جزوه درس

ریزپردازنده (میکروپروسسور)

گروه مهندسی پزشکی

جواد راستی

شاید تاکنون با واژه‌هایی مانند پردازنده‌های ۸ بیتی یا ۱۶ بیتی، ثبات یا رجیستر، ***ALU***، حافظه‌های ***RAM*** و ***ROM***، فضای آدرس‌دهی، گذرگاه یا باس داده و آدرس و... برخورد کرده باشید. این کلمات، واژه‌های معمول دنیای پردازنده‌ها هستند و هدف ما در این نوشتار، آشنا کردن شما با این مفاهیم می‌باشد.

۱-۱- پردازنده‌ها چگونه کار می‌کنند؟

سیستمهای کنترلی

به طور کلی هر سیستم کنترلی دارای سه بخش ورودی، پردازشگر و خروجی است (شکل ۱). چنین سیستمی داده‌ها را از دنیای خارج **دریافت** می کند (بخش **ورودی**)، روی این داده‌ها **عملیاتی** انجام می دهد (بخش **پردازش**) و بر اساس این پردازشها **خروجی** لازم را تولید می کند (بخش **خروجی**). کارخانه پارچه بافی یک مثال ساده برای یک سیستم کنترلی است که نخ بعنوان ماده اولیه وارد آن می شود و پس از انجام یک سری عملیات، پارچه بعنوان خروجی سیستم تولید می شود.

شکل ۱-۱- اجزاء یک سیستم کنترلی

تصمیم‌گیریهای منطقی، بخش اساسی پردازش در سیستمهای کنترلی هستند. در درس مدارهای منطقی با نحوه طراحی بخش پردازشگر سیستمهای الکترونیکی با استفاده از مدارات منطقی ترکیبی و ترتیبی آشنا شده اید.
عملکرد پردازنده‌ها از نظر کلی با مدارهای منطقی یکسان است و تنها چگونگی ورود و خروج داده‌ها و نحوه انجام پردازش روی آنهاست که باعث متفاوت شدن ویژگیهای آنها می‌شود.

چرا پردازنده‌ها به وجود آمدند؟

شاید این سؤال پیش بیاید که با وجود مدارهای ترکیبی و ترتیبی منطقی که انجام هر پردازشی را ممکن می‌سازند، چه نیازی موجب ایجاد و گسترش پردازنده‌ها شد؟

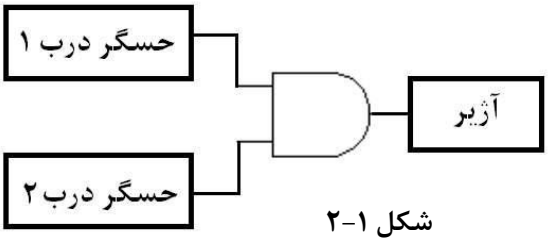
سادگی

ساده‌ترین دلیل این است که حجم و پیچیدگی مدارهای منطقی با زیاد شدن تعداد ورودیها و خروجیهای سیستم و نیز پردازشهای آن به شدت افزایش می‌یابد که باعث دشواری طراحی، نگهداری، پیاده‌سازی، اشکالزدایی و... سیستم می شود و استفاده از مدارهای منطقی به عنوان بخش پردازشگر سیستمهای پیچیده و بزرگ را عملاً غیرممکن می‌سازد.

برنامه پذیری

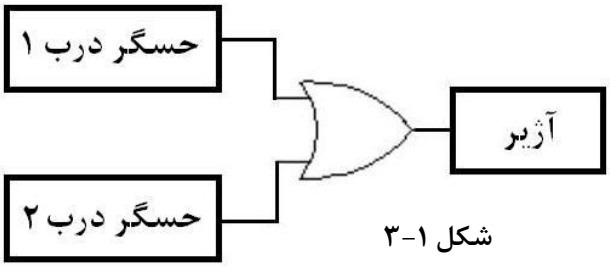
اما دلیل مهمتر، دشوار بودن پیکربندی مجدد و به عبارت دیگر تغییر ساختار مدارهای منطقی سیم‌بندی شده است. برای فهم این موضوع به مثال زیر توجه کنید:

فرض کنید یک سیستم امنیتی باید حالت دو درب را کنترل کرده و در صورت ایجاد خطر، آژیری را به صدا درآورد. یک حسگر[1]، حالت درب را به صورت یک سیگنال منطقی (صفر و یک) گزارش می‌کند؛ به این صورت که اگر درب باز باشد مقدار سیگنال یک و درغیراین‌صورت صفر است. خروجی سیستم که باید آژیر را فعال کند نیز یک سیگنال منطقی است که وقتی یک شود آژیر فعال می‌شود.



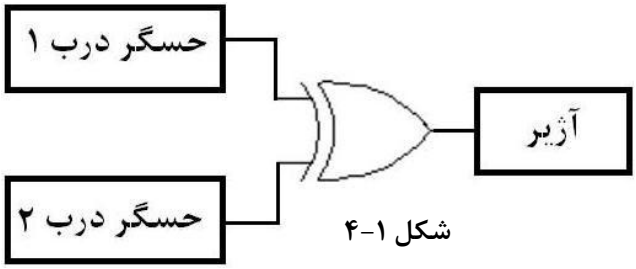
شکل ۱-۲

می‌خواهیم سیستم را به‌گونه‌ای طراحی کنیم که فقط اگر هر دو درب باز بودند، آژیر به صدا درآید. مدار ساده شده این سیستم در شکل ۱-۲ نشان داده شده است.



شکل ۱-۳

حال فرض کنید که از نظر امنیتی مدار باید به گونه‌ای تغییر یابد که اگر یکی از دربها هم باز شد، آژیر به صدا درآید. شکل ۱-۳ این سیستم را نشان می‌دهد.



شکل ۱-۴

اکنون سیاست امنیتی به گونه‌ای تغییر می‌یابد که فقط هنگامی که یکی از دربها باز بود، آژیر باید به صدا درآید. شکل ۱-۴ را ببینید.

همانطور که مشاهده می‌کنید هر بار که نظر کارفرما تغییر می‌کند، سیستم نیز باید عوض شود[2]؛ البته در این مثال ما تنها دو ورودی و یک خروجی داریم؛ در صورتی‌که همانند سیستمهای بزرگ و صنعتی دهها ورودی و خروجی داشته باشیم، تغییر کلی سیستم، هزینه‌های سرسام‌آوری درپی دارد.

ایده اصلی ایجاد و گسترش پردازنده‌ها، **قابلیت برنامه‌پذیری** آنهاست. یک سیستم کنترلی مبتنی بر پردازنده دارای تعدادی ورودی و خروجی است که تمام **پردازشهایی** که قرار است روی ورودیها انجام شود به صورت مجموعه ای از

[1] ***Sensor***

[2] به این سیستمها اصطلاحاً **طراحی شده به صورت سفارشی (Custom Designed)** گفته می شود.

دستورات نرم افزاری که **برنامه**[1] نام دارد، به سیستم داده می‌شود. تغییر نظر کارفرما، تنها منجر به **تغییر برنامه** سیستم می‌شود که هزینه آن در مقابل هزینه تغییر کلی سیستم بسیار ناچیز است. به بیان دیگر، پیچیدگی طراحی یک سیستم سخت‌افزاری، به طراحی یک برنامه مبدل می‌شود که بسیار ساده‌تر است. این موضوع اساس کار کنترل‌کننده‌های منطقی برنامه‌پذیر[2] که سیستمهایی مبتنی بر پردازنده (شکل ۱-۵) هستند، می‌باشد.

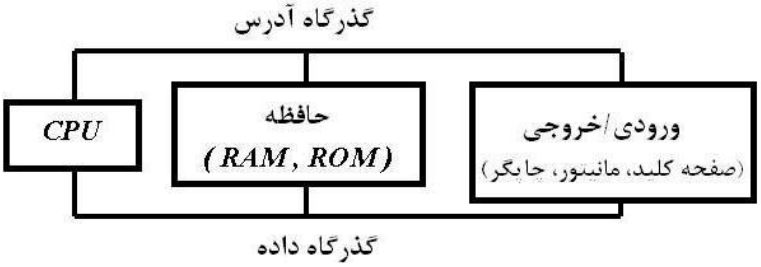
در این سیستمها، به جای طراحی بخش پردازشگر سیستم کنترلی به صورت سخت‌افزار سفارشی که مشکلات ذکر شده را به دنبال دارد، پردازش را به یک برنامه (نرم‌افزار) می سپاریم که هم طراحی و پیاده‌سازی و اشکالزدایی و هم تغییر کارکرد آن نسبت به سخت افزار، بسیار ساده‌تر و کم هزینه‌تر می باشد. وظیفه پردازنده **اجرای این برنامه** است.

شکل ۱-۵ – یک سیستم کنترلی مبتنی بر پردازنده

اکنون که با ضرورت وجود پردازنده‌ها آشنا شدیم، به بررسی چگونگی عملکرد آنها می‌پردازیم. در ادامه با ساختار کلی یک کامپیوتر به عنوان یک سیستم مبتنی بر پردازنده و سپس با چند مثال با نحوه اجرای برنامه‌ها در پردازنده‌ها آشنا خواهید شد.

یک سیستم پردازنده ای چه اجزایی دارد؟

شما با کامپیوتر به عنوان یک سیستم پردازنده ای آشنایی دارید. قطعه CPU داخل کامپیوتر همان پردازنده مورد نظر ماست؛ اما کامپیوتر اجزاء دیگری مانند دیسک سخت، RAM، کارت گرافیکی، صفحه کلید، نمایشگر[3] و ... دارد. تمام این اجزاء را می‌توان در سه بخش اصلی (که اجزاء اساسی هر سیستم مبتنی بر پردازنده نیز هستند)، طبقه‌بندی کرد (شکل ۱-۶):



شکل ۱-۶ – اجزاء یک سیستم کنترلی مبتنی بر پردازنده

[1] ***Program***
[2] ***Programmable Logic Controller: PLC***
[3] ***Monitor***

۱- *واحد پردازش مرکزی[1] (CPU)*

۲- *حافظه[2]*

۳- *وسایل ورودی و خروجی[3] (I/O)*

مجموعه دستورات یا برنامه ای که قرار است پردازشهای سیستم ما را انجام دهد، در واحد **حافظه** ذخیره می شود. کار **واحد پردازش مرکزی** (که از این به بعد آن را به اختصار **پردازنده** می نامیم) اجرای خط به خط این برنامه است. کار **واحدهای ورودی/خروجی**، تبادل اطلاعات با دنیای خارج است.

از نظر ساختاری می‌توان کامپیوتر را مانند یک انسان در نظرگرفت که داده‌هایی را از حواس خود (**ورودی**) مانند بویایی و بینایی می‌گیرد، در مغز خود (**پردازنده**) با توجه به اطلاعاتی که در ذهن (**حافظه**) دارد، پردازشهایی (شاید ناخودآگاه) را روی آنها انجام می‌دهد و متناسب با آنها عکس‌العمل (**خروجی**) نشان می‌دهد.
مانند اجزای بدن که از طریق رگها و اعصاب با هم مرتبط هستند، اجزاء داخلی یک کامپیوتر از طریق مجموعه‌ای از سیمها به نام گذرگاه یا **باس**[4] با یکدیگر در ارتباط می‌باشند.
اکنون به شرح مختصری راجع به بخشهای ذکر شده می‌پردازیم.

کار پردازنده چیست؟

کار پردازنده اجرای مجموعه دستورات یا برنامه‌ای است که در حافظه ذخیره شده است. برای این کار پردازنده باید:

- دستورات را به ترتیب از حافظه دریافت یا **واکشی** کند (Fetch)،
- معنای آن را **درک** کند (Decode)
- آن را **اجرا** کند (Execute).

برای درک سه مرحله فوق، کار یک نوازنده موسیقی را در نظر بگیرید. او برای اجرای یک آهنگ باید سمبل نت موسیقی را از روی یک‌صفحه که نتهای یک آهنگ روی آن نوشته شده‌اند، بخواند (Fetch)، اینکه آن سمبل نشان‌دهنده کدام نت است را درک کند (Decode)، و سپس آن نت را بنوازد (Execute).

برنامه چگونه به پردازنده داده می شود؟

اما پردازنده نه سمبلهای موسیقی را می‌فهمد و نه با هیچکدام از زبانهای محاوره‌ای ما آشناست! تنها زبانی که می‌توان به وسیله آن با پردازنده ارتباط برقرار کرد، زبان ارقام دودویی است. در واقع به دلیل سهولت استفاده از اجزاء الکترونیکی

[1] ***Central Processing Unit: CPU***
[2] ***Memory***
[3] ***Input / Output Devices: I/O***

[4] ***Bus***

مانند لامپ خلأ، رله و ترانزیستور که به صورت دوحالته نیز قابل استفاده هستند، از ابتدا مدارات منطقی و پردازنده‌ها با دستگاه دودویی استاندارد شدند.

کوچکترین واحد اطلاعاتی برای یک پردازنده، یک رقم دودویی یا یک **بیت**[1] است که حاوی یک سیگنال منطقی (صفر یا یک) می باشد[2]. تمام اطلاعات کامپیوتری از مجموعه‌ای از بیتها تشکیل می‌شوند. هر ۸ بیت یک **بایت**[3] نامیده می‌شود که یک واحد اطلاعاتی مهم در پردازنده‌هاست. واحد مهم دیگر، ***Nibble*** است که از ۴ بیت تشکیل می‌شود.

زبان ماشین[4]

برای ارتباط با یک پردازنده باید از ارقام دودویی استفاده کنیم. تمام دستورات، ورودیها و خروجیها در این قالب قرار می‌گیرند که آن را **زبان ماشین** یا **زبان صفر و یک** می‌نامیم. زبان ماشین، پایینترین سطح گفتگو با پردازنده است.

در ابتدا به دلیل عدم گستردگی کاربرد پردازنده‌ها، از زبان صفر و یک برای برنامه‌نویسی پردازنده‌ها استفاده می‌شد. مثلاً فرض کنید می‌خواهیم با استفاده از یک پردازنده فرضی، دو عدد را بخوانیم و با هم جمع کنیم و نتیجه را نشان دهیم. با مطالعه برگه اطلاعات پردازنده مورد نظر، متوجه می‌شویم که در این پردازنده رشته "10110001" به عنوان دستور خواندن عدد از ورودی در نظر گرفته شده است؛ یعنی پردازنده با دریافت این رشته صفر و یک از حافظه (به عنوان دستور)، یک عدد را از ورودی می خواند. به همین ترتیب دستور جمع دو عدد "00110011" و دستور ارسال نتیجه به خارج "00100010" است. بنابراین برنامه فوق به شکل زیر در می آید:

```
10110001
10110001
00110011
00100010
```

همانطور که می بینید برنامه نویسی و همچنین فهم این برنامه بسیار دشوار است.

پس از گسترده شدن کاربرد پردازنده‌ها در سیستمهای مختلف، نیاز به زبانی که علیرغم نزدیکی کافی به سطح ماشین، دارای دستورات ساده تر و قابل فهم تر باشد به شدت احساس می شد.

زبان اسمبلی[5]

در زبان اسمبلی که به همین هدف عرضه شد، برای هر دستور یک کلمه معادل که شبیه به کلمات انگلیسی است، وجود دارد؛ مثلاً در پردازنده فوق کلمه IN معادل رشته "10110001" در نظر گرفته شده است که با معنای این دستور که خواندن

---

[1] ***Binary digIT: BIT***

[2] در تراشه‌های با استاندارد TTL، ولتاژ حدود صفر تا ۰/۸ ولت برای بیان **صفر منطقی** و ولتاژ حدود ۳ تا ۵ ولت برای بیان **یک منطقی** به کار می رود. صفر و یک منطقی می تواند با دو مقدار متفاوت جریان (مثلاً ۴ و ۲۰ میلی آمپر) نیز نمایش داده شود.

[3] ***Byte***

[4] ***Machine Language***

[5] ***Assembly Language***

عدد از ورودی (INPUT) است نیز همخوانی دارد. برنامه قبلی با استفاده از معادل اسمبلی دستورات به صورت زیر در می آید:

```
IN
IN
ADD
OUT
```

برنامه نویسی به زبان اسمبلی، کار را برای برنامه نویس راحت‌تر می کند؛ اما پردازنده تنها زبان صفر و یک را متوجه می شود. نرم افزار اسمبلر[1] کار ترجمه دستورات زبان اسمبلی به دستورات زبان ماشین را بر عهده دارد.

زبان اسمبلی با ایجاد تحول در برنامه نویسی پردازنده‌ها، سالها بعنوان زبان برنامه‌نویسی متداول پردازنده‌ها به کار می رفت. اما از دشواری نسبی برنامه نویسی به زبان اسمبلی که بگذریم، برنامه‌نویس این زبان باید از ساختمان داخلی پردازنده‌ای که برای آن برنامه می نویسد، کاملاً آگاه باشد. بعلاوه برنامه ای که به زبان اسمبلی یک پردازنده نوشته می شود، روی پردازنده دیگری قابل اجرا نیست؛ چون هر برنامه اسمبلی (یا زبان ماشین) با توجه به ساختار داخلی یک پردازنده نوشته می شود که با پردازنده‌های دیگر متفاوت است. به همین دلیل زبان ماشین و زبان اسمبلی را **زبانهای وابسته به ماشین**[2] می گویند.

زبانهای سطح بالا[3]

دستورات زبانهای برنامه نویسی سطح بالا مانند زبانهای پاسکال، C، فرترن و ... علاوه بر شباهت زیاد به زبان انگلیسی که باعث سادگی برنامه نویسی می شود، به کاربر اجازه می دهد بدون نیاز به اطلاع از ساختار داخلی پردازنده، برنامه‌های خود را بنویسد؛ مثلاً دستور clrscr (Clear Screen)، دستور پاک کردن صفحه نمایش کامپیوتر در زبانهای پاسکال و C است؛ اما برای انجام همین کار به زبان اسمبلی کامپیوتر این دستورات باید نوشته شوند:

```
MOV     AH,6H
MOV     AL,25
MOV     CX,0
MOV     DH,24
MOV     DL,79
MOV     BH,14H
INT     10H
```

این مثال به خوبی تفاوت برنامه نویسی به زبان سطح بالا و زبان اسمبلی را نشان می‌دهد.

چون پردازنده تنها دستورات زبان ماشین را درک می کند، از نرم افزار مترجم (**کامپایلر**)[4] برای تبدیل دستورات زبان سطح بالا به سطوح پایین‌تر استفاده می شود.

---

[1] ***Assembler***
[2] ***Machine Dependent***
[3] ***High Level Languages: HLL***
[4] ***Compiler***

ذکر این نکته ضروریست که زبان اسمبلی هنوز در مواردی مانند وقتی که حجم و زمان اجرای برنامه‌ها مهم است، مورد استفاده قرار می گیرد، به طوری که حتی گاهی بخشی از یک برنامه سطح بالا را با استفاده از دستورات اسمبلی می نویسیم. این کار برای کاهش حجم کد ماشین نهایی و یا زمان اجرای آن و نیز استفاده مؤثر از سخت‌افزار انجام می‌شود.

اجزاء داخلی پردازنده‌ها

پردازنده برای انجام سه وظیفه اصلی خود یعنی واکشی، رمزگشایی و اجرای دستورات، به امکاناتی مجهز است که در زیر به مهمترین آنها می‌پردازیم:

۱ - ثبات‌ها[1]: ثبات‌ها، حافظه‌های کوچک و سریعی هستند که داخل پردازنده قرار دارند و برای ذخیره موقت داده‌ها و دستکاری آنها به کار می‌روند. این ثباتها بسته به نوع پردازنده می‌توانند ۸ بیتی، ۱۶بیتی، ۳۲ بیتی، ۶۴ بیتی و ... باشند. هرچه تعداد و اندازه ثباتها بیشتر باشد، کارایی پردازنده بالاتر است.

پردازنده‌ها دارای ثباتهای متنوعی هستند که در زیر چند نوع از آنها را معرفی می‌کنیم:

- **ثبات انباره[2]:** یک ثبات همه‌منظوره[3] است که در انواع دستورات حسابی و منطقی و انتقال داده‌ها بعنوان برگه کاری موقت به‌کار می‌رود.
- **ثبات حالت (پرچم)[4]:** بیتهای این ثبات بیانگر حالت پردازنده بعد از اجرای دستورات است. مثلاً یکی از بیتهای این ثبات، پرچم صفر[5] است که اگر نتیجه یک عمل منطقی یا حسابی برابرصفر باشد، مقدار آن برابر یک می‌شود؛ یعنی مقدارZF بعد از انجام یک عملیات منطقی یا حسابی، بیانگر صفر بودن یا صفر نبودن نتیجه عملیات است.

  از بیتهای ثبات پرچم که در اکثر پردازنده‌ها وجود دارند می‌توان به پرچم صفر، پرچم سرریز[6]، پرچم علامت[7] و پرچم رقم نقلی[8] اشاره کرد. کاربرد اصلی ثبات پرچم در تصمیم گیریهای لازم بعد از انجام یک عمل داخلی پردازنده است.
- **ثبات شمارنده برنامه[9]:** چون دستورات یک برنامه باید به ترتیب اجرا شوند، پردازنده باید به طریقی بداند دستور بعدی که باید اجرا کند کدام است. کار ثبات شمارنده برنامه، نگهداری آدرس دستور بعدی است که قرار است توسط پردازنده اجرا شود. با اجرای هر دستور، پردازنده به طور خودکار یک واحد به این ثبات اضافه می‌کند تا به

---

[1] ***Registers***
[2] ***Accumulator Register***
[3] ثباتهای همه منظوره (General Purpose Registers) ثباتهایی هستند که برای انجام هر عملیاتی داخل پردازنده می توانند مورد استفاده قرار گیرند.
[4] ***Flag Register***
[5] ***Zero Flag: ZF***
[6] ***Oveflow Flag: OF***
[7] ***Sign Flag: SF***
[8] ***Carry Flag: CF***
[9] ***Program Counter: PC*** یا ***Instruction Pointer: IP***

دستور بعدی اشاره کند. با کمک این ثبات و گذرگاههای داده و آدرس، پردازنده دستورات را از حافظه دریافت می‌کند.

گفتیم که پردازنده، دستوراتی را که از حافظه دریافت می کند، اجرا می نماید. این دستورات چگونه اجرا می شوند؟

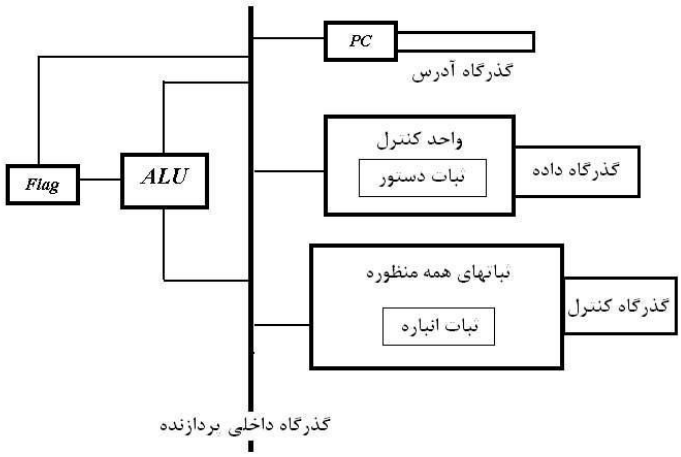
۲- *واحد محاسبه و منطق*[1]: این واحد پردازنده که اصطلاحاً به آن ALU گفته می‌شود، قسمتی از پردازنده است که مسؤول انجام اعمال ریاضی مانند جمع، تفریق، ضرب و تقسیم و اعمال منطقی مانند AND ، OR و NOT می‌باشد.

واحد ALU تنها یک واحد حسابگر است و برای عملکرد درست باید کنترل شود.

۳ - واحد کنترل[2]: کار این بخش، کنترل تمام فعالیتهای پردازنده است.

یکی از مهمترین قسمتهای واحد کنترل، ثبات رمزگشای دستور[3] است. می‌توان این قسمت را به‌عنوان یک فرهنگ لغت تصور کرد که معنای هر دستور و مراحلی را که پردازنده باید برای اجرای آن در پیش بگیرد، مشخص می‌کند. دستورات پس از واکشی از حافظه وارد این ثبات می‌شوند تا پس از رمزگشایی اجرا شوند.

بلوک دیاگرام داخلی پردازنده در شکل ۱-۷ آمده است.



شکل ۱-۷- بلوک دیاگرام داخلی پردازنده

سرعت پردازنده به چه معناست؟

در آخر به این نکته توجه کنید که پردازنده از نظر عملیاتی، مانند یک مدار ترتیبی منطقی همزمان (سنکرون) عمل می‌کند و تمام اعمال آن با یک موج مربعی که پالس ساعت نام دارد، هماهنگ می‌شود. فرکانس پالس ساعت، یکی از معیارهای

[1] ***Arithmetic & Logic Unit: ALU***
[2] ***Control Unit: CU***
[3] ***Instruction (Decoder) Register: IR***

کارایی پردازنده و بیانگر سرعت آن خواهد بود. معیار مشابه دیگر به صورت MIPS[1] یعنی تعداد دستورات قابل اجرا توسط پردازنده بر حسب میلیون دستور در ثانیه، بیان می‌شود.

برای بحث دقیقتر در مورد چگونگی اجرای یک دستورالعمل در پردازنده، به مراجع معماری کامپیوتر مراجعه کنید.

نقش حافظه در سیستمهای کامپیوتری

اطلاعاتی که باید توسط پردازنده پردازش شود، اعم از دستورات و بعضی داده‌های ورودی/خروجی، باید در حافظه ذخیره شود. در سیستمهای پردازنده‌ای معمولاً از دو نوع حافظه استفاده می‌شود:

- **حافظه فقط خواندنی (ROM)**[2]: اطلاعات این حافظه با قطع برق از میان نمی‌رود. بنابراین معمولاً (همانطور که از نام آن برمی‌آید) برای ذخیره اطلاعات دایمی و ثابت به کار می‌رود. مثلاً برنامه‌ای که حافظه کامپیوتر را تست می‌کند و حتماً اجرای آن را هنگام روشن کردن کامپیوتر خود دیده‌اید، باید پس از هر بار روشن شدن کامپیوتر اجرا شود؛ بنابراین باید در ROM کامپیوتر ذخیره شود[3]. معمولاً برای نوشتن در حافظه ROM به دستگاه مخصوصی به نام برنامه‌ریز[4] نیاز است.
- **حافظه خواندنی و نوشتنی (RAM[5] یا RWM[6])**: اطلاعات این نوع حافظه با قطع برق از میان می‌رود. به علاوه برای نوشتن در آن نیازی به دستگاه برنامه‌ریز نیست. بنابراین برای ذخیره داده‌های موقت مانند متغیرهای یک برنامه یا در سطوح بالاتر برای نگهداری سیستم عامل یا نرم‌افزارهای مختلف مانند انواع بازیها یا برنامه‌های کاربردی دیگر به کار می‌رود.

  اطلاعات ROM بر خلاف RAM دایمی است و با قطع برق از بین نمی‌رود؛ به همین دلیل، گاهی ROM را حافظه غیرفرّار[7] هم می‌نامند.

عملکرد حافظه

همانطور که قبلاً ذکر شد، حافظه محل ذخیره دستورات و نیز بعضی داده‌های ورودی و خروجی است.

مشخصات مهم یک حافظه عبارتند از:

الف) ظرفیت حافظه: بیانگر حجم اطلاعاتی است که می‌تواند در حافظه ذخیره شود و معمولاً بر حسب بیت یا بایت بیان می‌شود.

---

[1] ***Milion Instruction Per Second: MIPS***

[2] ***Read Only Memory: ROM***

[3] به برنامه فوق که عملیات ابتدایی سیستم از جمله تست حافظه کامپیوتر را انجام می‌دهد، اصطلاحاً بایوس (BIOS) گفته می‌شود.

[4] ***Programmer***

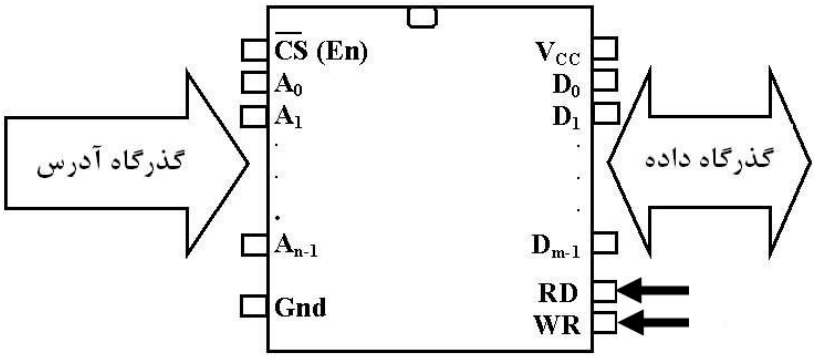
[5] ***Random Access Memory: RAM***

[6] ***Read/Write Memory***

[7] ***Non Volatile Memory: NV-Memory***

ب) ساختار حافظه: حافظه از تعدادی ثبات یکسان (خانه) تشکیل می‌شود که در هر خانه یک قلم داده ذخیره می‌شود. منظور از ساختار حافظه، حجم هر یک از خانه‌ها و تعداد خانه‌هاست.

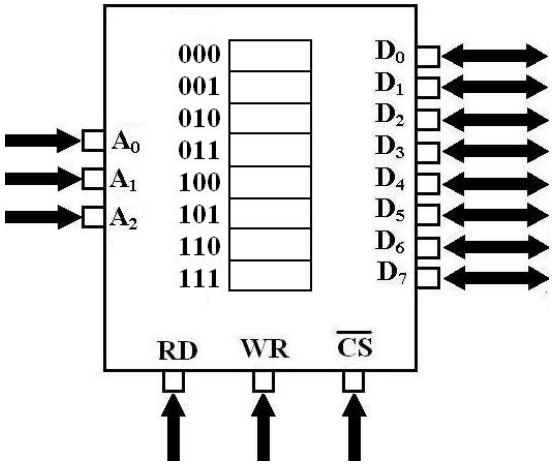
حجم هر خانه حافظه از یک تا ۸ و گاهی ۱۶ بیت معمول است. تعداد خانه‌های یک حافظه نیز معمولاً توانی از ۲ است. شکل کلی یک حافظه را در شکل ۱-۸ می‌بینید.



شکل ۱-۸- پینهای یک تراشه حافظه

خطوط A_0 تا A_{n-1} که گذرگاه آدرس حافظه نامیده می‌شود، آدرس خانه‌ای از حافظه که باید اطلاعات آن خوانده یا در آن اطلاعاتی نوشته شود را مشخص می‌کند. طول هر قلم اطلاعاتی (حجم هر خانه) که اطلاعات آن از طریق خطوط داده D0 تا D_{m-1} رد و بدل می‌شود، m بیت است.

پینهای RD و WR برای تعیین نحوه تبادل اطلاعات با حافظه (خواندن یا نوشتن) به‌کارمی‌روند. اگر سیگنال RD را فعال کنیم یعنی می‌خواهیم از حافظه اطلاعاتی بخوانیم. با فعال کردن سیگنال WR به حافظه اعلام می‌کنیم که اطلاعاتی در یکی از خانه‌هایش نوشته می‌شود. در هر دو عمل خواندن و نوشتن، تبادل اطلاعات با خانه‌ای از حافظه که شماره (آدرس) آن روی گذرگاه آدرس حافظه گذاشته می‌شود انجام می‌گیرد.



به عنوان مثال حافظه نوعی روبرو را در نظر بگیرید. این حافظه ۸ خط داده (D_0 تا D_7) و ۳ خط آدرس (A_0 تا A_2) دارد؛ چون در این حافظه می‌توان 2^3 خانه حافظه را آدرس‌دهی کرد (با ۳ بیت می‌توان 2^3 عدد دودویی مختلف را مشخص کرد) و ظرفیت هر خانه نیز ۸ بیت است (چون ۸ پین برای تبادل اطلاعات با خارج دارد)، پس این حافظه دارای ۸ (2^3) خانه یک بایتی (۸ بیتی) است. پس ظرفیت این تراشه ۸ بایت یا ۶۴ بیت است.

به عنوان یک مثال عملی، تراشه 2732، ۸ خط داده (D_0 تا D_7) و ۱۲ خط آدرس (A_0 تا A_{11}) دارد؛ چون در این حافظه می‌توان 2^{12} خانه حافظه را آدرس‌دهی کرد (با ۱۲ بیت می‌توان 2^{12} عدد دودویی مختلف را مشخص کرد)، پس این حافظه دارای $2^{10} \times 4 = 1024 \times 4$ (2^2 (یا ۴ کیلو) خانه یک بایتی (۸ بیتی) است. پس ظرفیت این تراشه ۴ کیلوبایت یا ۳۲ کیلوبیت (۸ × ۱۰۲۴ × ۴) است که دو رقم آخر شماره آن نیز این مطلب را نشان می‌دهد.

پرسش: ظرفیت حافظه شکل ۱-۱۰ بر حسب بیت چقدر است؟

ذخیره و بازیابی اطلاعات در حافظه

اکنون فرض کنید می‌خواهیم در خانه شماره ۷ حافظه 2732، عدد دودویی ۸ بیتی **10110001** را ذخیره کنیم. کافی است شماره خانه فوق یعنی معادل دودویی ۱۲ بیتی عدد ۷ (000000000111) را روی پینهای A_0 تا A_{11} و عدد مورد نظر (**10110001**) را روی پینهای D_0 تا D_7 تراشه قرار داده و پین WR (Write) را فعال کنیم تا عدد فوق در خانه شماره ۷ ذخیره شود. اگر این حافظه از نوع RAM باشد تا زمان قطع برق و اگر از نوع ROM باشد به طور دائم این داده در این خانه باقی خواهد ماند.

برای خواندن محتویات یک خانه مثلاً همان خانه شماره ۷، کافی است عدد ۱۲ بیتی معادل ۷ یعنی 000000000111 را روی پینهای آدرس حافظه قرار داده و پین RD (Read) را فعال کنیم. عدد ذخیره شده در خانه ۷ که در اینجا 10110001 است روی پینهای D_0 تا D_7 (گذرگاه داده حافظه) قرار می‌گیرد.

توجه کنید که چه برای نوشتن در حافظه و چه برای خواندن از آن، پین فعال ساز تراشه (CS یا En) باید فعال باشد.

مراحل خواندن و نوشتن در حافظه را می‌توان به صورت زیر خلاصه کرد؛

برای نوشتن عدد a در آدرس b حافظه:

- آدرس b را روی پینهای آدرس حافظه (گذرگاه آدرس حافظه) قرار می‌دهیم.
- عدد a را روی پینهای داده حافظه (گذرگاه داده حافظه) قرار می‌دهیم.
- پین WR حافظه را فعال می‌کنیم تا عدد a در آدرس b حافظه ثبت شود.

برای خواندن محتویات خانه حافظه به آدرس x:

- آدرس x را روی پینهای آدرس حافظه (گذرگاه آدرس حافظه) قرار می‌دهیم.
- پین RD حافظه را فعال می‌کنیم تا محتویات خانه حافظه به آدرس x روی پینهای داده حافظه (گذرگاه داده حافظه) قرار گیرد.
- پینهای داده حافظه (گذرگاه داده حافظه) را می‌خوانیم.

پرسش) راجع به دو مفهوم گسترش طول کلمه و گسترش طول آدرس حافظه تحقیق کنید.

ج) زمان دسترسی: مشخصه سرعت حافظه و عبارتست از زمانی که بعد از گذاشتن آدرس خانه مورد نظر روی خطوط آدرس و فعال کردن پین RD به طول می‌انجامد تا داده خانه فوق روی خطوط داده تراشه حافظه به صورت پایدار ظاهر شود.

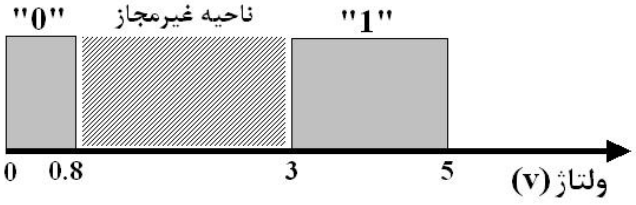
نقش وسایل ورودی/خروجی در سیستمهای کامپیوتری

در حقیقت پردازنده‌ها برای رفع نیازهای کاربران طراحی شده‌اند؛ به این منظور باید داده‌هایی به پردازنده وارد شده و پس از انجام عملیات مورد نظر، نتایج از پردازنده خارج شوند. مجموعه‌ای شامل پردازنده، گذرگاهها و وسایل ورودی/خروجی، این وظیفه را در سیستمهای کامپیوتری به عهده دارند.

وسایل ورودی می‌توانند بسیار ساده مانند یک کلید دوحالته و یا پیچیده مانند صفحه کلید یا ماوس باشند. به طور مشابه، وسایل خروجی نیز انواع ساده مانند دیود نوری تا انواع پیچیده مانند نمایشگر را دربرمی‌گیرند.

ورودی/خروجی‌های ساده

اطلاعاتی که در سیستمهای مبتنی بر پردازنده و مدارهای منطقی وجود دارد، همه در قالب «صفر و یک منطقی» است. بنابراین اگر بتوانیم یک بیت (یک رقم دودویی) را تولید کنیم، می‌توانیم اطلاعات چند بیتی را نیز به عنوان ورودی به سیستم بدهیم. به همین ترتیب اگر بتوان یک بیت را نمایش داد، می‌توان اطلاعات چندبیتی را نیز به راحتی به نمایش درآورد.

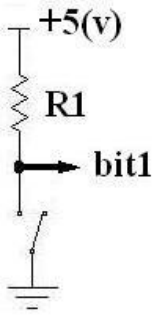


در مدارهای منطقی و سیستمهای مبتنی بر پردازنده، از «صفر و یک منطقی» تعبیر ولتاژی می‌شود. شکل مقابل نشان می‌دهد، اگر ولتاژی بین صفر تا ۰/۸ ولت باشد از آن به «صفر منطقی» و اگر بین ۳ تا ۵ ولت باشد از آن به «یک منطقی» تعبیر می‌شود[1]. مثلاً اگر به پین ورودی یک گیت NOT (پین ۱ تراشه 7404) ولتاژی حدود صفر (مثلاً ۰/۲ ولت) اعمال کنیم، روی پین خروجی این گیت (پین ۲ تراشه 7404) ولتاژی حدود ۵ ولت (مثلاً ۴/۸ ولت) به کمک ولت‌متر قابل مشاهده است. ولتاژ بین ۰/۸ تا ۳ ولت در منطق ولتاژی فوق غیرمجاز است؛ یعنی اگر خروجی یک مدار منطقی مثلاً ۲ ولت باشد، مدار خراب است و به درستی کار نمی‌کند. اگر به ورودی یک مدار منطقی ولتاژ ۲ ولت داده شود، مدار ممکن است از آن تعبیر «صفر» یا «یک» داشته باشد.

پس برای ایجاد یک بیت، باید مداری دوحالته داشته باشیم که دو ولتاژ صفر و ۵ ولت را بتواند تولید کند. برای نمایش یک بیت نیز باید مداری ببینیم که در مقابل دو ولتاژ صفر و ۵ ولت، دو واکنش متفاوت ایجاد کند.

[1] این منطق ولتاژ موسوم به TTL است. در منطق‌های ولتاژی دیگر مانند CMOS، ولتاژهای دیگری به «صفر» و «یک» منطقی نسبت داده می‌شوند.

چگونه یک بیت را تولید کنیم؟



ساختار مقابل با استفاده از یک کلید دوحالته این کار را انجام می دهد.

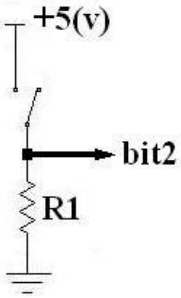
در این ساختار اگر کلید باز باشد، سیگنال bit1 توسط مقاومت R1 به ۵ ولت متصل است؛ بنابراین «یک منطقی» ایجاد می شود.

اگر کلید بسته باشد، سیگنال bit1 به زمین متصل شده و «صفر منطقی» ایجاد می شود.

به طور خلاصه:

bit1	کلید
"1"	باز
"0"	بسته

یک بیت با استفاده از ساختار روبرو نیز قابل ایجاد است.



در این ساختار اگر کلید باز باشد، سیگنال bit2 توسط مقاومت R1 به زمین متصل است؛ بنابراین «صفر منطقی» ایجاد می شود.

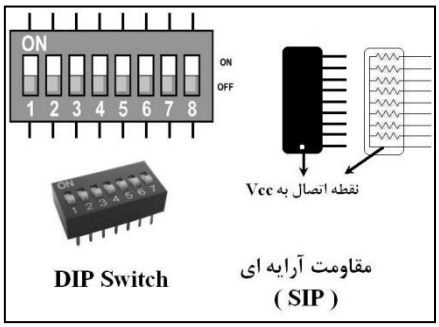
اگر کلید بسته شود، سیگنال bit2 به ۵ ولت متصل شده و «یک منطقی» ایجاد می‌شود.

به طور خلاصه:

bit2	کلید
"0"	باز
"1"	بسته

مقاومت R1 در هردو ساختار برای جلوگیری از اتصال کوتاه شدن منبع تغذیه هنگام بسته شدن کلید به کار رفته و مقدار نوعی آن 10 KΩ است.

همانطور که مشاهده می کنید **bit1** در هنگام **باز بودن کلید** برابر «یک منطقی» است و **bit2** در هنگام **بسته بودن کلید**.

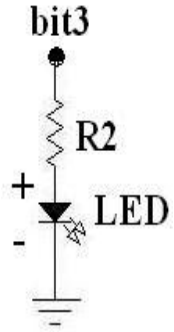


دو ساختار ذکر شده از نظر فنی متفاوت هستند که بررسی این تفاوت به عنوان تحقیق به شما واگذار شده است.

برای تولید یک عدد ۸ بیتی باید هشت عدد از ساختارهای فوق را به هم متصل کرد. معمولاً برای اتصال ساختار فوق (۸ کلید دوحالته)، از کلیدهای کوچک کنار هم که در تعداد ۴ تایی یا ۸ تایی ساخته می‌شوند و به نام میکروسوییچ یا

DIP Switch معروفند، استفاده می‌شود. همچنین برای خودداری از به کار بردن ۸ مقاومت مجزا که باعث شلوغی مدار می‌شود، از مقاومتهای آرایه ای که به نام SIP معروفند، استفاده می شود که هم جای کمتری اشغال می کنند و هم اتصال آنها ساده تر است.

چگونه یک بیت را نمایش دهیم؟



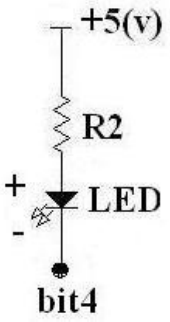
ساده ترین راه برای نمایش یک بیت، استفاده از دیود نوری (LED[1]) است. دیود نوری یک عنصر الکترونیکی شبیه یک لامپ کوچک با دوپایه مثبت (آند) و منفی (کاتد) است. اگر ولتاژ پایه مثبت دیود نوری از ولتاژ پایه منفی آن از حدی بیشتر شود، دیود نوری روشن و در غیر این صورت خاموش می‌شود. ساختار روبرو برای نمایش یک بیت با دیود نوری به کار می رود.

اگر bit3، «صفر منطقی» باشد، ولتاژی حدود صفر دارد و با توجه به صفر بودن ولتاژ سر منفی، دیود نوری روشن نخواهد شد. اگر bit3، «یک منطقی» باشد دارای ولتاژی بالاتر از ۳ ولت است و باعث روشن شدن دیود نوری می شود.

به طور خلاصه:

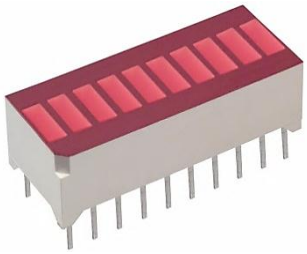
وضعیت دیود نوری	bit3
خاموش	"0"
روشن	"1"

ساختار دیگر نمایش یک بیت را در شکل روبرو می بینید.



پرسش) با توجه به ساختار مقابل، جدول زیر را کامل کنید.

وضعیت دیود نوری	bit4
	"0"
	"1"



مقاومت R2 در هر دو ساختار برای جلوگیری از سوختن دیود نوری در اثر عبور جریان زیاد به کار رفته و مقدار نوعی آن 330 Ω است.

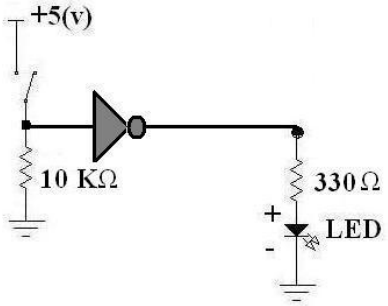
برای نمایش یک عدد ۸ بیتی باید هشت عدد از ساختارهای فوق را به هم متصل کرد. معمولاً برای اتصال ساختار فوق (۸ عدد LED)، از LEDهای کوچک کنار هم که در تعداد ۴

[1] **L**ight **E**mitting **D**iode

تایی یا ۸ تایی ساخته می شوند و به نـام LED Bar معروفنـد، اسـتفاده مـی‌شـود. همچنـین بـرای خـودداری از بـه کـار بـردن ۸ مقاومت مجزا که باعث شلوغی مدار می شود، از مقاومتهای آرایه ای که به نام SIP معروفند، استفاده می شـود کـه هـم جـای کمتری اشغال می کنند و هم اتصال آنها ساده‌تر است.

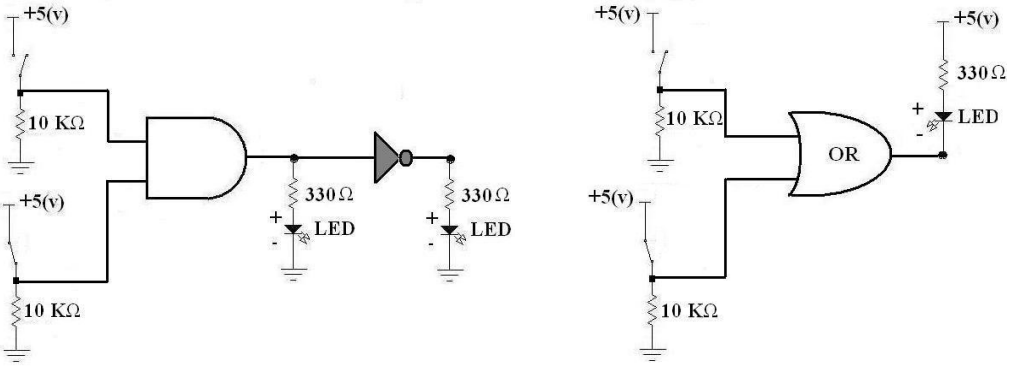
معمولاً در پردازنده‌های هشت بیتی به هر هشت بیت ورودی یا خروجی یک آدرس (مانند آدرس حافظه) نسبت داده می‌شود.

مثال) در ساختار روبرو مشخص کنید دیود نوری خاموش است یا روشن؟

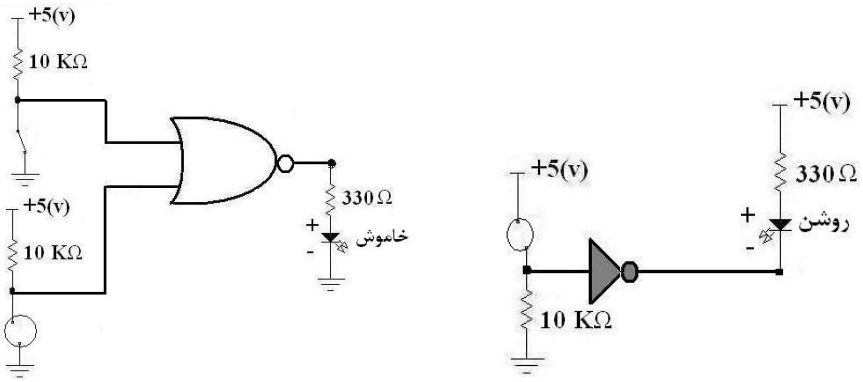


چون کلید باز است، ورودی گیت NOT با مقاومت 10KΩ به زمین متصـل شـده اسـت و «صفر منطقی» است. اگر ورودی گیت NOT «صفر» باشد، خروجی آن «یک» (یعنی حدود ۵ ولت) است؛ چون ولتاژ سر مثبت دیـود نـوری از سـر منفـی آن بیشـتر اسـت، دیود نوری روشن می شود.

پرسش) در هریک از ساختارهای زیر مشخص کنید هر یک از دیودهای نوری روشن است یا خاموش؟



پرسش) در هریک از شکلهای زیر معلوم کنید کلید مشخص شده باز است یا خیر؟



گذرگاهها در سیستمهای کامپیوتری

در هر کامپیوتر به طور معمول سه نوع گذرگاه وجود دارد: گذرگاه داده[1]، گذرگاه آدرس[2] و گذرگاه کنترلی[3].
تمام اطلاعاتی که باید در یک سیستم کامپیوتری جابجا شوند، از **گذرگاه داده** عبور می‌کنند. **گذرگاه آدرس** مشخص می‌کند که گذرگاه داده در هر لحظه باید در اختیار چه وسیله‌ای باشد و **گذرگاه کنترلی** این ارتباط را نظم می‌بخشد.

اهمیت گذرگاهها به حدی است که یکی از مهمترین معیارهای کارایی پردازنده‌ها، پهنای (تعداد خطوط) گذرگاه آدرس و گذرگاه داده آنها و تنوع سیگنالهای گذرگاه کنترلی آنهاست. کمی بیشتر با این گذرگاهها آشنا می‌شویم:

- **گذرگاه داده:** همانطور که گفته شد، گذرگاه داده محل عبور تمام اطلاعاتی است که باید داخل سیستم جابجا شوند؛ بعنوان مثال برای انتقال دستورات از حافظه به پردازنده، برای خواندن اطلاعات از ورودی و نیز برای ارسال اطلاعات به خروجی باید از گذرگاه داده استفاده شود.

پهنای گذرگاه داده (یعنی اینکه گذرگاه دادهٔ یک پردازنده چند بیتی است) تا آنجا مهم است که یکی از مهمترین معیارهای طبقه‌بندی پردازنده‌ها به شمار می‌رود که ابتدایی‌ترین نوع آن ۴ بیتی (مانند پردازنده‌های ۴۰۰۴ و ۴۰۴۰) بود و سپس انواع کاملتر ۸ بیتی (Z80، ۶۸۰۰، ۸۰۸۵)، ۱۶ بیتی (۸۰۸۶، ۸۰۸۸، ۸۰۲۸۶) و ۳۲ بیتی (۸۰۳۸۶، ۸۰۴۸۶، پنتیوم) به بازار آمد. وقتی گفته می‌شود پردازنده ۸۰۸۵، ۸ بیتی است یعنی می‌تواند هر بار (در هر سیکل کاری) ۸ بیت را پردازش کند؛ به بیان دیگر ورود و خروج اطلاعات در آن باید در قالبهای ۸ بیتی صورت گیرد.
واضح است که قدرت پردازش یک پردازنده به طور مستقیم با اندازه گذرگاه داده آن در ارتباط است؛ به عنوان مثال، یک پردازنده ۸ بیتی در هر سیکل کاری ۸ بیت را می‌تواند پردازش کند؛ اما یک پردازنده ۱۶ بیتی در همین مدت زمان می‌تواند ۱۶ بیت را مورد پردازش قرار دهد، به تعبیری سرعت پردازش ۲ برابر حالت قبل است.
گذرگاه داده یک گذرگاه دوجهته[4] است؛ یعنی داده‌ها از طریق آن هم می‌توانند به پردازنده وارد و هم از آن خارج شوند.
در بعضی پردازنده‌ها، پهنای گذرگاه داده خارجی (پینهای داده پردازنده) با پهنای گذرگاه داده داخلی (مسیر انتقال داده‌ها درون پردازنده) متفاوت است؛ مثلاً پردازنده ۸۰۸۸ دارای گذرگاه داده داخلی ۱۶ بیتی و گذرگاه داده داخلی ۸ بیتی است، در حالی که گذرگاههای داده داخلی و خارجی پردازنده هر دو ۱۶ بیتی‌اند.

[1] ***Data Bus***
[2] ***Address Bus***
[3] ***Control Bus***
[4] ***Bi-directional***

- **گذرگاه آدرس:** همانطور که گفته شد، گذرگاه داده محل مشترک عبور اطلاعات سیستم است. پرسشی که بلافاصله به ذهن می آید این است که در هر لحظه چه کسی حق استفاده از این گذرگاه مشترک را دارد؟

برای حل این مشکل، هر پردازنده دارای تعدادی پین آدرس است که با گذاشتن یک شماره روی آن، هویت وسیله ای را می تواند از گذرگاه داده استفاده کند را معلوم می کند. به مجموعه این پینها **گذرگاه آدرس پردازنده** گفته می‌شود.

نحوه استفاده از گذرگاه آدرس به این صورت است که طراح سیستم به همه **واحدهایی** که می خواهند از گذرگاه داده استفاده کنند (خانه‌های حافظه و تمام ورودیها و خروجیها)، یک شماره (**آدرس**) می دهد. در هنگام عملکرد سیستم، شماره یا آدرسی که پردازنده روی گذرگاه آدرسش قرار می دهد نشان دهنده شماره واحدی است که حق استفاده از گذرگاه داده را دارد. مثلاً اگر پردازنده‌ای ۸ خط آدرس (۸ پین آدرس) داشته باشد، با قرار گرفتن عدد 00110010 روی این پینها توسط پردازنده، به کل سیستم اعلام می شود که در این لحظه واحدی به شماره 00110010 (که ممکن است یک خانه حافظه، یک ورودی یا یک خروجی باشد) حق استفاده از گذرگاه داده (خواندن یا نوشتن) را دارد. این شماره در هنگام طراحی سیستم توسط طراح به این واحد داده شده است.

از آنجا که گذرگاه آدرس برای اعلان شماره واحدی که می تواند از گذرگاه داده استفاده کند به کار می‌رود، هر چه تعداد خطوط آن بیشتر باشد، پردازنده می‌تواند مقدار بیشتری حافظه یا تعداد بیشتری ورودی/خروجی را شناخته و از آنها استفاده کند. به‌عبارت دیگر، تعداد خطوط آدرس یک پردازنده، معیاری از تعداد واحدهایی است که می‌تواند آدرس‌دهی کند.

این تعداد واحدها همیشه توانی از ۲ است؛ به این ترتیب که یک پردازنده با n خط آدرس، می‌تواند به ۲^n واحد مختلف آدرس بدهد[۱]. مثلاً پردازنده Z80، ۱۶ خط آدرس دارد، این ۱۶ خط یا ۱۶ بیت می‌توانند $۲^{۱۶}$ یا ۶۵۵۳۶ حالت (آدرس دودویی) را بسازند، بنابراین Z80 می‌تواند تا ۶۵۵۳۶ واحد مختلف (حافظه یا ورودی/خروجی) (که هر کدام حاوی یک بایت می‌باشد) را آدرس‌دهی کند. اصطلاحاً گفته می‌شود که فضای آدرس‌دهی Z80 برابر ۶۵۵۳۶ بایت یا ۶۴ کیلوبایت است. توجه کنید که در اندازه‌گیری ظرفیت حافظه و فضای آدرس‌دهی، معنای کلمه کیلو، با سایر علوم متفاوت می باشد؛ به طوری‌که هر کیلوبایت شامل ۱۰۲۴ بایت است؛ به طور مشابه هر مگابایت برابر ۱۰۲۴ کیلو بایت، هرگیگابایت برابر ۱۰۲۴ مگابایت و هر ترابایت برابر ۱۰۲۴ گیگابایت است.

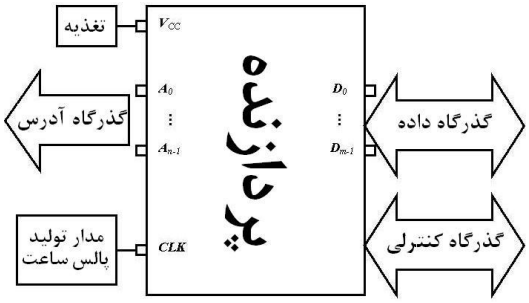
[۱] چون یک عدد دودویی با n رقم، می‌تواند ۲^n مقدار مختلف داشته باشد

پردازنده 8086/88 دارای ۲۰ خط آدرس و فضای آدرس‌دهی یک مگابایت است؛ بنابراین آدرسهای حافظه در آن ۲۰ بیتی هستند. در آینده خواهیم دید چگونه با ثباتهای آدرس‌دهی ۱۶ بیتی پردازنده و سیستم سگمنت/آفست، این آدرس ۲۰ بیتی تولید می‌شود.

چون گذرگاه آدرس فقط برای آدرس‌دهی و توسط پردازنده به کار می‌رود، بنابراین یک گذرگاه یک‌جهته[1] و سمت آن به سوی خارج پردازنده است.

- **گذرگاه کنترل:** این گذرگاه شامل پینهایی از پردازنده است که برای کنترل ارتباطات داخل سیستم مورد استفاده قرار می‌گیرد و در پردازنده‌های مختلف، نوع و تعداد آنها متفاوت است. مثلاً تعدادی از این سیگنالها مشخص می‌کنند آیا پردازنده مشغول دریافت اطلاعات است یا ارسال اطلاعات؟ دسته دیگر مشخص می‌کنند وسیله‌ای که آدرس آن توسط گذرگاه آدرس مشخص گردیده، از نوع حافظه است یا ورودی/خروجی؟ و ... .

شکل عمومی یک پردازنده با n خط آدرس و m خط داده را در شکل ۱-۹ مشاهده می‌کنید.



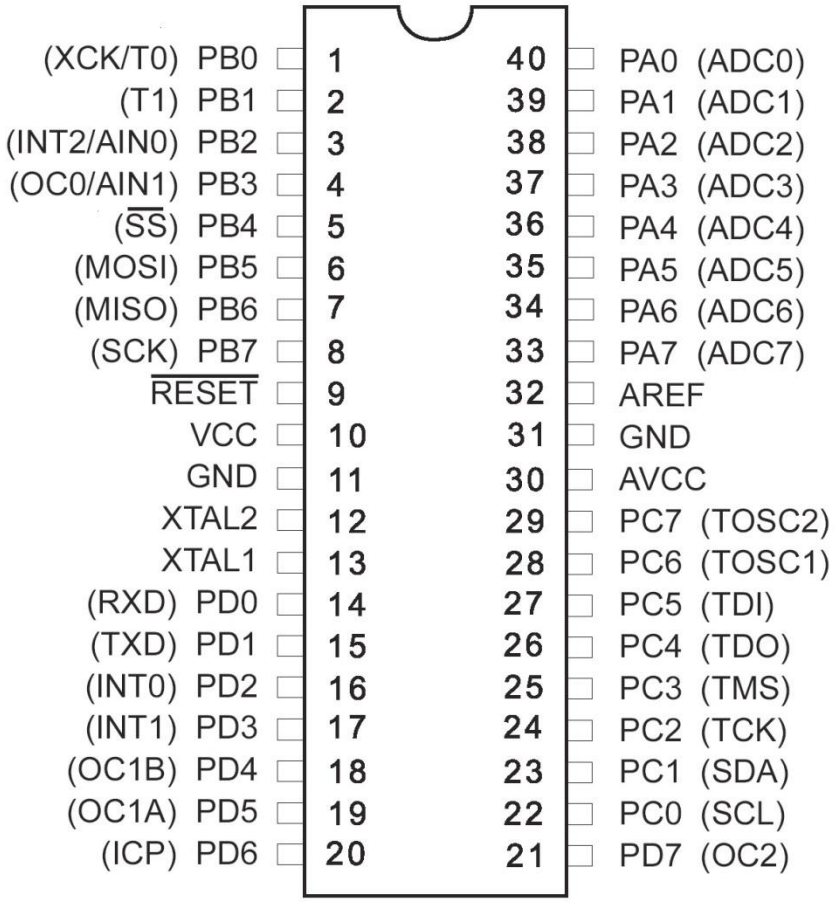
شکل ۱-۹ – شکل عمومی پینهای یک پردازنده

چنانچه به یادگیری شیوه اتصال حافظه و ورودی/خروجی به پردازنده‌ها علاقمند هستید، جزوه شماره ۲ را از وب‌سایت jrasti.com دانلود و مطالعه کنید.

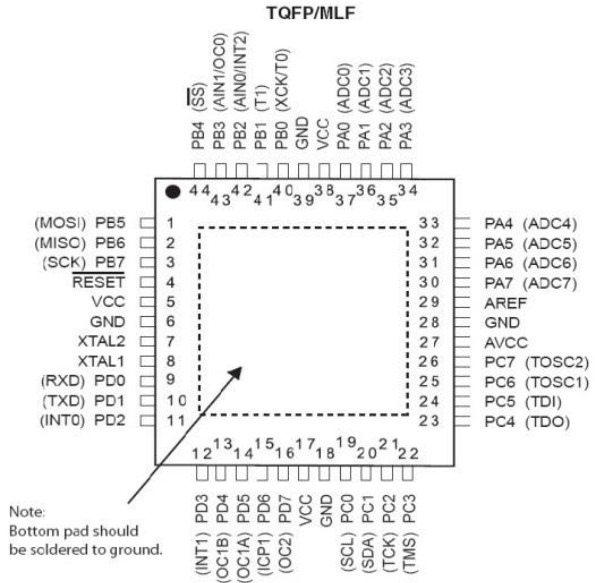
[1] ***Uni-directional***

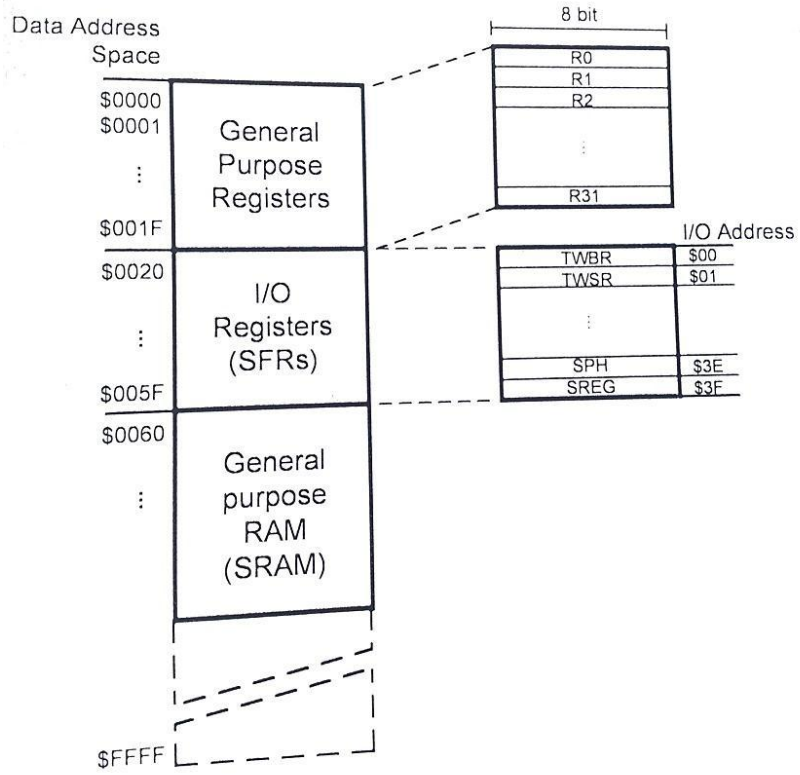
تراشه ATMEGA32

PDIP

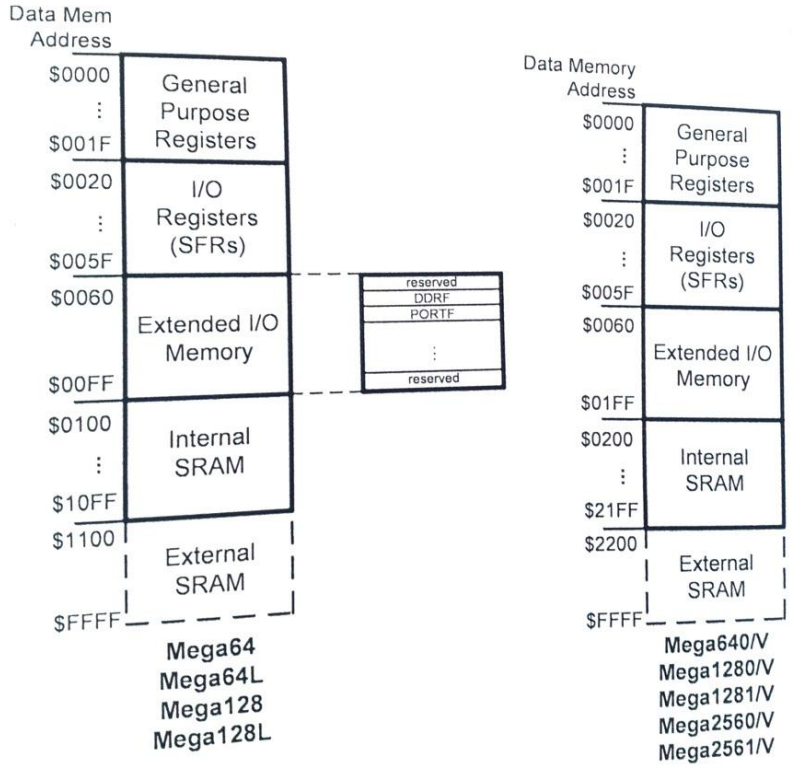


TQFP/MLF





حافظه داده برای AVRهایی که حافظه I/O توسعه یافته ندارند



حافظه داده برای AVRهایی که حافظه I/O توسعه یافته دارند.

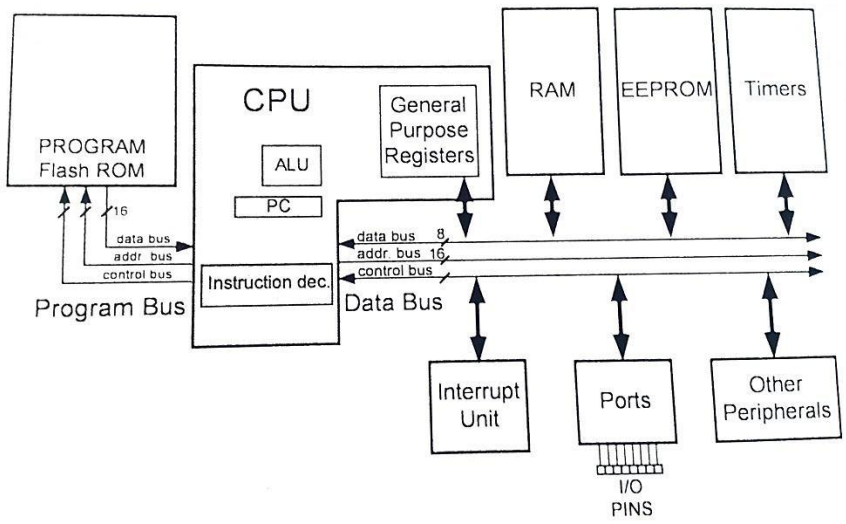
Address		Name
Mem.	I/O	
$20	$00	TWBR
$21	$01	TWSR
$22	$02	TWAR
$23	$03	TWDR
$24	$04	ADCL
$25	$05	ADCH
$26	$06	ADCSRA
$27	$07	ADMUX
$28	$08	ACSR
$29	$09	UBRRL
$2A	$0A	UCSRB
$2B	$0B	UCSRA
$2C	$0C	UDR
$2D	$0D	SPCR
$2E	$0E	SPSR
$2F	$0F	SPDR
$30	$10	PIND
$31	$11	DDRD
$32	$12	PORTD
$33	$13	PINC
$34	$14	DDRC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

Address		Name
Mem.	I/O	
$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		Name
Mem.	I/O	
$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
$5E	$3E	SREG

Note: Although Memory Address $20-$5F is set aside for IO Registers (SFR) we can access them as IO locations with addresses starting at $00

ثبات‌های I/O در ATmega32 و آدرس خانه‌های حافظه داده در آنها



معماری هاروارد در AVR

برنامه نویسی میکروکنترلر AVR به زبان C

مقدمه

برای آشنایی با معماری و ساختمان داخلی یک میکروکنترلر، بهترین زبان برنامه‌نویسی زبان سطح پایین اسمبلی است. سطح این زبان از زبان ماشین بالاتر است و کاربر را از کار با کدهای پیچیده ماشین معاف می کند؛ اما پیاده سازی برنامه های مفصل و پیچیده به زبان اسمبلی کاری دشوار و وقتگیر است که نمونه آن را می‌توان در پیاده سازی شرطها مشاهده کرد.

مزایای استفاده از زبانهای سطح بالا برای برنامه نویسی میکروکنترلرها عبارتند از :

- سادگی و صرف زمان کمتر برای نوشتن برنامه های بزرگ و پیچیده
- سهولت تغییر و به روز کردن برنامه‌ها
- قابل فهم‌تر بودن برنامه های زبانهای سطح بالا نسبت به برنامه های زبان اسمبلی
- قابلیت استفاده از توابع از پیش آماده شده کامپایلر و کتابخانه‌های موجود
- قابلیت استفاده از کد نوشته شده برای یک میکروکنترلر در میکروکنترلرهای دیگر با تغییرات ناچیز

امروزه برای برنامه نویسی میکروکنترلرها اغلب از زبانهای سطح بالا استفاده می شود که زبان C، یکی از زبان‌های متداول در این زمینه است. برنامه های زبانهای سطح بالا توسط کامپایلر به زبان ماشین قابل فهم میکروکنترلر تبدیل می شوند؛ هرچند تلاشهای زیادی برای بهینه کردن کامپایلرها در روش تبدیل دستورات سطح بالا به دستورات ماشین انجام می شود، اما فایلهای زبان ماشین تولید شده توسط کامپایلرها نسبت به فایلهای زبان ماشین تولید شده توسط اسمبلرها به نحو قابل ملاحظه‌ای بزرگتر و کندتر است. به بیان دیگر اگر برای انجام یک عملیات واحد، یک برنامه به زبان سطح بالا نوشته و توسط کامپایلر به زبان ماشین تبدیل کنیم و برنامه ای دیگر به زبان اسمبلی نوشته و با اسمبلر کد زبان ماشین معادل آن را به دست آوریم، حجم فایل زبان ماشین اول بین ۱/۵ تا دو برابر حجم فایل زبان ماشین دوم بوده و اجرای آن نیز به همین نسبت بیشتر به طول می‌انجامد. در کل می‌توان گفت برای کاربردهای معمولی که حجم کد ماشین و زمان اجرای برنامه چندان مهم نیست، استفاده از زبان سطح بالا برای برنامه نویسی میکروکنترلر نسبت به زبان اسمبلی ساده‌تر، سریعتر و کاراتر است.

در این نوشته ابتدا به مقدمات برنامه نویسی میکروکنترلر AVR با زبان C خواهیم پرداخت و سپس با ارائه تعدادی مثال کاربردی، ویژگیهای مختلف زبان C را بررسی خواهیم کرد. برای مطالعه مطالبی که در ادامه می‌آید، آشنایی با ساختارهای زبان C ضروری است.

متداول‌ترین نرم‌افزار برای برنامه‌نویسی AVR به زبان C، نرم‌افزار CodeVision است که دارای کتابخانه‌های کاربردی برای راه‌اندازی دستگاههای جانبی مختلف می‌باشد و می‌تواند به کمک برنامه‌ساز CodeWizard، قسمت اعظم کدهای مورد نیاز یک پروژه را به صورت خودکار تولید کند. برای برنامه‌نویسی‌های حرفه‌ای از این نرم‌افزار استفاده می‌شود؛ اما فاقد شبیه‌ساز اجرا و اشکالزدا می‌باشد و از این نظر وابسته به نرم‌افزارهای اشکالزدایی مانند AVRStudio است. محیط

برنامه‌نویسی IAR نیز از نسخه‌های کارآمد، اما گران‌قیمت برنامه‌نویسی میکروکنترلر AVR است. در این مطلب برای نوشتن برنامه های C، ترجمه، شبیه سازی و اشکالزدایی آنها از محیط مجتمع رایگان و کارآمد AVRStudio استفاده می کنیم. فراموش نکنید که برای این کار علاوه بر نرم‌افزار فوق باید WinAVR را نیز نصب کنید. استفاده از کتابخانه‌های avr-libc می تواند کمک خوبی در نوشتن پروژه‌های حرفه‌ای زبان C برای شما باشد. برای اطلاعات بیشتر به آدرس ذیل مراجعه کنید:

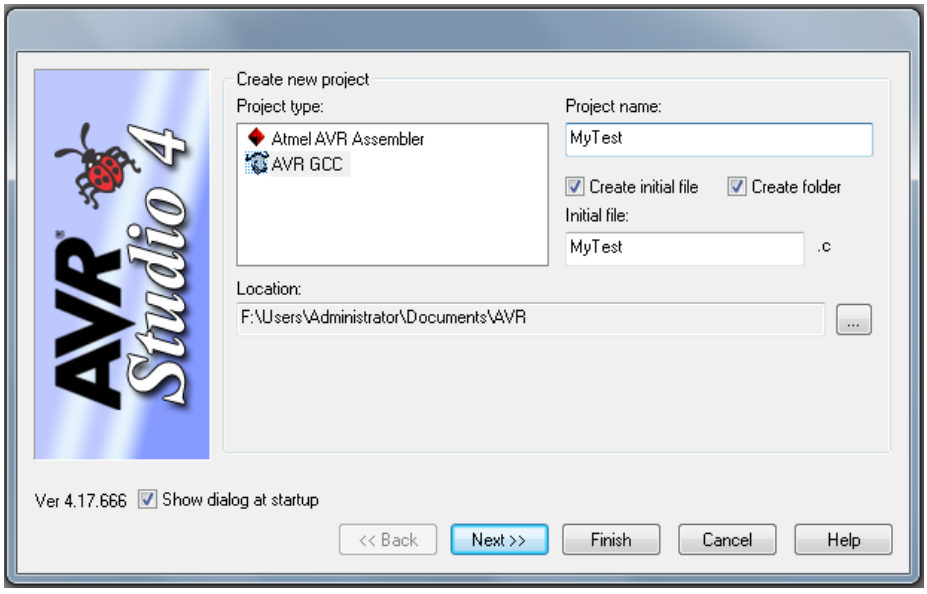
http://www.nongnu.org/avr-libc/

به علاوه در ادامه نحوه برنامه‌نویسی در Codevision و اتصال آن به AVRStudio برای اشکالزدایی را خواهیم آموخت.

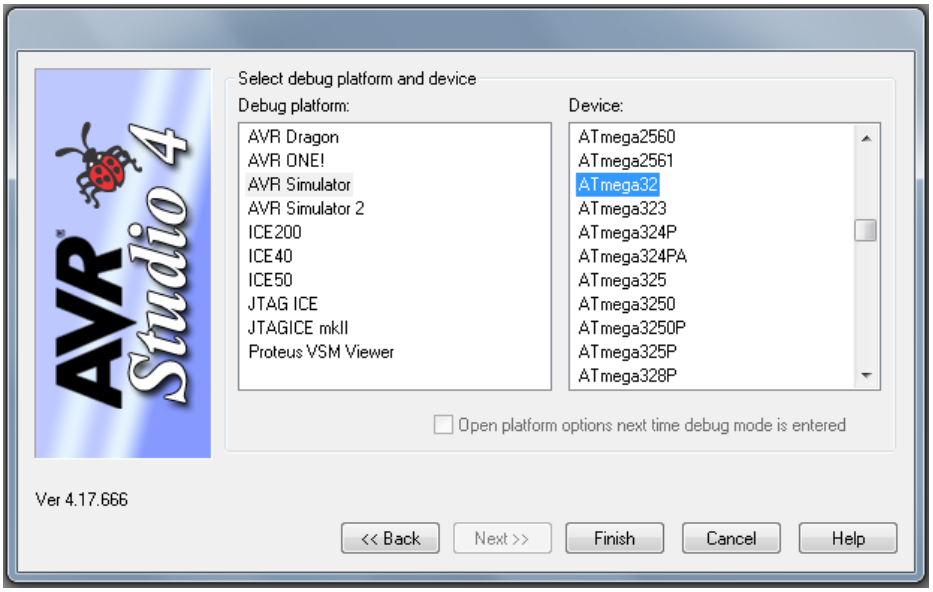
زبان C میکروکنترلر AVR کاملاً شبیه به زبان C کامپیوتر است. البته هیچگاه فراموش نکنید که یک برنامه C در یک بستر کامپیوتری و یک برنامه C میکروکنترلر AVR در بستر یک میکروکنترلر اجرا می‌شود؛ استفاده از برخی تکنیکهای سطح بالا در برنامه‌نویسی C ممکن است منجر به ایجاد کدهای ماشین بزرگ با زمان اجرای طولانی شود. در این موارد بهتر است برنامه خود را هرچه ساده تر بنویسید تا کامپایلر بتواند کد ماشین کوتاهتری از آن بسازد.

برنامه‌نویسی و اشکالزدایی در محیط AVRStudio

برای ایجاد یک پروژه زبان C در AVRStudio، از منوی Project، گزینه New Project را انتخاب کنید. در کادری که ظاهر می‌شود، از قسمت نوع پروژه گزینه AVR GCC را انتخاب کنید و نام پروژه را در سمت راست وارد کنید:

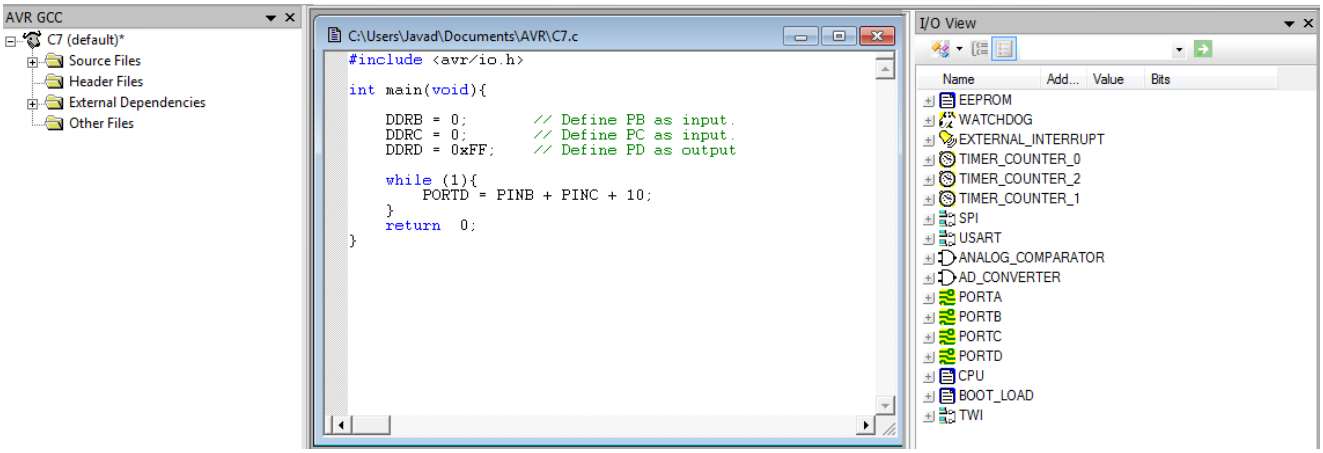


با فشردن کلید Next، کادری ظاهر می‌شود که در آن نوع بستر اشکالزدایی و میکروکنترلر مورد استفاده را مشخص می کنیم:

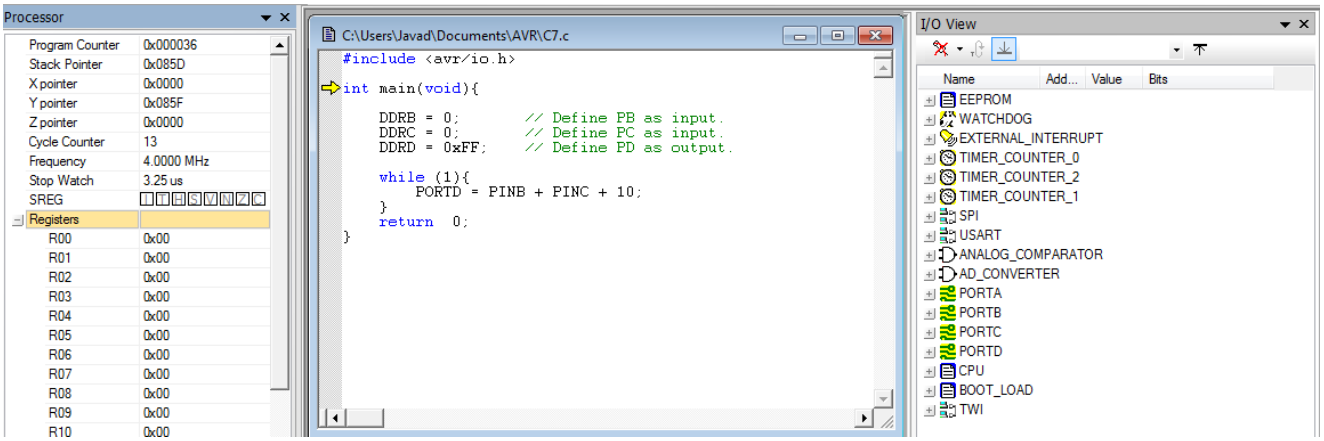


با فشردن کلید Finish پروژه ایجاد شده و یک فایل خالی برای برنامه‌نویسی ایجاد می‌شود.

اکنون برنامه زیر را در محیط ایجاد شده بنویسید.

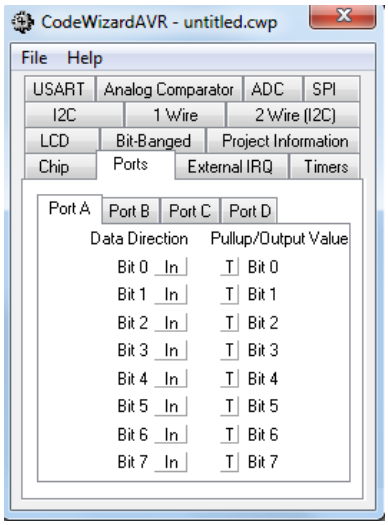


با انتخاب گزینه Build از منوی Build (یا فشردن کلید F7 یا انتخاب آیکون مربوطه از بالای صفحه)، برنامه را کامپایل کنید. در پنجره پایین صفحه، خطاهای و هشدارهای احتمالی برنامه شما دیده می‌شود. در صورتی که ترجمه با موفقیت انجام شده بود، با انتخاب گزینه Start Debugging از منوی Debug می‌توانید برنامه خود را خط‌به‌خط اجرا و نتیجه اجرا را به شکل زیر مشاهده کنید:



مقادیر ثباتها در کادر سمت چپ و مقادیر واحدهای ورودی/خروجی در کادر سمت راست دیده می‌شود. با انتخاب گزینه‌های دیگر منوی Debug می‌توانید به امکانات بیشتری برای اجرای مرحله‌به‌مرحله و اشکالزدایی برنامه خود دست یابید.

برنامه‌نویسی و اشکالزدایی در محیط CodeVision



برای ایجاد یک پروژه در محیط CodeVision، ابتدا از منوی فایل گزینه New و از کادری که ظاهر می‌شود گزینه Project را انتخاب کنید تا پروژه جدیدی گشوده شود. در کادر بعدی از شما پرسیده می‌شود آیا مایل به استفاده از برنامه‌ساز CodeWizard هستید یا خیر؟ استفاده از برنامه‌ساز تولید برنامه‌ها را بسیار ساده می‌کند؛ مثلاً با انتخاب مشخصات مورد نظرتان برای تنظیم بخش‌های مختلف میکروکنترلر، بسیاری از دستوراتی که باید توسط برنامه‌نویس نوشته شود، به صورت خودکار توسط برنامه‌ساز تولید می‌شود. عیب استفاده از CodeWizard تولید کدهای مفصل با بخش‌های بلااستفاده است که علاوه بر اشغال بی‌مورد فضای ROM میکروکنترلر، اشکالزدایی برنامه را نیز با مشکل مواجه می‌کند. اکنون که در ابتدای راه برنامه‌نویسی به زبان C برای میکروکنترلر هستیم، از استفاده از برنامه‌ساز CodeWizard صرفنظر می‌کنیم و برنامه‌ها را به صورت ساده می‌نویسیم. بنابراین از کادر اخیر گزینه No را انتخاب کنید. با تعیین نام پروژه در مرحله بعد، یک پروژه خالی ساخته شده و آماده است. در کادری که ظاهر می‌شود، زبانه C Compiler و از قسمت Chip گزینه ATmega32 را انتخاب کنید. در این کادر مشخصات دیگری از مدل برنامه‌نویسی میکروکنترلر نیز قابل تنظیم است.

اکنون مجدداً از منوی File گزینه New و این بار از کادری که ظاهر می‌شود گزینه Source را انتخاب کنید تا فایل اصلی برنامه شما ایجاد شود. کد زیر را در این فایل وارد و سپس آن را با یک نام دلخواه و پسوند c. ذخیره کنید:

```
#include <mega32.h>
void main(void){
	DDRD.0 = 1;
	while (1) {
		PORTD.0 = 0;
		PORTD.0 = 1;
	}
}
```

اکنون باید فایل برنامه ایجاد شده را وارد پروژه‌ای که در مرحله قبل ساختید بنمایید. برای این کار از منوی Project گزینه Configure را انتخاب کنید. در کادری که ظاهر می‌شود دکمه Add را بزنید و در لیستی که از فایل‌های برنامه نمایش داده می‌شود، نام فایل بالا را انتخاب کنید و سپس به تنظیمات پروژه پایان دهید (به اتفاقاتی که در کادر سمت چپ صفحه می‌افتد دقت کنید). اکنون از منوی Project گزینه Build All را انتخاب کنید. چنانچه در کادری که ظاهر می‌شود، خطا یا هشداری وجود نداشته باشد، یعنی پروژه و برنامه شما صحیح و قابل اجراست.

برای اشکالزدایی برنامه از منوی Setting گزینه Debugger را انتخاب کنید و در کادری که ظاهر می‌شود، آدرس نرم‌افزار AVRStudio را وارد کنید تا از این به بعد از این نرم‌افزار برای شبیه‌سازی و اشکالزدایی برنامه شما استفاده شود. با انتخاب گزینه Debugger از منوی Tools، نرم‌افزار AVRStudio اجرا می‌شود.

در نرم‌افزار AVRStudio از منوی File گزینه Open را انتخاب کنید. به مسیری که پروژه CodeVision شما در آن ذخیره شده بروید و فایلی که هم‌نام با پروژه شماست و پسوند cof. دارد را انتخاب کنید. اکنون AVRStudio ایجاد پروژه‌ای برای شبیه‌سازی برنامه شما را آغاز می‌کند که مراحل آن مانند آنچه در ابتدای این مطلب شرح داده شد، می‌باشد. در انتها برنامه شما قابل شبیه‌سازی و اشکالزدایی است.

در ادامه به مدل برنامه‌نویسی C در نرم‌افزارهای AVRStudio و CodeVision خواهیم پرداخت.

قالب برنامه های C

قالب کلی یک برنامه C به صورت زیر است :

```
#include <نام فایل سرآیند۱.h>
#include <نام فایل سرآیند۲.h>
...
اعلان متغیرهای عمومی
اعلان و تعریف توابع کاربر

int    main(void){

        اعلان متغیرهای محلی
```

```
    return  0;
}
```

فایلهای سرآیند[1] که به همراه کامپایلر C ارایه می شوند (یا می توانند توسط کاربر نوشته شوند)، شامل تعریف توابعی هستند که برای ساده تر شدن برنامه نویسی در اختیار کاربر قرار داده شده است؛ مثلاً در فایل سرآیند io.h که در اکثر برنامه هایی که برای AVRStudio نوشته می‌شود مورد استفاده قرار می گیرد، نام ثباتهای ورودی/خروجی، پورتها، پرچمها و دیگر اجزا داخلی AVR و نیز توابعی برای کار کردن با آنها اعلان شده است؛ با شامل[2] کردن این فایل در برنامه خود می‌توانید مستقیماً با این اجزا و توابع کار کنید.

انواع داده‌ها در C

اکثر انواع داده‌های زبان C در برنامه‌های C میکروکنترلر نیز قابل استفاده هستند؛ اما به دلیل اینکه AVR یک میکروکنترلر ۸ بیتی است و کامپایلر باید متغیرهای زبان C را به واحدهای ۸ بیتی تبدیل کند، برای تعریف متغیرهای برنامه خود سعی کنید از انواع داده زبان C به ترتیب زیر استفاده کنید تا حجم کد ماشین حاصل از برنامه های شما کوتاهتر شود:

انواع داده یک بایتی : (signed) char – unsigned char

انواع داده دوبایتی : unsigned int یا (signed) int – short int یا (signed) short

انواع داده ۴ بایتی : float – (signed) long – unsigned long

مثلاً برای تعریف شمارنده یک حلقه for که حداکثر آن ۲۰۰ است، به جای روش معمول در برنامه‌های C کامپیوتر که از داده های int استفاده می کند، از داده نوع unsigned char استفاده کنید.

نرم‌افزار CodeVision علاوه بر داده‌های استاندارد C، می‌تواند از انواع داده‌های دیگری نیز استفاده کند که در انتهای این مطلب به آنها خواهیم پرداخت.

مثالهایی از برنامه های C

اکنون تعدادی از مثالهای برنامه نویسی کاربردی به زبان C را بررسی می‌کنیم. برای هر مثال در ابتدا برنامه AVRStudio و سپس در مواقع لزوم برنامه CodeVision نوشته شده است تا با مشخصات برنامه‌نویسی در هر دو محیط آشنا شوید. توصیه می شود این مثالها که حاوی نکاتی مهم درباره برنامه نویسی به زبان C هستند را با دقت مطالعه کنید.

[1] ***Header Files***
[2] ***include***

مثال) برنامه ای بنویسید که دو عدد ۸ بیتی را بخواند، با هم جمع کند، حاصل را با عدد ۱۰ جمع کند و نتیجه را نمایش دهد.

در اینجا پورتهای PB و PC را ورودی و پورت PD را خروجی در نظر می گیریم.

```
#include <avr/io.h>

int main(void){

        DDRB = 0;          // Define PB as input.
        DDRC = 0;          // Define PC as input.
        DDRD = 0xFF;    // Define PD as output.


        while (1){
                PORTD = PINB + PINC + 10;
        }
        return  0;
}
```

در مورد این برنامه چند نکته قابل توجه است :

۱) همانطور که گفته شد، فایل سرآیند io.h حاوی تعریف و نامگذاری ثباتهای داخلی AVR و توابع دستکاری آنهاست و شامل کردن آن در ابتدای برنامه ضروری است. در این فایل کلیه اجزای داخلی AVR با حروف **بزرگ** انگلیسی نامگذاری شده است. از آنجا که زبان C به بزرگی و کوچکی حروف حساس است، در نوشتن برنامه ها به این نکته دقت کنید.

۲) در ابتدای برنامه پورتهای PB و PC به عنوان ورودی تعریف شده اند. در زبان C پیشوند **0x** قبل از یک عدد ثابت ، نشانگر بیان آن عدد در دستگاه ۱۶تایی است؛ اما اعداد دودویی مستقیماً قابل استفاده نیستند.

۳) در برنامه های C، بعد از علامت // تا انتهای خط به عنوان توضیح در نظر گرفته می‌شود. از توضیحات برای واضح کردن قسمتهای مختلف برنامه استفاده می کنیم.

۴) حلقه while (1) به دلیل درست بودن دایمی شرط ادامه حلقه، بی نهایت مرتبه اجرا می‌شود. چنانچه بخواهیم این عملیات تنها یک مرتبه اجرا شود، برنامه باید به صورت زیر بازنویسی شود :

```
#include <avr/io.h>

int main(void){
        DDRB = 0;          // Define PB as input.
        DDRC = 0;          // Define PC as input.
        DDRD = 0xFF;    // Define PD as output.
```

```
        PORTD = PINB + PINC + 10;
        while (1);
        return  0;
}
```

پس از اجرای دستورات، برنامه در حلقه بینهایت `while (1);` به دام می افتد و باقی می‌ماند. این دستور مشابه خط فرمان Here: RJMP Here است که برای قطع اجرای برنامه در انتهای برنامه های اسمبلی مورد استفاده قرار می گرفت.

برای ایجاد حلقه بینهایت، به جای دستور `while(1)`، می توان از دستور `for(;;)` نیز استفاده کرد. توجه کنید که دستور متوقف کننده اجرای برنامه برای شبیه‌سازی برنامه در محیط‌هایی که دارای برنامه ناظر (monitor) هستند (مانند AVRStudio) ضروری نیست. اما برای اجرای عملی برنامه روی میکروکنترلر باید حتماً آن را بنویسید.

نکته دیگر اینکه برنامه فوق ساده‌ترین نسخه ممکن برای عملیات مورد نظر است؛ اما چنانچه بخواهید آن را برای اشکالزدایی مناسب‌تر کنید، بهتر است به صورت زیر آن را بازنویسی کنید:

```
#include <avr/io.h>

int main(void){


        DDRB = 0;           // Define PB as input.
        DDRC = 0;           // Define PC as input.
        DDRD = 0xFF;    // Define PD as output.

        unsigned char a,b,c;
        while (1){
                a = PINB;
                b = PINC;
                c = a + b + 10;
                PORTD = c;
        }
        return  0;
}
```

برنامه CodeVision این مثال به صورت زیر نوشته می‌شود:

```
#include <mega32.h>


void main (void) {
        DDRB = 0;       // Define PB as input.
        DDRC = 0;       // Define PC as input.
        DDRD = 0xFF;    // Define PD as output.
```

```
    while (1){
        PORTD = PINB + PINC + 10;
    }
}
```

تفاوتهای عمده یکی در نام فایلهای سرآیند و دیگری در نوع خروجی تابع main است که در محیط CodeVision باید از نوع void تعریف و دستور ;0 return حذف شود.

مثال) برنامه ای بنویسید که مرتباً PB را بخواند و معکوس آن را در PC نمایش دهد.

```
#include <avr/io.h>

int main(void){
    DDRB = 0;
    DDRC = 255;
    while (1)
        PORTC = ~PINB;
    return 0;
}
```

علامت ~ باعث معکوس شدن بیتهای P1 می شود. خوانا و ساده بودن این برنامه نسبت به برنامه اسمبلی با همین هدف در مثال بالا مشخص است.

برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

مثال) برنامه ای بنویسید که دو عدد m و n را بخواند و حاصل 2m-n را در نمایش دهد.

عدد m را از پورت PD و عدد n را از پورت PB خوانده و حاصل مورد نظر را در پورت PC نمایش می‌دهیم.

```
#include <avr/io.h>
#define     m     PIND
#define     n     PINB
#define     Out   PORTC

int main(void){
    DDRD = DDRB = 0;
    DDRC = 255;
    while (1) {
        Out = 2*m - n;
    }
    return 0;
}
```

می بینید که با استفاده از رهنمود #define می توان پورتها یا ثباتها را نامگذاری و به خواناتر شدن برنامه کمک نمود. به‌علاوه این کار باعث می‌شود بتوانید برنامه‌ای را که برای میکروکنترلر AVR نوشته شده است، به راحتی روی میکروکنترلرهای دیگر که از یک زبان شبیه به C حمایت می‌کنند، اجرا کنید؛ برای این کار تنها کافی است تعاریف #define را مطابق با سخت‌افزار آن میکروکنترلر تغییر دهید. مزیت دیگر استفاده از رهنمود فوق، این است که اگر تصمیم برنامه‌نویس راجع به پورت‌های ورودی و خروجی تغییر کرد، نیازی نیست کل برنامه را تغییر دهد و می‌تواند تنها با اصلاح این رهنمود تغییر مورد نظرش را اعمال کند.

برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

مثال) برنامه ای بنویسید که بیتهای پورت PC را یکی‌درمیان مرتباً خاموش و روشن کند.

```
#include <avr/io.h>

void MyDelay(void){
        unsigned int i;
        for (i = 0;i < 50000;i++);
}
int main(void){
        DDRC = 255;
        PORTC = 0xAA;
        while (1) {
                MyDelay();
                PORTC = ~PORTC;
        }
        return 0;
}
```

در این برنامه از تابع MyDelay که شامل یک حلقه بدون دستور است، برای ایجاد تأخیر استفاده شده است (با اسیلوسکوپ یا به کمک نرم‌افزار شبیه‌ساز Proteus این تأخیر را اندازه بگیرید). البته در اکثر کامپایلرها برای اجرای صحیح این برنامه، باید سطح بهینه‌سازی (Optimization Level) کامپایلر را به صفر تغییر دهید؛ وگرنه کامپایلر حلقه for تأخیر را به جهت اینکه کاری انجام نمی‌دهد، به عنوان دستور زائد تشخیص داده و حذف می‌کند.

راه دیگر ایجاد تأخیر، استفاده از تایمرهاست که در ادامه خواهیم دید. به علاوه در فایل سرآیند delay.h توابع تأخیر _delay_ms و _delay_us برای ایجاد تأخیرهای برحسب میلی‌ثانیه و میکروثانیه تعریف شده‌اند.

در محیط CodeVision نیز همین فایل سرآیند و توابع وجود دارند؛ فقط کاراکتر _ درابتدای نام تابع حذف شده است.

برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

مثال) برنامه‌ای بنویسید که مقادیر صفر تا ۱۰۰ را مرتباً با تأخیر زمانی ۵۰۰ میلی‌ثانیه به پورت PC ارسال کند.

```
#include <avr/io.h>
#include <util/delay.h>

int main(void){
	DDRC = 255;
	while (1) {
		unsigned char i;
		for (i = 0; i <= 100;i++){
			PORTC = i;
			_delay_ms(500);
		}
	}
	return 0;
}
```

برنامه محیط CodeVision به صورت زیر است:

```
#include <mega32.h>
#include <delay.h>


void main(void){
	DDRC = 255;
	while (1) {
	    unsigned char i;
	    for (i = 0; i <= 100;i++){
		PORTC = i;
		delay_ms(500);
	    }
      }
}
```

مثال) برنامه‌ای بنویسید که مرتباً PB را بخواند و تا زمانی که صفر نشده، عدد ۱۰۰ را در PC بنویسد. با صفر شدن PB، عدد ۲۰۰ در PC نوشته شده و برنامه به پایان می‌رسد.

```
#include <avr/io.h>
int main(void){
	DDRB = 0;
	DDRC = 255;
	while (PINB != 0) {
		PORTC = 100;
	}
	PORTC = 200;
	while (1);
	return 0;
}
```

برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

مثال) برنامه ای بنویسید که مرتباً عددی بین صفر تا ۹ را از PB بخواند و آن را در نمایشگر 7-segment آند مشترک متصل به PC نمایش دهد. اگر عدد خوانده شده بزرگتر از ۹ باشد، نمایشگر باید خاموش شود.

```
#include <avr/io.h>
int main(void){
unsigned char index,Seven_Seg_Codes[ ] = {0x03,0x9F,0x25,0x0D ,0x99,
0x49,0x41,0x1F,0x01,0x09};


        DDRB = 0;
        DDRC = 255;
        while (1) {
                if (PINB >= 10)
                        PORTC = 0xFF; // Turn off the Common Anode 7-Seg
                else {
                        index = PINB;
                        PORTC = Seven_Seg_Codes[index];
                }
        }
        return 0;
}
```

فرض بر این است که پینهای a و b و ... و h نمایشگر هفت‌قسمتی، به ترتیب به پینهای PC7 و PC6 و ... و PC0 میکروکنترلر متصل شده‌اند.

کامپایلر برای ذخیره متغیرهای تعریف شده در برنامه مانند آرایه Seven_Seg_Codes از فضای RAM میکروکنترلر استفاده می‌کند. چون محتویات این آرایه در طول برنامه تغییر نمی‌کند، در بعضی کامپایلرها با تعریف آن به صورت const (یا flash در CodeVision) می‌توان آن را در حافظه ROM قرار داد.

برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

پرسش) نشان دهید برنامه قبلی را به صورت ذیل هم می‌توان نوشت:

```
#include <avr/io.h>
int main(void){
        DDRB = 0;
        DDRC = 255;
        for (;;) {
                switch(PINB){
                        case 0  : PORTC = 0x03;
                                  break;
                        case 1  : PORTC = 0x9F;
```

```
                        break;
                case 2  : PORTC = 0x25;
                        break;
                case 3  : PORTC = 0x0D;
                        break;
                case 4  : PORTC = 0x99;
                        break;
                case 5  : PORTC = 0x49;
                        break;
                case 6  : PORTC = 0x41;
                        break;
                case 7  : PORTC = 0x1F;
                        break;
                case 8  : PORTC = 0x01;
                        break;
                case 9  : PORTC = 0x09;
                        break;
                default : PORTC = 0xFF;
                        break;


        }
    }
    return 0;
}
```

مثال) برنامه ای بنویسید که زوج پورت PA-PB را با عدد ۱۶ بیتی 127A H جمع کند و حاصل را در زوج پورت PC-PD را نمایش دهد.

```
#include <avr/io.h>
int main(void){
    DDRA = DDRB = 0;        // Define as input
    DDRC = DDRD = 0xFF; // Define as output


    for (;;){
        unsigned char carry = 0;
        if (PINB + 0x7A > 255)
            carry = 1;
        PORTD = PINB + 0x7A;
        PORTC = PINA + 0x12 + carry;
    }
    return 0;
}
```

برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

مثال) برنامه ای بنویسید که رشته *This is a message* را به PD ارسال کند.

```
#include <avr/io.h>
#include <util/delay.h>

int main(void){

	DDRD = 255;
	const unsigned char Table[18] = "This is a message";
	unsigned char i;
	for (i = 0;i < 17;i++){
		PORTD = Table[i];
		_delay_ms(500);
	}

	for (;;);
	return 0;
}
```

دقت کنید رشته ۱۷ کاراکتر دارد و با لحاظ کردن کاراکتر NULL انتهای رشته، طول آن ۱۸ کاراکتر خواهد بود. اما تنها ۱۷ کاراکتر اول آن باید به پورت D ارسال شود.

برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

مثال) برنامه ای بنویسید که PB را بخواند و مجموع ارقام دهدهی آن را در PC بنویسد.

```
#include <avr/io.h>
int main(void){
	DDRB = 0;
	DDRC = 255;
	while (1){
		unsigned char n = PINB,sum = 0;
		sum = sum + n % 10; // or sum += n % 10;
		n = n / 10;
		sum = sum + n % 10;
		sum = sum + n / 10;
		PORTC = sum;
	}
	return 0;
}
```

این برنامه را از نظر خوانایی با نسخه اسمبلی آن مقایسه کنید. برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

برنامه‌نویسی بیتی

زبان C محیط AVRStudio دستورات بیتی ندارد و باید از دستورات دستکاری بیتی زبان C استاندارد و قواعد زیر برای برنامه‌نویسی بیتی استفاده کرد:

برای «یک» کردن بیت iام یک عدد بدون تغییر بقیه بیت‌ها، آن عدد را با الگویی OR می‌کنیم (با استفاده از عملگر | در زبان C) که در آن الگو، بیت iام «یک» و بقیه بیت‌ها «صفر» هستند. این الگو «ماسک یک» نامیده می‌شود.

برای «صفر» کردن بیت iام یک عدد بدون تغییر بقیه بیت‌ها، آن عدد را با الگویی AND می‌کنیم (با استفاده از عملگر & در زبان C) که در آن الگو، بیت iام «صفر» و بقیه بیت‌ها «یک» هستند. این الگو «ماسک صفر» نامیده می‌شود.

برای معکوس کردن بیت iام یک عدد بدون تغییر بقیه بیت‌ها، آن عدد را با الگویی XOR می‌کنیم (با استفاده از عملگر ^ در زبان C) که در آن الگو، بیت iام «یک» و بقیه بیت‌ها «صفر» هستند. این الگو «ماسک معکوس» نامیده می‌شود.

مثال) برنامه‌ای بنویسید که LED متصل به PB4 را بدون تغییر بقیه بیت‌های این پورت خاموش و روشن کند.

```
#include <avr/io.h>

void MyDelay(void){
	unsigned int i;
	for (i = 0;i < 55000;i++);
}

int main(void){

	DDRB  = DDRB | 0x10;

	while (1) {
		PORTB = PORTB | 0x10;
		MyDelay();
		PORTB = PORTB & 0xEF;
		MyDelay();
	}
	return 0;
}
```

برای «یک» کردن PB4، ثبات PORTB را با 00010000 یا 0x10، OR و برای «صفر» کردن این بیت، پورت مذکور را با 11101111 یا 0xEF، AND می‌کنیم.

محیط CodeVision دستورات بیتی دارد و به همین لحاظ نوشتن این گونه برنامه‌ها در این محیط بسیار ساده است. برنامه زیر را ببینید:

```
#include <mega32.h>

void MyDelay(void){
        unsigned int i;
        for (i = 0;i < 55000;i++);
}

void main(void){

        DDRB.4  = 1;

        while (1) {
            PORTB.4 = 1;
               MyDelay();
               PORTB.4 = 0;
               MyDelay();
        }
}
```

توجه کنید دستورات بیتی CodeVision برای آدرس‌دهی بیتی ثباتهای I/O در بازه آدرسی صفر تا 0x1F معتبر هستند.

پرسش) نشان دهید حلقه اصلی برنامه را می‌توان با استفاده از ماسک تغییر به صورت زیر بازنویسی کرد:

```
        while (1) {
                PORTB = PORTB ^ 0x10;
                MyDelay();
        }
```

می‌توان از دستور ;PORTB.4 = ~PORTB.4 نیز در محیط CodeVision استفاده کرد.

مثال) برنامه‌ای بنویسید که یک رقص نور روی پورت PC ایجاد کند.

```
#include <avr/io.h>
#include <util/delay.h>
int main(void){

        DDRC = 0xFF;
```

```
	for (;;) {
		unsigned char i;
		unsigned char a = 1;
		for (i = 1; i <= 7;i++){
			PORTC = a;
			_delay_ms(500);
			a = a << 1;
		}
	}
		return 0;
}
```

عملگر >> برای شیفت به چپ استفاده می‌شود. دستور a = a << 1 مقدار a را یک واحد به چپ شیفت داده و مقدار جدید را در متغیر a قرار می‌دهد.

برنامه CodeVision این مثال با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

پرسش) برنامه بالا را طوری اصلاح کنید که الف) جهت رقص نور را عوض کند ب) LED روشن دوتا دوتا جلو برود.

مثال) برنامه‌ای بنویسید که هرگاه کلید فشاری متصل به PB.0 فشرده («صفر») شود، یک واحد به محتویات PC اضافه کند.

```
#include <avr/io.h>
#include <util/delay.h>
void Debounce(void);
int main(void){

	DDRB = DDRB & 0xFE;
	DDRC = 0xFF;


	while (1){
	  while (PINB & 0x01); // Wait until push button is pressed.
	  Debounce();
	  PORTC = PORTC + 1;
	  while ((PINB & 0x01) == 0); // Wait until push button is released.
	  Debounce();
	}
	return 0;
}
////////////////////////////////////////////////////////////////////////////
void Debounce(void){
	_delay_ms(100);
}
```

در ابتدا باید منتظر بمانیم تا کلید فشرده شود. سپس به محتویات PC یک واحد اضافه کرده و منتظر می‌مانیم تا کلید رها شود. بعد از تشخیص فشرده یا رها شدن کلید، به کمک تابع Debounce تأخیر کوتاهی ایجاد می‌کنیم تا لرزش‌های کلید از بین برود و برنامه را در تشخیص فشرده یا رها شدن کلید دچار اشتباه نکند.
برنامه CodeVision را خودتان بنویسید.

پرسش) برنامه قبلی را طوری اصلاح کنید که هرگاه کلید فشاری متصل به PB.0 فشرده («صفر») شود، محتویات PC را در PD بنویسد.

مثال) برنامه‌ای بنویسید که حالت دو درب که به پین‌های صفر و یک پورت PB متصل هستند را کنترل کند؛ اگر یکی از آنها باز شود چراغ متصل به PC0 و اگر هر دو باز شود آژیر متصل به PC1 روشن شود. مدار تشخیص باز و بسته بودن درب در حالت باز بودن درب، سیگنال منطقی "یک" تولید می کند.

```
#include <avr/io.h>

int main(void){

	DDRB = DDRB & 0xFC;
	DDRC = DDRC | 0x03;

	while (1){
		if ((PINB & 0x03) == 0x03){
			PORTC = PORTC | 0x02;
			PORTC = PORTC & 0xFE;
		}
		else if (0 < (PINB & 0x03) && (PINB & 0x03) < 0x03){
			PORTC = PORTC | 0x01;
			PORTC = PORTC & 0xFD;
		}
		else if ((PINB & 0x03) == 0){
			PORTC = PORTC & 0xFC;
		}
	}
	return 0;
}
```

برنامه CodeVision را خودتان بنویسید.

مثال) برنامه ای بنویسید که PC و PD را بخواند و اگر PC > PD، PB.6 را روشن و در غیر این صورت خاموش کند.

```
#include <avr/io.h>

int main(void){

	DDRC = DDRD = 0;
	DDRB = DDRB | 0x40;

	for (;;) {
		if (PINC > PIND)
			PORTB = PORTB | 0x40;
		else
			PORTB = PORTB & 0xBF;
	}
	return 0;
}
```

برنامه CodeVision را خودتان بنویسید.

مثال) برنامه ای بنویسید که بیتهای زوج و فرد PC را به ترتیب روشن و خاموش کند. تأخیر مابین روشن و خاموش شدن بیتها با عددی که از PB خوانده می شود تنظیم می‌گردد.

تاکنون فقط از توابع بدون پارامتر استفاده کرده ایم. این مثال از یک تابع پارامتردار استفاده می‌کند.

```
#include <avr/io.h>
#include <util/delay.h>
void MyDelay(unsigned char time); // 'time' is parameter of Delay function.
int main(void){

	DDRC = 0xFF;
	DDRB = 0;

	while (1) {
		unsigned char t = PINB;
		PORTC = 0x55;
		MyDelay(t);
		PORTC = 0xAA;
		MyDelay(t);
	}
	return 0;
}
////////////////////////////////////////////////////////////////////////////
void MyDelay(unsigned char time){
	_delay_ms(50*time);
}
```

همانگونه که مشاهده می کنید، الگوی تابع MyDelay قبل و بدنه آن بعد از تابع main تعریف شده است تا برنامه ساخت‌یافته‌تر شود.

برنامه CodeVision را خودتان بنویسید.

استفاده از دستورات اسمبلی در پروژه‌های C

گاهی برای کاهش طول کد ماشین یا زمان اجرای برنامه، در نقاط بحرانی یک نرم افزار سطح بالا از کدهای اسمبلی استفاده می کنیم. کامپایلر AVR GCC این قابلیت را دارد که در یک پروژه به صورت همزمان با فایلهای c و اسمبلی کار کند. برای اطلاعات بیشتر به مراجع AVR GCC مراجعه کنید. در CodeVision می‌توانید دستورات اسمبلی را بین عبارات #asm و #endasm بنویسید.

تعیین محل ذخیره داده‌ها در CodeVision

در CodeVision می‌توان محل ذخیره انواع داده را تعیین کرد. با تعریف متغیرها به صورت عادی، فضای مورد نیاز آنها از حافظه‌ی SRAM تأمین می‌شود. مثال زیر یک متغیر عمومی نوع int را در خانه‌ی شماره 80h حافظه SRAM ذخیره می کند:

```
int x @0x80;
```

با تعریف داده به صورت register به کامپایلر فرمان داده می‌شود تا متغیر را در ثبات‌های میکروکنترلر ذخیره کند. البته حتی در صورت استفاده نکردن از این عبارت هم کامپایلر ممکن است این کار را انجام دهد. اگر از کلمه volatile قبل از تعریف داده استفاده کنید، آن متغیر در ثبات‌ها ذخیره نخواهد شد.

با تعریف متغیر به صورت const یا flash، داده مزبور به صورت ثابت در حافظه ROM و با تعریف متغیر به صورت eeprom، داده مزبور در حافظه EEPROM میکروکنترلر ذخیره خواهد شد.

اگر یک متغیر از نوع bit تعریف شود، در یکی از بیتهای ثباتهای R2 تا R14 جایابی می‌شود و فقط می‌تواند مقادیر صفر یا یک را بپذیرد:

```
bit  x = 1;
```

پرسشهای دوره ای

۱) برنامه ای بنویسید که مقادیر صفر تا ۲۵۵ را مرتباً به PB ارسال کند.

۲) برنامه ای بنویسید که یک رمز ۸ بیتی را بخواند و اگر برابر 10111001 بود، یک درب را باز کند و برنامه به پایان برسد. از یک کلید فشاری برای ورود رمز استفاده کنید. کاربر تا ۳ مرتبه فرصت دارد رمز ورودی را به درستی وارد کند، وگرنه یک آژیر به صدا در خواهد آمد.

۳) برنامه ای بنویسید که PC را بخواند و اگر بزرگتر از ۱۰۰ بود آن را به PA و در غیر این صورت به PB ارسال کند.

۴) الف) برنامه‌ای بنویسید که محتویات پورت PB را به صورت سریال با شروع از LSB روی PC5 ارسال کند.

ب) برنامه ای بنویسید که بایتهای سریال ارسال شده با برنامه قسمت **الف** را روی پین PD.0 میکروکنترلر مقصد دریافت کند.

۵) برنامه‌ای بنویسید که توازن فرد پورت PD را روی بیت PA.7 نمایش دهد.

۶) برنامه‌ای بنویسید که محتویات پورت PB را به صورت دهدهی روی سه نمایشگر هفت‌قسمتی که به پورت PC متصل هستند، به صورت Refreshing نمایش دهد.

۷) برنامه‌ای بنویسید که حاصل PB / 8 را تا دو رقم اعشار روی سه نمایشگر هفت‌قسمتی که به پورت PC متصل هستند، به صورت Refreshing نمایش دهد.