با نام خدا

برنامه نویسی سخت افزار

مدرس
کیان ابراهیم کافوری
ترم دوم سال تحصیلی ۱۴۰۳-۱۴۰۲

برنامه نویسی سخت افزار

- مدرس: کیان ابراهیم کافوری
- email: kian_kafoori@yahoo.com
- Web page: kafoori.blog.ir (مراجعه همیشگی به این وبلاگ الزامی است.)
- پیشنیاز درس: مدار منطقی، آشنایی با زبان های برنامه نویسی مفید خواهد بود.
- سرفصل مطالب درس:
  - اصول کارکرد کامپیوترها و اصول برنامه نویسی
  - معرفی میکروکنترلرها
  - آشنایی با زبان برنامه نویسی مرجع (C و C++)
  - برنامه نویسی برای میکروکنترلر
  - معرفی سخت افزار و برنامه نویسی برد های Arduino
  - آشنایی با ابزارهای جانبی متداول
  - آشنایی با Raspberry Pi

برنامه نویسی سخت افزار

- روزهای برگزاری: شنبه
- نکته های مهم برای موفقیت در این درس:
  - توجه کامل به مطالب در کلاس.
  - حل تمرین ها، و پرسیدن مشکلات.
  - فشردگی مطالب و لزوم مطالعه هفتگی دروس.
  - هشدار: مطالب خوانده نشده به سرعت انباشته خواهد شد.
  - هشدار: این درس شب امتحانی نیست.
  - اگر خودتان تمرین ها را حل نکنید در جلسه امتحان قادر به حل مسائل نخواهید بود.
  - حضور به موقع در کلاس.
  - ساعت کلاس: ۴:۱۰ تا ۶:۴۰ بعد از ظهر
  - اسلاید ها در اختیار شما قرار خواهد گرفت.
- منبع امتحان: اسلاید ها + مطالب گفته شده در کلاس+ تمرین ها.

برنامه نویسی سخت افزار (ارزشیابی)

- ارزشیابی درس:
  - فعالیت کلاسی (تمرین های پایان فصل) + امتحان پایان ترم

- نمره پایانی شما تنها حاصل تلاش شما است و نه هیچ عامل دیگر
- امتحان پایان ترم از همه مطالب خواهد بود.
- حضور در کلاس الزامی است.
- تحویل پاسخ تمرین ها در زمان تعیین شده دارای اهمیت زیادی است.

برنامه نویسی سخت افزار (منبع های درس)

- "The AVR microcontroller and embedded systems using assembly and C" by M. Mazidi et al. , Prentice Hall , 2011.
- AVR datasheets
- کتاب های تالیفی فارسی زبان درباره میکروکنترلرها
- "Arduino for dummies" by John Nussey, John Wiley & Sons, 2018.
- "Arduino Microcontroller Processing for Everyone!" by Steven F. Barrett, Morgan & Claypool Publishers, 2010.
- "Getting Started with Raspberry Pi" by Matt Richardson and Shawn Wallace, O'Reilly Media, 2013.
- کتاب های تالیفی فارسی زبان درباره Arduino و Raspberry Pi.

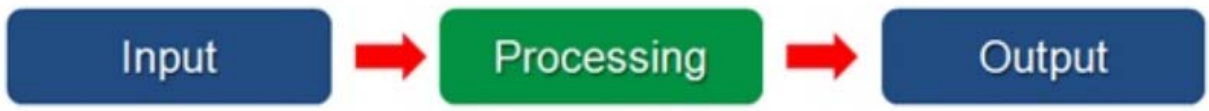
اصول کارکرد کامپیوترها و اصول برنامه نویسی

برنامه نویسی سخت افزار

بخش اول

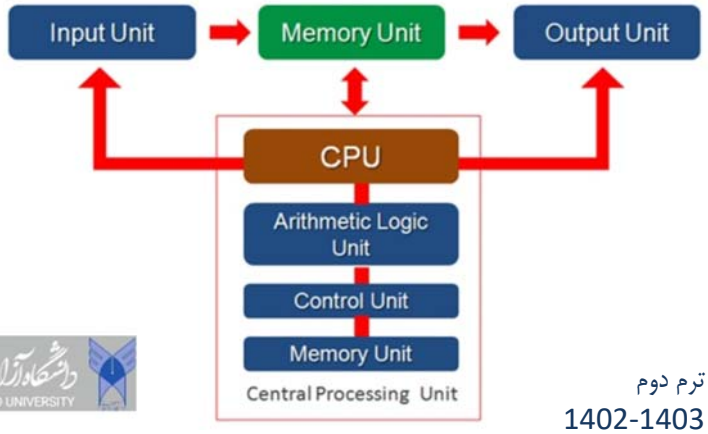
کامپیوتر چیست و چگونه کار می کند؟

- کامپیوتر ابزاری است که برنامه نوشته شده کاربر را اجرا می کند.
- برنامه شامل مجموعه دستورهایی که به شکل خط به خط و پیاپی اجرا می شوند.
- عملکرد اجرایی کامپیوتر به طور کلی یکی از چهار مورد زیر است:
  - دریافت از اطلاعات از ورودی، ارسال داده به خروجی (ارتباط با I/O)
  - ذخیره یا بازیابی اطلاعات از حافظه (Memory Read/Write)
  - پردازش اطلاعات (عملیات ریاضی، مقایسه، و...) (Data Processing)
  - کنترل روند اجرای برنامه (تصمیم گیری، حلقه، زیر برنامه، و ...)
- به برنامه اجرایی توسط کامپیوتر نرم افزار گفته می شود.



کامپیوتر چیست و چگونه کار می کند؟ (ادامه)

- کامپیوتر برنامه کاربر را به کمک اجزای داخلی خود انجام می دهد.
- اجزای داخلی کامپیوتر را به طور کلی می توان در سه بخش طبقه بندی کرد:
  - واحد پردازش مرکزی (CPU): که وظیفه آن اجرای برنامه است.
  - حافظه (Memory):برنامه کاربر و داده ها در آن نگهداری می شود.
  - دستگاه های ورودی/خروجی (Input/output): ارسال/دریافت داده به بیرون.
- عملکرد اجزای داخلی و نحوه اتصال آنها موضوع بحث سخت افزار است.



برنامه نویسی برای کامپیوتر چگونه انجام می شود؟

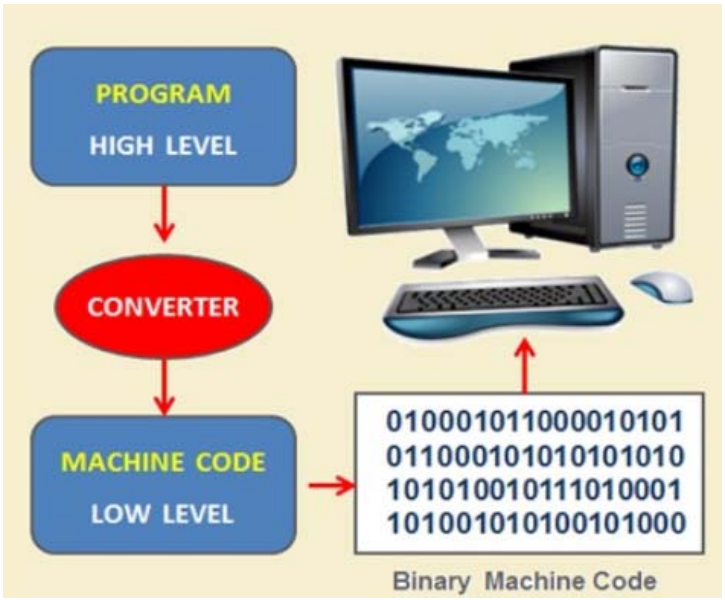
- برنامه شامل مجموعه از دستورهای مشخص شده است که کامپیوتر آنها را به ترتیب اجرا می کند. برای اجرای یک برنامه در هر لحظه یک دستور اجرا می شود.
- ساختار دستورها برمبنای قوانین «زبان برنامه نویسی» تعریف شده اند.
- زبان های برنامه نویسی متعددی ایجاد شده اند: C/C++، Basic، Python، و ...
- این زبان های معیار «زبان سطح بالا» نام دارند.
- این برنامه ها برای اجرا روی کامپیوتر ها به زبان ماشین تبدیل می شوند.

برنامه نویسی برای کامپیوتر چگونه انجام می شود؟ (ادامه)

- روند اجرای یک برنامه در زبان سطح بالا در کامپیوتر

Computer Program :

```
int main()
{
// Variable declaration
int a, b, sum;
// Take two numbers as input from the
user
scanf("%d %d", &a, &b);
// Add the numbers and assign the value
// to some variable
sum = a + b;
// Use the calculated value
printf("%d\n", sum);
return 0;
// End of program
}
```

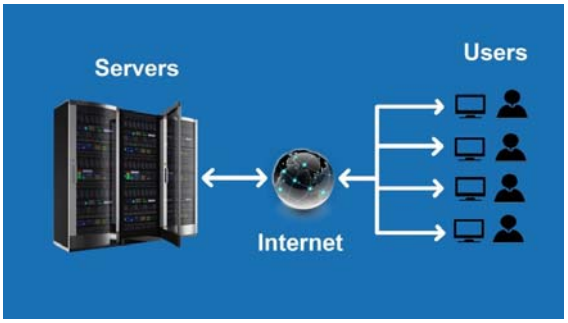


انواع کامپیوترها

- با توجه به کاربرد، میزان قدرت پردازش مورد نیاز، قیمت و سایر عوامل کامپیوترهای مختلفی تولید شده اند. می توان کامپیوترهای موجود را دسته های زیر طبقه بندی کرد:
  - کامپیوتر های Desktop (رومیزی): از جمله PC ها و لپ تاپ ها
    - هدف آنها کاربرد روزمره تا کاربرد حرفه است. مهمترین معیار آنها ایجاد کارایی و قیمت مناسب است.
  - کامپیوترهای همراه: گوشی ها و ساعت های هوشمند، تبلت ها، و ...
    - مهمترین معیار در این کامپیوترها، توان مصرف پایین، اندازه کوچک، کارایی، و قیمت مناسب است.

انواع کامپیوترها (ادامه)

- ابر کامپیوترها (Supercomputers)
  - مجموعه ای از کامپیوترهای قدرتمند که در کنار هم و به شکل موازی کار می کنند.
  - قدرت پردازش بیشتر و سرعت بالاتر هدف اصلی طراحی این کامپیوترها است.
- Server ها و Mainframe ها
  - هدف آنها پاسخ گویی سریع و با تعداد بالا است.
  - وب سایت ها و اپلیکیشن ها با این کامپیوترها مدیریت می شوند.

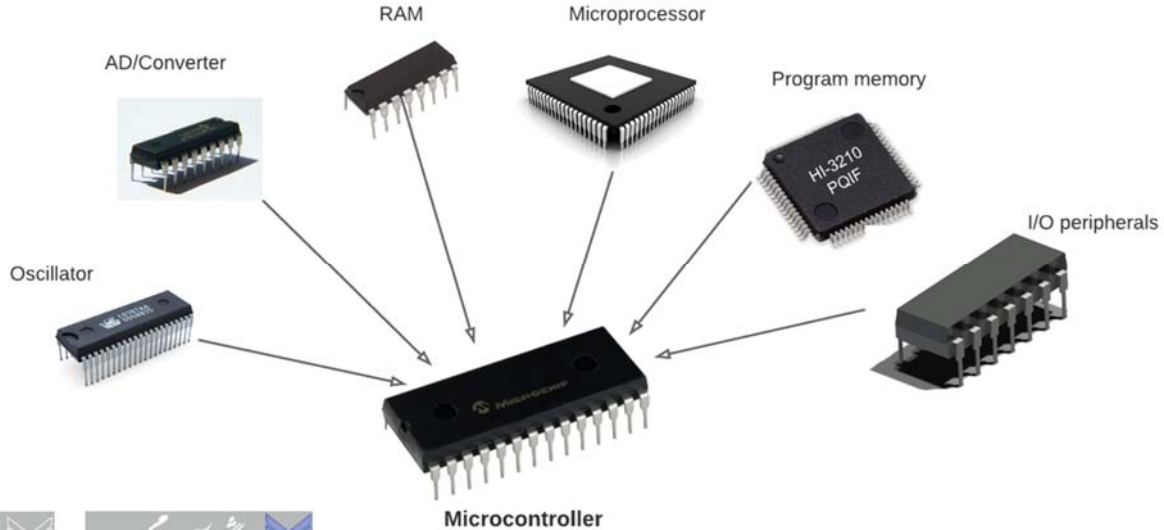


انواع کامپیوترها (ادامه)

- کامپیوترهای Embedded:
  - همه اجزای لازم برای راه اندازی کامپیوتر در یک IC گردآوری شده است.
  - هدف آنها اجرای برنامه ای نسبتاً ساده با قیمت پایین است.
  - اندازه کوچک، راه اندازی سریع و مصرف انرژی کم از دیگر مشخصات این دسته است.
  - قدرت پردازشی بسیار کمتر نسبت به دسته های قبل دارند.
  - در بسیاری از کاربردهای روزمره و دستگاه های مختلف به کار گرفته شده اند.

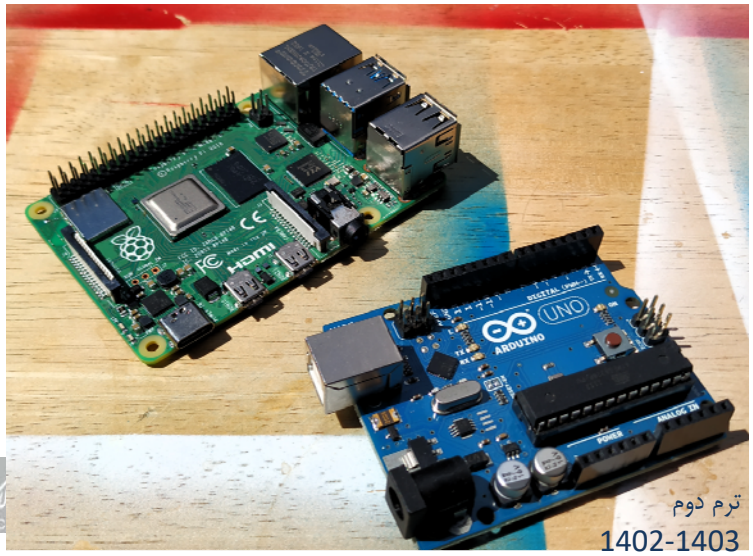
میکروکنترلرها

- میکروکنترلرها کامپیوترهایی از نوع Embedded هستند که به طور گسترده در کاربردهای مختلف به کار گرفته شده اند.
- سازندگان متعددی خانواده های مختلفی از میکروکنترلرها طراحی و تولید کرده اند.



محتوای این درس

- در این درس تمرکز ما بر برنامه نویسی برای کامپیوترهای Embedded است.
  - ابتدا برنامه نویسی برای یک خانواده میکرو کنترلر را می آموزیم
  - برنامه نویسی برای برد های Arduino
  - آشنایی با کامپیوتر های Raspberry Pi

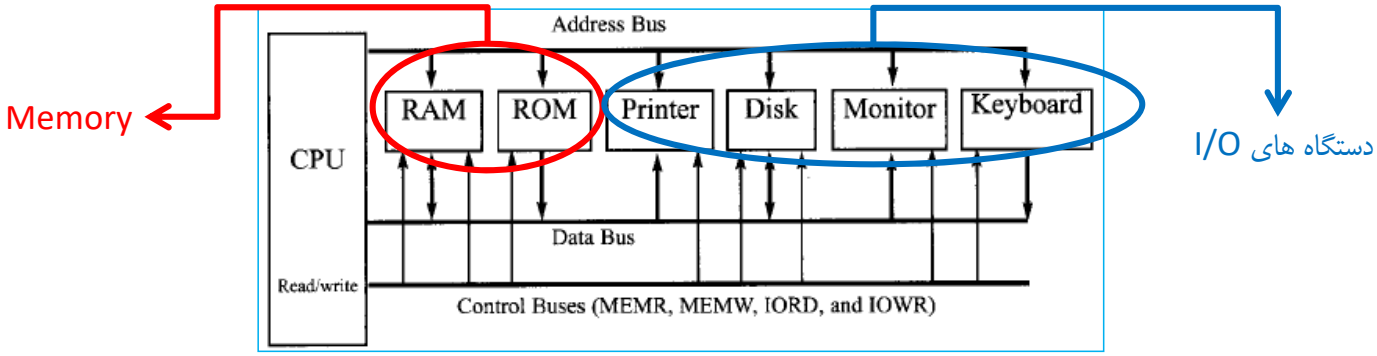


محتوای این درس (ادامه)

- همچنین ارتباط میکروکنترلرها و کامپیوترهای Embedded با مدارهای واسطه متداول مورد بررسی قرار می گیرد.
- در ادامه این فصل مقدمات مورد نیاز برای اشنایی با سخت افزار و نرم افزار معرفی می شود

سازماندهی درونی کامپیوتر

- سازمان داخلی هر کامپیوتر را می توان به سه بخش تقسیم کرد: CPU، Memory و دستگاه های I/O.



- وظیفه اصلی هر کامپیوتر: اجرای برنامه های نوشته شده کاربر.
- CPU برنامه ی موجود در Memory را اجرا می کند و با جهان بیرون به کمک دستگاه های I/O ارتباط برقرار می کند. همچنین داده های جاری را نیز در Memory ذخیره می کند.
- کاربر برنامه دلخواه خود را در Memory قرار می دهد.
- در حین اجرای برنامه، کاربر می تواند داده های و نتایج را از طریق دستگاه های I/O ارسال و دریافت کند.
- CPU با Memory و I/O به وسیله مجموعه سیم هایی به نام Bus در ارتباط است.
- سه نوع باس در هر کامپیوتر وجود دارد: داده، آدرس، کنترل.

ارتباط سه بخش اصلی کامپیوتر با مجموعه ای از سیم ها با نام باس (bus) انجام می شود.

- هر سه بخش به یک مجموعه سیم متصل می شود.
- هر دستگاه I/O یا هر خانه حافظه باید یک آدرس داشته باشد تا قابل تشخیص توسط CPU باشد.
- آدرس هر دستگاه باید یکتا باشد.
- CPU برای فراخوانی دستگاه دلخواه، آدرس باینری را آن را روی باس آدرس قرار می دهد.
- CPU از باس داده برای دریافت یا فرستادن داده به سمت دستگاه ها استفاده می کند.
- CPU از باس کنترل برای ارسال فرمان به دستگاه ها استفاده می کند (از قبیل خواندن، نوشتن و ...)
- هر چه باس داده بزرگتر باشد سرعت انتقال داده ها بیشتر شده و کارایی CPU بالاتر می رود. در عوض هزینه ساخت و پیچیدگی مدارها بیشتر می شود.
- میانگین اندازه باس داده در CPU های امروزی بین ۸ تا ۶۴ بیت است.
- باس داده خانواده AVR ۸ بیتی است. و باس داده PC CPU ها ۳۲ و ۶۴ بیت است.
- باس داده دو طرفه است: یعنی CPU می تواند هم بر روی آن داده گذاشته و هم از آن بخواند، اما نه همزمان و از طریق فرمان های روی باس کنترل جهت آن تعیین می شود.

خصوصیت های انواع باس

- هر چه باس آدرس بزرگتر باشد تعداد دستگاه های بیشتری قابل آدرس دهی است، و مخصوصاً مقدار حافظه بیشتری برای CPU قابل دسترسی است.
- x خط آدرس ⇔ تعداد حافظه قابل آدرس دهی = 2^x
- باس آدرس یک طرفه است: تنها CPU روی آن مقدار می نویسد. و از آن نمی خواند.
- CPU تعیین می کند که در هر لحظه کدام دستگاه به باس متصل است.
- همه CPU های General-purpose به صورت بایتی آدرس دهی می شوند. یعنی هر بایت از حافظه یک آدرس اختصاصی دارد. و این ربطی به اندازه باس داده ندارد.
- بنابراین می توان گفت که:
- x خط آدرس ⇔ حافظه قابل آدرس دهی = 2^x بایت.
- به CPU ی که x خط آدرس دارد، می توان 2^x بایت حافظه وصل کرد.
- همچنین حافظه ای که x خط آدرس دارد، 2^x بایت ظرفیت خواهد داشت.
- باس کنترل نیز یک طرفه است. (فرمان ها از سمت CPU صادر می شود.)

چند اصطلاح مهم در حوزه حافظه

- bit، nibble، byte، word: به ترتیب نشاندهده ۱، ۴، ۸ و ۱۶ بیت در فضای حافظه هستند.

کیلو بایت، مگابایت، گیگابایت، ترابایت			
Kilo byte (KB)	2^{10} byte	1024 byte	
Mega byte (MB)	2^{20} byte	1024 Kbyte	1048576 byte
Giga byte (GB)	2^{30} byte	1024 Mbyte	1.073×10^9 byte
Tera byte (TB)	2^{40} byte	1024 Gbyte	1.099×10^{12} byte

RAM: Random Access Memory.

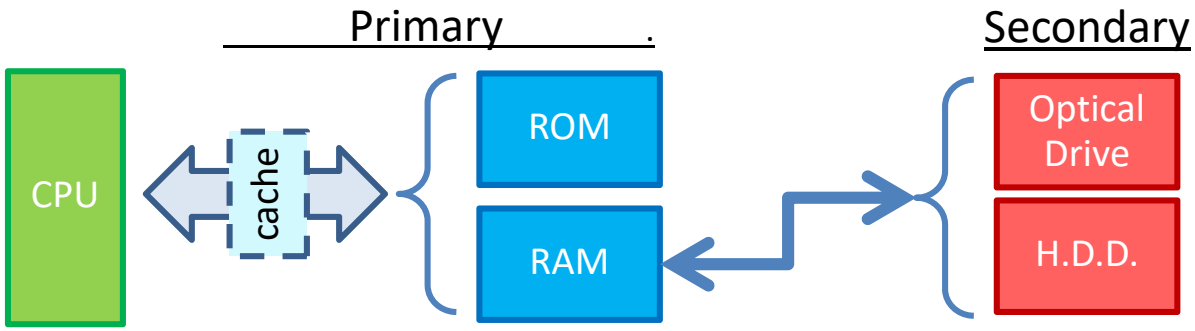
- ذخیره موقتی داده ها و برنامه ها
- خواندنی و نوشتنی
- پاک شدن داده با قطع منبع تغذیه. (volatile)

ROM: Read-only Memory.

- ذخیره داده های اساسی
- فقط خواندنی
- پاک نشدنی (non-volatile)

حافظه اصلی (اولیه)، حافظه فرعی (ثانویه)

- CPU فقط می تواند داده های موجود در RAM یا ROM را پردازش کند.
- داده های اساسی و تغییر ناپذیر در ROM دخیره می شود.
- برنامه های در حال اجرا که هر زمان در حال تغییراند و داده های آنها در RAM ذخیره می شوند.
- برنامه ها معمولاً از دیسک سخت (hard disk) به RAM منتقل می شود.
- CPU مستقیم برنامه ها را از روی دیسک سخت اجرا نمی کند، چون عملکرد دیسک سخت بسیار کندتر از عملکرد CPU است.
- دیسک سخت قابلیت نگه داری حجم بیشتری از داده را دارد (ظرفیت بالاتری دارد).
- معمولاً به دیسک سخت حافظه فرعی (ثانویه) secondary، و به RAM و ROM حافظه اصلی (اولیه) primary گفته می شود.



نحوه کارکرد CPU (معماری CPU)

- برنامه (مجموعه دستورهایی که کامپیوتر باید اجرا کند) در حافظه ذخیره می شود.
- وظیفه CPU فراخوانی یک به یک این دستورها و اجرای آن است.
- هر CPU برای اینکه بتواند دستورها را اجرا کند باید بخش های زیر را داشته باشد:

1. تعدادی register عمومی: آرایه هایی از واحد های حافظه برای نگه داری موقتی داده ها، و نگه داری آدرس ها. رجیسترهای ۸ تا ۶۴ بیتی با توجه به نوع CPU وجود دارد. هر چه رجیسترها بزرگ تر باشد کارایی CPU بالاتر می رود اما پیچیدگی و قیمت CPU بالاتر می رود. (خانواده AVR ۸ بیتی است.)
2. یک واحد ALU: انجام دستورهای حسابی مانند جمع، تفریق و ... و دستور های منطقی مانند AND، OR و....
3. رجیستر خاص به نام PC یا IP: آدرس دستور بعدی اجرایی را نگه داری می کند. با اجرای هر برنامه مقدار این رجیستر تغییر می کنند. محتوای این رجیستر برای فراخوانی دستور بعدی روی باس آدرس قرار می گیرد.
4. واحدی به نام instruction decoder: مدار ترکیبی یا ترتیبی که دستور خوانده شده از حافظه را تفسیر کرده و دستور های لازم را به واحد های CPU می فرستد.
5. واحد زمان بندی و کنترل: مداری ترتیبی برای کنترل بر روند اجرای برنامه.

نحوه کارکرد CPU (معماری CPU) (۲)

بخش های اصلی CPU

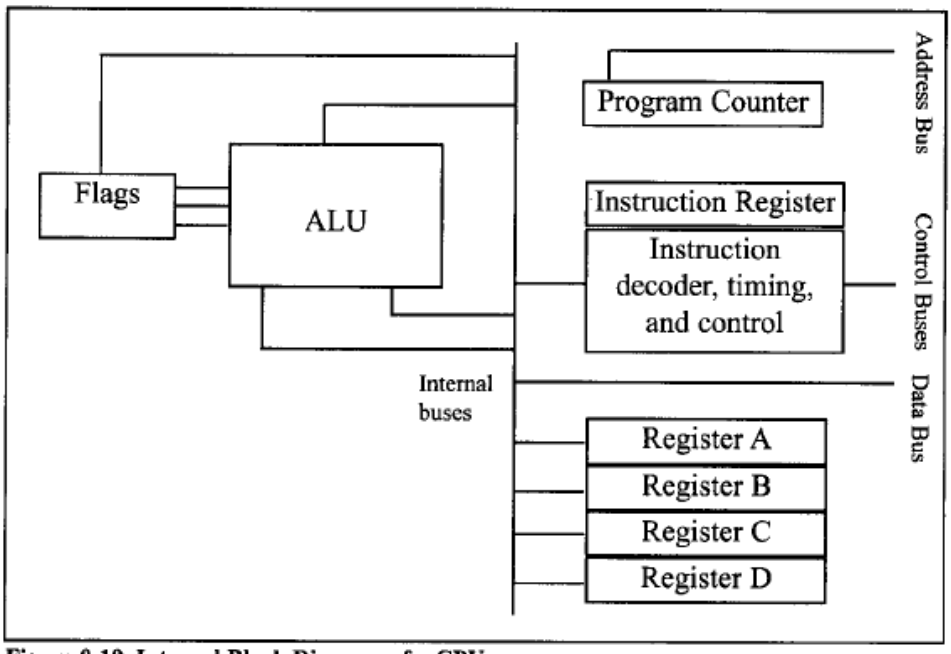


Figure 0-19. Internal Block Diagram of a CPU

ذخیره سازی اعداد و داده ها در کامپیوتر:

شمارش و سیستم های کدینگ

- نحوه ذخیره سازی اعداد در کامپیوتر:
- شمارش انسان: سیستم مبنای ۱۰ (decimal) یا دهدهی
  - ده سمبل مجزا برای شمارش به کار می رود: 0,1,..,9
  - ده انگشت دو دست ابزار شمارش بوده است.
  - بعد از ۹ به ۱۰ می رسیم یعنی سمبل مکان سمت راست به اولین مقدار (صفر) برمی گردد و مکان سمت چپ یکی افزایش می یابد.
- شمارش در کامپیوتر: مبنای ۲ (binary) یا دودویی
  - دو سمبل مجزا برای شمارش به کار می رود: 0,1
  - منطق خاموش یا روشن مبنای شمارش بوده.
  - نمایش بهتر: مبنای ۱۶ یا (hexadecimal).
  - به هر رقم در مبنای ۲، bit گفته می شود.
  - بعد از ۱ به ۱۰ می رسیم یعنی سمبل مکان سمت راست به اولین مقدار (صفر) برمی گردد و مکان سمت چپ یکی افزایش می یابد.

تبدیل عدد decimal به binary

- تقسیم متوالی تا رسیدن به صفر و ثبت باقیمانده ها.
- تقسیم بر ۲ را ادامه می دهیم تا خارج قسمت صفر شود.
- عدد تبدیل شده از باقیمانده ها تشکیل می شود. (به ترتیب معکوس)

Example 0-1

Convert 25_{10} to binary.

Solution:

	Quotient	Remainder	
25/2 =	12	1	LSB (least significant bit)
12/2 =	6	0	
6/2 =	3	0	
3/2 =	1	1	
1/2 =	0	1	MSB (most significant bit)

Therefore, $25_{10} = 11001_2$.

تبدیل عدد binary به decimal

- مکان رقم از راست نشان دهنده وزن آن در رقم است.
- هر رقم را در ارزش مکانی آن ضرب می کنیم.

740683_{10} =

3×10^0	=	3
8×10^1	=	80
6×10^2	=	600
0×10^3	=	0000
4×10^4	=	40000
7×10^5	=	700000
		740683

110101_2 =

				Decimal	Binary
1×2^0	=	1×1	=	1	1
0×2^1	=	0×2	=	0	00
1×2^2	=	1×4	=	4	100
0×2^3	=	0×8	=	0	0000
1×2^4	=	1×16	=	16	10000
1×2^5	=	1×32	=	32	100000
				53	110101

Weight:	16	8	4	2	1
Digits:	1	1	0	0	1
Sum:	16 +	8 +	0 +	0 +	$1 = 25_{10}$

سیستم شانزده شانزدهی یا hexadecimal

- سیستم شانزده شانزدهی یا hexadecimal یا به طور مختصر Hex
- روش نمایش مناسب تر برای binary (به دلیل تبدیل سراست بین این دو).
- ۱۶ سمبل متمایز برای هر رقم وجود دارد.

Example 0-4

Represent binary 100111110101 in hex.

Solution:
First the number is grouped into sets of 4 bits: 1001 1111 0101.
Then each group of 4 bits is replaced with its hex equivalent:

1001	1111	0101
9	F	5

Therefore, $100111110101_2 = 9F5$ hexadecimal.

Example 0-5

Convert hex 29B to binary.

Solution:

	2	9	B
29B =	0010	1001	1011

Dropping the leading zeros gives 1010011011.

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

تبدیل decimal به Hex و برعکس

- تبدیل decimal به Hex
  1. تبدیل به باینری و سپس تبدیل به Hex.
  2. تبدیل مستقیم به مبنای ۱۶ با تقسیم های متوالی و پیگیری باقیمانده ها.

(a) Convert 45_{10} to hex.

32	16	8	4	2	1	First, convert to binary.
1	0	1	1	0	1	32 + 8 + 4 + 1 = 45

$45_{10} = 0010\ 1101_2 = 2D$ hex

- تبدیل Hex به decimal
  1. تبدیل به باینری و سپس تبدیل به decimal.
  2. تبدیل مستقیم براساس ارزش مکانی.

(a) $6B2_{16} = 0110\ 1011\ 0010_2$

1024	512	256	128	64	32	16	8	4	2	1
1	1	0	1	0	1	1	0	0	1	0

$1024 + 512 + 128 + 32 + 16 + 2 = 1714_{10}$

جمع و تفریق باینری

Table 0-3: Binary Addition

A + B	Carry	Sum
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0

- جمع دو عدد باینری تک رقمی

	Binary	*Decimal*
	1101	13
+	1001	9
	10110	22

- جمع دو عدد باینری و معادل دهدهی آن
- تفریق به کمک مفهوم مکمل دو انجام می شود: a - b = a + (-b)
- مکمل دوی هر عدد = (مکمل یک + ۱)
- مکمل یک = برعکس کردن همه بیت ها

	10011101	binary number
	01100010	1's complement
+	1	
	01100011	2's complement

جمع و تفریق Hex

- روند عادی جمع انجام می شود، تنها تفاوت در تعداد سمبل ها است که ۱۶ است. و اگر جمع دو رقم از $۱۵_{۱۰}$ بیشتر شود carry منتقل می شود.

Perform hex addition: 23D9 + 94BE.

Solution:

	23D9	LSD:	9 + 14 = 23	23 – 16 = 7 with a carry
+	94BE		1 + 13 + 11 = 25	25 – 16 = 9 with a carry
	B897		1 + 3 + 4 = 8	
		MSD:	2 + 9 = B	

- تفریق نیز به روش قرض گرفتن از رقم پر ارزش تر انجام می شود.

Perform hex subtraction: 59F – 2B8.

Solution:

	59F	LSD:	8 from 15 = 7
–	2B8		11 from 25 (9 + 16) = 14 (E)
	2E7		2 from 4 (5 – 1) = 2

کد ASCII

- متن با تعریف کد در کامپیوتر ذخیره می شود.
- یکی از این کد ها کدی با نام ASCII است.
- به همه کاراکترها (از جمله حروف بزرگ و کوچک، علامت ها، ارقام، فرمان ها و ..) یک کد باینری تعلق می گیرد.
- کدینگ استاندارد نمایش کارکترها (۷ بیت)
- انتخاب کد هر کاراکتر هوشمندانه انجام شده است:
  - کد حروف بزرگ و کوچک تنها در بیت پنجم اختلاف دارد.
  - (a⇦110 0001 , A⇦100 0001)
  - ارقام بین کدهای ۳۰ تا ۳۹ Hex قرار گرفته اند.

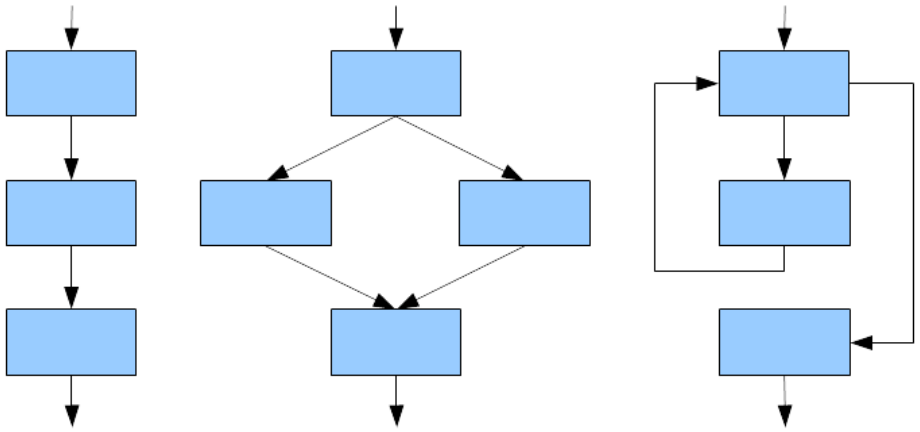
Hex	Symbol	Hex	Symbol
41	A	61	a
42	B	62	b
43	C	63	c
44	D	64	d
...	...	...	...
59	Y	79	y
5A	Z	7A	z

Figure 0-1. Selected ASCII Codes

اصول برنامه نویسی برای کامپیوتر

- برنامه کامپیوتری تشکیل شده از دستورها است که مستقل و پیاپی اجرا می شوند.
- ساختار و عملکرد دستورها در هر زبان برنامه ریزی تعریف شده است.
- در برنامه ریزی روند های زیر قابل استفاده است.

اجرای پی در پی تصمیم گیری (برمبنای شرط) حلقه (تکرار بخشی از برنامه)



معرفی میکروکنترلرها

بخش دوم

میکروکنترلرها در برابر ریزپردازنده های همه منظوره

میکروکنترلرها

- مثال ها: خانواده 8051، خانواده AVR، خانوداه PIC، خانواده STM32 و سایر میکروکنترلرهای مبتنی بر معماری ARM
- با مقادیر و تعداد مشخص و تغییر ناپذیری از RAM، ROM، و پایه های I/Oو تایمر.
- قابلیت راه اندازی به تنهایی و بدون نیاز به قطعات اضافی.
- پیاده سازی ساده و ارزان.
- مشخصات تغییر ناپذیر.
- راه اندازی سریع مدار.
- کارایی پردازشی متوسط.

پردازنده های همه منظوره

- مثال ها: خانواده x86 اینتل (8086، 80286، 80386، 80486، Pentium، و...) خانواده موتورولا، PowerPC ARM-Cortex A
- بدون RAM، ROM، و پایه های I/Oو تایمر، روی چیپ پروسسور.
- به تنهایی کاربردی نیستند و قطعات اضافی باید در کنار آنها قرار گیرد.
- مدار ساخته شده آنها بزرگتر و گران تر خواهد بود.
- انعطاف پذیری در کاربرد های مختلف.
- زمان بر بودن طراحی مدار.
- کارایی پردازشی بالا.

میکروکنترلر در برابر ریزپردازنده های همه منظوره (۲)

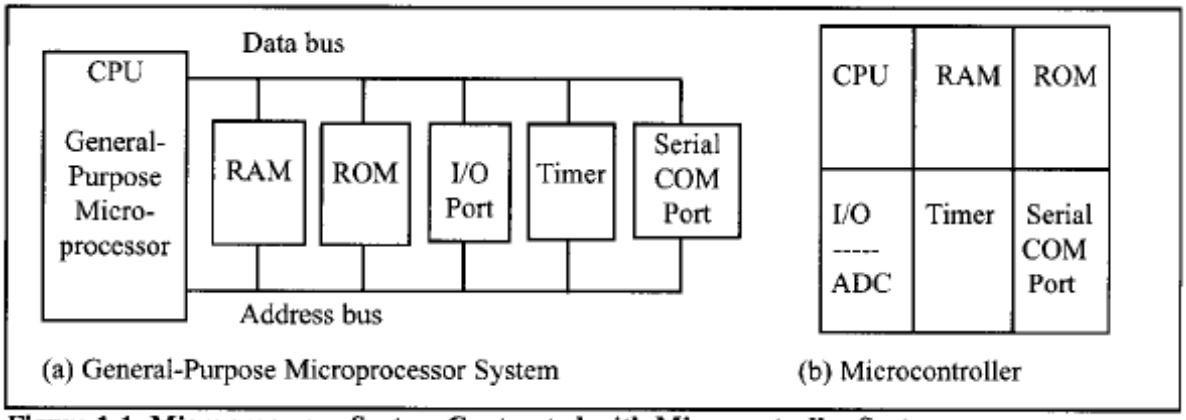


Figure 1-1. Microprocessor System Contrasted with Microcontroller System

کاربرد های میکروکنترلرها

- در جایی که کوچکتر بودن، توان مصرفی، قیمت، و طراحی سریع مدار خیلی مهم تر از قدرت پردازشی باشد، میکروکنترلرها مطرح می شوند.
- در embedded system ها کاربرد دارند:
  - سسیستمی با یک پردازنده داخلی که معمولاً یک وظیفه را انجام می دهد.
  - برنامه معمولاً روی ROM قرار می گیرد و غیر قابل تغییر است.
  - معمولاً سیستم عاملی برای اجرای نرم افزار های متعدد وجود ندارد.
  - بعضی از کاربردها:

انتخاب میکروکنترلر

- خانواده های متداول میکروکنترلر: (موجود به صورت تجاری، ۸ بیتی)
  - 68HC08/68HC11 از Freescale Semiconductor، 8051 از Intel، Z8 از Zilog، PIC و AVR از Microchip Technology.
- بعضی از این تولید کننده ها میکروکترلر های ۱۶ و ۳۲ بیتی نیز تولید کرده اند.
- هر خانواده مجموعه دستورهای مربوط به خود را دارند. بنابرین برنامه های آنها با یکدیگر منطبق نیستند، اما اصول کارکرد مشابه ای دارند.
- معیارهای انتخاب میکروکنترلر چیست؟

1. تطابق با نیازهای محاسباتی کاربرد مورد نظر به گونه ای که کارایی مناسب به همراه قیمت مناسب وجود داشته باشد.
2. وجود نرم افزارها و سخت افزاهای مناسب کامپایلر، اسمبلر، دیباگر، امولاتور، پروگرامر.

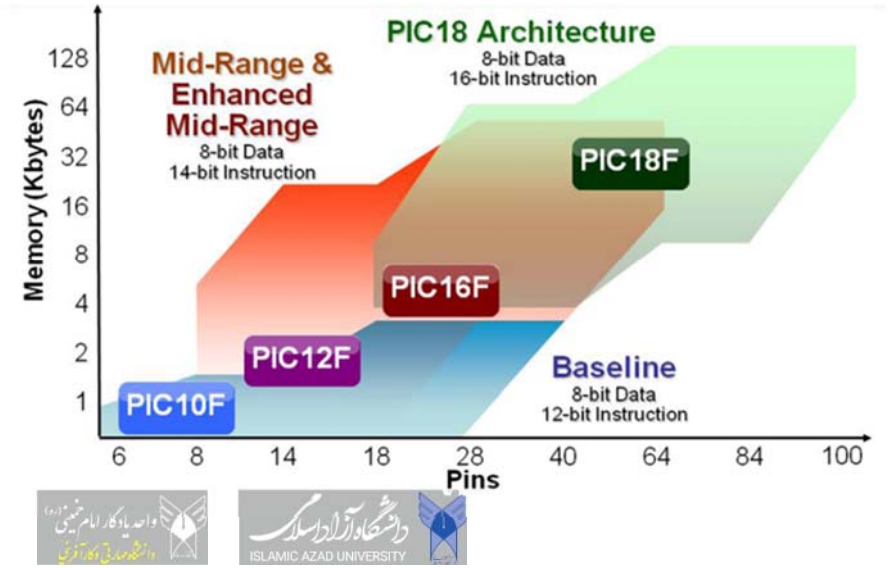
خانواده های مختلف میکروکنترلرهای ۸ بیتی

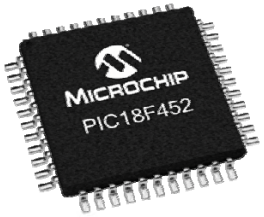
- خانواده 8051
  - معرفی توسط Intel در سال ۱۹۸۰ و ادامه توسط سایر تولید کنندگان

- خانواده Z80
  - معرفی توسط شرکت Zilog سال ۱۹۷۴، بسیار پرکاربرد در اولین سالهای معرفی میکروکنترلر

خانواده های مختلف میکروکنترلرهای ۸ بیتی (ادامه)

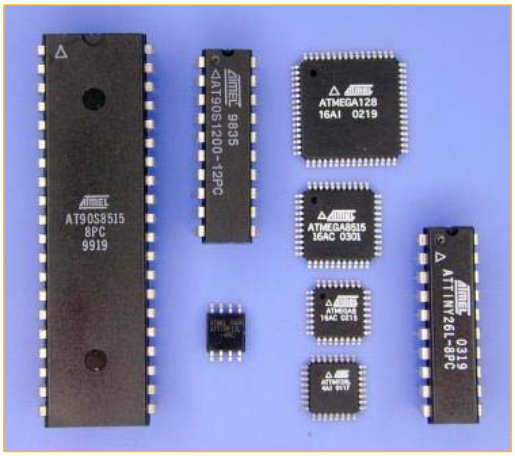
- خانواده PIC
  - متعلق به شرکت Microchip اولین نمونه در سال ۱۹۷۶
  - توسعه یافته و بسیار پرکاربرد در استفاده های امروزی



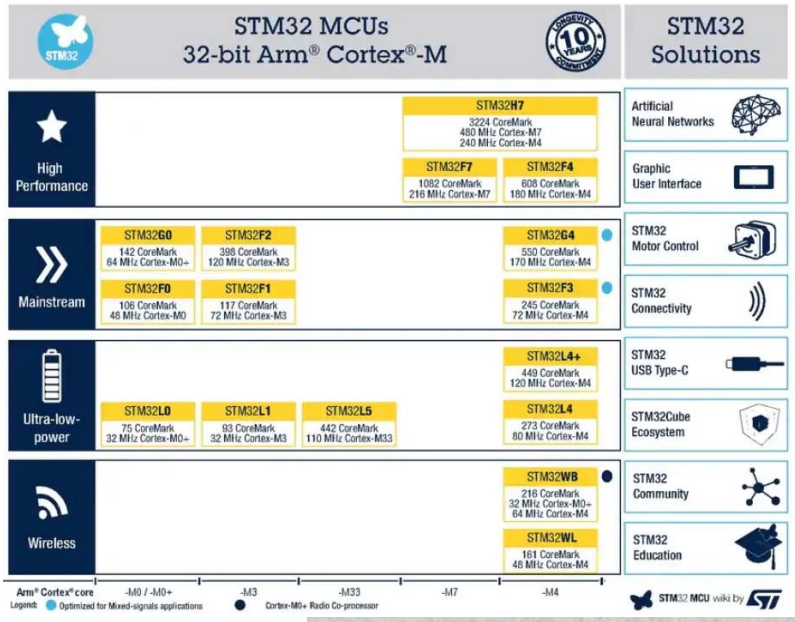


خانواده های مختلف میکروکنترلرهای ۸ بیتی (ادامه)

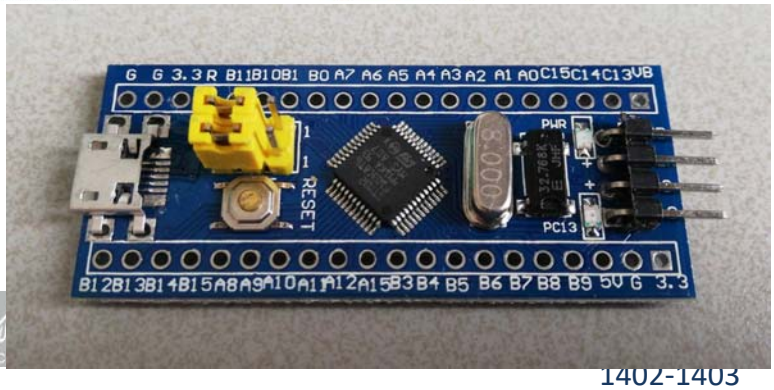
- خانواده AVR
  - اولین نمونه در سال ۱۹۹۶.
  - داده ها (رجیسترها) ۸ بیتی هستند.
  - ۱۰۰٪ تطابق سخت افزاری وجود ندارد مخصوصاً بین خانواده های مختلف (مثل tiny و mega).
- خانواده های AVR:
  - classic, Tiny, Mega, Special purpose
  - خانواده جدید: Xmega.
- تمرکز ما بر Mega خواهد بود، با یادگیری Mega تسلط بر بقیه نیز مشکل نخواهد بود.



خانواده های مختلف میکروکنترلرهای ۳۲ بیتی



- خانواده های معرفی شده بر مبنای معماری ARM Cortex M
  - خانواده STM32 از شرکت ST



خانواده های مختلف میکروکنترلرهای ۳۲ بیتی (ادامه)

- سایر خانواده های معرفی شده بر مبنای معماری ARM Cortex M
  - خانواده LPC از شرکت NXP
  - خانواده SAM از Atmel
- خانواده AVR32 از Atmel
- خانواده PIC32 از Microchip
- و...

عوامل موثر در انتخاب میکروکنترلرها

- اندازه باس داده: ابتدا باید دید که کاربرد مورد نظر با کدام یک از میکروکنترلرهای ۸ یا ۱۶ یا ۳۲ بیتی قابل پیاده سازی است.
- سرعت اجرای برنامه: حداکثر فرکانس کارکرد میکروکنترلر چه مقدار است؟
- بسته بندی: DIP و QFP

QFP

DIP

- توان مصرفی: مدل های کم مصرف دارای پسوند L هستند.
- مقدار RAM و ROM: پس از انتخاب، مقدار اینها قابل تغییر نیست.
- تعداد پایه های I/O: پس از انتخاب، قابل تغییر نیست.
- قابلیت انطباق پذیری میان مدل های مختلف از یک خانواده. (سخت افزاری و نرم افزاری)
- قیمت آن چقدر است؟ این عامل معمولاً عوامل بالا را محدود می کند.

ویژگی های خانواده AVR

- ۸ بیتی، میکروکنترلر تک چیپ (همه بخش ها روی یک IC). دارای دو باس داده مجزا: برنامه و داده.
- ویژگی های پایه (ویژگی هایی که در همه مدل های این خانواده وجود دارد): دارای ROM از نوع flash روی چیپ، RAM داده، EEPROM داده، تایمر، پایه های I/O، وقفه های داخلی و خارجی.
- ویژگی های اضافه (شامل بعضی از مدل ها): ارتباط های سری USART، SPI، I2C، CAN، USB.

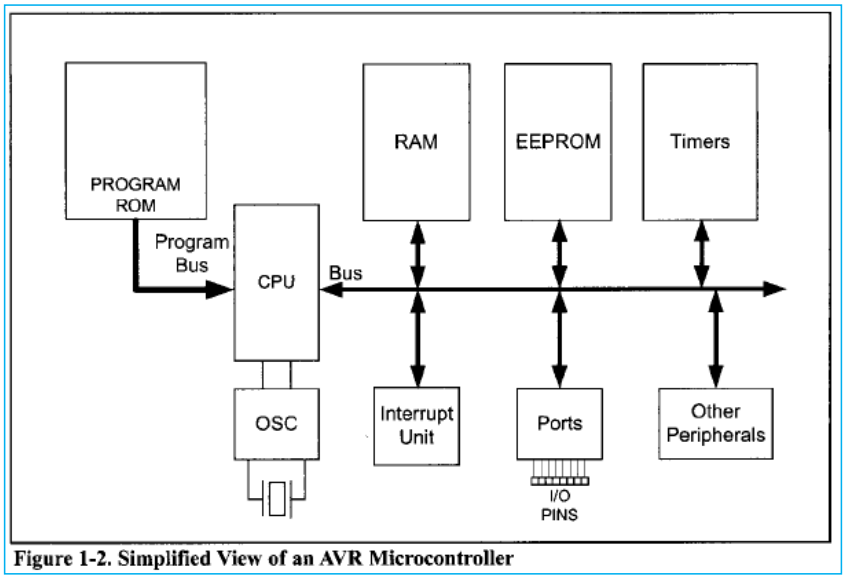
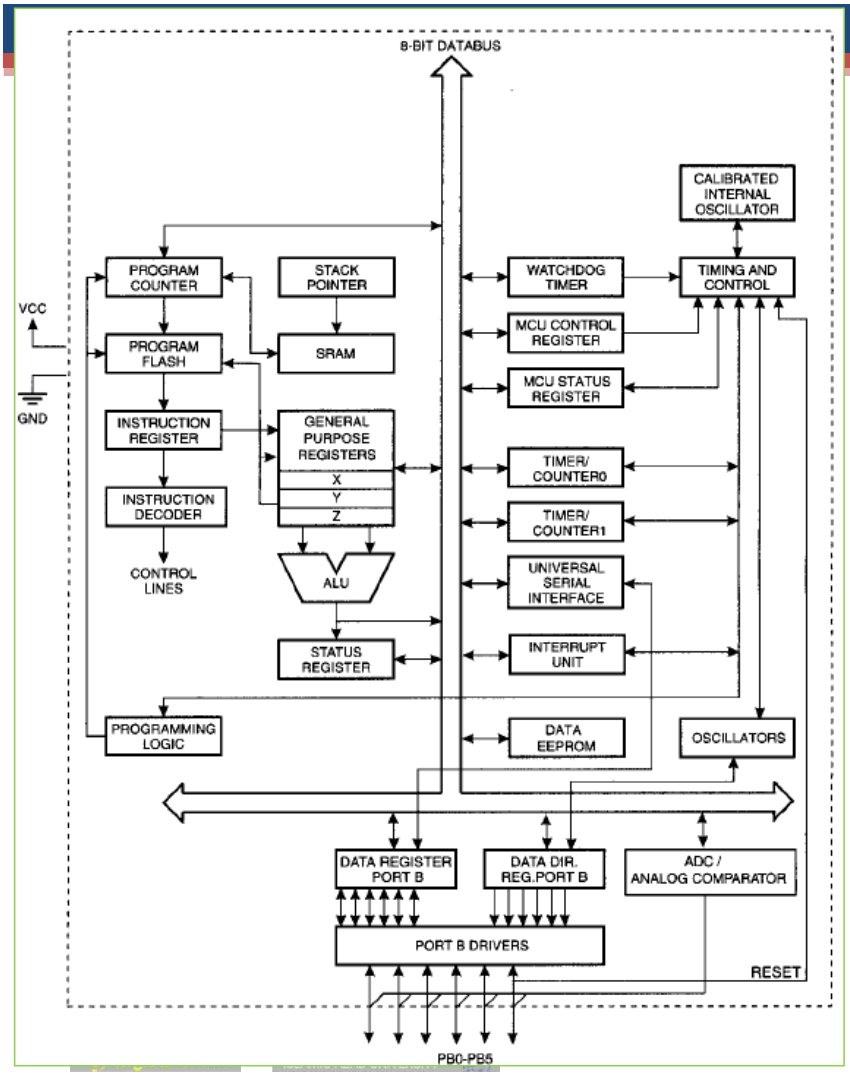


Figure 1-2. Simplified View of an AVR Microcontroller

نمای خلاصه شده درون میکروکنترلرهای AVR

ساختار کامل AVR (مثال ۱: ATtiny25)



بخش های آشنا:

۱- باس داده ۸ بیتی
- همه بخش ها به باس داده وصل هستند..
- باس داده به بیرون IC متصل نیست بنابرین نمی توان قطعه ای به این کامپیوتر اضافه کرد.

۲- حافظه ها
- RAM از نوع static
- Program Flash: ROM برای نگه داری برنامه
- رجیسترهای عمومی

۳- رجیستر PC
- شمارنده سطر برنامه.

۴- بخش ID
- مفسر دستور و تولید فرمان ها (control lines)
- control lines: همان باس کنترل است.

۵- ALU: بخش محاسبات ریاضی و منطقی.

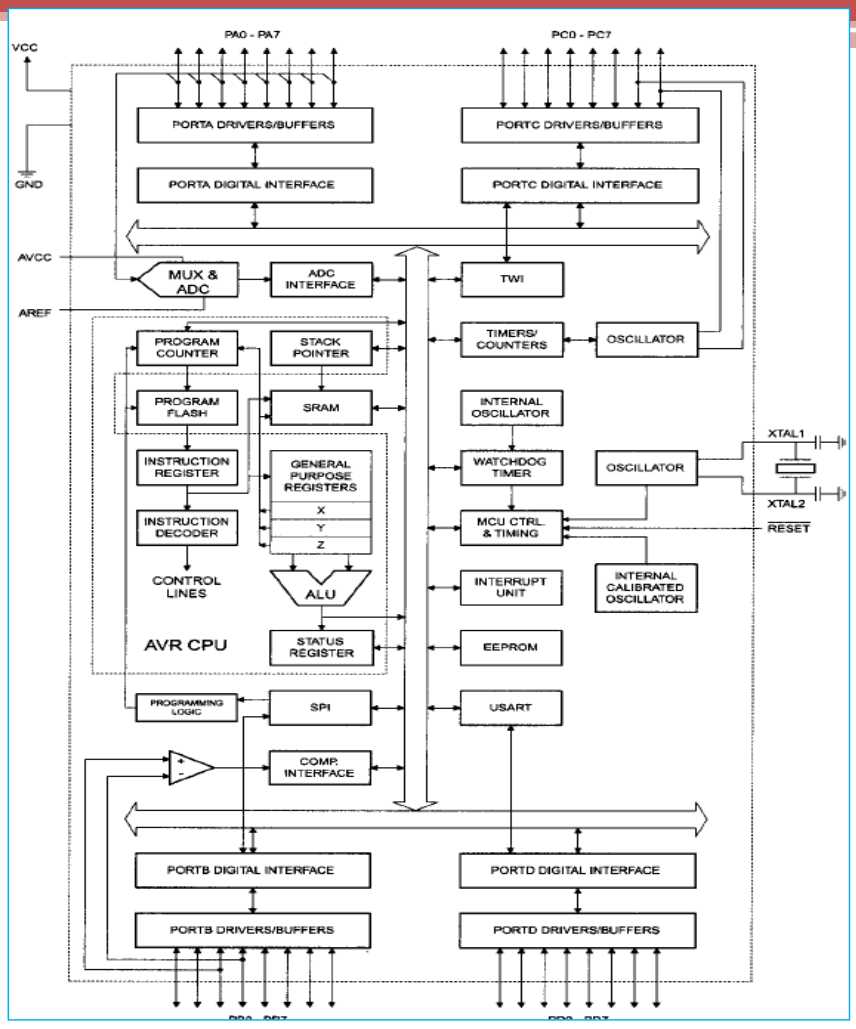
۶- واحد Timing and Control
- زمان بندی و کنترل اجرای برنامه.

۷- رجیستر Status
- رجیستر وضعیت یا همان فلگ ها.

۸- اسیلاتور ها
- تولید کننده فرکانس کلاک میکروکنترلر

۹- روی این شکل باس آدرس رسم نشده است.

ساختار AVR (مثال ۲: ATmega32)



ساختار اصلی این مدل پیشرفته تر تفاوتی با مدل ساده صفحه قبل ندارد و تنها امکانات بیشتری به آن اضافه شده است.

مانند افزایش پایه های I/O، پورت های سری، مقایسه کننده آنالوگ و...

ویژگی های سخت افزاری AVR

- فضای حافظه برنامه قابل آدرس دهی معماری 8M است. ولی ROM موجود روی AVR های فعلی از ۱ تا ۲۵۶ Kbyte است. (flash) (اولین عدد توان ۲ در شماره مدل نشان دهنده میزان ROM آن دستگاه بر حسب Kbyte است.)
- حداکثر RAM قابل آدرس دهی ۶۴ Kbyte است.
- حداکثر SRAM داخلی ۸ و ۱۶ Kbyte.
- در بعضی از مدل ها امکان افزایش RAM خارجی وجود دارد.
- فضای آدرس داده به سه بخش SFR (I/O)، رجیسترهای عمومی و SRAM داخلی تقسیم شده است.
- فضای کوچکتری نیز به EEPROM اختصاص داده شده است.
- پایه های I/O: ۳ تا ۸۶ عدد. (وابسته به خانواده، مدل و بسته بندی)
- بسته بندی ها: ۸ پایه تا ۱۰۰ پایه.
- تایمرها: از یک تا ۶ تا + watchdog.
- مبدل آنالوگ به دیجیتال (ADC): ۱۰ بیتی، حداکثر تا ۱۶ کانال ورودی.
- پورت های سری: USART, I2C, SPI, USB,CAN.

خانواده های مختلف AVR

- classic: اولین مجموعه AVR، با مدل های جدید جایگزین شده است.

Table 1-2: Some Members of the Classic Family

Part Num.	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
AT90S2313	2K	128	128	15	0	2	SOIC20, PDIP20
AT90S2323	2K	128	128	3	0	1	SOIC8, PDIP8
AT90S4433	4K	128	256	20	6	2	TQFP32, PDIP28

Notes:
1. All ROM, RAM, and EEPROM memories are in bytes.
2. Data RAM (general-purpose RAM) is the amount of RAM available for data manipulation (scratch pad) in addition to the register space.

- tiny: بسته بندی های کوچکتر، دستور های کمتر، پایه های کمتر، امکانات جانبی کمتر، قیمت کمتر، مصرف انرژی کمتر.

- Program memory: 1K to 8K bytes
- Package: 8 to 28 pins
- Limited peripheral set
- Limited instruction set: The instruction sets are limited. For example, some of them do not have the multiply instruction.

خانواده های مختلف AVR (۲)

- چند مدل از خانواده tiny:

Part Num.	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
ATtiny13	1K	64	64	6	4	1	SOIC8, PDIP8
ATtiny25	2K	128	128	6	4	2	SOIC8, PDIP8
ATtiny44	4K	256	256	12	8	2	SOIC14, PDIP14
ATtiny84	8K	512	512	12	8	2	SOIC14, PDIP14

Table 1-4: Some Members of the Tiny Family

- خانواده mega: دستورهای قدرتمند، قابلیت های مدارهای واسطه کامل،
  - Program memory: 4K to 256K bytes
  - Package: 28 to 100 pins
  - Extensive peripheral set
  - Extended instruction set: They have rich instruction sets.

Table 1-3: Some Members of the ATmega Family

Part Num.	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
ATmega8	8K	1K	0.5K	23	8	3	TQFP32, PDIP28
ATmega16	16K	1K	0.5K	32	8	3	TQFP44, PDIP40
ATmega32	32K	2K	1K	32	8	3	TQFP44, PDIP40
ATmega64	64K	4K	2K	54	8	4	TQFP64, MLF64
ATmega1280	128K	8K	4K	86	16	6	TQFP100, CBGA

خانواده های مختلف AVR (۳)

- Special Purpose: دارای مدارهای جانبی ویژه (علاوه بر جدول زیر مدل های با قابلیت zigbee، Ethernet controller نیز وجود دارد.)

Table 1-5: Some Members of the Special Purpose Family

Part Num.	Code ROM	Data RAM	Data EEPROM	Max I/O pins	Special Capabilities	Timers	Pin numbers & Package
AT90CAN128	128K	4K	4K	53	CAN	4	LQFP64
AT90USB1287	128K	8K	4K	48	USB Host	4	TQFP64
AT90PWM216	16K	1K	0.5K	19	Advanced PWM	2	SOIC24
ATmega169	16K	1K	0.5K	54	LCD	3	TQFP64, MLF64

- خانواده Xmega: در دو نسخه ۸ و ۱۶بیتی ارائه شده است.
  - DAC، و ADC ۱۲ بیتی.
  - عملکرد real-time: زمان پاسخگویی مدارهای جانبی قابل پیشینی است.
  - کارکرد با ولتاژ ۱/۶ ولت.
  - USB Host، کتابخانه های نرم افزاری تکمیل شده از جمله برای صفحات لمسی.

مقایسه ای میان سه میکروکنترلر از سه سازنده

Table 1-7: Comparison of 8051, PIC18 Family, and AVR (40-pin package)

Feature	8052	PIC18F452	ATmega32
Program ROM	8K	32K	32K
Data RAM (maximum space)	256 bytes	2K	2K
EEPROM	0 bytes	256 bytes	1K
Timers	3	4	3
I/O pins	32	35	32

برنامه نویسی به زبان C

بخش سوم

ساختار برنامه به زبان C

- متن برنامه اصلی زبان C در بدنه ()main نوشته می شود.
- در زبان C شروع و پایان هر بخش با {} ایجاد می شود.
- در پایان هر دستور که مطمئن هستیم به پایان رسیده است باید ; قرار داده شود.
- برنامه خط به خط اجرا می شود.
- اگر بخشی از برنامه که با {} مشخص شده است فقط شامل یک دستور باشد می توان {} را حذف کرد
- اولین دستور در ()main اولین خط اجرایی برنامه است.

```
main()
{
   -----;
   -----;
   -----;
    .
    .
    .
}
```


نمونه هایی از برنامه های زبان C با ساختار درست

- جلوتر نوشته شدن دستور ها در زبان C تاثیری در ساختار بخش ها ندارد.
- قرار دادن space و enter بین کلمات تآثیری در برنامه ندارد

```
main()
{
   int i,j;
   i=0;
   while (i<5)
   {
        i=i+1;
        j=i*2;
   }
}
```

```
main()
{
   int i,j;
   i=0;
   while (i<5)
        i=i+1;
}
```

```
main()
{
   int i,j;
   i=0;
   while (i<5)
        i=i+1;
   j=i*2;
}
```

```
main()
{
   int i,j; i=0;
   while (i<5) i=i+1;
}
```

نمونه هایی از برنامه هایی با ساختار نادرست

- زوج {} نامتقارن
- فراموش شدن ;
- گذاشتن اشتباه ;

```
main()
{
   int i,j;
   i=0;
   while (i<5)
   {
        i=i+1;
        j=i*2;
}
```

```
main()
{
   int i,j;
   i=0;
   while (i<5)
   {
        i=i+1
        j=i*2
   }
}
```

```
main()
{
   int i,j;
   i=0;
   while (i<5);
   {
        i=i+1;
        j=i*2;
   }
}
```


متغیرها در زبان C

- برای نگه داری اعداد صحیح و اعشاری و کاراکترها در زبان C باید ابتدا متغیر ها تعریف شوند.
- به طور معمول متغیرهای محلی در خط اول main تعریف می شوند.
- انواع متغیرهای از پیش تعریف شده در زبان C:

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	–32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	–2,147,483,648 to +2,147,483,648
float	32-bit	±1.175e-38 to ±3.402e38
double	32-bit	±1.175e-38 to ±3.402e38

روش درست تعریف و مقداردهی متغیرها

- نام متغیرها باید با حروف لاتین یا _ شروع شود. در حروف بعدی اعداد هم مجاز هستند.
- می توان در هنگام تعریف، مقدار اولیه هم برای متغیرها تعریف کرد.
- چند تعریف متغیر و مقدار دهی درست:

```
main()
{
 int i1,i2;
 char k=12;
 char key='D';
 long n= -230490;
 unsigned int x= 123;
 float F1= 23.24;
 double F2= -12.34;
}
```


عملگرهای کاربردی زبان C

- عملگرهای زبان C:

عملگر	مفهوم
!	NOT منطقی (تک بیتی)
~	مکمل یک
+	مثبت کردن مقدار
-	منفی کردن مقدار
++	افزودن یک
--	کاهش یک

ریاضی

عملگر	مفهوم
*	ضرب
/	تقسیم صحیح
%	باقیمانده تقسیم
+	جمع
-	تفریق

- چند مثال از کاربرد عملگرها در زبان C

عملگرهای کاربردی زبان C (۲)

- مقایسه ای (قابل استفاده در شرط ها)

عملگر	مفهوم
<	کوچکتر
<=	کوچکتر مساوی
>	بزرگتر
>=	بزرگتر مساوی
==	برابر
!=	مخالف

- بیتی و منطقی

عملگر	مفهوم
&	AND بیتی
|	OR بیتی
^	XOR بیتی
||	OR منطقی (تک بیت)
&&	AND منطقی (تک بیت)

- مثال هایی از کاربرد این عملگرها

تصمیم گیری در زبان C

- ابزار اصلی تصمیم گیری در زبان C دستور if (و else) است.
- شرط دستور if فقط یکبار بررسی می شود.
- نوشتن else اختیاری است.

```
if (شرط)
{
    دستورهایی که در حالت درستی شرط اجرا میشوند
}
else
{
    دستورهایی که در حالت نادرستی شرط اجرا میشوند
}
```

- چند مثال از عملکرد if

حلقه ها در زبان C: حلقه شرطی

```
while (شرط)
{
    ...;
    ...;
    ...;
    .
    .
}
```

```
main()
{
int j=0,i=10;
while (i>5)
{
    i--;
    j=j+i;
}
}
```

- حلقه شرطی while:
- شرط حلقه بررسی می شود، اگر شرط درست باشد، دستورهای داخل حلقه یکبار اجرا می شود. سپس دوباره شرط بررسی می شود و این روند تکرار می شود.
- اگر شرط از همان ابتدا درست نباشد. دستورهای حلقه هیچگاه اجرا نمی شود.
- اگر شرط همیشه درست باشد. حلقه بی پایان می شود. و هیچگاه از حلقه خارج نمی شویم.
- اگر به جای شرط عدد گذاشته شود. هر عدد غیر صفر درست محسوب می شود.

- چند مثال از حلقه while.

حلقه ها در زبان C: حلقه با تعداد تکرار مشخص

- اگر بخواهیم بخشی از برنامه را با تعداد مشخص تکرار کنیم از حلقه for استفاده می کنیم.

```
main()
{
for (گام حلقه ;شرط ادامه ;مقداردهی اولیه)
{
    ...;
    ...;
    .
    .
}
}
```

- ساختار حلقه for:

- بررسی چند مثال با حلقه for.

مثال از حلقه for

- محاسبه میانگین مقدار اعداد از یک تا n
- برنامه برای دو حالت محاسبه صحیح و اعشاری نوشته شده است.

```
main()
{
int i,sum=0,n=10,m;
float mean,fsum,fn;
for (i=1;i<=n;i++)
{
   sum=sum+i;
}
m= sum/n;
fsum=sum; fn=n;
mean = fsum/fn;
}
```


آرایه ها در زبان C

- آرایه ها برداری از متغیرهای همنوع هستند که با شماره قابل دسترسی هستند.
  - آرایه ای از متغیرها را می توان با قرار دادن [] جلوی نام متغیر تعریف کرد.
  - دسترسی به هر عضو با مقدار دهی داخل [] انجام می شود، مقدار داخل [] می تواند مقدار ثابت و یا متغیر باشد، این مقدار باید صحیح، و غیر منفی باشد.
  - به آرایه ای از char ها string می گوییم که با "" مقدار دهی می شود.
  - char تنها را ' ' نمایش می دهیم که معادل کد ASCII آن است.

```
int n[20];
int m[]={2,5,8};
char k[]={4,7};
char q[]="abcd";
float f[]={12.4,4.01,3.5};
```

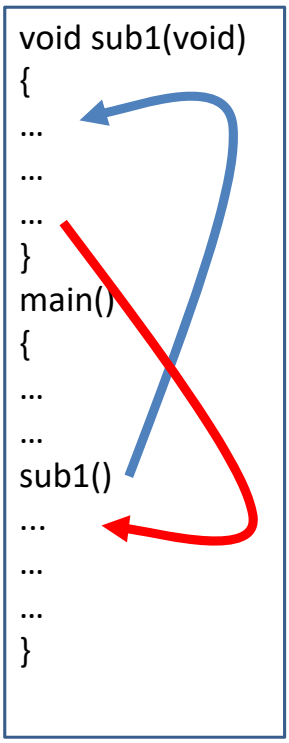
مثال از تعریف و دسترسی به آرایه ها

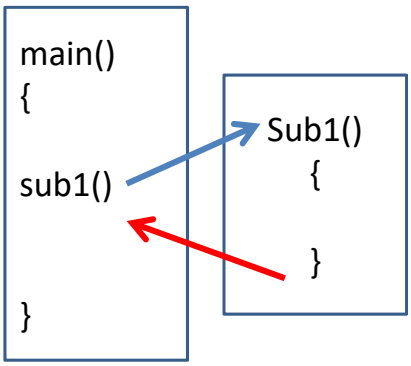
- برنامه مقابل توان های دوی اعداد ۰ تا ۹ را در آرایه ar2 ذخیره می کند.
- همچنین محتوای دو string را با هم مقایسه می کند و نتیجه را در آرایه ar1 ذخیره می کند.

```
main()
{
int i;
int ar1[]={1,4,5};
int ar2[10];
char St1[]="acb";
char st2[]= {'a','b','c'};
for (i=0;i<=9;i++)
    ar2[i]=i*i;
for (i=0;i<=2;i++)
    if (st1[i]==st2[i])
        ar1[i]=1;
    else
        ar1[i]=0;
}
```


تابع (function)

- در زبان C می توان بخشی از برنامه را جداگانه تعریف کرد.
- می توان این بخش را از قسمت های مختلف برنامه فراخوانی کرد.
- این بخش تابع (function) یا زیربرنامه نام دارد.
- تعریف تابع‌ها می تواند قبل یا بعد از main انجام شود.
- متغیرهای درون تابع مستقل هستند و به طور جداگانه باید تعریف شوند.





```
long power(int x,y)
{
   int i;
   long p=1;
   for (i=1;i<=y,i++)
   {
       p=p*x;
   }
   return p;
}
main()
{
   int a=2,b=15;
   long c,d;
   c=power(a,b);
   d=power(3,9);
}
```

ورودی‌ها و خروجی‌های تابع

- تابع می توانند یک یا چند خروجی و یک خروجی داشته باشد.
- ورودی‌ها در بین پرانتز جلوی نام تابع تعریف می‌شوند
- در مثال روبرو تابعی با دو ورودی و یک خروجی داریم.
- دو ورودی از نوع int و خروجی از نوع long است.
- تابع برای محاسبه توان نوشته شده است.
- در انتها ۲ به توان ۱۵ و ۳ به توان ۹ محاسبه شده است و در دو متغیر c و d ذخیره شده است.

برنامه نویسی برای میکروکنترلر

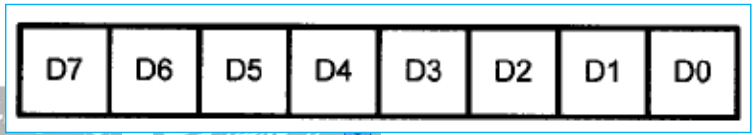
برنامه ریزی سخت افزار

بخش چهارم

ساختار میکروکنترلر AVR

رجیسترهای همه منظوره (General Purpose) در AVR

- هر کامپیوتری دارای یک زبان ماشین است. در این زبان به ازای هر امکان سخت افزاری آن کامپیوتر یک دستور قرار داده شده است. نحوه برنامه نویسی در این زبان کمی با برنامه نویسی به زبان های C یا Basic متفاوت است.
- برای انجام دستورهای حسابی و منطقی داده ها باید به رجیسترهای همه منظوره (GPR) منتقل شوند.
- رجیسترها مکان هایی برای نگه داری داده هستند (خارج از RAM) و همه ۸ بیتی هستتند.
- AVR دارای ۳۲ رجیستر GPR است به نام R0 تا R31.
- برای انجام دستورهای ALU، داده ها باید به این رجیسترها منتقل شوند.
- داده های بزرگتر از ۸ بیت باید به قسمت های ۸ بیتی شکسته شده و با عملیات ۸ بیتی پردازش شوند.
- این رجیسترها در پایین ترین آدرس فضای حافظه قرار گرفته اند. (از آدرس 0x00 تا 0x1F)



ساختار GPR ها و ALU

- با توجه به شکل روبرو همه عملیات ریاضی و منطقی باید بین دو رجیستر انجام شود.
- نتیجه محاسبه دوباره باید درون یک رجیستر ذخیره شود.
- بعد از اجرای هر دستور ریاضی یا منطقی، مشخصه هایی از نتیجه به دست آمده در رجیستری به نام status register ثبت می شود.

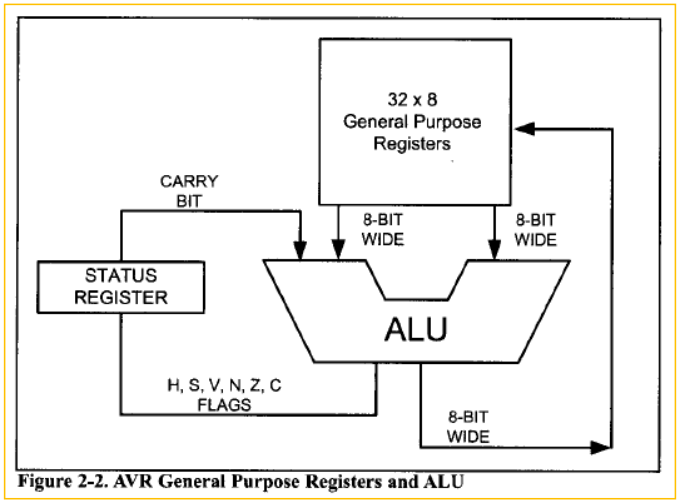


Figure 2-2. AVR General Purpose Registers and ALU

فضای حافظه داده در AVR

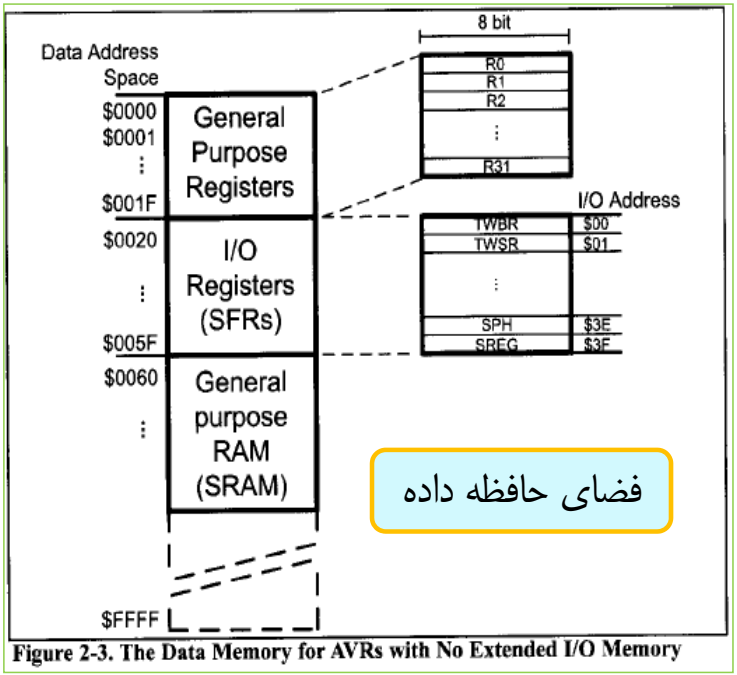


Figure 2-3. The Data Memory for AVRs with No Extended I/O Memory

- دو فضای حافظه وجود دارد: داده، برنامه (کد)
- خود فضای داده سه بخش است: فضای GPR، فضای رجیسترهای I/O، و فضای SRAM
- در مدل های کوچک تر به فضای رجیسترهای I/O، ۶۴ بایت تخصیص داده شده، در مدل های بزرگ تر فضای بیشتری تخصیص داده شده است.
- رجیستر های I/O یا SFR ها برای تنظیم بخش های مختلف میکروکنترلر به کار می روند.
- هر یک از رجیسترهای SFR دارای اسم مخصوص به خود است.

رجیسترهای SFR (I/O)

- این رجیسترها هر یک وظیفه کنترل و تنظیم بخشی از دستگاه های جانبی میکرو کنترلر را به عهده دارند.

فهرست رجیستر های I/O در AVR

Address Mem.	Address I/O	Name
$20	$00	TWBR
$21	$01	TWSR
$22	$02	TWAR
$23	$03	TWDR
$24	$04	ADCL
$25	$05	ADCH
$26	$06	ADCSRA
$27	$07	ADMUX
$28	$08	ACSR
$29	$09	UBRRL
$2A	$0A	UCSRB
$2B	$0B	UCSRA
$2C	$0C	UDR
$2D	$0D	SPCR
$2E	$0E	SPSR
$2F	$0F	SPDR
$30	$10	PIND
$31	$11	DDRD
$32	$12	PORTD
$33	$13	PINC
$34	$14	DDRC
$35	$15	PORTC

Address Mem.	Address I/O	Name
$36	$16	PINB
$37	$17	DDRB
$38	$18	PORTB
$39	$19	PINA
$3A	$1A	DDRA
$3B	$1B	PORTA
$3C	$1C	EECR
$3D	$1D	EEDR
$3E	$1E	EEARL
$3F	$1F	EEARH
$40	$20	UBRRC
		UBRRH
$41	$21	WDTCR
$42	$22	ASSR
$43	$23	OCR2
$44	$24	TCNT2
$45	$25	TCCR2
$46	$26	ICR1L
$47	$27	ICR1H
$48	$28	OCR1BL
$49	$29	OCR1BH
$4A	$2A	OCR1AL

Address Mem.	Address I/O	Name
$4B	$2B	OCR1AH
$4C	$2C	TCNT1L
$4D	$2D	TCNT1H
$4E	$2E	TCCR1B
$4F	$2F	TCCR1A
$50	$30	SFIOR
$51	$31	OCDR
		OSCCAL
$52	$32	TCNT0
$53	$33	TCCR0
$54	$34	MCUCSR
$55	$35	MCUCR
$56	$36	TWCR
$57	$37	SPMCR
$58	$38	TIFR
$59	$39	TIMSK
$5A	$3A	GIFR
$5B	$3B	GICR
$5C	$3C	OCR0
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

Note: Although memory address $20-$5F is set aside for I /O registers (SFR) we can access them as I /O locations with addresses starting at $00.

Figure 2-7. I/O Registers of the ATmega32 and Their Data Memory Address Locations

برنامه ریزی پورت های ورودی/خروجی

- ATmega32 ۳۲ پایه قابل برنامه ریزی دارد. (برنامه ریز می تواند نقش این پایه ها را مشخص کند و به کمک آنها داده ها را از میکرو به خارج بفرستد یا داده ها را از بیرون دریافت کند.)
- این پایه ها را اصطلاحاً پورت های I/O نام گذاری کرده اند.
- در شکل روبرو این پایه ها با عنوان های PA تا PD مشخص شده است (۳۲ پایه در ۴ گروه ۸ تایی).
- در جدول زیر تعداد پورت های I/O برخی از مدلهای دیگر AVR آورده شده است. (در جدول زیر X به معنای ۸ بیت است.)
- به عنوان مثال ATmega64 دارای ۸x۶ + ۵ = ۵۳ پایه قابل برنامه ریزی است.

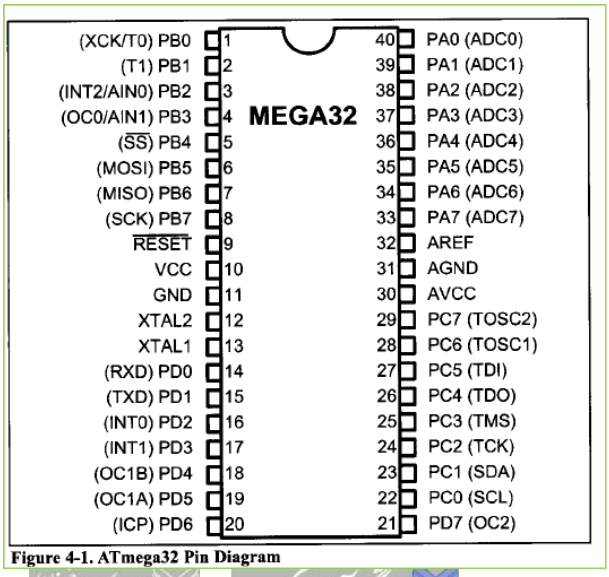


Figure 4-1. ATmega32 Pin Diagram

Table 4-1: Number of Ports in Some AVR Family Members

Pins	8-pin	28-pin	40-pin	64-pin	100-pin
Chip	ATtiny25/45/85	ATmega8/48/88	ATmega32/16	ATmega64/128	ATmega1280
Port A			X	X	X
Port B	6 bits	X	X	X	X
Port C		7 bits	X	X	X
Port D		X	X	X	X
Port E				X	X
Port F				X	X
Port G				5 bits	6 bits
Port H					X
Port J					X
Port K					X
Port L					X

پورت های I/O

- هر گروه از پورت ها I/O ۸ بیتی است، و ۳ رجیستر در فضای رجیسترهای I/O به هر گروه اختصاص داده شده است (شکل پایین سمت چپ).
- برنامه ریز با مقدار دهی و خواندان از این رجیسترها نقش پورت ها (ورودی یا خروجی) را تعیین می کند و همچنین داده ها را انتقال می دهد.
- هر پایه خروجی دارای سه بیت متناظر در این سه رجیستر است. (شکل پایین سمت راست)
- به عنوان مثال: برای کار با پایه PA5 باید با بیت پنجم رجیسترهای زیر کار کرد.

Table 4-2: Register Addresses for ATmega32 Ports

Port	Address	Usage
PORTA	$3B	output
DDRA	$3A	direction
PINA	$39	input
PORTB	$38	output
DDRB	$37	direction
PINB	$36	input
PORTC	$35	output
DDRC	$34	direction
PINC	$33	input
PORTD	$32	output
DDRD	$31	direction
PIND	$30	input

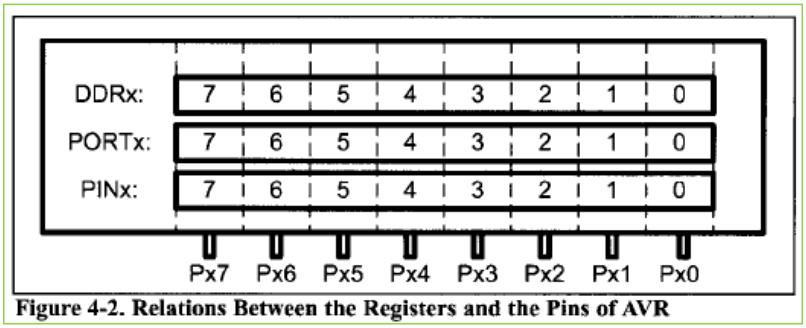
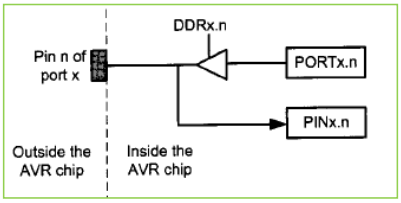


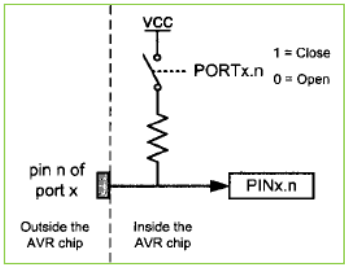
Figure 4-2. Relations Between the Registers and the Pins of AVR

پورت های I/O (۲)

- رجیستر DDRx ورودی یا خروجی بودن هر پایه را تعیین می کند.
- در این رجیستر، ۱ کردن هر بیت به معنای خروجی و ۰ کردن آن به معنای ورودی بودن پایه متناظر است.
- اگر پایه به شکل خروجی تنظیم شود: مقدار ولتاژ آن (۰ یا ۵ ولت) در اختیار برنامه نویس است.
- ورودی باشد: مداری که از بیرون به پایه وصل می شود ولتاژ آن را تعیین می کند. (از داخل قطع است.)



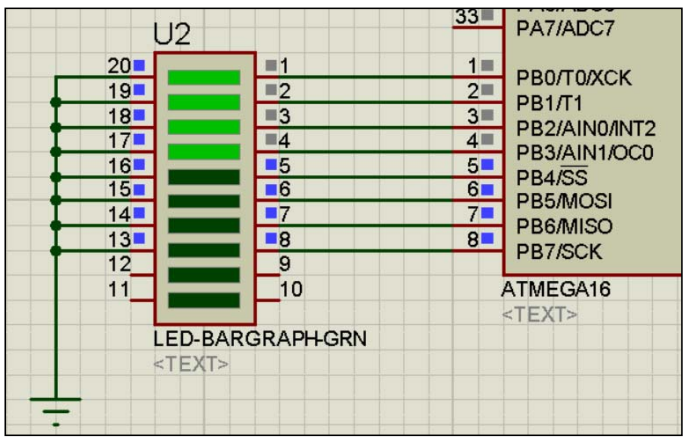
- اگر پایه خروجی تنظیم شده باشد، مقدار نوشته شده در PORTx به پایه های خروجی منتقل می شود.
- اگر پایه ورودی تنظیم شود، مقدار PORTx به خروجی منتقل نمی شود.
  بلکه در این حالت ورودی PORTx نقش مقابل را دارد. ←
  در این حالت می توان مقاومت داخلی را قطع یا وصل کرد.
- برای خواندن وضعیت فعلی پایه باید مقدار رجیستر PINx را بخوانیم.
- در ابتدای روشن شدن میکرو (یا Reset)، همه پایه ها ورودی هستند.



مثال اول از پایه های ورودی/خروجی

- هشت LED را به پایه های PB متصل کنید و به ترتیب زیر روشن کنید و سپس ترکیب را تغییر دهید.
- 00001111

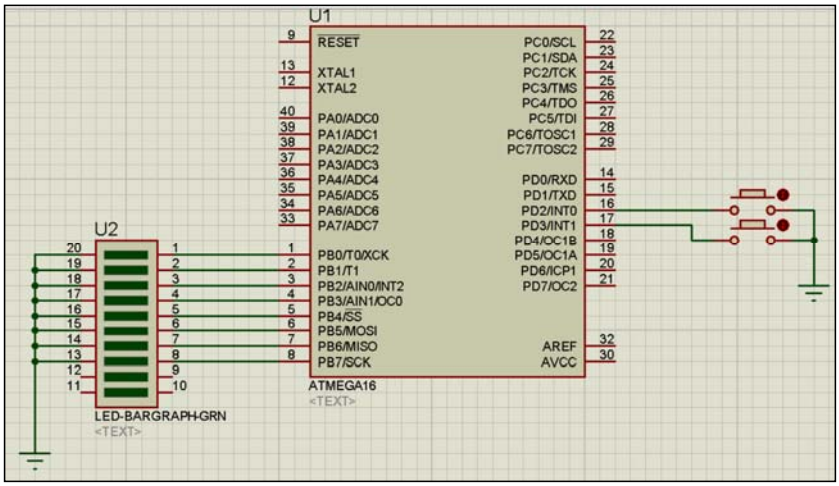
```
#include <mega16.h>
void main(void)
{
DDRB =0xFF;
PORTB =0b00001111;
while (1)
      {
      }
}
```



مثال دوم از پایه های ورودی/خروجی

- دو کلید به پایه های PD2 و PD3 وصل کنید با فشردن این کلیدها دو LED خاموش و روشن شوند. در تغییر بعدی برنامه یک کلید ۴ LED را خاموش و روشن کند. در تغییر بعدی عملکرد کلید ها معکوس شوند.

```
#include <io.h>
void main(void)
{
DDRB =0xFF;
while (1)
      {
            if (PIND.2==0)
                  PORTB.0=1;
            else
                  PORTB.0=0;
            PORTB.1=PIND.3;
      }
}
```



ادامه تمرین اسلاید قبل

- روشن و خاموش کردن ۴ LED با یک کلید:

```
#include <io.h>
void main(void)
{
DDRB =0xFF;
while (1)
      {
            if (PIND.2==0)
                  PORTB=0x0f;
            else
                  PORTB=0x00;
      }
}
```

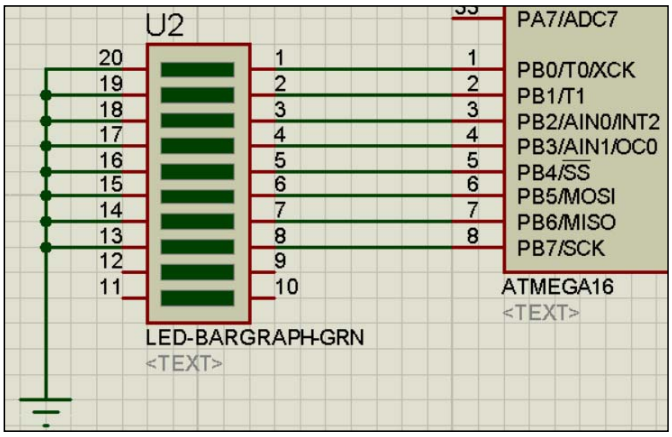
- معکوس کردن کارکرد کلیدها:

```
#include <io.h>
void main(void)
{
DDRB =0xFF;
while (1)
      {
            if (PIND.2==1)
                  PORTB.0=1;
            else
                  PORTB.0=0;
            PORTB.1=!PIND.3;
      }
}
```

مثال از ایجاد تاخیر بین دستورها

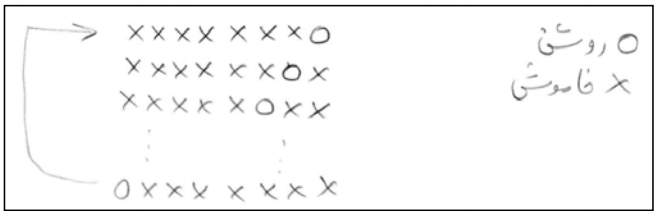
- یک LED را با یک ثانیه تاخیر روشن و خاموش کنید.

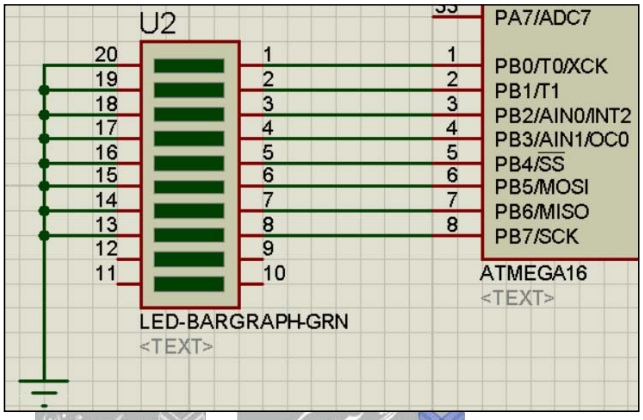
```
#include <io.h>
void main(void)
{
DDRB =0xFF;
while (1)
    {
        PORTB.0=1;
        delay_ms(1000);
        PORTB.0=0;
        delay_ms(1000);
    }
}
```



مثال دوم از ایجاد تاخیر بین دستورها

- ۸ LED را به ترتیب زیر و با تاخیر ۰.۵ ثانیه روشن کنید:






```
DDRB=0xFF;
while (1)
        {
        a=1;
        for (i=1;i<=8;i++)
        {
        delay_ms(500);
        PORTB = a;
        a=a*2;
        }
        }
}
```