیمت و کارآیی متفاوت از یک سخت‌افزار خاص وجود داشته باشد و همه‌ی آن‌ها بتوانند نرم‌افزار واحدی را اجرا کنند.»



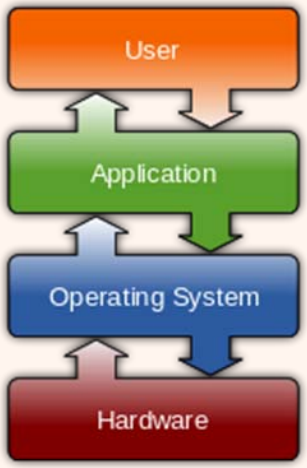
معماری مجموعه‌ی دستورالعمل

- «یکی از مهم‌ترین تجریدها واسط بین سخت‌افزار و پایین‌ترین سطح نرم‌افزار است که به دلیل اهمیت آن، به این سطح از تجرید نام ویژه‌ای اعطا شده است:«معماری مجموعه‌ی دستورالعمل» یا در یک کلمه‌ی ساده «معماری». معماری مجموعه دستوالعمل‌های یک کامپیوتر (که از آن با کوته‌واژه‌ی ISA یاد خواهد شد) مشتمل بر تمام آن چیزی است که برنامه‌نویسان برای خلق برنامه‌ای به زبان دودویی ماشین و با عملکردی درست نیازمند دانستن آن‌ها هستند (شامل تمام دستوالعمل‌ها، دستگاه‌های ورودی/خروجی و نظائر آن).»

conventional machine language

واسط دودویی برنامه‌های کاربردی

- «معمولا جزئیات انجام عملیات ورودی/خروجی، تخصیص حافظه و تمام عملیات سطح پایین سیستمی توسط سیستم‌عامل مدیریت و انجام می‌شود و از این رو برنامه‌نویسان نباید نگران چنین جزئیاتی باشند. تلفیق مجموعه‌ی دستوالعمل‌های پایه‌ی یک کامپیوتر و واسطی که توسط سیستم‌عامل در اختیار برنامه‌نویسان قرار داده می‌شود اصطلاحا به «واسط دودویی برنامه‌های کاربردی» یا ABI شهرت دارد.»



گام‌های تبدیل برنامه به زبان ماشین

Display the sum of A times B plus C.

داده‌ها متغیر سراسری هستند

C

```
printf("a*b+c\n",a*b+c);
```

یک به چند

کامپایلر

زبان اسمبلی x86

```
movl   a, %edx
movl   b, %eax
imull  %eax, %edx
movl   c, %eax
addl   %eax, %edx
movl   $.LC0, %eax
movl   %edx, 4(%esp)
movl   %eax, (%esp)
call   printf
```

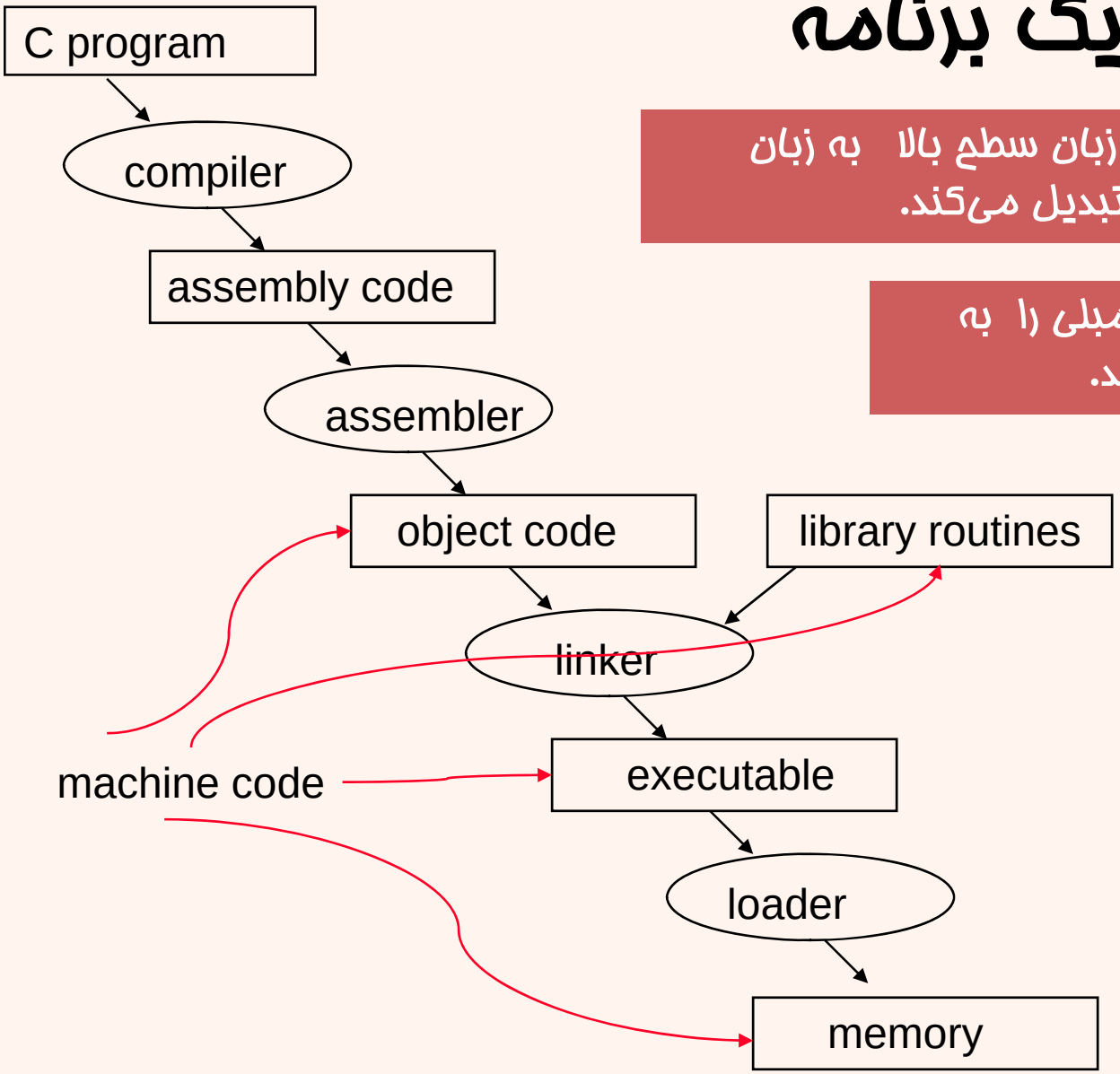
اسمبلر

تناظر یک به یک

زبان ماشین Intel

```
8b 15 14 a0 04 08
a1 18 a0 04 08
0f af d0
a1 1c a0 04 08
01 c2
b8 e0 84 04 08
89 54 24 04
89 04 24
e8 09 ff ff ff
```

ترجمه‌ی یک برنامه



کامپایلر برنامه به زبان سطح بالا به زبان اسمبلی (ماشین) تبدیل می‌کند.

اسمبلر برنامه به زبان اسمبلی را به زبان ماشین تبدیل می‌کند.

معمولا کامپایلرها کد زبان سطح بالا را مستقیم به زبان ماشین تبدیل می‌کنند، به عنوان مثال gcc، همه مراحل را همزمان انجام می‌دهد.
با استفاده از سوییچ S- کد اسمبلی را از کامپایلر گرفت.
با استفاده از سوییچ c– object code را تولید می‌کند.

پیونده و بارکننده

Linker or Link editor

- پیونده‌ده یک برنامه‌ی سیستمی است که برنامه‌هایی که به زبان ماشین تبدیل شده‌اند را با هم ترکیب کرده، آدرس برچسب‌ها را مشخص می‌کند و در نهایت یک فایل اجرایی تولید می‌کند.

loader

- بخشی از سیستم‌عامل است.
  - خواندن سرآیند برای مشخص کردن حجم برنامه، آماده کردن فضای حافظه‌ی برای داده‌ها و دستورها، منتقل کردن محتویات برنامه و داده‌ها به حافظه، آماده‌سازی پشته و انتقال آرگومان‌ها، آماده‌سازی ثبات‌ها و پرش به آدرس شروع برنامه

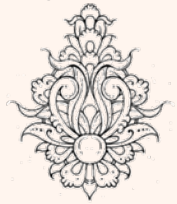
در باره‌ی جزییات فرآیند تبدیل برنامه به کد ماشین قابل اجرا، در بخش دوم درس توضیحات بیشتری ارائه خواهد شد.

زبان‌های سطح بالا

- بیشتر برنامه‌ها به زبان‌های سطح بالا (high level) نوشته می‌شوند.
- این برنامه‌ها شامل عباراتی به زبان انگلیسی به همراه یک سری عبارات ریاضی هستند.
- با کمک زبان‌های سطح بالا جزییات اجرای یک برنامه از دید برنامه‌نویس پنهان می‌شود.
- در عمل برنامه‌نویس ساختار واقعی کامپیوتر را فراموش می‌کند، نوشتن برنامه به این زبان‌ها بسیار ساده خواهد بود.
- زبان‌های سطح بالا برای نوشتن برنامه‌های کاربردی مورد استفاده قرار می‌گیرد.

زبان‌های سطح بالا (ادامه...)

- زبان‌های سطح بالا به برنامه‌نویس کمک می‌کنند به جای درگیری با جزییات سخت‌افزار، بر روی مسأله تمرکز کند.
- هدف یک برنامه‌نویس حل مسأله است، بدون این که بخواهد در جریان نحوه‌ی کار یک سری ادوات الکترونیکی(وسیله) قرار بگیرد.
- بدین‌ترتیب، مسأله زودتر حل می‌شود.
- برنامه‌ای به زبان سطح بالا نوشته می‌شود، قابل اجرا بر روی سکوهای مختلفی است.



زبان ماشین چیست؟

- برای فرمان دادن به سخت‌افزار می‌باید به زبان آن صحبت کنید. واژگانی که فرهنگ زبان کامپیوتر را تشکیل می‌دهند، «دستورالعمل» می‌نامند، این دستورالعمل‌ها به صورت یک عدد در مبنای دو برای ماشین قابل درک هستند. به مجموعه‌ی این دستورالعمل‌ها «زبان ماشین» می‌گویند.

instruction

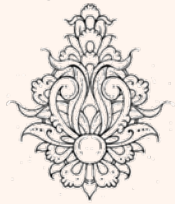
machine language

- هر پردازنده زبان خاص خود را دارد.
- برخلاف تنوع و گوناگونی زبان آدمیان، زبان‌های کامپیوترهای مختلف بسیار به هم شبیه هستند و بیشتر مشابه لهجه‌ها و گویش‌ها متفاوت هستند. بدین‌ترتیب، با یاد گرفتن یک زبان، فراگرفتن زبان دیگر کاری دشوار نخواهد بود. ریشه‌ی این شباهت، قواعد و اصول مشابهی است که فناوری سخت‌افزار بر پایه‌ی آن بنا شده است.

یکی از خصوصیات مهم در مورد مجموعه‌ی دستورات این است که سخت‌افزار مورد نیاز آن ساده باشد.

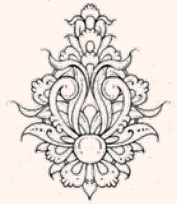
زبان اسمبلی چیست؟

- استفاده از زبان ماشین تقریبا امکان‌پذیر نیست.
- از این رو از یک معادل ساده‌تر استفاده می‌شود.
- «زبان اسمبلی» تناظر یک به یک با «زبان ماشین» دارد.
- در این زبان به جای «صفر و یک»های زبان ماشین یک سری نماد جایگزین می‌شود.
- اسمبلر برنامه‌ی نوشته شده به زبان اسمبلی را به زبان ماشین تبدیل می‌کند.
- برنامه نوشته شده به زبان اسمبلی(زبان ماشین) تنها بر روی سخت‌افزاری که برای آن زبان ساخته شده است، قابل اجرا خواهد بود.



چند نکته در مورد اسمبلی

- کامپیوترها موظف به ارائه‌ی عملیات پایه‌ی کم‌تعداد و پرکاربرد هستند.
- با آشنایی با دستوالعمل‌های زبان ماشین، رموز نهفته‌ی در مفهوم «برنامه‌های ذخیره شده» بر شما آشکار خواهد شد.
- بر خلاف سایر زبان‌های برنامه‌نویسی، برای زبان اسمبلی استاندارد واحدی وجود ندارد.
- اسمبلرهای مختلف از قواعد متفاوتی استفاده می‌کنند.

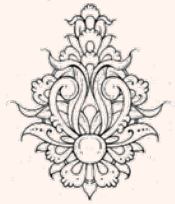


چرا اسمبلی؟

- برای درک نحوه‌ی کارکرد کامپیوتر باید اسمبلی بدانیم، بدین ترتیب ارتباط سیستم‌عامل، سخت‌افزار و برنامه‌های کاربردی را بهتر خواهیم فهمید. بدین‌ترتیب
  - می‌توانیم علت بروز خطاها را بهتر درک کنیم.
  - می‌توانیم از الگوریتم‌های بهینه بهره بگیریم و برنامه‌های بهتری بنویسیم.
  - می‌توانیم رفتار یک کامپیوتر را بهتر پیش‌بینی کنیم.
  - می‌توانیم کامپایلرهای بهتری بنویسیم.
  - می‌توانیم کامپیوترهای بهتر(سریع‌تر و یا ارزان‌تری) طراحی کنیم.
  - می‌توانیم برای کاربرد مورد نظر پردازنده‌ی مناسب را انتخاب کنیم.

چرا اسمبلی؟(ادامه...)

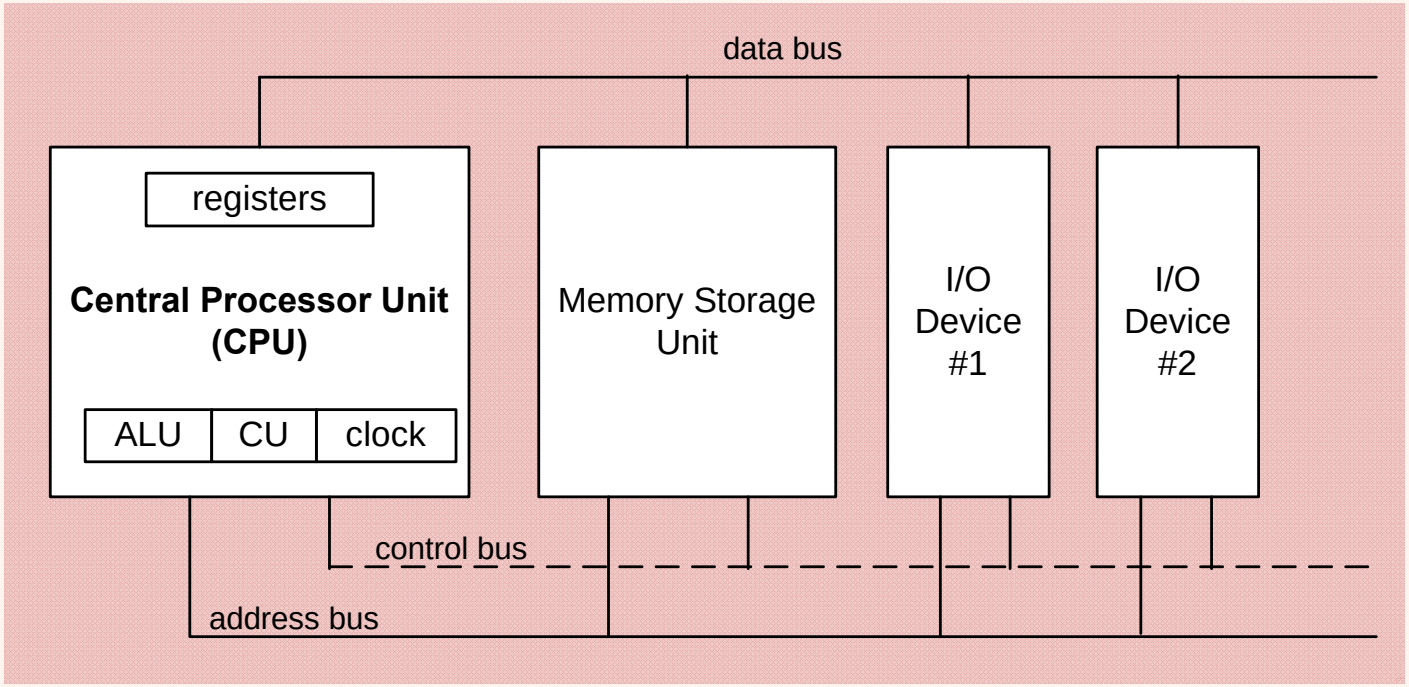
- بهترین برنامه‌نویسان به خوبی با عملکرد سخت‌افزار آشنا هستند.
- برای نوشتن برنامه‌های زیر به زبان اسمبلی نیاز خواهیم داشت.
  - برای سیستم‌های توکار
  - برای نوشتن برنامه‌های راه‌انداز (driver)، برنامه‌ای که وظیفه‌ی تعامل با سخت‌افزارهای خاص را دارد.
  - برای نوشتن برنامه‌های که در آن کارایی اهمیت دارد، بسیاری از وسایل ویژه‌ی بازی محدودیت‌هایی در رابطه با حافظه دارند، از این رو برنامه‌هایی که برای این گونه وسایل نوشته می‌شود، می‌باید به خوبی بهینه‌سازی شود.
  - امنیت نرم‌افزار



کاربردهای سطوح مختلف زبان

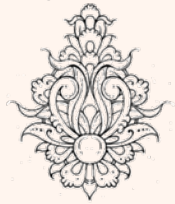
نوع کاربرد	زبان سطح بالا	زبان اسمبلی
برنامه‌های تجاری که برای اجرا بر روی سکوی (platform) واحدی در نظر گرفته شده‌اند.	به راحتی قابل سازماندهی است و به روزرسانی آن آسان است.	کار دشواری است و نیاز به مهارت و تجربه‌ی بالایی دارد.
برنامه‌های راه‌انداز(driver)	امکان دسترسی مستقیم به سخت‌افزار وجود ندارد، اگر هر ممکن باشد، به شیوه‌ای دشوار خواهد بود که به روز رسانی را دشوار می‌کند.	بسیار راحت و سرراست خواهد بود، به خوبی قابل مستندسازی و به روزرسانی است
برنامه‌های تجاری که بناست بر روی سکوهای (platform) گوناگون اجرا شوند.	به راحتی می‌توان آن‌ها را از نو کامپایل کرد.	کدها باید برای هر سکو از نو نوشته شده و با اسمبلر متفاوتی، اسمبل شوند.
سیستم‌های توکار و بازی‌های کامپیوتری که به دسترسی مستقیم به حافظه احتیاج دارند.	حجم کد بسیار بالا خواهد بود و کارآیی مناسبی نخواهد داشت.	کد تولید شده کوچک و سریع خواهد بود.

ساختار پایه‌ی یک کامپیوتر



مفهوم گذرگاه

- گذرگاه(Bus) یک سیستم ارتباطی است که امکان انتقال داده بین بخش‌های مختلف یک کامپیوتر را فراهم می‌کند.
- در واقع یک مجموعه سیم است که بخش‌های مختلف را به هم متصل می‌کند(در درس مدار منطقی به طور کامل با جزییات چنین سیستمی آشنا خواهید شد).
- در این سیستم یک مقصد و یک مبدأ وجود دارد.
- گذرگاه از نظرهای مختلفی قابل تقسیم هستند:
  - گذرگاه داده
  - گذرگاه آدرس
  - گذرگاه کنترل

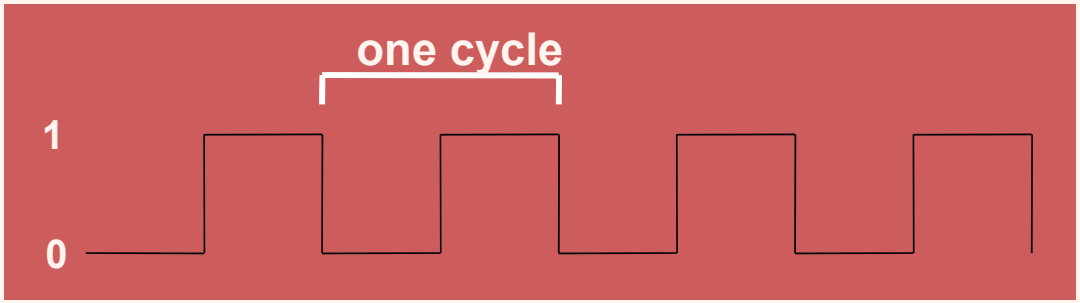


مفهوم گذرگاه (ادامه...)

- برای نوشتن در حافظه باید آدرس خانه‌ی مورد نظر مشخص باشد، همچنین دستگاه‌های ورودی-خروجی(I/O) آدرس مشخص و یکتایی دارند. پردازنده این آدرس از طریق گذرگاه آدرس منتقل می‌کند، همه‌ی بخش‌ها این آدرس‌ها را می‌خوانند، تا تشخیص دهند مورد خطاب پردازنده هستند یا نه؟ این گذرگاه یک طرفه است.
- پردزانده چه بخواهد داده‌ای را در حافظه بنویسد یا بخواند این انتقال از طریق گذرگاه داده انجام می‌شود، در مورد دستگاه‌های I/O نیز وضعیت بدین منوال است. این گذرگاه دو طرفه است.
- دو گذرگاه قبلی مجموعه‌ای از سیم‌های موازی بودند که وظیفه‌ی جابجایی داده‌ها و آدرس‌ها را داشتند، گذرگاه کنترل مجموعه‌ای از سیم‌های مستقل است که نوع عملکرد را مشخص می‌کند، مثلا در مورد حافظه مشخص می‌کند که کدام یک از دو عمل خواندن یا نوشتن باید انجام شود؟

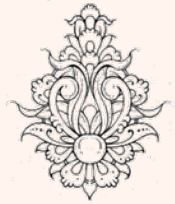
ساعت سیستم (clock)

- برای همگام‌سازی فعالیت‌ها مورد استفاده قرار می‌گیرد.
- معیاری برای مدت زمان لازم برای انجام یک کار است.



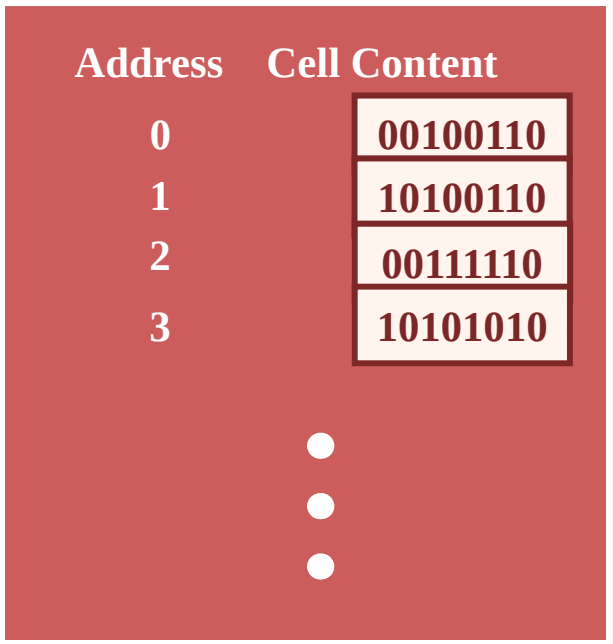
حافظه

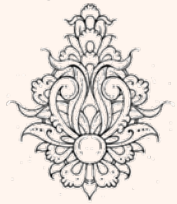
- حافظه را می‌توان به صورت آرایه‌ای بزرگ از داده‌های یک بایتی تصور کرد.
  - واقعیت، با این ذهنیت دارای تفاوت‌هایی است، البته این مساله از دید برنامه‌نویس مخفی است، در آینده بیشتر در مورد این جزییات خواهید دید.
- در بسیاری از پردازنده‌ها مانند خانواده‌ی x86 دستیابی به تک‌تک بایت‌های حافظه امکان‌پذیر است.
- با این حال به هر بار دسترسی به حافظه امکان خواندن دو و یا چهار بایت داده نیز وجود دارد.
- آدرس، در واقع عددی است که به یک موقعیت خاص در حافظه مراجعه می‌کند.



آدرس‌های حافظه

- هر آدرس، یک عدد صحیح بدون علامت است. تمامی دستیابی‌ها به حافظه از طریق این آدرس صورت می‌پذیرد.





حافظه

آدرس‌ها در مبنای شانزده

جایگاه بیت‌ها

7 6 5 4 3 2 1 0

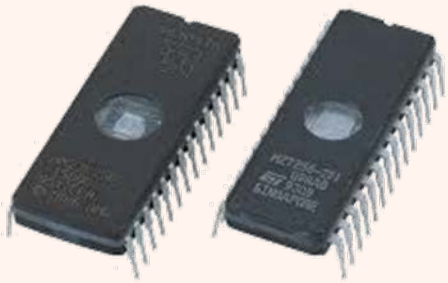
FFFF

0002
0001
0000

byte-addressable

این حافظه چند بایتی است؟

چند خط آدرس دارد؟



n address bits → 2^n addresses

تمرین

- یک حافظه‌ی 2MB شامل چند بیت است؟

$2 = 2^1$, $M = 2^{20}$, $B = byte = 8 = 2^3$

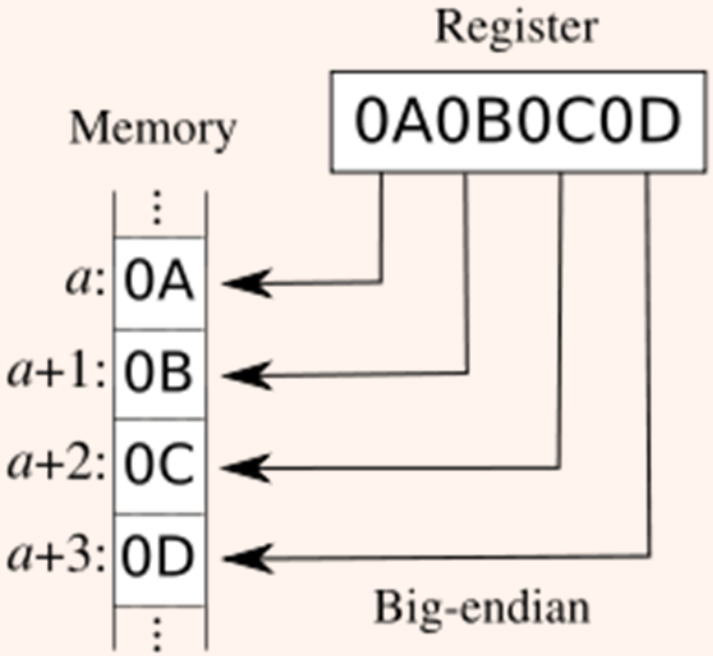
$2^1 \times 2^{20} \times 2^3 = 2^{24} = 16,777,216$

- این حافظه به چند خط آدرس نیاز دارد؟
  - بیست و یک خط
- یک پردازنده دارای ۳۲ خط آدرس می‌باشد، این پردازنده حداکثر چه میزان حافظه را می‌توان آدرس‌دهی کند؟
  - **4GB**

وقتی داده‌ای بیش از یک بایت فضا نیاز داشته باشد، چه آدرسی به آن اختصاص داده می‌شود و به چه ترتیب در حافظه قرار خواهد گرفت؟

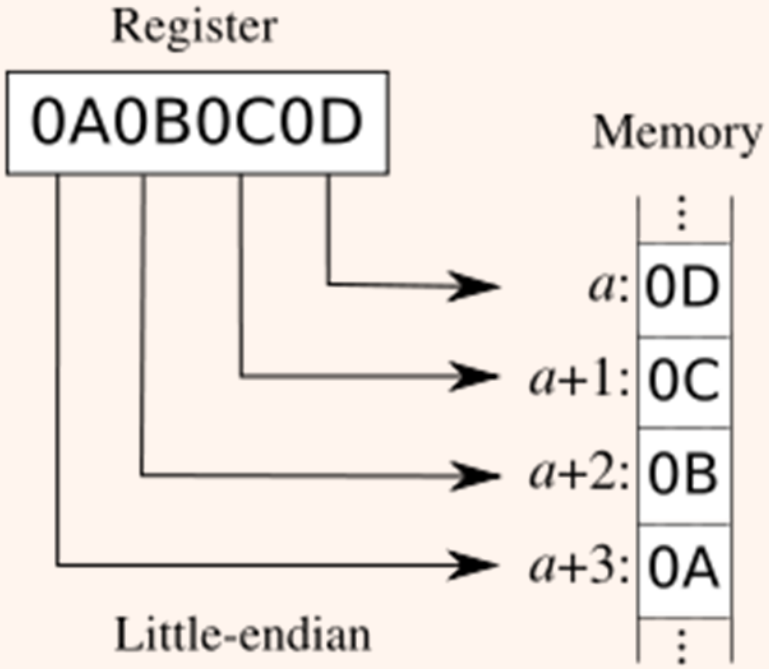
نحوه‌ی نمایش اعداد در حافظه

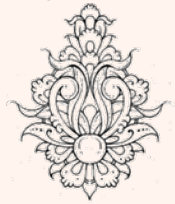
- در شیوه‌ی نمایش big endian باارزش‌ترین رقم در کوچک‌ترین آدرس ذخیره می‌شود.
  - Motorola، Sun



نحوه‌ی نمایش اعداد در حافظه (ادامه...)

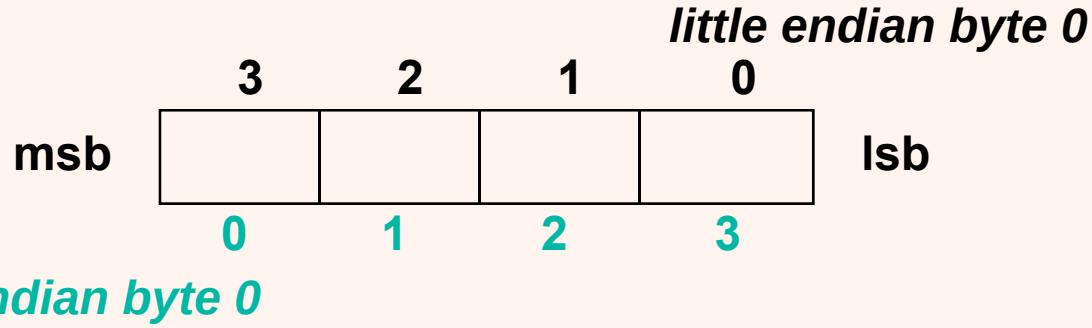
- در شیوه‌ی نمایش little endian باارزش‌ترین رقم در بزرگ‌ترین آدرس ذخیره می‌شود.
  - DEC، IBM و Intel



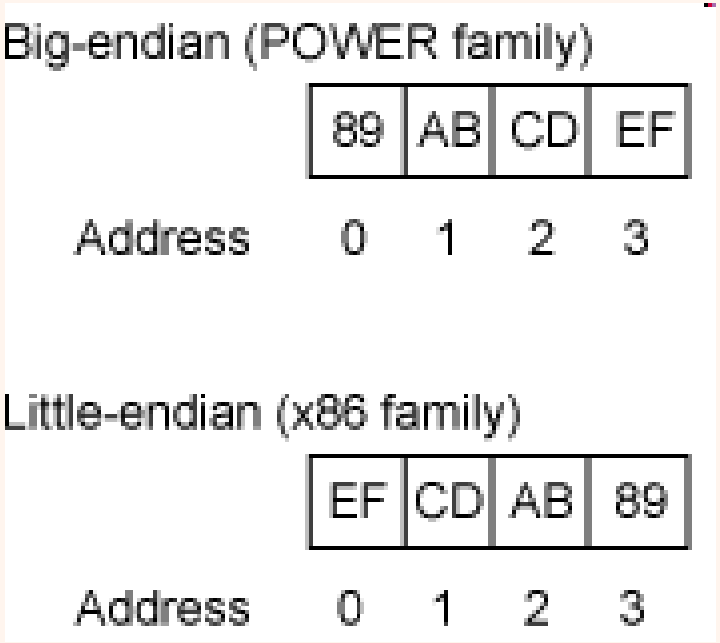


نحوه‌ی نمایش اعداد در حافظه (ادامه...)

- به نظر شما کدام شیوه دارای برتری است؟

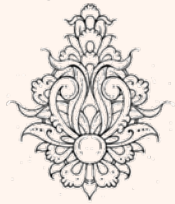


89ABCDEF



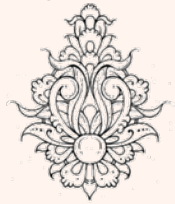
ثبات چیست؟

- حافظه‌ای سریع و کم‌حجم درون پردازنده است. برای دخیره‌ی موقت داده توسط پردازنده مورد استفاده قرار می‌گیرد.
- بعضی از ثبات‌ها برای کارهای خاص سیم‌کشی شده‌اند! و در نتیجه کاربردهای خاصی دارند.
- ممکن است شامل داده و یا حتی آدرس باشند.
- تعداد و انواع آن بسته به نوع پردازنده متفاوت است.



ثبات چیست؟(ادامه...)

- ثبات را هم می‌توان به دو گروه تقسیم کرد:
  - ثبات‌های در دسترس (آدرس‌پذیر):
    ثبات‌هایی که توسط برنامه‌نویس قابل استفاده هستند.
  - ثبات‌های (آدرس‌ناپذیر): ثبات‌هایی که توسط سخت‌افزار مورد استفاده قرار می‌گیرد.



ثبات‌های حافظه

- این ثبات‌ها جزء ثبات‌های آدرس‌ناپذیر می‌باشند.

Memory Address Register (MAR)

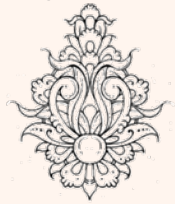
– ثبات آدرس حافظه: این ثبات درون پردازنده قرار دارد و به خطوط آدرس متصل است.

Memory Data Register (MDR)

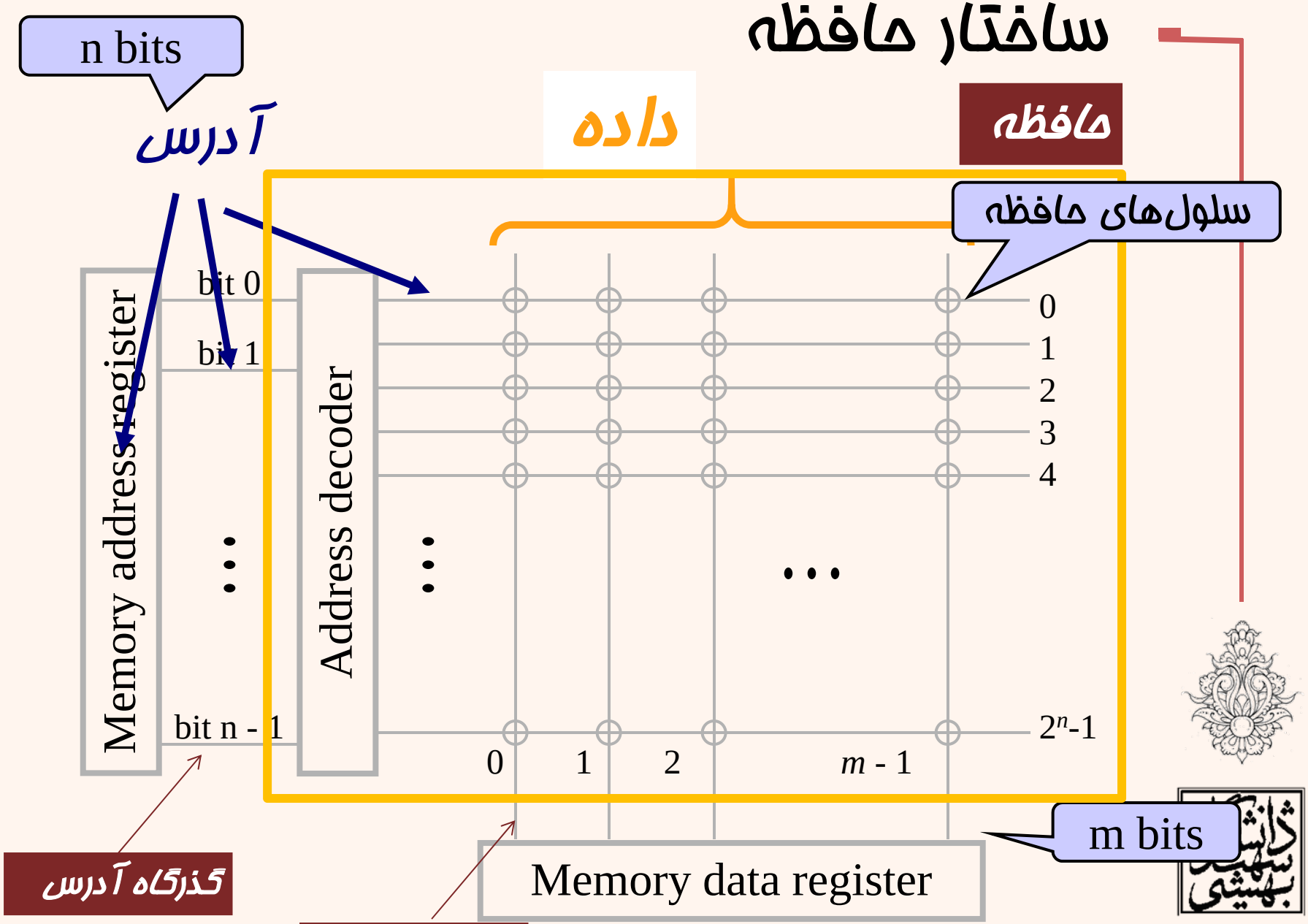
Memory Buffer Register (MBR)

– ثبات داده‌ی حافظه:

– داده‌ای که از حافظه خوانده می‌شود، در این ثبات قرار می‌گیرد.



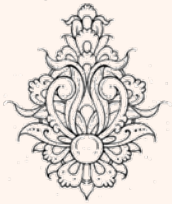
ساختار حافظه



تمرین

- در یک پردازنده MAR بیست و چهار بیتی است، حداکثر فضایی که این پردازنده قادر به آدرس‌دهی است را حساب کنید؟

  – **16MB**



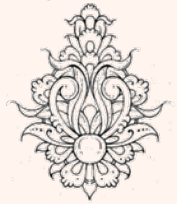
برخی ثبات‌های مهم

- ثبات دستورالعمل (Instruction Register):
  این ثبات بخشی از واحد کنترل است که دستورالعمل جاری را در خود ذخیره می‌کند.

Instruction Pointer

program counter

- شمارنده‌ی دستورالعمل (program counter):
  این ثبات، توالی اجرای برنامه را حفظ می‌کند، بسته به نوع طراحی آدرس دستور جاری و یا دستورالعمل بعدی را در خود نگه می‌دارد.
- ثبات پرچم (Flag Register) یا ثبات وضعیت (register status):
  نتیجه‌ی محاسبات ALU از قبیل صفر شدن نتیجه، علامت نتیجه، بروز سرریز و .. توسط بیت‌های مختلف این ثبات مشخص می‌شوند.



گام‌های مختلف اجرای یک برنامه

- واکشی (Fetch)
- رمزگشایی (Decode)
- خواندن عملوندها (Fetch operands)
- اجرای دستور (Execute)
- نوشتن نتیجه‌ی دستورالعمل (Store output)