جزوه درس ریزپردازنده

مرجع:

میکروکنترلرهای AVR

تألیف:

مهندس جابر الوندی

تهیه کننده:

فصل ۱

ساختار داخلی میکروکنترلرهای AVR

اهداف

آشنایی با میکروکنترلر Atmega16

معرفی فیوز بیت های میکروکنترلر

آشنایی با پورت های ورودی و خروجی

ارائه منبع تغذیه میکروکنترلر با ورودی AC و DC

آشنایی با ساختار داخلی میکرو کنترلر Atmega16

معرفی کلاک سیستم و انواع منابع پالس ساعت در میکروکنترلر

آشنایی با مدهای Sleep و معرفی تایمر نگهبان (Watchdog)

۱-۱ تعاریف اولیه از ساختار میکروکنترلر

در این قسمت می خواهیم شما را با چند تعریف ساده از اجزای یک میکروکنترلر آشنا کنیم تا ذهن شما آماده پذیرش مطالب در ادامه آموزش کتاب باشد.

انواع حافظه ماندگار(دائمی):

حافظه های ماندگار به حافظه هایی گفته می شود که بتوانند اطلاعات داده شده را نگه دارند حتی اگر تغذیه آنها قطع شود نباید این اطلاعات پاک شود.

حافظه PROM: این نوع حافظه فقط خواندنی، تنها می تواند یکبار برنامه ریزی شود.

حافظه EPROM: حافظه فقط خواندنی است که اطلاعات توسط مدار واسط، با ولتاژ الکتریکی نوشته و با نور ماورای بنفش مثل نور خورشید توسط پنجره شیشه ای که معمولاً با یک بر چسب پوشیده می شود پاک می گردد.

حافظه EEPROM: حافظه فقط خواندنی است که اطلاعات توسط مدار واسط، با ولتاژ الکتریکی نوشته و پاک می شود این نوع حافظه دو مدل است:

الف- موازی(Parallel): آدرس دهی، نوشتن و خواندن حافظه به صورت موازی انجام می گیرد.

ب- سریال(Serial): این حافظه اغلب در بسته بندی های کوچک ۸ پایه قرار می گیرند و آدرس دهی حافظه و آدرس دهی سخت افزاری و همچنین عمل نوشتن و خواندن به صورت سریال و فقط توسط ۲ پایه انجام می گیرد به این نوع ارتباط دهی I2C گفته می شود.

انواع حافظه فرار(غیر دائمی):

حافظه های فرار به حافظه هایی گفته می شود که بتوانند اطلاعات داده شده را نگه دارند اما بر خلاف حافظه های ماندگار، اگر تغذیه آنها قطع شود اطلاعات نیز پاک می شود.

SRAM: این نوع حافظه از نوع Static یا پایدار است. منظور از پایداری این است که نیازی به تازه سازی ندارد و تا وقتی که تغذیه آن بر قرار است می تواند اطلاعات داده شده را حفظ نماید.

DRAM: این نوع حافظه از نوع Dynamic یا غیر پایدار است. یعنی CPU باید مدام با یک فصله زمانی مشخص، دیتای ذخیره شده در این نوع حافظه را بازسازی کند این قبیل از حافظه ها با ظرفیت های بالا ساخته می شوند و بیشتر در کنار ریز پردازنده هایی با سرعت بالا قرار می گیرند.

پورت های ورودی و خروجی:

منظور از Port یا در گاه این است که یک سری پایه برای ارتباط با دنیای بیرون تراشه فراهم شده است که هم می توانند ورودی و هم خروجی باشند. هر میکروکنترلر با توجه به نوع بسته بندی دارای یک الی چندین پورت است و الزاما یک پورت دارای ۸ پایه نیست.

Buffer (تقویت کننده جریان):

تراشه هایی به نام بافر وجود دارند که ما در راه اندازی جریان بیشتر از ۲۰ میلی آمپر از آنها استفاده می کنیم مانند: 74HC245و 74HC244 و ULN2003

سیکل ماشین:

CPU و سایر قسمت های میکروکنترلر برای انجام هر دستور به میزان خاصی زمان نیاز دارند که به آن سیکل ماشین (پالس ساعت) گفته می شود.

(فرکانس اسیلاتور÷1) = سیکل ماشین

تایمر یا کانتر: Timer & counter

timer : برای زمان سنجی دقیق از تایمر استفاده می شود. تایمر، پالس ساعت خود را از کلاک سیستم دریافت می کند و شروع به شمارش می کند.

counter :برای شمارش پالس های بیرونی استفاده می شود. کانتر، پالس ساعت خود را از پایه Tn دریافت و شمارش می کند.

به طور مثال برای ساختن یک ساعت دیجیتال به تایمر و برای شمارش شیشه های نوشابه عبوری از جلوی سنسور به کانتر نیاز داریم.

وقفه ها:

وقفه به معنی قطع کردن برنامه جاری و سرویس دادن به تابع وقفه است و نه تاخیر زمانی. برای این که میکروکنترولر بتواند علاوه بر برنامه جاری به سایر قسمت ها یا المان دیگری سرویس بدهد باید از وقفه استفاده کنیم. اجزای جانبی CPU مانند: تایمر و کانتر، مبدل ADC، مقایسه کننده آنالوگ، ارتباط سریال، TWI و ... دارای وقفه مخصوص به خود هستند.

ارتباط سریال USART:

این ارتباط دهی برای تبادل اطلاعات بین دو سیستم بوده و ممکن است این ارتباط بین دو میکروکنترلر یا یک میکروکنترلر با PC و یا یک میکروکنترلر با هر تراشه دیگری که دارای این پروتکل باشد، بر قرار شود. در این ارتباط دهی، اطلاعات ارسالی و دریافتی بر روی ۲ خط صورت می گیرد.

مبدل آنالوگ به دیجیتال(ADC):

برای تبدیل ولتاژ خوانده شده از یک المان با یک سنسور، از مبدل آنالوگ به دیجیتال استفاده می کنیم. برخی از میکروکنترلرهای AVR دارای مبدل ADC با دقت حداکثر ۱۰ بیت هستند.

مبدل دیجیتال به آنالوگ(DAC):

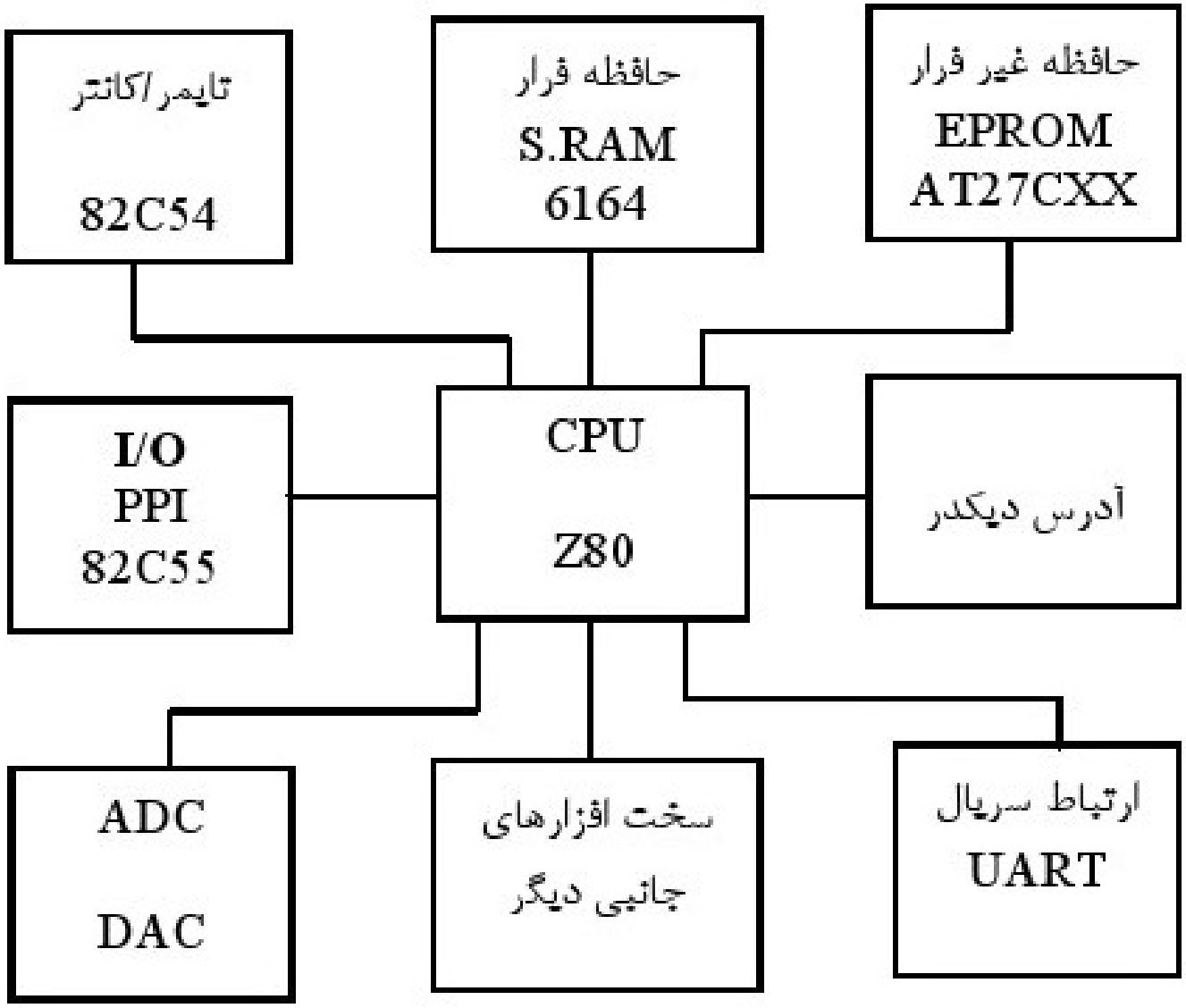
برای این که بتوانیم ولتاژ متغیر در خروجی، توسط میکروکنترلر ایجاد کنیم باید از مبدل دیجیتال به آنالوگ استفاده کنیم. مانند: (تولید موج سینوسی) لازم به ذکر است که میکروکنترلرهای AVR فاقد مبدل DAC هستند اما از ویژگی PWM تایمرهای آن می توان DAC را شبیه سازی کرد.

سنسور (Sensor):

المان یا وسیله ای که یک پارامتر فیزیکی مانند: دما، فشار، رطوبت، گاز، مادون قرمز، میدان مغناطیسی و ... را به پارامتر الکتریکی (ولتاژ یا جریان) تبدیل می کند.

۲-۱ مقدمه ای بر میکروکنترلرها

با پیشرفت علم و تکنولوژی در الکترونیک تراشه هایی به عنوان میکروپروسسورها طراحی و تولید شدند تا قبل از سال ۱۹۷۱ میلادی اگر شخص طراح، سیستمی را می خواست طراحی کند باید سیستم مورد نظر خود را به شرکت های سازنده میکروپروسور ارائه می داد تا طراحی و ساخته شود و یا اینکه مجبور بود با استفاده از آی سی های دیجیتالی، سیستم مورد نظر خود را طراحی کند. از این پس شرکت های سازنده میکروپروسسورها، از جمله شرکت Zilog تصمیم به ساخت میکروپروسسوری نمود که بتوان آن را در اختیار کاربر قرار داد و به هر صورت ممکن که می خواهد سیستم مورد نظر خود را طراحی کند و به همین دلیل میکروپروسسور Z80 را به بازار عرضه کرد و نرم افزا کامپایلر به زبان اسمبلی و پروگرامر آن را نیز ارائه داد. به طور کلی اگر یک شخص از میکروپروسسور ۸ بیتی Z80 برای سیستمی استفاده کند، باید المان های جانبی CPU را نیز علاوه بر سخت افزار سیستم مورد نظر، در کنار میکروپروسسور Z80 قرار دهد.



شکل 1-1 بلوک دیاگرام یک CPU به همراه اجرای جانبی آن

همان طور که در بلوک دیاگرام شکل ۱-۱ مشاهده می نمایید، برای اینکه از یک میکروپروسسور، حتی برای ساده ترین سیستم بخواهیم استفاده کنیم، باید از المان های جانبی دیگری نیز بهره بگیریم. این عمل سبب افزایش قیمت و پیچیده شدن سخت افزار پروژه مورد نظر می گردد. از این پس شرکت های سازنده قطعه، به فکر تراشه ای بودند که تمام امکانات جانبی میکروپروسسور را به همراه خود CPU داشته باشد، تا شخص طراح با قیمت مناسب و سخت افزار کمتر، بتواند سیستم مورد نظر خود را بسازد.
در سال ۱۹۸۱ میلاد شرکت Intel تراشه ای را به عنوان میکروکنترلر خانواده 8051 به بازار عرضه کرد. این میکروکنترلر دارای CPU ۸ بیتی، تایمر یا کانتر، وقفه، تبادل سریال، حافظه SRAM و حافظه غیر فرار(FLASH) داخلی می باشد. در اوایل، میکروکنترلر 8051، از نوع حافظه PROM یعنی نوع OTP (One Time Programmable) بهره می برد، این نوع میکروکنترلر فقط یک بار قابل برنامه ریزی بود. بعداً که این میکروکنترلر توسعه یافت، از حافظه های EPROM استفاده کردند. این نوع میکروکنترلر با شماره 87Cxx آغاز می شد و حسن این حافظه در این بود که می توانست توسط پنجره شیشه ای که در بالای تراشه قرار داشت، در مجاورت نور ماورای بنفش مثل نور خورشید قرار گرفته و بعد از چند دقیقه پاک شود. شرکت سازنده میکروکنترلر 8051، تصمیم به استفاده از حافظه ای گرفت که بتواند با ولتاژ الکتریکی نوشته و پاک شود(حافظه Flash)، این سری با شماره 89Cxx آغاز می شود که امروزه نیز همچنان مورد استفاده قرار می گیرد.

شرکت های سازنده دیگری، تحت لیسانس شرکت Intel از میکروکنترلر 8051 تولید کردند، از جمله این شرکت ها می توان به شرکت Atmel اشاره کرد. این شرکت بعداً نوع توسعه یافته 8051 را با سری شماره AT89Sxx ارائه کرد، تفاوت نوع S با نوع C در نحوه ی برنامه ریزی میکروکنترلر است،زیرا نوع s را می توان داخل مدار، پروگرام کرد. برای پروگرام کردن نوع S نیازی به خارج کردن میکروکنترلر از مدار نداریم و می توانیم در داخل سیستم، توسط یک آی سی بافر، عمل پروگرام کردن را از طریق پورت LPT انجام دهیم. سرانجام شرکت Atmel نیز خانواده AVR را در سال ۱۹۹۷ میلادی به بازار عرضه نموده است.

خانواده AVR به سه سری تقسیم می شوند:

۱. سری AT90S ۲. سری ATtiny ۳. Atmega

۱. **سری AT90S :** این سری، اعضای کلاسیک خانواده AVR را تشکیل می دهند و قابلیت های کمتری نسبت به دو سری بعدی دارند و کمتر استفاده می

AT90S4434	AT90S8535	AT90S4433	AT90S2323	AT90S1200
AT90S8534	AT90S4414	AT90S8515	AT90S2343	AT90S2313

جدول 1-1 میکروکنترلرهای سری AT90S

2. سری ATtiny:

این میکروکنترلرها در ابعاد کوچک 20,8 و 28 پایه هستند و قابلیت های خوبی نسبت به سری اول دارند و بیشتر در سیستم هایی که نیاز به پورت بالا نیست استفاده می شوند. یکی از اعضای ۸ پایه این سری ATtiny85 است که دارای امکانات خوبی از جمله مبدل ADC می باشد.

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

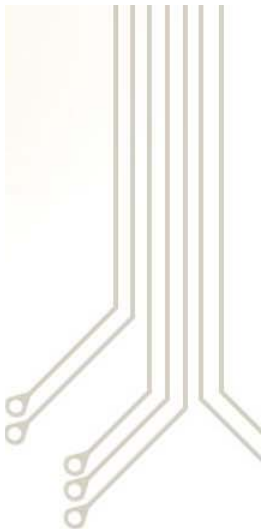
جدول 1-2 میکروکنترلرهای سری ATtiny

۳.سری ATmega:

این سری از میکروکنترلرهای AVR، امکانات بیشتری نسبت به دو سری قبلی دارند و توجه مخاطبان را به خود جلب نموده اند. میکروکنترلرهای ATmega8، ATmega32 و ATmega128 را می توان به عنوان میکروکنترلرهای پرکاربرد این سری نام برد.

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

جدول 1-3 میکروکنترلرهای سری ATmega



معماری میکروکنترلرهای AVR (RISC):

به طور کلی دو نوع معماری برای ساخت میکروکنترلرها وجود دارد:

۱. **معماری CISC** (Complex Instruction Set Computer)

تاریخچه این نوع معماری به قبل از سال ۱۹۸۰ میلادی بر می گردد. اکثر میکروپروسسورها و میکروکنترلرهای قدیمی از این نوع معماری، در آنها استفاده شده است. در این معماری تعداد دستورات بیشتر و پیچیده تر است اما برنامه نویسی آن به خصوص اسمبلی ساده تر شده است و از طرفی سرعت اجرایی دستورات پایین تر است.

۲. **معماری RISC** (Reduced Instruction Set Computer)

در این نوع معماری تعداد دستورات کاهش پیدا کرد و از طرفی سرعت اجرایی دستورات ۱۰ برابر نسبت به معماری قبلی افزایش یافت و برنامه نویسی به زبان اسمبلی را قدری پیچیده و سخت کرد اما با وجود ساختار بهینه شده میکروکنترلرهای AVR با حافظه های ظرفیت بالا و همچنین استفاده از معماری RISC، امکان برنامه نویسی به زبان های سطح بالاتر مانند C و بیسیک فراهم گردید.

تفاوت معماری RISC با معماری CISC

۱. تعداد و اندازه دستورات در RISC کمتر از CISC است در نتیجه سرعت RISC بیشتر است.
۲. در معماری RISC انتقال داده از رجیستر به رجیستر و یا حافظه به حافظه صورت نمی گیرد.
۳. تعداد رجیسترها در معماری RISC بیشتر است.
۴. برنامه نویسی به زبان اسمبلی در معماری RISC پیچیده تر از CISC است.
۵. اکثر دستورات در معماری RISC در یک کلاک سیکل اجرا می شوند.
۶. مصرف توان معماری CISC بیشتر از RISC است.
۷. برنامه های MSDOS در معماری RISC قابل اجرا نیست همین دلیل باعث شده است تا اکثر میکروپروسورها (نظیر 80X86 ساخت شرکت Intel) از معماری CISC استفاده کنند.

تفاوت میکروپروسسورها با میکروکنترلرها:

میکروپروسسورها فقط یک پردازنده کوچک هستند و باید المان های جانبی نظیر ROM،RAM، تایمر یا کانتر، وقفه و سایر المان های جانبی در کنار میکروپروسسور قرار داده شود تا بتوان یک سیستم چند منظوره را طراحی کرد. به طور مثال CPU کامپیوتر یک میکروپروسسور است و PC یک سیستم چند منظوره است که می تواند چندین عمل را اجرا کند ولی میکروکنترلرها دارای المان های جانبی محدود در یک بسته بندی نظیر FLASH SRAM تایمر یا کانتر، وقفه و غیره هستند و فقط می توانند در سیستم های تک منظوره استفاده شوند. میکروکنترلر به معنی کنترل کننده کوچک است به طور مثال کنترل دمای یک دستگاه صنعتی توسط میکروکنترلر یک سیستم تک منظوره است. میکروپروسسورها از معماری CISC استفاده می کنند اما میکروکنترلرهای پیشرفته از جمله AVR و PIC از معماری RISC استفاده می کنند البته میکروکنترلرهای قدیمی تر از جمله 8051 از معماری CISC استفاده می کند و اساسی ترین تفاوت میکروپروسسورها انعطاف پذیری آنهاست که می توان RAM و ROM آنها را افزایش داد اما در میکروکنترلرها میزان ظرفیت حافظه ها ثابت است.

۳-۱ خصوصیات میکروکنترلر ATmega16

- ✓ قابلیت اجرایی بالا و توان مصرفی پایین
- ✓ **معماری پیشرفته RISC**
- ✓ ۱۳۱ دستورالعمل قدرتمند که تنها در یک کلاک سیکل اجرا می شوند
- ✓ دارای ۳۲ رجیستر ۸ بیتی همه منظوره
- ✓ سرعتی حداکثر تا ۱۶ مگاهرتز برای نوع ATmega16
- ✓ **حافظه غیر فرار برنامه و دیتا**
- ✓ 16k بایت حافظه Flash با قابلیت خود برنامه ریزی
- ✓ تحمل حافظه Flash: قابلیت 10.000 بار نوشتن و پاک کردن
- ✓ 512 بایت حافظه EEPROM داخلی قابل برنامه ریزی
- ✓ تحمل حافظه EEPROM: قابلیت 100.000 بار نوشتن و پاک کردن
- ✓ 1k بایت حافظه SRAM داخلی
- ✓ قفل برنامه Flash و EEPROM جهت حفاظت از برنامه نوشته شده

- ✓ **قابلیت ارتباط دهی JTAG(.IEEE Std)**
- ✓ حمایت از اشکال زدایی تراشه داخل سیستم
- ✓ قابلیت برنامه ریزی Lock bits ,Fuse Bits, EEPROM,FLASH از طریق JTAG

ویژگی های جانبی:

- ❖ دو تایمر یا کانتر ۸ بیتی با مقسم فرکانسی مجزا و مد مقایسه ای
- ❖ یک تایمر یا کانتر ۱۶ بیتی با مقسم فرکانسی مجزا و دارای مد مقایسه ای و مد Capture
- ❖ RTC با اسیلاتور مجزا
- ❖ دارای 4 کانال PWM (میتواند برای DAC استفاده شود)
- ❖ 8 کانال مبدل آنالوگ به دیجیتال 10 بیتی
- ❖ 8 کانال Single-ended (سیگنالی که نسبت به زمین سنجیده می شود)
- ❖ دارای 7 کانال تفاضلی فقط برای بسته بندی نوع TQFP

- دارای 2 کانال تفاضلی با قابلیت برنامه ریزی گین 1x،10x و 200x
- ارتباط سریال دو سیمه (Tow Wire)
- USART سریال قابل برنامه ریزی
- ارتباط سریال SPI به صورت Master / Slave
- دارای تایمر Watchdog قابل برنامه ریزی با اسیلاتور مجزا داخلی
مقایسه کننده آنالوگ داخلی

ویژگی های خاص میکروکنترلر

- Reset فعال در موقع وصل تغذیه با قابلیت برنامه ریزی آشکارساز Brown-Out
- دارای اسیلاتور RC کالیبره شده داخلی
- دارای منابع وقفه داخلی و خارجی
- دارای شش مد Sleep:

Idle, ADC Noise Reduction, Power-save, Power-down, Standby Extended Standby

- **پایه های ورودی و خروجی و نوع بسته بندی**
- 32 خط ورودی و خروجی قابل برنامه ریزی
- 40 پایه PDIP، 44 پایه TQFP و 44 پایه MLF
- **ولتاژهای عملیاتی**

2.7v تا 5.5v برای ATmega16L

4.5v تا 5.5v برای ATmega16

- **فرکانس های کاری**

0 تا 8MHZ برای ATmega16L

0 تا 16MHZ برای ATmega16

- **مصرف توان با شرایط 1MHZ,3V,25°C برای ATmega16L**

در حالت فعال 1.1mA

توان در مد 0.35mA:Idle

توان در مد Power-down : کمتر از 1μA

۴-۱ فیوز بیت های میکروکنترلر ATmega16

فضای اختصاص داده شده به فیوز بیت ها، از نوع حافظه ماندگار است. فیوز بیت ها برای تنظیمات خاصی استفاده می شوند و با پاک کردن میکروکنترلر از بین نمی روند و تغییر آنها فقط از طریق پروگرامر امکان پذیر است و برای تنظیم آنها نیاز به برنامه نویسی خاصی نداریم و موقع پروگرام کردن توسط ابزار نرم افزار Code VisionAVR آنها را تنظیم و برنامه ریزی می کنیم. فیوز بیت ها با 0 برنامه ریزی و با 1 غیر فعال می شوند. توجه کنید که برنامه ریزی فیوز بیت ها باید قبل از قفل کردن تراشه صورت گیرد. میکرو

فیوز بیت OCDEN(On Chip Debug Enable)

موقعی که فیوز بیت ارتباط دهی JTAG فعال شده باشد و برنامه میکروکنترلر را قفل نکرده باشیم می توانیم با فعال کردن فیوز بیت OCDEN برنامه میکروکنترلر را به طور آنلاین در حین اجرا توسط مدار واسط که از ارتباط سریال JTAG استفاده می کند و توسط نرم افزار AVR Studio مشاهده کنیم. به این نوع آنالیز امولاتور (Emulator) یا شبیه ساز سخت افزاری گفته می شود. لازم به ذکر است که فعال کردن این فیوز بیت مصرف توان میکروکنترولر را افزایش می دهد. (این فیوز بیت به طور پیش فرض غیر فعال است.)

فیوز بیت JTAGEN

با فعال کردن این فیوز بیت می توان میکروکنترلر را از طریق ارتباط دهی استاندارد JTAG برنامه یزی کرد.(این فیوز بیت به طور پیش فرض فعال است)

نکته مهم:

چون پایه های ارتباط دهی JTAG در میکروکنترلر ATmega16 بر روی PC2 تا PC5قرار دارد باید در زمانی که ما از این ارتباط دهی استفاده نمی کنیم آن را غیر فعال کنیم، در غیر اینصورت نمی توانیم از پایه های PC2 تا PC5 استفاده کنیم.

Fuse High Byte	Bit No.	Description	Default Value
OCDEN[4]	7	Enable OCD	1(unprogrammed, OCD disabled)
JTAGEN	6	Enable OCD	0(programmed, JYAG enabled)
SPIE[1]	5	Enable SPI Serial Program and Data Downloading	0(programmed, SPI prog. Enabled)
EESAVE	4	Oscillator options	1(programmed)
BOOTSZ1	3	EEPROM memory is preserved through the Chip Erase	1(programmed, EEPROM not preserved)
BOOTSZ0	2	Select Boot Size (see Table 100 for details)	0(programmed)[3]
BOOTSZ0	1	Select Boot Size (see Table 100 for details)	0(programmed)[3]
BOOTRST	0	Select reset vector	1(unprogrammed)

جدول 1-4 بایت بالایی فیوز بیت های ATmega16

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out Detector trigger level	1(unprogrammed)
BODEN	6	Brown-out Detector enable	1(unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1(unprogrammed)(1)
SUT0	4	Select start-up time	0(programmed)(1)
CKSEL3	3	Select Clock source	0(programmed)(2)
CKSEL2	2	Select Clock source	0(programmed)(2)
CKSEL1	1	Select Clock source	0(programmed)(2)
CKSEL0	0	Select Clock source	1(unprogrammed)(2)

جدول 1-5 بایت پایینی فیوز بیت های ATmega16

فیوز بیت SPIEN

اگر این فیوز بیت فعال باشد می توان میکروکنترلر را از طریق ارتباط دهیSPI برنامه ریزی کرد. این فیوز بیت در نرم افزار codeVisionAVR قابل دسترس نیست.
(فیوز بیت SPIEN به طور پیش فرض فعال است)

فیوز بیت CKOPT

با فعال کردن این فیوز بیت، می توانیم از حداکثر دامنه نوسان اسیلاتور خارجی استفاده کنیم. اگر این فیوز بیت فعال باشد، خروجی اسیلاتور به صورت Rail-to-Rail کار می کند. یعنی دامنه نوسانات برابر تغذیه میکروکنترلر می شود. این حالت در مکان هایی که نویز دارند موثر است ولی از سویی باعث افزایش توان مصرفی در میکروکنترلر می شود.

(این فیوز بیت به طور پیش فرض غیر فعال است)

فیوز بیت EESAVE

در موقعی که ما میکروکنترلر را پاک می کنیم EEPROM نیز پاک می شود. اگر بخواهیم در موقع پاک شدن حافظه Flash از محتوای حافظه EEPROM محافظت کنیم باید این فیوز بیت را فعال کنیم.
(این فیوز بیت به طور پیش فرض غیر فعال است)

فیوز بیت های BOOTSZ0 و BOOTSZ1

این دو فیوز بیت میزان حافظه اختصاص داده شده BOOT را تعیین می کنند و برنامه ریزی آنها طبق جدول ۱-۶ می باشد. (به طور پیش فرض هر دو فیوز بیت فعال هستند)

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application section	Boot Reset Address(start Boot Loader Section)
1	1	128 words	2	$0000-$1F7F	$1F80-$1FFF	$1F7F	$1F80
1	0	256 words	4	$0000-$1EFF	$1F00-$1FFF	$1EFF	$1F00
0	1	512 words	8	$0000-$1DFF	$1E00-$1FFF	$1DFF	$1E00
0	0	1024 words	16	$0000-$1BFF	$1C00-$1FFF	$1BFF	$1C00

جدول 1-6 تعیین ظرفیت حافظه Boot

فیوز بیت BOOTRST

این فیوز بیت برای انتخاب بردار Reset است اگر غیر فعال باشد بردار Reset از آدرس 0X0000 حافظه خواهد بود اما اگر این فیوز بیت فعال شود به ابتدای آدرسی که فیوز بیت های BOOTSZ1 و BOOTSZ0 طبق جدول ۱-۶ تعیین کرده اند پرش می کند. (این فیوز بیت به طور پیش فرض غیر فعال است)

فیوز بیت BODEN

برای فعال کردن مدار Brown-out باید این فیوز بیت فعال شود. مدار Brown-out آشکار ساز ولتاژ تغذیه است که اگر از 2.7 یا 4 ولت کمتر شود میکروکنترلر را Reset می کند. (این فیوز بیت به طور پیش فرض غیرفعال است)

فیوز بیت BODLEVEL

اگر فیوز بیت BODEN فعال شده باشد و فیوز بیت BODLEVEL برنامه ریزی نشده باشد آنگاه با کاهش ولتاژ VCC کمتر از 2.7v میکروکنترلر Reset می شود و اگر فیوز بیت BODLEVEL فعال شود آنگاه با کاهش ولتاژ VCC کمتر از 4v میکروکنترلر Reset می شود. (این فیوز بیت به طور پیش فرض غیرفعال است)

فیوز بیت های SUT0 و SUT1

این دو فیوز بیت، زمان شروع (Start-up) را در موقع وصل تغذیه طبق جدول ۱-۷ تعیین می کنند.(به طور پیش فرض SUT0 فعال و SUT1 غیرفعال است)

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{cc}=5.0v)	Recommended Usage
0	00	258 CK(1)	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK(1)	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK(2)	-	Ceramic resonator, BOD enabled
0	11	1K CK(2)	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK(2)	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	-	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

جدول 1-7 تنظیم فیوز بیت های Start-up

فیوز بیت های CKSEL3, CKSEL2, CKSEL1, CKSEL0

توسط این فیوز بیت ها نوع و مقدار فرکانس اسیلاتور را تعیین می کنیم.
به طور پیش فرض فیوز بیت CKSEL0 غیر فعال و بقیه فعال هستند یعنی فرکانس 1MHZ داخلی انتخاب شده است.
اگر بخواهیم فرکانس کاری اسیلاتور داخلی را تنظیم کنیم این فیوز بیت ها را طبق جدول ۱-۸ تنظیم می کنیم و اگر بخواهیم از کریستال خارجی استفاده کنیم باید این فیوز بیت ها را طبق جدول ۱-۹ در حالت یک یعنی غیر فعال قرار دهیم. در تنظیم این فیوز بیت ها دقت نمایید به طور مثال اگر اشتباهی تمام این فیوز بیت ها را فعال کنیم طبق جدول ۱-۹ مد کلاک خارجی انتخاب می شود که در این حالت میکروکنترلر نه با نوسان ساز داخلی و نه با کریستال خارجی کار می کند بلکه توسط کلاک خارجی که به پایه XTAL1 اعمال می شود کار می کند. همچنین توجه کنید اگر شما از کریستال خارجی برای میکروکنترلر خود استفاده می نمایید باید حتماً موقع پروگرامر کردن نیز کریستال به میکروکنترلر وصل باشد ولی در حالت استفاده از نوسان ساز داخلی نیازی به قرار دادن کریستال بیرونی ندارید.

CKSEL3..0	Nominal Frequency (MHz)
0001[1]	1.0
0010	2.0
0011	4.0
0100	5.0

1- میکرو کنترلر با مقدار پیش فرض 1MHZ تنظیم شده است.

جدول 1-8 تنظیم نوسان ساز کالیبره شده داخلی

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111.1010
External Low-frequency Crystal	1001
External RC Oscillator	1000.0101
Calibrated Internal RC Oscillator	0100.0001
External Clock	0000

جدول 1-9 تعیین منبع کلاک سیستم

نکته: اگر به طور تصادفی فیوز بیت ها را اشتباه تنظیم کردید و با قرار دادن کریستال خارجی، میکروکنترلر توسط پروگرامر شناسایی نشد، یک فرکانش 1MHZ توسط یک میکروکنترلر دیگری به پایه XTAL1 میکروکنترلر مذکور اعمال کنید و توسط پروگرامر، فیوز بیت ها را صحیح تنظیم نمائید.

۵-۱ پورت های ورودی و خروجی میکروکنترلر ATmega16

هر میکروکنترلر AVR بسته به نوع قابلیت و بسته بندی که دارد دارای یک سری پایه های ورودی و خروجی است. ممکن است عموما یک پورت دارای ۸ پایه نباشد به طورمثال پورت c میکروکنترلر ATmega8 دارای ۵ پایه می باشد. پورت های تمام میکروکنترلرهای AVR می تواند به صورت ورودی وخروجی عمل کنند. درحالت اولیه Reset تمام ورودی وخروجی ها را در حالت Tri-state قرار می گیرند. Tri-state یعنی حالتی که پایه پورت امپدانس بالا می باشد. همچنین تمامی پایه های پورت ها مجهز به مقاومت بالاکش (pull-up) داخلی هستند که می توانند درحالت ورودی فعال شوند. از آن جایی که جریان دهی پورت های میکروکنترلرهای قدیمی نمی توانستند حتی یک LED را روشن کنند شرکت های سازنده میکروکنترلرهای جدید سعی کرده اند که جریان دهی پایه های پورت ها را افزایش دهند. بافر Latch داخلی میکروکنترلرهای AVR می تواند در حالت جریان دهی (Source) و جریان کشی (Sink) در حالت فعال، جریانی تا 20mA را تامینکند البته در حالت حداکثر می تواند جریان 40mAرا تحمل کند. بنابراین به راحتی می تواند یک LED و یا سون سگمنت را جریان دهی کند.

به طورکلی تمام پورت های میکروکنترلرهای AVR دارای ۳ رجیستر تنظیم کننده به فرم زیر هستند:

۱-DDRx.n:

این رجیستر(data direction register) برای تنظیم هر پایه به عنوان ورودی و خروجی در نظرگرفته شده است. اگر بیتی از این رجیستر یک شود نشان دهنده تعیین آن پایه به عنوان خروجی و اگر صفر شود آن پایه ورودی خواهد بود.

مثال:

```
DDRA=0xFF;          //تمام پایه های پورت Aبه عنوان خروجی
DDRA=0X00;          // تمام پایه های پورت Aبه عنوان ورودی
DDRC.0=1,           //  پایه PC.0ازپورتCبه عنوان خروجی
DDRC.0=0 ,          //  پایه PC.0ازپورتCبه عنوان ورودی
```

۲-PORTx.n:

این رجیستر(port Data Register) برای ارسال دیتا به خروجی می باشد. هرموقع میکروکنترلر بخواهد داده ای را به خروجی بفرستد باید ابتدا رجیستر DDRx.n در حالت خروجی تنظیم شده باشد و سپس داده مورد نظر در رجیستر PORTx.n قرار می گیرد.

مثال:

```
DDRB=0XFF;          // تمام پایه های پورت B به عنوان خروجی
PORTB=46;           عدد ۴۶ دسیمال به خروجی پورت B ارسال می گردد.
```

۳-PINx.n:

این رجیستر(Port Input Pin Adress) برای دریافت دیتا از ورودی است. هرگاه میکروکنترلر بخواهد داده ای را از ورودی بخواند باید ابتدا ریجستر DDRx.n در حالت ورودی تنظیم شده باشد و سپس داده مورد نظر از رجیستر PINx.n به صورت بیتی توسط دستورهای شرطی خوانده می شود. همچنین خواندن به صورت بایتی، با یک متغیر ۸ بیتی انجام می شود.

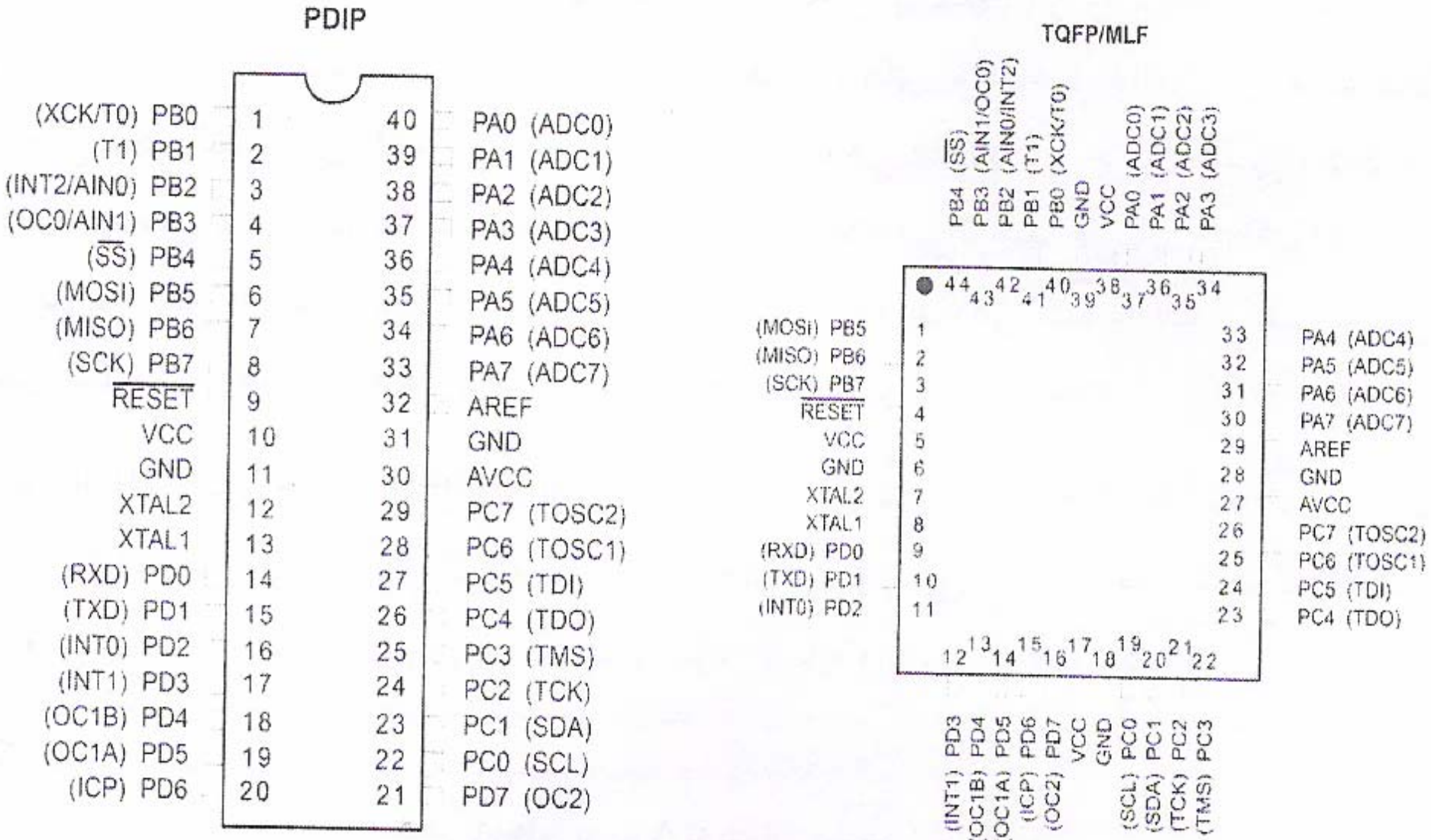
مثال:

```
PORTC.5=1;          // فعال کردن مقاومت pull-up داخلی پایه PC
DDRC.5=0;           // تعیین پایه PC5 به عنوان ورودی

// اگر پایه PC5 برابر صفر شد دستورالعمل ها اجراشوند.
IF(PINC.5==0){
  دستورالعمل ها;
}
```

مثال:

```
DDRC=0X00;          //تعیین تمام پایه های پورت C به عنوان ورودی
Data=PINC;          // Data خواندن دیتا از پورت C و قرار دادن آن در متغیر
```

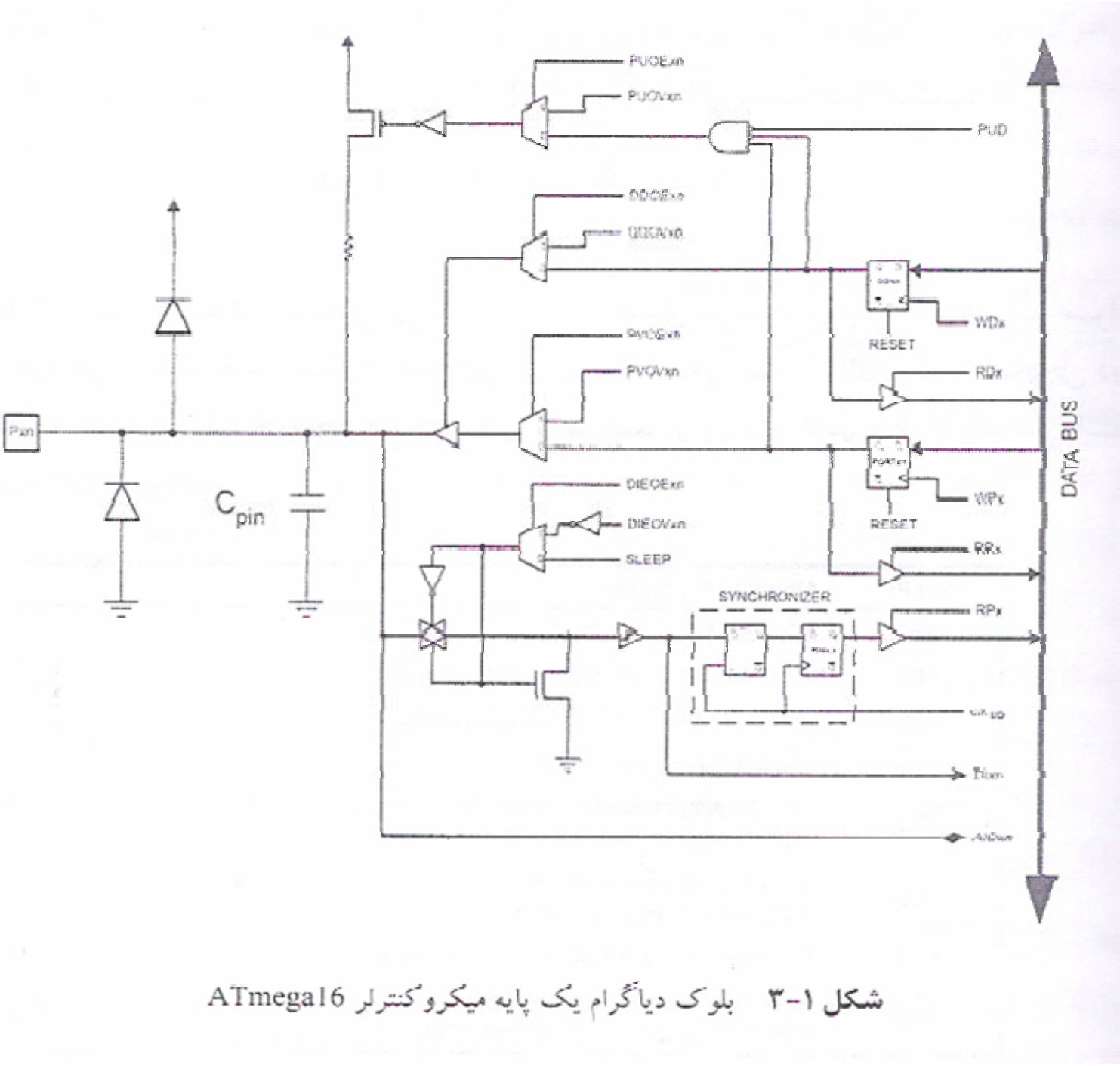


شکل ۱-۲ فرم بسته‌بندی ATmega16 با ترکیب PDIP و TQFP/MLF

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

جدول ۱-۱۰ نحوه‌ی پیکربندی پورت‌ها

همان طور که در شکل ۱-۳ پیداست، هر پایه از پورت میکروکنترلر، به وسیله دو دیود که به VCC و GND وصل شده اند در برابر الکتریسیته ساکن محافظت می شود و توسط یک خازن داخلی فرکانس های بالا و مخرب که باعث اثرات ناخواسته می شوند خنثی می شود.



شکل ۱-۳ بلوک دیاگرام یک پایه میکروکنترلر ATmega16

کاربردهای دیگر پورت های میکروکنترلر ATmega16

پورت A

پایه های PA0 تا PA7 پورت A را تشکیل می دهند. در حالت عادی می توان از این پایه ها به عنوان ورودی و خروجی استفاده کرد.

کاربرد بعدی پایه های پورت A، به عنوان ورودی مالتی پلکسر مبدل آنالوگ به دیجیتال می باشد. توجه کنید در صورتی که شما به طور مثال از کانال ADC0 استفاده می کنید، می توانید از دیگر پایه های پورت A برای کاربردهای دیگر به عنوان ورودی و خروجی استفاده کنید. ولی بهتر است در هنگام عمل تبدیل مبدل آنالوگ به دیجیتال داده ای به خروجی پورت A ارسال نشود. برای آشنایی بیشتر با عملکرد دوم این پورت می توانید به فصل مبدل آنالوگ به دیجیتال مراجعه کنید.

Port Pin	Alternate Function
PA7	ADC7 (ADC input channel 7)
PA6	ADC6 (ADC input channel 6)
PA5	ADC5 (ADC input channel 5)
PA4	ADC4 (ADC input channel 4)
PA3	ADC3 (ADC input channel 3)
PA2	ADC2 (ADC input channel 2)
PA1	ADC1 (ADC input channel 1)
PA0	ADC0 (ADC input channel 0)

جدول ۱-۱۱ کاربردهای دیگر پورت A

پورت B

پایه های PB0 تا PB7 پورت B را تشکیل می دهند. در حالت عادی می توان از این پایه ها به عنوان ورودی و خروجی استفاده کرد. کاربرد بعدی این پورت ارتباط دهی سریال SPI، وقفه خارجی دو، ورودی مقایسه کننده آنالوگ، خروجی مد مقایسه ای تایمر صفر و ورودی کانتر صفر و یک می باشد که به توضیح آن ها می پردازیم.

Port Pin	Alternate Functions
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	$\overline{SS}$ (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negative Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Positive Input) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

جدول ۱-۱۲ کاربردهای دیگر پورت B

پایه (XCK/T0) PB0

اگر کانتر صفر استفاده شود ورودی کانتر صفر پایه T0 خواهد بود. همچنین اگرUSART در مد سنکرون کارکند، به عنوان خروجی کلاک همزمان کننده USART عمل می کند.

پایه (T1) PB1

اگر کانتر یک استفاده شود ورودی کانتر یک پایه T1خواهد بود.

پایه (INT2/AINO) PB2

اگر وقفه خارجی دو توسط رجیسترهای مربوطه فعال شود آن گاه پایه INT2 به عنوان ورودی وقفه خارجی دو عمل می کند. همچنین اگر مقایسه کننده آنالوگ داخلی فعال شده باشد پایه AIN0 به عنوان ورودی مثبت OPAMP داخلی عمل می کند.

پایه (OC0/AIN1) PB3

در صورتی که از مد مقایسه ای تایمر صفر استفاده کنیم و خروجی مقایسه ای فعال شده باشد پایه OC0 به عنوان خروجی مد مقایسه ای تایمر صفر و در مد PWM تایمر صفر به عنوان خروجی سیگنال PWMتولید شده عمل می کند. همچنین اگر مقایسه کننده آنالوگ داخلی فعال شده باشد پایه AINI به عنوان ورودی منفی OPAMP داخلی عمل می کند.

پایه (SS) PB4

در شرایطی که از ارتباط دهی SPI استفاده کنیم و میکروکنترلر در حالت Slave باشد پایه SS به عنوان ورودی انتخاب Slave عمل می کند.

پایه (MOSI) PB5

اگر از ارتباط دهی سریال SPI استفاده کنیم پایه MISI به عنوان خروجی در حالت Master و به عنوان ورودی درحالت Slave عمل می کند. در حالت ورودی برای Slave تنظیم DDRB.5 تاثیری بر عملکرد پایه ندارد.

پایه (MISO) PB6

اگر از ارتباط دهی سریال SPI استفاده کنیم پایه MISO به عنوان ورودی در حالت Master و به عنوان خروجی در حالت Stave عمل می کند در حالت ورودی برای Master تنظیم DDRB.6 تأثیری بر عملکرد پایه ندارد.

پایه (SCK) PB7

در ارتباط دهی SPI پایه SCK به عنوان خروجی Master و به عنوان ورودی کلاک Slave عمل می کند. زمانی که Slave انتخاب شود این پایه ورودی خواهد بود و اگر Master انتخاب شود باید توسط DDRB.7 جهت داده تنظیم شود.

پورت C

پایه های PC0 تا PC7 پورت C را تشکیل می دهند. در حالت عادی می توان از این پایه ها به عنوان ورودی و خروجی استفاده کرد. عملکـرد بعـدی این پورت ارتباط دهی استاندارد JTAG و کریستال پـالس سـاعت واقعـی RTC تایمر دو است.

Port Pin	Alternate Function
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC1 (Timer Oscillator Pin 1)
PC5	TDI (JTAG Test Data In)
PC4	TDO (JTAG Test Data Out)
PC3	TMS (JTAG Test Mode Select)
PC2	TCK (JTAG Test Clock)
PC1	SDA (Two-wire Serial Bus Data Input/Output Line)
PC0	SCL (Two-wire Serial Bus Clock Line)

جدول ۱-۱۳ کاربردهای دیگر پورت C

پایه PC0 (SCL)

زمانی که از ارتباط دهی سریال دو سیمه TWI استفاده شود، پایه SCL به عنوان کلاک عمل می کند. این پایه کلاک استاندارد I2C است.

پایه PC1 (SDA)

زمانی که از ارتباط دهی سریال دو سیمه TWI استفاده شود، پایه SDA به عنوان خط دیتا عمل می کند. این پایه، دیتا استاندارد I2C است.

پایه PC2 (TCK)

در ارتباط دهی استاندارد JTAG که برای برنامه ریزی میکروکنترلر نیز استفاده می شود. پایه TCK به عنوان کلاک تست به صورت سنکرون عمل می کند. توجه کنید در صورت فعال بودن ارتباط دهی JTAG نمی توان از این پایه به عنوان ورودی و خروجی استفاده کرد. برای غیر فعال کردن این ارتباط دهی به بخش توضیح فیوز بیت ها مراجعه کنید.

پایه PC3 (TMS)

در ارتباط دهی JTAG پایه TMSبرای انتخاب مد تست می باشد. در صورت فعال بودن JTAG دیگر نمی توان از این پایه به عنوان ورودی و خروجی استفاده کرد.

پایه PC4 (TDO)

در ارتباط دهی JTAG پایه TDOبرای خروجی سریال عمل می کند. درصورت فعال بودن JTAG دیگر نمی توان از این پایه به عنوان ورودی و خروجی استفاده کرد.

پایه PC5 (TDI)

در ارتباط دهی JTAG پایه TDI به عنوان ورودی داده سریال عمل می کند و در صورت فعال بودن JTAG دیگر نمی توان از این پایه به عنوان ورودی و خروجی استفاده کرد.

پایه PC6 (TOSC1)

در صورتی که بخواهیم از RTC تایمر دو استفاده کنیم باید از کریستال 32.768KHZ پالس ساعت استفاده کنیم، پایه TOSC1 به عنوان پایه اول اسیلاتور پالس زمان واقعی می باشد. در صورت فعال شدن بیت AC2 در ریجستر ASSR، تایمر دو، پالس خود را از کریستال پالس ساعت تامین می کند و دیگر نمی توان از این پایه در این حالت به عنوان ورودی و خروجی استفاده کرد.

پایه PC7 (TOSC2)

در صورتی که بخواهیم از RTC تایمر دو استفاده کنیم باید از کریستال 32.768KHZ پالس ساعت استفاده کنیم، پایه TOSC2 به عنوان پایه دوم اسیلاتور پالس زمان واقعی می باشد. در صورت فعال شدن بیت AC2 در ریجستر ASSR، تایمر دو، پالس خود را از کریستال پالس ساعت تامین می کند و دیگر نمی توان از این پایه در این حالت به عنوان ورودی و خروجی استفاده کرد.

پورت D

پایه های PD0 تا PD7 پورت D را تشکیل می دهند، در حالت عادی می توان از این پایه ها به عنوان ورودی و خروجی استفاده کرد. عملکردهای بعدی این پورت ارتباط دهی سریال USART، وقفه های خارجی صفر و یک، خروجی های مد مقایسه ای تایمر یک و خروجی مد مقایسه ای تایمر ۲ و ورودیcapture تایمر یک می باشد.

Port Pin	Alternate Function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

جدول ۱-۱۴ کاربردهای دیگر پورت D

پایه (RXD) PD0

در ارتباط دهی سریال USART پایه RXD بدون در نظر گرفتن DDRD.0 به عنوان ورودی داده سریال پیکربندی می شود.

پایه (TXD) PD1

در ارتباط دهی سریال USART پایه TXD بدون در نظر گرفتن DDRD.1 به عنوان خروجی داده سریال پیکربندی می شود.

پایه (INT0) PD2

اگر وقفه خارجی صفر فعال شده باشد پایه INT0 به عنوان منبع ورودی وقفه صفر عمل می کند.

پایه (INT1) PD3

اگر وقفه خارجی یک فعال شده باشد پایه INT1 به عنوان منبع ورودی وقفه یک عمل می کند.

پایه (OC1B) PD4

در صورت استفاده از مد مقایسه ای تایمر یک و فعال کردن خروجی مقایسه ای، پایه OC1B به عنوان خروجی دوم مقایسه ای تایمر یک عمل می کند همچنین در مد PWM تایمر یک، این پایه می تواند خروجی سیگنال PWM تولید شده باشد.

پایه (OC1A) PD5

در صورت استفاده از مد مقایسه ای تایمر یک و فعال کردن خروجی مقایسه ای، پایه OC1A به عنوان خروجی اول مقایسه ای تایمر یک عمل می کند همچنین در مد PWM تایمر یک، این پایه می تواند خروجی سیگنال PWM تولید شده باشد.

پایه (ICP) PD6

اگر واحد تسخیر کننده یعنی مد Capture تایمر یک فعال شده باشد پایه ICP به عنوان ورودی Capture تایمر یک عمل می کند.

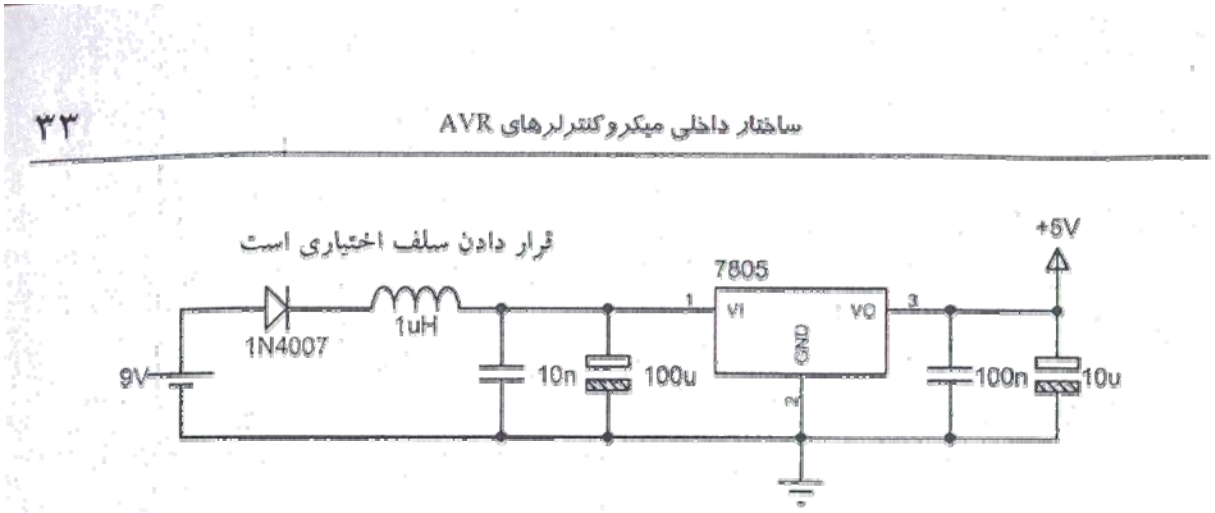
پایه PD7 (OC2)

در صورت استفاده از مد مقایسه ای تایمر ۲ و فعال کردن خروجی مقایسه ای، پایه OC2به عنوان خروجی مقایسه ای تایمر دو عمل می کند. همچنین در مد PWM تایمر دو، این پایه می تواند خروجی سیگنال PWM تولید شده باشد.

۶-۱ تغذیه مناسب جهت بایاس میکروکنترولر

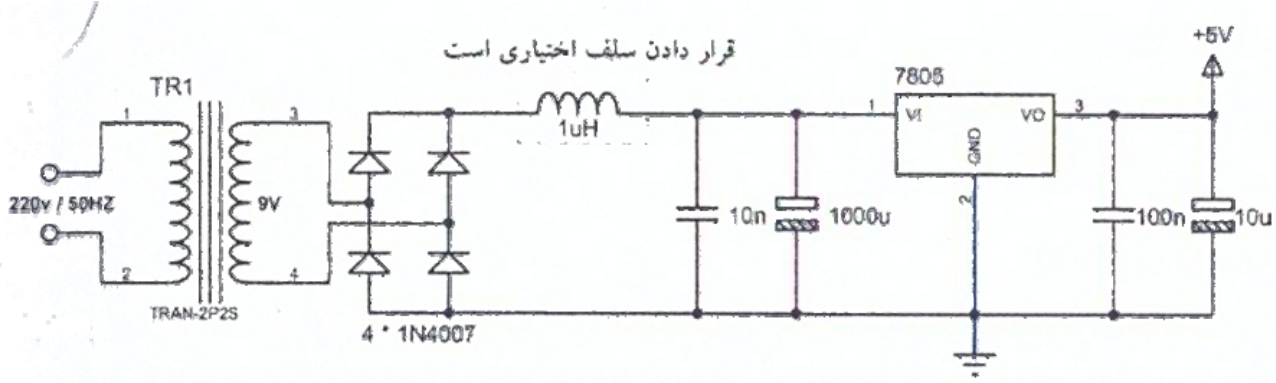
پایه های VCC و GND تغذیه میکروکنترولر را تامین می کنند. میکروکنترلرهای AVR می توانند با ولتاژ ۲,۷ تا ۵,۵ ولت برای نوع L و ولتاژ ۴ تا ۵,۵ ولت برای نوع بدون L کار کنند. اما از آن جایی که اکثر تراشه ها و المان هایی نظیر LCD و آی سی های دیجیتال TTL و... از ولتاژ ۵ ولت استفاده می کنند بنابرین به طور استاندارد تغذیه میکروکنترولر را ۵ ولت انتخاب می کنند و چون ولتاژ بیشتر از ۵ ولت، باعث سوختن میکروکنترلر می شود برای تغذیه آن از رگولاتور ۵ ولتی ۷۸۰۵ استفاده می شود. البته باید نویز محیط را نیز در نظر گرفت.

از منبع تغذیه DC ورودی ۹ ولتی استفاده کرده ایم زیرا ولتاژ ورودی رگولاتور بهتر است ۳ ولت بیشتر از ولتاژ نامی باشد. همچنین ورودی را می توان ۱۲ ولت انتخاب کرد اما این امر باعث افزایش تلفات و گرما در رگولاتور می شود. استفاده از دیود برای محافظت رگولاتور در برابر پلاریته معکوس می باشد. سلف ۱Hμ به همراه خازن عدسی ۱۰nF به عنوان فیلتر حذف نویز ورودی رگولاتور عمل می کنند و خازن عدسی ۱۰۰nF نویزهای تغذیه ۵ ولت خروجی رگولاتور را حذف می کند و خازن الکترولیتی ۱۰۰F μ در ورودی رگولاتور به صاف کردن تغذیه ورودی کمک می کند و همچنین خازن الکترولیتیμ ۱۰F خروجی رگولاتور باعث جلوگیری از افت ولتاژ تغذیه ۵ ولت می شود.



شکل ۱-۴ تغذیه میکروکنترلر با ورودی DC

در صورتی که بخواهیم از تغذیه با ورودی AC برای میکروکنترلر خود استفاده نماییم باید طبق مدار شکل ۱-۵ عمل کنیم. همگی توضیح های داده شده در مدار پیشین، در این مدار نیز صدق می کند. تنها تفاوت این مدار وجود ترانس کاهنده ۲۲۰ ولت به ۹ ولت می باشد که به وسیله پل دیود به گونه تمام موج یکسو می شود و به وسیله خازن شیمیایی $1000\,\mu F$ صاف می گردد.

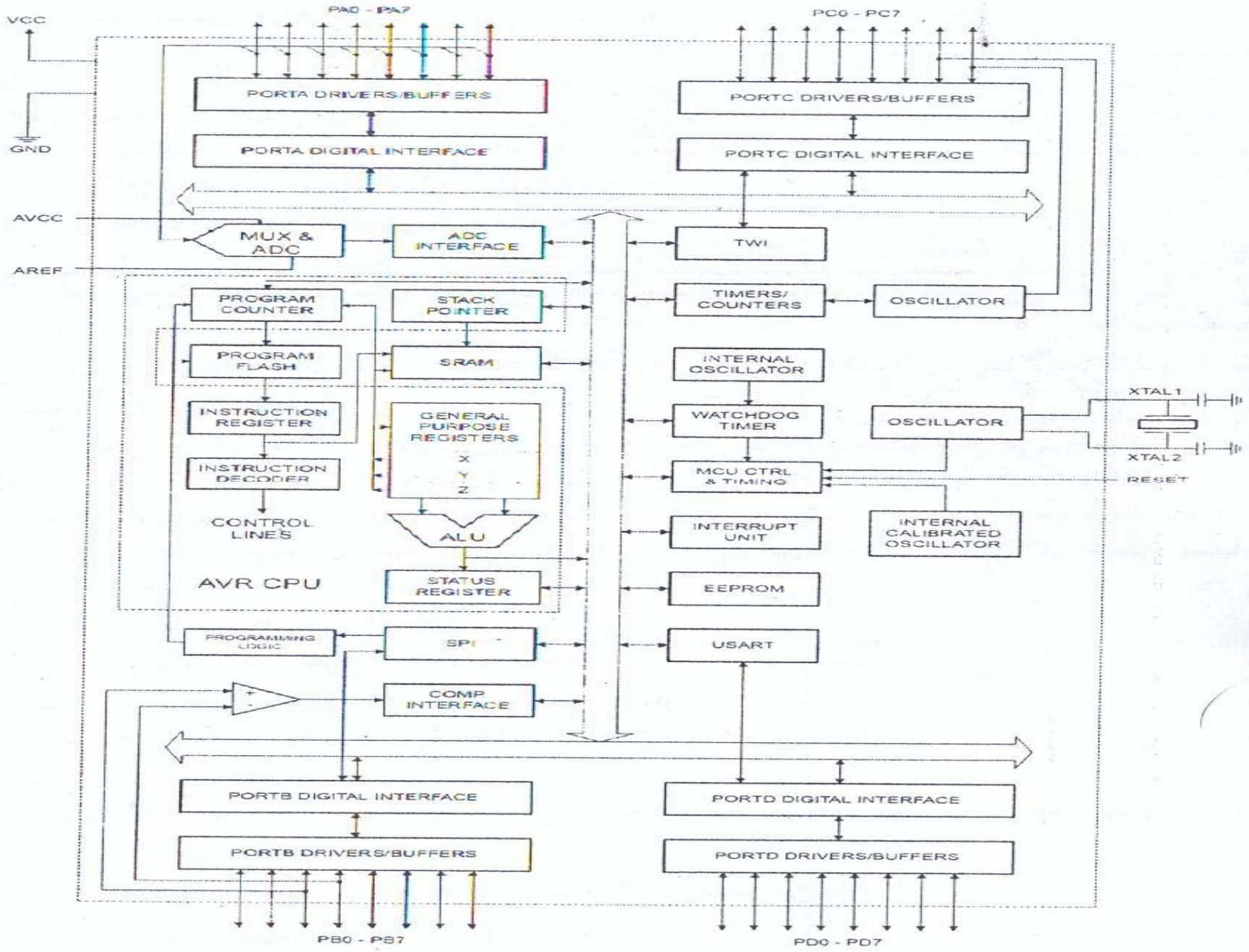


شکل ۱-۵ تغذیه میکروکنترلر با ورودی AC

توجه: رگولاتورهای موجود در بازارهای الکترونیکی می توانند بیشینه تا 500mA جریان بدهند. در صورتی که بخواهیم جریانی بیشتر از حد مجاز استفاده کنیم میتوانیم از یک ترانزیستور و یا موازی کردن دو رگولاتور و یا رگولاتورهایی با جریان بیشتر و یا از منبع تغذیه سوییچینگ استفاده کنیم. همچنین برای استفاده، جریانی بیشتر از 100mA از خروجی رگولاتور مدار فوق، باید از گرماگیر (Heat Sink) استفاده کنیم، ابعاد هیت سینک را ۲*۲ سانتی متر انتخاب کنید.

۱-۷ ساختار داخلی میکروکنترولر ATmaga16

همان گونه که در بلوک دیاگرام شکل ۱-۶ مشاهده می کنید اجزای درونی میکروکنترولر به وسیله باس های داخلی به هم متصل شده اند و یک بخش از این بلوک دیاگرام با خط چین با نام AVR VPU مشخص شده است که در واقع پردازنده اصلی AVR می باشد. در این بخش قصد داریم اجزای تشکیل دهنده CPU میکروکنترلر AVR را توضیح دهیم.

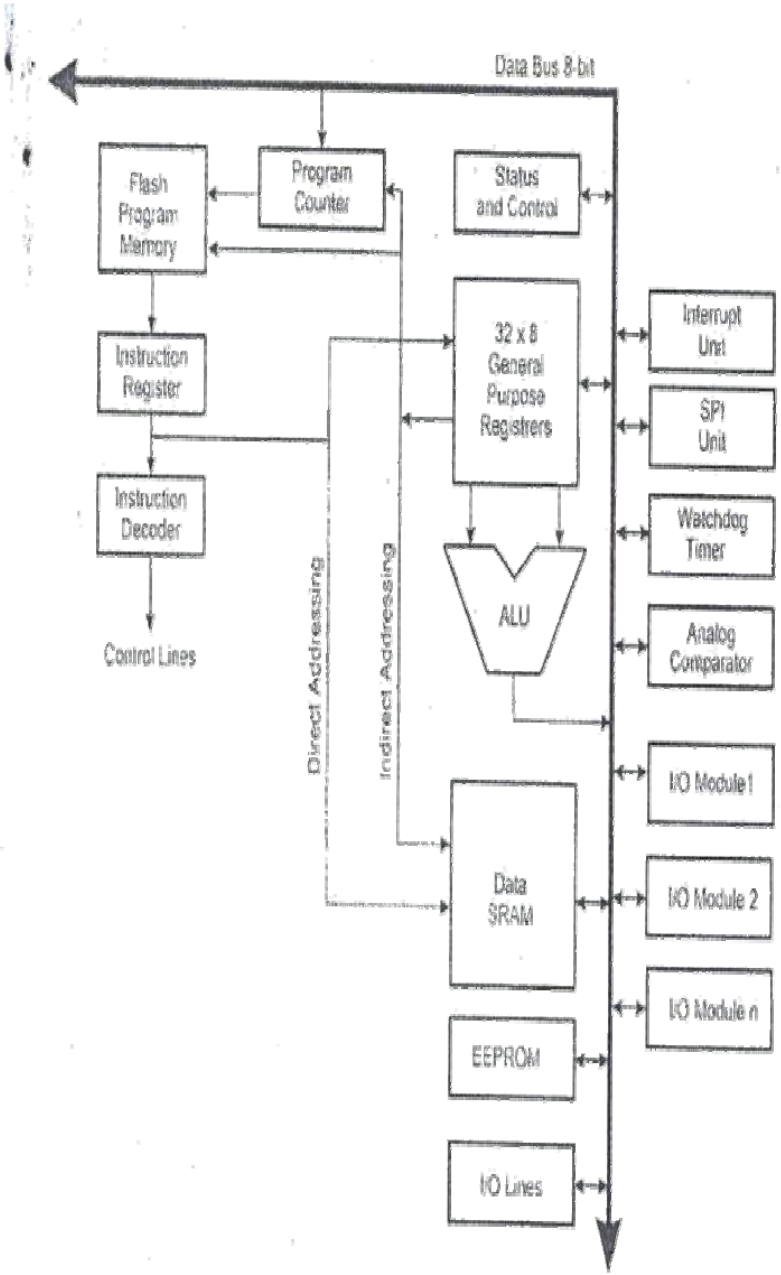


شکل ۱-۶ بلوک دیاگرام داخلی میکروکنترلر ATmega16

کار اصلی یک CPU دسترسی به حافظه ها، محاسبات ریاضی و منطقی، کنترل وسایل جانبی و بررسی وقفه های هر یک از قسمت هاست.

شمارنده برنامه، اشاره گر پشته، رجیستر دستورات، آشکارساز دستورات، رجیسترهای X,Y,Z ، رجیسترهای همه منظوره، واحد محاسبه و منطق (ALU) و رجیستر وضعیت از اجزای تشکیل دهنده CPU میکروکنترولر AVR هستند.

میکروکنترولر AVR از معماری RISC استفاده می کند و برای کارایی بهتر از ساختار Harvard و همچنین از حافظه ها و باس های جداگانه برای انتقال داده استفاده می کند. دستورات با یک سطح pipelining اجرا می شوند و هنگامی که یک دستور در حال اجرا می باشد، دستور بعدی از حافظه برنامه pre-fetched می شود. با این روش هر دستور تنها در یک کلاک سیکل اجرا می شود.



شکل ۱-۷ بلوک دیاگرام ساختار میکروکنترلرهای AVR

شمارنده برنامه PC (Program Counter)

CPU میکروکنترولر برای این که دستورات را از اولین آدرس حافظه برنامه، خط به خط بخواند نیاز به یک شمارنده برنامه می باشد. افزایش PC آدرس خط بعدی را برای اجرای دستورات فراهم می کند.

اشاره گر پشته SP (Stack Pointer)

در میکروکنترولر AVR اشاره گر پشته از دو رجیستر ۸ بیتی استفاده می کند. اشاره گر پشته برای ذخیره موقت اطلاعات در دستور های فراخوانی CALL ، PUSH و POP و متغیرهای محلی، روتین های وقفه و توابع استفاده می شود.

رجیستر های دستورات (Instruction Register)

در اصل CPU میکروکنترولر برای فهمیدن انجام یک دستور العمل از کد ماشین استفاده می کند و به هر یک از این کدها یک سمبل در زبان اسمبلی اختصاص داده می شود. رجیستر دستورات، تمامی دستورهای اسمبلی در نظر گرفته شده از طرف شرکت سازنده Atmel را شامل می شود.

آشکارساز دستورات (Instruction Decoder)

شمارنده برنامه افزایش می یابد و کد هر دستور به وسیله CPU خوانده می شود و با توجه به رجیستر دستورات، کد خوانده شده آشکار می شود و CPU متوجه می شود که کد دستور به چه معنی و مفهومی است.

رجیسترهای همه منظوره

میکروکنترولر های AVR دارای ۳۲ رجیستر همه منظوره هستند. این رجیسترها قسمتی از حافظه SRAM داخلی میکروکنترولر می باشند که اکثر دستورات اسمبلی AVR مستقیما با این رجیسترها دسترسی دارند و در یک کلاک سیکل اجرا می شوند.

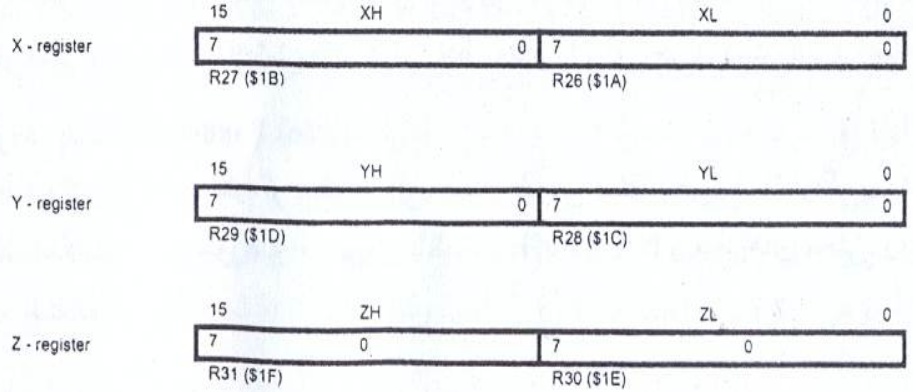
شکل ۱-۸

رجیسترهای همه منظوره CPU میکروکنترلرهای AVR

		Addr.	
	R0	$00	
	R1	$01	
	R2	$02	
	...		
	R13	$0D	
General	R14	$0E	
Purpose	R15	$0F	
Working	R16	$10	
Registers	R17	$11	
	...		
	R26	$1A	X-register Low Byte
	R27	$1B	X-register High Byte
	R28	$1C	Y-register Low Byte
	R29	$1D	Y-register High Byte
	R30	$1E	Z-register Low Byte
	R31	$1F	Z-register High Byte

رجیسترهای X، Y و Z

وظیفه این سه رجیستر که البته از ترکیب رجیسترهای R26 تا R31 بوجود می‌آیند اشاره‌گر ۱۶ بیتی جهت آدرس‌دهی غیر مستقیم فضای داده هستند و بسیاری از دستورات اسمبلی AVR با این سه رجیستر عمل می‌کنند.



واحد محاسبه و منطق ALU (Arithmetic Logic Unit)

ALU در میکروکنترلر AVR به صورت مستقیم با تمام ۳۲ رجیستر همه منظوره ارتباط دارد. عملیات های محاسباتی با رجیستر های همه منظوره در یک کلاک سیکل اجرا می شوند. به طور کلی عملکرد ALU را می توان به سه قسمت اصلی ریاضیاتی، منطقی و توابع بیتی تقسیم بندی کرد. در برخی از ALU های توسعه یافته در معماری میکروکنترلر های AVR از یک ضرب کننده با قابلیت ضرب اعداد بدون علامت و علامتدار و نیز اعداد اعشاری استفاده شده است.

رجیستر وضعیت SREG (Status Register)

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

بیت‌های این رجیستر در واقع پرچم‌هایی هستند که CPU را از نتایج دستورات و وضعیت برنامه آگاه می‌کنند.

بیت ۰- C پرچم Carry

انجام دستوراتی که عملیات محاسباتی یا منطقی را انجام میدهند سبب تاثیر بر پرچم نقلی (Carry) می شود. برای نمونه اگر نتیجه جمع ۸ بیتی بیشتر از ۸ بیت شود، کری برابر یک خواهد شد.

بیت Z-1 پرچم zero

برخی از دستورات منطقی و محاسباتی سبب تاثیر بر روی این پرچم می شوند. اگر حاصل عملیات صفر شود این پرچم فعال می گردد.

بیت N-2 پرچم Negative

این بیت نشانگر نتایج منفی در عملیات های محاسباتی یا منطقی است.

بیت V-3پرچم Two's complement overflow

در دستورات محاسباتی این بیت نشانگر سرریز مکمل عدد دو می باشد.

بیت S-4 بیت علامت sign Bit

این بیت حاصل XOR دو پرچم N و V می باشد. S=N xor V (s=0 علامت + و s=1 علامت-)

بیت H-5 پرچم Half Carry

در انجام دستورات محاسباتی BCD پرچم نقلی کمکی تاثیر می بیند.

بیت T-6 پرچم Bit Copy Storage

میتوانیم از بیت T به عنوان مبدا و یا مقصد یک عملیات بیتی استفاده کنیم.

بیت I-7 پرچم Global Interrupt Enable

این بیت برای فعال سازی وقفه همگانی است؛ اگر این بیت یک شود به دیگر وقفه ها در صورت فعال شدن پاسخ داده می شود. در موقع رخ دادن یکی از وقفه ها برنامه وارد تابع وقفه مربوطه می شود. در این حالت این پرچم توسط سخت افزار صفر می شود و به وقفه دیگری پاسخ داده نمی شود و در برگشت از تابع وقفه مربوطه این بیت به طور اتوماتیک توسط سخت افزار یک می شود.

در زبان برنامه نویسی C این بیت به گونه زیر فعال و غیرفعال می شود:

#asm (“sei”) // فعال کردن وقفه همگانی

#asm (“cli”) // غیر فعال کردن وقفه همگانی

۸-۱ انواع حافظه در میکروکنترولر های AVR

میکروکنترلرهای AVR دارای سه نوع حافظه هستند:

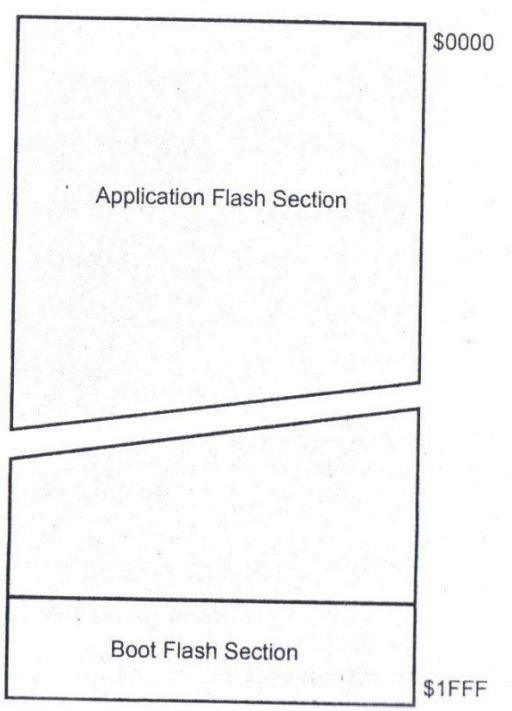
حافظه Flash

در این حافظه کدهای برنامه یعنی همان فایل *.hex که توسط پروگرامر بر روی تراشه load می شود، قرار می گیرد و cpu میکروکنترلر برنامه را که اجرا می کند کد دستورالعمل را از این حافظه برداشت می کند.
حافظه ثابت (Flash) میکروکنترلرهای AVR از نسل جدید این حافظه می باشد و دارای دو قسمت Application و Boot Loader هستند. در قسمت Application کدهای برنامه قرار می گیرد ولی ناحیه Boot این امکان را فراهم می کند که میکروکنترلر بدون استفاده از ابزار پروگرامر، برنامه حافظه Flash را تغییر دهد.

برای نمونه نحوه تعیین یک آرایه در حافظه ثابت به گونه زیر است:

```
Flash char row [] = {0xfe,0xfd,0xfb,0xf7};    //  flash ۴ عدد ۸ بیتی ذخیره شده در
```

شکل ۱-۹ آدرس و تقسیم بندی حافظه flash را به دو قسمت نشان می دهد.



شکل ۱-۹
نقشه حافظه برنامه (flash)

حافظه EEPROM

این حافظه جز حافظه های ماندگار می باشد که میکروکنترلر می تواند اطلاعاتی را در این حافظه داخلی بنویسد و یا اطلاعاتی را از آن بخواند. همچنین نیاز به یادآوری است که این حافظه در صورت قطع تغذیه میکرکنترلر پاک نمی گردد.

از این حافظه زمانی استفاده می شود که میکروکنترلر باید دیتایی را در خود ثبت کند و بعدا آن دیتا را به کاربر اعلام کند و در صورتی که میکرکنترلرReset شد یا تغذیه آن قطع گردید، داده ذخیره شده از بین نرود. باید توجه کنید که برای خواندن و نوشتن در حافظه eeprom باید یک زمان تاخیری در نظر بگیرید. در جدول ۱-۱۵ مدت زمان مناسب با کلاک 1MHZ، ۸,۵ میلی ثانیه بیان شده است.

تغییر متغیر در حافظه eeprom به گونه زیر است:

```
eeprom unsigned int x=0xff ;      // متغیر x در حافظه eeprom ذخیره شده است
```

Symbol	Number of Calibrated RC Oscillator Cycles(1)	Typ Programming Time
EEPROM write (from CPU)	8448	8.5 ms

Note: 1. Uses 1 MHz clock, independent of CKSEL Fuse setting.

جدول ۱-۱۵ زمان برنامه‌ریزی حافظه eeprom

حافظه SRAM

همان گونه که گفته شد؛ کدهای برنامه در حافظه Flash قرار می گیرند و CPU میکروکنترلر کدهای دستورات را آشکار می کند. حال باید حاصل دستورات انجام شده در یک حافظه موقت ذخیره گردد. این حافظه در میکروکنترلر AVR از نوع static RAM می باشد. رجیسترهای همه منظوره و رجیستر های ورودی خروجی نیز جز این حافظه می باشند. محتوای این حافظه با قطع تغذیع پاک می گردد و در صورتی که میکروکنترولر را Reset کنیم، محتوای رجیسترها صفر می شود ولی محتوای حافظه SRAM صفر نمی شود.

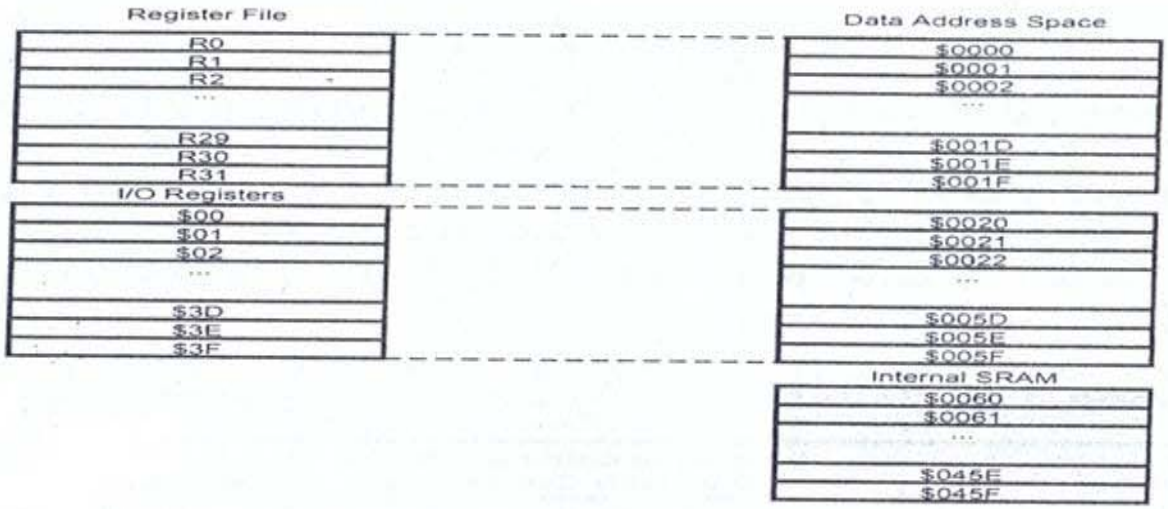
برای نمونه متغیر M را در حافظه ذخیره کنید.

```
Unsigned char M=0x12;
```

بنابراین مشخص می شود؛ در تعریف هر متغیری که پیش از آن واژه کلیدی flash و eeprom به کار برده نشود، به مفهوم ذخیره متغیر در حافظه SRAM می باشد.
شکل ۱-۱۰ تقسیم بندی حافظه SRAM داخلی میکروکنترلر ATmega16 را نشان می دهد.

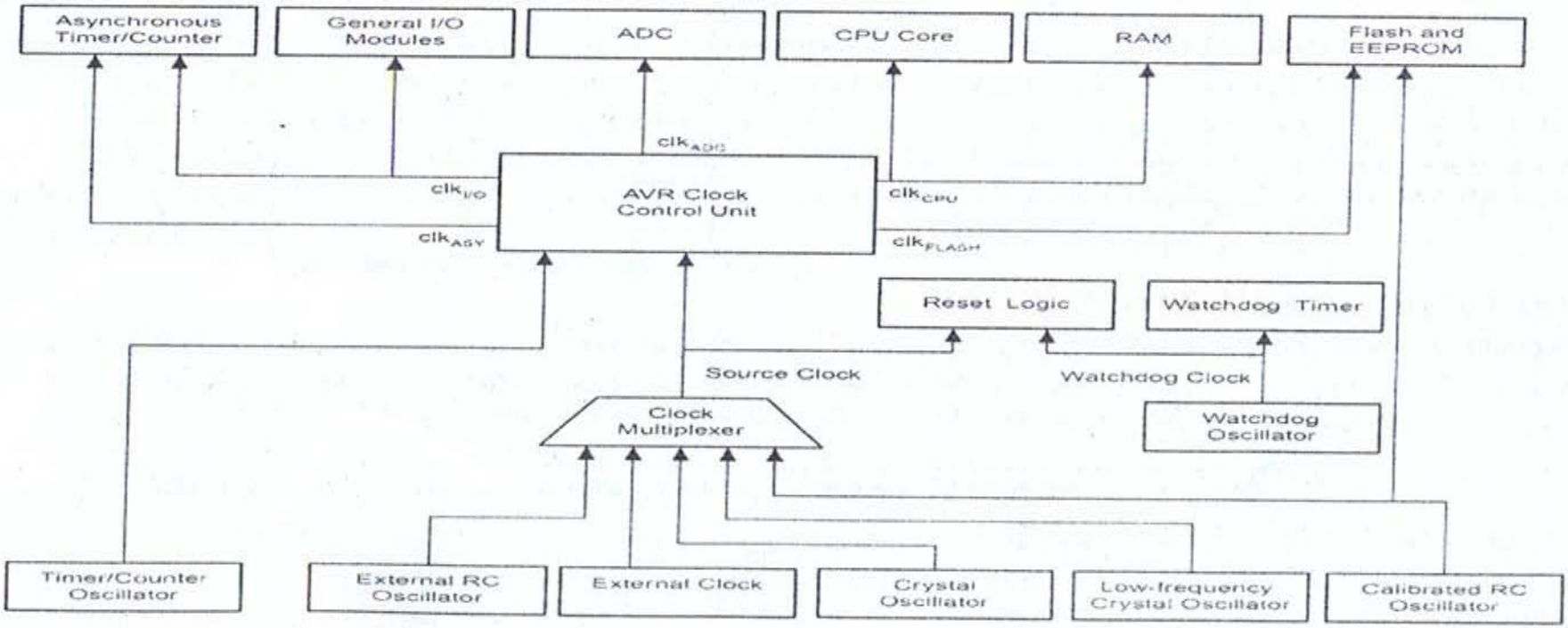
۹-۱ کلاک سیستم در میکروکنترلرهای AVR

سیستم پالس ساعت (سیکل ماشین) در میکروکنترلرهای AVR بسیار متنوع است. وارون میکروکنترلر ۸۰۵۱ که تنها با کریستال خارجی کار می کرد، میکروکنترلرهای جدید می توانند افزون بر کریستال خارجی از نوسان ساز داخلی نیز استفاده کنند. در این بخش می خواهیم سیستم توزیع کلاک و انواع نوسان سازها را توضیح دهیم.



شکل ۱-۱۰ نقشه حافظه دیتا (RAM)

شکل ۱-۱۱ سیستم پالس ساعت را برای قسمت‌های مختلف درونی نشان می‌دهند.



شکل ۱-۱۱ بلوک دیاگرام کلاک سیستم

همان گونه که در شکل ۱-۱۱ مشاهده می فرمایید منبع کلاک به وسیله یک مالتی پلکسر، پالس لازم را به واحد کنترل کننده کلاک اعمال می کند. در واقع کلاک لازم جهت راه اندازی می تواند یکی از نوسان سازها باشد و امکان استفاده همزمان از آن ها وجود ندارد.

پالس CLKCPU به هسته (اجزای درونی) cpu و SRAM داخلی اعمال می شود، پالس CLKADC کلاک لازم را جهت مبدل آنالوگ به دیجیتال فراهم می کند که به وسیله قسمت مبدل ADC می تواند به طور مجزا تقسیم گردد. پالس CLKFLASH کلاک لازم را برای حافظه فلش و eeprom داخلی فراهم می سازد. پالس clkI/O برای تولید پالس ماژول های ورودی و خروجی نظیر USART، SPI، شمارنده و وقفه ها به کار برده می شود، پالس CLKASY برای راه اندازی آسنکرون تایمر یا کانتر دو برای استفاده از فرکانس 32.768KHZ اسیلاتور RCT است. همچنین تایمر Watdhodog از یک اسیلاتور مجزا داخلی استفاده می کند.

منابع پالس ساعت میکروکنترلرهای AVR

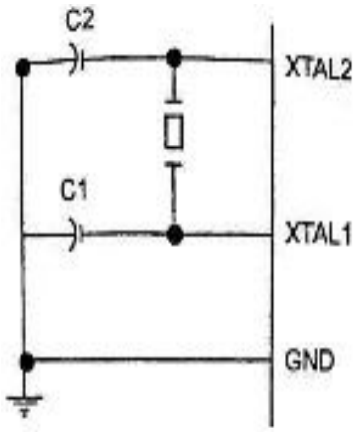
همان گونه که گفته شد میکروکنترلرهای AVR از نوسان سازهای داخلی و یا خارجی استفاده می کنند. برای تعیین نوسان ساز سیستم باید از فیوز بیت های میکروکنتر Atmega16 طبق جدول ۱-۱۶ استفاده کنیم که در بخش توضیح فیوز بیت ها نیز توضیح داده شد.

Device clocking option	CKSEL3..0
External Crystal/Ceramic Resonator	1111-1010
External Low-frequency Crystal	1001
External RC Oscillator	1000-0101
Calibrated internal RC Oscillator	0100-0001
External clock	0000

جدول ۱-۱۶ تعیین منبع کلاک سیستم

نوسان ساز با کریستال خارجی

برای استفاده از کریستال خارجی، باید فیوز بیت های CKSEL3..0 را طبق جدول ۱-۱۶ به صورت "1111" برنامه ریزی کنیم. پایه های خارجی XTAL1 و XTAL2 طبق شکل ۱-۱۲ توسط دو خازن عدسی (خازن های بالانس) با مقادیر یکسان به یک کریستال متصل می گردند.



شکل۱-۱۲ نحوه اتصال کریستال خارجی

خازن های C1 و C2 را معمولا 22PF انتخاب می کنیم.

وظیفه این دو خازن حذف نویز الکترومغناطیس اطراف کریستال می باشد که طبق جدول۱-۱۷ با توجه به کریستال استفاده شده تعیین می شوند. در PCB برای حذف نویز، بدنه کریستال خارجی را به زمین وصل می کنند اما نباید بدنه کریستال حرارت ببیند زیرا ممکن است به آن آسیب برسد.

CKOPT	CKSEL3...1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101	0.4 - 0.9	-
1	110	0.9 - 3.0	12 – 22
1	111	3.0 - 8.0	12 – 22
0	101, 110, 111	1.0 ≤	12 – 22

جدول ۱-۱۷ انتخاب خازن ها در فرکانس های مختلف

هنگام استفاده از کریستال خارجی می توان با فعال کردن فیوز بیت CKOPT دامنه نوسان اسیلاتور را حداکثرکرد. در این حالت اسیلاتور به صورت Rail-to-Rail عمل می کند، یعنی اگر تغذیه میکروکنترلر ۵ ولت باشد، بیشینه پالس ساعت نیز ۵ ولت خواهد بود. این حالت برای محیط های پر نویز مانند کارخانه های صنعتی بسیار مناسب است. البته فعال کردن این فیوز بیت به اندازه چند میلی آمپر جریان مصرفی میکروکنترلر را افزایش می دهد. در میکروکنترلر های بدون پسوند L اگر بخواهیم از کریستال 16MHZ استفاده کنیم باید فیوز بیت CKOPT فعال گردد، در غیر این صورت بیشینه کریستال خارجی 8MHZ خواهد بود.

زمان Start-up برای استفاده از کریستال خارجی به وسیله فیوز بیت های SUT0 و SUT1 طبق جدول ۱-۱۸ تعیین می گردد. منظور از زمان start-up ، مدت زمانی است که تغذیه به میکروکنترلر وصل می شود و بعد از مدتی که نوسانات اسیلاتور پایدار شد میکروکنترلر reset شده و برنامه را اجرا می کند. این مدت زمان در حدود چند میلی ثانیه است.

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (VCC = 5.0V)	Recommended Usage
0	00	258 CK(1)	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK(1)	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK(2)	-	Ceramic resonator, BOD Enabled
0	11	1K CK(2)	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK(2)	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	-	Crystal Oscillator, BOD Enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

۱- این گزینه ها تنها برای زمانی استفاده می شود که فرکانس بالا نباشد و برای کریستال ها مناسب نیست.
۲- این گزینه ها تنها برای نوسان سازهای رزوناتور سرامیکی مفید هستند.

جدول ۱-۱۸ انتخاب زمان شروع (Start-Up) در حالت های نوسان ساز خارجی

نوسان ساز با کریستال فرکانس پایین

منظور از کریستال با فرکانس پایین، کریستال ۳۲,۷۶۸ کیلو هرتز می باشد. درصورتی که فیوز بیت های CKSEL3..0 به صورت "۱۰۰۱" برنامه ریزی شوند، پالس ساعت سیستم از کریستال خارجی فرکانس پایین استفاده می کند. در این حالت اگر فیوز بیت CKOPT فعال شود، خازن داخلی بین دو پایه XTAL1 و XTAL2 فعال می گردد و مقدار این خازن ۳۶ پیکوفاراد است. نحوه استفاده از این نوسان ساز طبق شکل ۱۲ می باشد و همچنین زمان START-Up در این حالت طبق جدول 19-1 تعیین می شود.

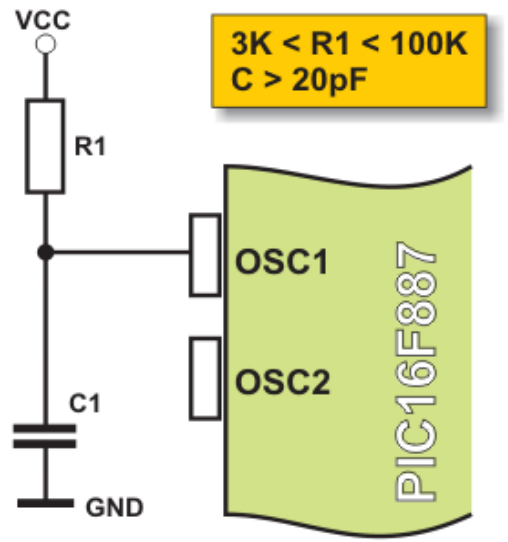
SUT1..0	Start-up time from power-down and power-save	Additional delay from reset(Vcc=5.0v)	Recommended usage
00	1K CK	4.1ms	Fast rising power or BOD enabled
01	1K CK	65ms	Slowly rising power
10	32K CK	65ms	Stable frequency at start-up
11	RESERVED	RESERVED	RESERVED

جدول ۱-۱۹انتخاب زمان شروع درحالت نوسان سازخارجی فرکانس پایین

نوسان ساز با RC خارجی

ازاسیلاتوربا RC خارجی در کاربردهایی که به تغییرات زمان و فرکانس حساسیت نداشته باشیم استفاده می کنیم. فرکانس این اسیلاتور از رابطه زیر به دست می آید که در آن مقدار خازن حداقل باید 22PF انتخاب شود و مقدار مقاومت بین ۳ تا ۱۰۰ کیلو اهم انتخاب می شود.

f=1/3RC



شکل ۱-۱۳ نحوه استفاده از اسیلاتور خارجی RC را نشان می دهد.

شکل ۱-۱۳ نحوه اتصال نوسان ساز RC خارجی

با برنامه ریزی فیوز بیت CKOPT در حالت استفاده از این نوع نوسان ساز، می توان خازن 36pf داخلی بین پایه XTAL1 و GND را فعال نموده و از خازن بیرونی استفاده نکنیم. در این نوع نوسان ساز چهار مد برای محدوده های مختلف فرکانسی طبق جدول ۱-۲۰ وجود دارد که می توان با فیوزبیت های CKSEL3..0 تنظیم گردد. همچنین در این نوع نوسان ساز زمان Start-Up به وسیله فیوزبیت های SUT0 و SUT1 طبق جدول ۱-۲۱ تعیین می شود.

CKSEL3..0	Frequency Range(MHz)
0101	_<0.9
0110	0.9-3.0
0111	3.0-8.0
1000	8.0-12.0

جدول۱-۲۰ انتخاب محدوده نوسان ساز RCخارجی

SUT1..0	Start-up time from power-down and power-save	Additional delay from reset(Vcc=5.0v)	Recommended usage
00	18CK	-	BOD enable
01	18CK	4.1ms	Fast rising power
10	18CK	65ms	Slowly rising power
11	6CK	4.1ms	Fast rising power Or BOD enable

جدول۱-۲۱ انتخاب زمان شروع درحالت نوسان ساز RCخارجی

نوسان ساز با اسیلاتور RC کالیبره شده داخلی

اسیلاتور RC کالیبره شده داخلی می تواند فرکانس های ثابت 1MHZ، 2MHZ، 4MHZ، 8MHZ، را در شرایط تغذیه ۵+ ولت و در دمای ۲۵ درجه سانتی گراد ایجاد نماید. فرکانس کاری این اسیلاتور به شدت به ولتاژ تغذیه، درجه حرارت محیط و مقدار بایت رجیستر OSCCAL وابسته می باشد.

از آن جایی که این نوع نوسان ساز به دما و ولتاژ وابسته است، پیشنهاد می کنیم در موقع استفاده از تبادل سریال USART و دیگر پروتکل ها و برنامه هایی که به زمان بسیار وابسته هستند از کریستال خارجی استفاده کنید. در همه پروژه های این کتاب از کریستال خارجی استفاده شده است که در برخی می توانستیم از اسیلاتور داخلی استفاده کنیم. همچنین نیاز به گفتن است که میکروکنترلر ATmega16 به گونه پیش فرض از اسیلاتور کالیبره شده داخلی با فرکانس 1MHZ استفاده می کند که شما باید فیوزبیت های CKSEL3..0 را طبق این بخش تنظیم کنید.

زمانی که از اسیلاتور داخلی استفاده می شود نیازی به قرار دادن کریستال خارجی نیست و پایه های XTAL1 و XTAL2 آزاد گذاشته می شود و همچنین در این نوسان ساز، نباید فیوز بیت CKOPT فعال باشد. مقدار فرکانس این اسیلاتور به وسیله بیت های CKSEL3..0 طبق جدول ۲۲-۱ تعیین می شود.

CKSEL3..0	Nominal Frequency (MHz)
0001	1.0
0010	2.0
0011	4.0
0100	8.0

جدول۱-۲۲ انتخاب فرکانس نوسان ساز کالیبره شده داخلی

زمان Start-Up نیز به وسیله فیوز بیت های SUT0 و SUT1 طبق جدول ۱-۲۳ تنظیم می گردد.

SUT1...0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (VCC = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10	6 CK	65 ms	Slowly rising power
11	Reserved	Reserved	Reserved

جدول ۱-۲۳ انتخاب زمان شروع (Start-Up) در حالت نوسان ساز کالیبرشده داخلی

رجیستر کالیبراسیون OSCCAL

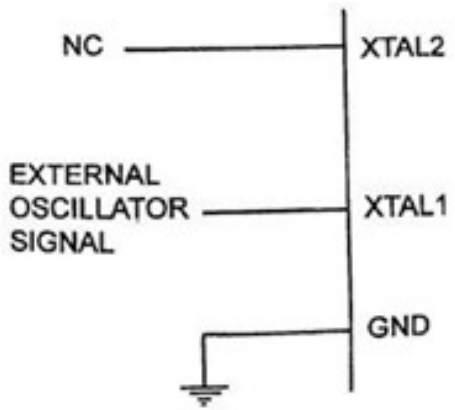
در صورت استفاده از اسیلاتور RC کالیبره شده داخلی، در هر بار که میکرکنترلر Reset می شود، مقدار رجیستر OSCCAL، Load می شود و اسیلاتور به طور خودکار تنظیم می گردد. در حالت عادی استفاده از این نوع اسیلاتور ٪۳± خطا دارد اما اگر از شرایط تغذیه ۵+ ولت و یا دمای ۲۵ درجه سانتی گراد خارج گردد، ممکن است این خطا به ٪۱۰ افزایش یابد که با توجه به این که برای دسترسی به حافظه Flash و حافظه eeprom از این اسیلاتور می خواهد استفاده شود ممکن است نتایج نامشخصی داشته باشد. با نوشتن مقدار 0x00 کم ترین و با نوشتن مقدار 0xff در این رجیستر بیشترین فرکانس ممکن انتخاب می گردد. تنظیم دقیق این کالیبراسیون خیلی تضمینی نیست ولی مقدار بایت کالیبراسیون را می توانیم در هر شرایط دمایی طوری تنظیم کنیم که خطا به 1± کاهش یابد.

OSCCAL value	Min frequency in percentage of normal frequency	Max frequency in percentage of normal frequency
$00	50	100
$7F	75	150
$FF	100	200

جدول ۱-۲۴ محدوده فرکانسی نوسان ساز RC کالیبره شده داخلی

نوسان ساز با کلاک خارجی

در صورت تنظیم فیوز بیت های CKSEL3..0 به صورت "0000" میکروکنترلر AVR پالس ساعت خودرا از یک منبع خارجی که به پایه ورودی تقویت کننده نوسان ساز یعنی XTAL1 اعمال می شود، دریافت می کند. با برنامه ریزی فیوز بیت CKOPT می توان خازن ۳۶پیکوفاراد داخلی بین پایه XTAL1 و GND را فعال کرد. تغییرات فرکانس در این حالت نباید بیشتر از 2± باشد، در غیر این صورت میکروکنترلر ممکن است رفتار نامشخصی داشته باشد.



شکل۱-۱۴ نحوه اعمال کلاک خارجی رابه میکروکنترلرنشان میدهد

توجه: ممکن است فیوز بیت های CKSEL3..0 را به طور ناخواسته به صورت "0000" برنامه ریزی کنید، اما قصد استفاده از کریستال خارجی را داشته باشید، می بینید میکروکنترلر شما کار نمی کند و توسط پروگرامر شناسایی نمی شود برای تغییر فیوز بیت با پروگرامر به شکل صحیح باید یک فرکانس 1MH با دامنه ۵ ولت به پایه XTAL1 اعمال کنید.

زمان Start-Up در موقع استفاده از کلاک خارجی به عنوان پالس ساعت سیستم، می تواند به وسیله فیوز بیت های SUT0 و SUT1 طبق جدول ۱-۲۵ تنظیم گردد.

SUT1...0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (VCC = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10	6 CK	65 ms	Slowly rising power
11	Reserved	Reserved	Reserved

جدول۲۵-۱انتخاب زمان شروع در حالت نوسان ساز کلاک خارجی

نوسان ساز مجزا تایمر یا کانتر دو

برخی از میکروکنترلرهای AVR از جمله ATmega16 دارای پایه های TOSC1 و TOSC2 می باشند. زمانی که تایمر دو به عنوان RTC عمل می کند از کریستال پالس ساعت (32.768KHZ) که به این دو پایه متصل می گردد کلاک دریافت می کند. اسیلاتور برای استفاده از این نوع کریستال بهینه شده و نیازی به قراردادن خازن های حذف نویز بیرونی نمی باشد. زیرا اسیلاتور فرکانس پایین است و ممکن است سبب توقف نوسانات اسیلاتور گردد.

۱۰-۱ مدهای مختلف Sleep

همان طور که در ویژگی های میکروکنترلر ATmega16 گفته شد، این میکروکنترلر دارای شش مد Sleep می باشد. هریک از این مدها در عملکردهای متفاوتی کاربرد دارند. هدف از مدهای توان، کاهش توان مصرفی میکروکنترلر می باشد. این مدها باعث می شوند تا در پروژه هایی که از باتری برای تغذیه استفاده می شود مصرف جریان میکروکنترلر کاهش یابد. هنگامی که از مدهای sleep استفاده می کنیم، کلاک قسمتی از اجزای درونی متوقف می شود و با رخ دادن یک وقفه، cpu و دیگر قسمت های میکروکنترلر از مد خواب (Sleep) بیدار شوند.

رجیستر MCUCR (MCU Control Register)

رجیستر MCUCR

7	6	5	4	3	2	1	0
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

نیبل بالایی این رجیستر برای تنظیم مدهای توان می باشد.

- **بیت های SM2..0-7,5,4**

این بیت ها طبق جدول ۱-۲۶ یکی از مدهای Sleep را تعیین می کنند.

- **بیت SE-6**

اگر این بیت یک شود مد Sleep فعال میشود.

SM 2	SM 1	SM 0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	standby
1	1	1	Extended standby

جدول ۱-۲۶ انواع مدهای Sleep

کتابخانه Sleep.h

برای فعال کردن مدهای توان code vision AVR می توانید از فایل سرآمد Sleep.h استفاده کنید.

void sleep-enable(void); // فعال کردن sleep

void sleep-disable(void); //غیر فعال کردن مد sleep

مد Idle

در این حالت CPU میکروکنترلر متوقف می شود اما ارتباط دهی سریال SPI، TWI و USART، مقایسه کننده آنالوگ، مبدل ADC، تایمر یا کانترها، Watchdog و وقفه ها می توانند به کار خود ادامه دهند. در این مد مصرف تست شده با ولتاژ 3v و دمای ۲۵ درجه سانتیگراد و نوسان ساز 1MHZ برای ATmega16L، 0.35mA بوده است.

void idle (void); // میکرو کنترلر به مد Idle می رود

مد ADC Noise Reduction

در این حالت نیز CPU میکروکنترلر متوقف می شود اما ارتباط دهی سریال TWI، مبدل ADC، تایمر یا کانتر ۲، Watchdog و وقفه های خارجی می توانند به کار خود ادامه دهند. کاربرد این مد برای کاهش نویز محیط، برای ADC می باشد. وقتی میکروکنترلر وارد این مد شده باشد، اعمال یک Reset خارجی، Reset به وسیله تایمر Watchdog، Reset از طریق مدار Brown-out، وقفه کامل شدن تبدیل ADC، وقفه ارتباط دهی TWI، وقفه سرریز تایمر یا کانتر ۲، وقفه آماده بودن eeprom و هر یک از وقفه های خارجی می توانند میکروکنترلر را از مد Sleep بیدار کنند.

مد Power-down

در این حالت نوسان ساز خارجی متوقف می گردد. در این مد وقفه های خارجی، ارتباط سریال TWI و تایمر Watchdog می توانند به کار خود ادامه دهند. در این مد مصرف تست شده با ولتاژ 3v و دمای ۲۵ درجه سانتی گراد و نوسان ساز 1MHZ برای ATmega16، کم تر از 1µA بوده است.

اعمال یک Reset خارجی، Reset به وسیله تایمر Watchdog، Reset از طریق مدار Brown-out، وقفه کامل شدن تبدیل ADC، وقفه ارتباط دهی TWI و هر یک از وقفه های خارجی می توانند میکروکنترلر را از این مد Sleep بیدار کنند.

```
void powerdown (void);      //  میکروکنترلر به مد powerdown می رود
```

مد Power-Save

این حالت بسیار شبیه مد powerdown است. تنها تفاوت بین آن ها این است که تایمر یا کانتر دو، می توانند به صورت غیر همزمان (یعنی بیت AS2 در رجیستر ASSR یک شده باشد) در این مد به کار خود ادامه می دهد. در این حالت میکروکنترلر با رخ دادن وقفه سرریز تایمر یا کانتر۲ و همچنین رخ دادن وقفه تطبیق مقایسه تایمر یا کانتر دو، از این مد خارج می شود.

```
void powersave (void);      //  میکروکنترلر به مد powersave می رود
```

مد Standby

زمانی که از کریستال یا رزوناتور خارجی به عنوان منبع پالس ساعت استفاده می شود. می توان از این مد که بسیار عملکردی مشابه مد Power-down دارد استفاده کرد، با این تفاوت که اسیلاتور می تواند به کار خود ادامه دهد.

```
void Standby (void);        // میکروکنترلر به مد Standby می رود
```

مد Extended Standby

زمانی که از کریستال یا رزوناتور خارجی به عنوان منبع پالس ساعت استفاده می شود، می توان از این مد که بسیار عملکردی مشابه مد Power-Save دارد استفاده کرد، با این تفاوت که اسیلاتور می تواند به کار خود ادامه دهد.

```
void Extended_Standby (void);    // میکروکنترلر به مد Extended Standby می رود
```

۱-۱۱ منابع Reset میکروکنترلر ATmega16

Reset در واقع به نوعی یک وقفه است که می تواند توسط هر یک از منابع تحریک کننده آن رخ دهد. هنگامی که Reset رخ می دهد، تمامی رجیسترهای ورودی و خروجی و همچنین دیگر رجیسترهای کنترلی با توجه به مقادیر پیش فرض در نظر گرفته شده، تنظیم می شوند. بعد از Reset یک مدت زمان کوتاه به اندازه زمان تعیین شده توسط start-up طول می کشد، سپس بعد از پایداری نوسان ساز، برنامه از بردار Reset آدرس 0x0000 آغاز می شود. در صورتی که بردار Reset را توسط فیوز بیت های BOOTSZ1..0 تعیین کرده باشیم آن گاه بردار Reset از قسمت Boot حافظه Flash داخلی آغاز می شود. سیستم Reset میکروکنترلر ATmega16، از طریق رجیستر MCUCSR منبع Reset را تشخیص می دهد و از ۵ طریق ممکن Reset می شود.

رجیستر MCUCSR

7	6	5	4	3	2	1	0
JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
0	0	0					

1. Power-on reset:

هنگامی که ولتاژ تغذیه از ولتاژ آستانه (VPOT) پایین تر شود، بیت PORF در رجیستر MCUCSR یک شده و میکروکنترلر RESET می شود.

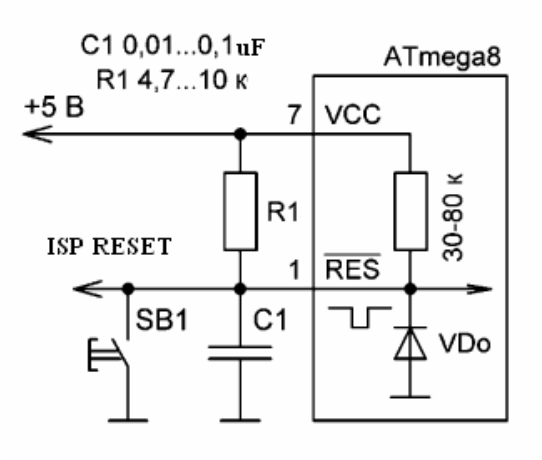
۲.External reset:

اگر یک پالس باسطح صفر منطقی به مدت طولانی تر از سیکل پالس ساعت میکروکنترلر، به پایه خارجی Reset اعمال شود، بیت EXTRF در رجیستر MCUCSR یک شده و میکروکنترلر Reset می شود. در حالت اجرای برنامه باید این پایه توسط یک مقاومت ۱۰K به VCC وصل گردد. شکل ۱-۱۵ نحوه وصل کردن مقاومت R را به پایه خارجی Reset نشان می دهد.

همچنین به دلیل این که این پایه، تنها دارای یک دیود داخلی متصل به زمین است و فاقد دیود متصل به VCC است بنابراین می توان با قرار دادن یک دیود خارجی به صورت معکوس از اثر ناخواسته الکتریسیته ساکن جلوگیری کرد و نویزهای احتمالی محیط را نیز می توان به وسیله یک خازن عدسی خنثی نمود و به وسیله یک کلید فشاری می توان میکروکنترلر را به گونه دستی Reset کرد. نیاز به گفتن است که قرار دادن دیود، خازن و کلید فشاری اختیاری است.

3. Watchdog Reset:

اگر تایمر نگهبان زمان (Watchdog Timer) فعال شود و از مقدار نهایی خود سرریز کند، بیت WDRF در رجیستر MCUCSR یک شده و میکروکنترلر را Reset می کند.



شکل ۱-۱۵ نحوه استفاده از پایه Reset

۴. Brown-Out Reset:

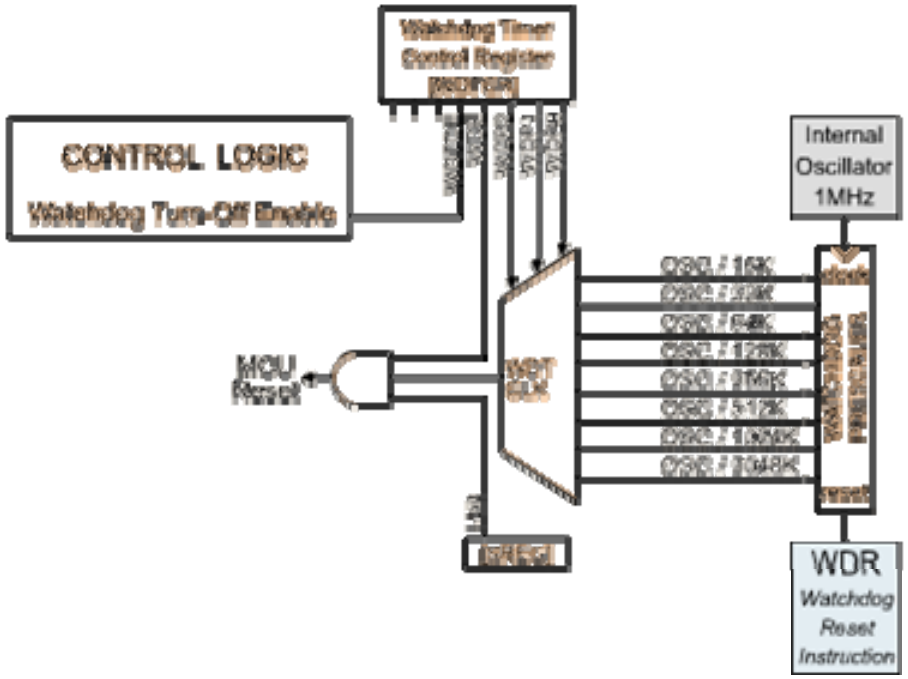
در صورت فعال بودن مدار Brown-out توسط فیوزبیت BODEN، اگر مقدار ولتاژ تغذیه از حد ولتاژ ۲,۷ یا ۴ ولت که توسط فیوز بیت BODLEVEL تعیین می گردد پایین تر بیاید، بیت BORF در رجیستر MCUCSR یک شده و میکروکنترلر را Reset می کند.

5. JTAG AVR Reset:

تا وقتی که در حلقه اسکن سیستم JTAG، رجیستر Reset یک منطقی باشد، بیت JTRF در رجیستر MCUCSR یک شده و میکروکنترلر RESET خواهد شد.

۱۲-۱ تایمر Watchdog:

میکروکنترلرهای AVR دارای یک تایمر نگهبان زمان هستند. وظیفه این تایمر در صورت فعال شدن توسط برنامه نویسی، این است که از عملکرد صحیح برنامه میکروکنترلر می توان اطمینان حاصل کرد. این تایمر زمانی که فعال می شود توسط کلاک داخلی مجزا با فرکانس 1MHZ (تحت شرایط تغذیه ۵ ولت) شروع به شمارش می کند و بعد از یک مدت زمانی طبق تقسیم فرکانسی تنظیم شده توسط کاربر، این تایمر به مقدار نهایی خود می رسد و میکروکنترلر را Reset می کند.



شکل ۱-۱۶ بلوک دیاگرام تایمر Watchdog

WDTCR - رجیستر کنترل تایمر نگهبان زمان

7	6	5	4	3	2	1	0
-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

بیت های WDP2,WDP1,WDP0-2:0 (watchdog Timer Prescaler)

توسط این سه بیت طبق جدول ۱-۲۷ می توان تقسیم فرکانسی را برای مدت زمان لازم جهت Reset میکروکنترلر، توسط تایمر نگهبان زمان تنظیم نمود.

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at Vcc=3.0v	Typical Time-out at Vcc=5.0v
0	0	0	16k(16.384)	17.1ms	16.3ms
0	0	1	32k(32.768)	34.3ms	32.5ms
0	1	0	64k(65.536)	68.5ms	65ms
0	1	1	128k(131.072)	0.14s	0.13s
1	0	0	256k(262.144)	0.27s	0.26s
1	0	1	512k(524.288)	0.55s	0.52s
1	1	0	1024k(1.048.576)	1.1s	1.0s
1	1	1	2048k(2.097.152)	2.2s	2.1s

بیت WDE-3 (Enable watchdog)

با یک کردن این بیت، تایمر نگهبان زمان فعال می شود و برای غیرفعال کردن تایمر Watchdog باید علاوه بر صفر کردن بیت WDE، بیت WDTOE را نیز یک کرد.

روش غیر فعال کردن تایمر نگهبان زمان:

۱. نوشتن یک منطقی در بیت های WDTOE و WDE
۲. نوشتن صفر منطقی پس از چهار سیکل در بیت WDE

بیتWDTOE-4 watchdog Turn-off Enable

زمانی که در بیت WDE صفر منطقی نوشته می شود باید این بیت یک شود. در غیر این صورت تایمر نگهبان زمان نمی تواند غیرفعال شود. با نوشتن یک منطقی در این بیت، پس از ۴ سیکل توسط سخت افزار به طور اتوماتیک صفر می شود.

استفاده از تایمر نگهبان زبان در برنامه نویسی C

شاید سوال شما این باشد که چه لزومی به استفاده از تایمر watchdog وجود دارد؟ اصلا چرا باید میکروکنترولر reset شود؟ جواب این است که میکروکنترلر ممکن است در حلقه ای از برنامه منتظر دیتایی شده باشد ولی آن دیتا را به علت نامشخصی دریافت نمی کند و یا این که نویز سبب شده است که برنامه میکروکنترلر به بردار آدرس نامشخصی از حافظه پرش کرده باشد. پس در هر صورت میکروکنترلر هنگ کرده است. با فعال کردن تایمر watchdog می توان محتوای این تایمر را قبل از رسیدن به مقدار نهایی خود، توسط دستور اسمبلی زیر reset کرد :

```
#asm   ("WDR")
```

بنابراین تا زمانی که برنامه به درستی کار میکند محتوای این تایمر توسط برنامه reset می شود و اجازه reset کردن میکروکنترلر را نمی دهد. اما اگر میکروکنترلر به قسمتی از برنامه رسید و نتوانست به موقع تایمر نگهبان را reset کند آن گاه تایمر watchdog این فرصت را پیدا می کند که به مقدار نهایی خود برسد و میکروکنترلر را reset کند و برنامه را از سرگردانی نجات دهد.

جهت فعال کردن تایمر watchdog ، به طور مثال با زمان سرریز یک ثانیه، باید رجیستر WDTCR به صورت ;WDTCR=0X0E مقداردهی شود و برای غیر فعال کردن این تایمر از تابع زیر استفاده می کنیم:

```
Void  WDT_off (void)   {          / / تابع غیر فعال کردن تایمر نگهبان
WDTCR=0X18;                       / / نوشتن یک در بیت های WDE و WDTOE
WDTCR=0X00;                       / / خاموش شدن تایمر نگهبان زمان
}
```

فصل ۲

اصول طراحی و آموزش زبان C

اهداف

۱. آشنایی با اصول طراحی و ترسیم الگوریتم

۲. آموزش دستورات برنامه نویسی C مخصوص میکروکنترلر

۳. معرفی توابع کتابخانه ای و کاربرد آن ها در برنامه نویسی

۲-۱ اصول طراحی

در یک طراحی الکترونیکی می بایست اصول و قوانین آن رعایت شود. هر شخص طراح می بایست دارای سه خصوصیت باشد:

۱. دایره اطلاعاتی (علم روز و ابزارات جدید را خوب بشناسد و خود را بتواند updateکند)
۲. شناخت سخت افزار (طراحی PCB، شناخت سنسور و میکرومنترلر و شناخت PLC و...)
۳. آگاهی کامل از یک زبان برنامه نویسی (زبان C، پاسکال ، بیسیک، اسمبلی، VHDL و...)

هر طراحی سه مرحله دارد که یک به یک به این مراحل می پردازیم.

۱. فهمیدن صورت مساله اولین گام برای یک طراحی است

برای این که یک شخص طراح بتواند موفق باشد می بایست صورت مساله را درک کند. گاهی اوقات پیشنهاد طراحی با جمله ای بسیار کوتاه و یا بسیار بلند داده می شود ولی یک طراح باید در درجه نخست خوب بفهمد که متقاضی طرح چه سیستمی مورد نظرش می باشد و در درجه دوم باید معیارهایی که در صورت مساله وجود ندارد را در نظر بگیرد.

مثال ۲-۱: سیستمی را طراحی کنید که دمای یک دستگاه صنعتی را اندازه بگیرد و هر موقع که از دمای تنظیم شده، درجه حرارت بالاتر رفت دستگاه را با خاموش کردن آن محافظت کند.

توضیح **مثال ۲-۱ مطرح شده**: در این مثال طراحی، شما می بایست در ظاهر سیستمی را طراحی کنید که بتواند دما را اندازه بگیرد و قابلیت تنظیم را نیز داشته باشد. ولی اگر به باطن صورت مساله توجه کنیم در می یابیم که این دستگاه در صنعت می خواهد استفاده شود، پس قضیه نویز نباید فراموش شود. طراحی باید به گونه ای باشد که در قبال نویز مصونیت داشته باشد. نکته دیگر این که دمای محیط اطراف سنسور موجب تغییرات لحظه ای سیستم می شود یعنی اگر برای نمونه دمای تنظیم شده ۴۰ درجه سانتی گراد باشد و بالای این دما دستگاه به وسیله سیستم طراحی شده خاموش شود و دما برای لحظه ای به ۳۹ درجه سانتی گراد برسد، دستگاه دوباره روشن می شود. پس این یعنی به نوسان افتادن ناخواسته سیستم و این مشکل بزرگی است زیرا در این حالت نه تنها دستگاه محافظت نمی شود بلکه با روشن و خاموش شدن مکرر، ممکن است دستگاه آسیب ببیند. برای حل این مشکل باید یک محدوده مرده تعریف کنیم که بالای عدد تنظیم شده دستگاه را خاموش کند و در دمایی با اختلاف قابل تنظیم دستگاه را روشن کند.

برای نمونه بالای ۴۰ درجه دستگاه را خاموش کند و در دمای زیر ۳۶ درجه سانتی گراد دستگاه را روشن کند. ضمنا در صورت مساله عنوان نشده است که ماکزیمم دما چقدر است که ما آن را در انتخاب دمای سیستم معیار قرار دهیم. همچنین فاصله قرار گرفتن سنسور از محل مورد نظر تا سیستم نادیده گرفته شده است.

مثال ۲-۲: سیستمی برای راه اندازی یک موتور سه فاز طراحی کنید که اول به صورت ستاره شروع به کار کند و بعد از مدتی به حالت مثلث برود.

توضیح: در این طراحی نیز، شما باید علاوه بر قابلیت تغییر زمان بین راه اندازی ستاره به مثلث، برای این سیستم کلیدهای start و stop در نظر گرفته و ضمنا موقعی که می خواهد از حالت ستاره به مثلث تغییر حالت بدهد بهتر است یک زمان تاخیری کوتاهی قرار داده شود.

۲. طراحی الگوریتم برنامه (فلوچارت)

دومین مرحله از اصول طراحی، الگوریتم برنامه نرم افزاری پروژه می باشد. الگوریتم ها در طراحی و عیب یابی کارایی ویژه ای دارند و برای تمام برنامه نویسان یک اصل مهم می باشد و حسن آن در این است که کاربر بر اساس آن می تواند برنامه بنویسد. مزیت دیگر آن این است که کاربر بعد از یک مدتی اگر به الگوریتم برنامه نگاه کند سریعا متوجه عملکرد برنامه می شود. در این مرحله طراح می بایست فارغ از این که با چه میکروکنترلری کار می کند و به چه زبان برنامه نویسی مسلط است باید برنامه را به گونه پرسش و پاسخ که همان طراحی الگوریتم (فلوچارت) است در بیاورد و مطابق آن مراحل بعدی را طی کند. البته زمانی که شما یک برنامه نویس حرفه ای شوید شاید الگوریتم یک برنامه را در ذهن خود تجسم کنید و نیازی به ترسیم آن نداشته باشید ولی اندکی دشوار به نظر می رسد. پروژه های این کتاب چندان پیچیده نبوده و نیازی به ترسیم الگوریتم نداشته اند.

در طراحی فلوچارت باید نکات زیر رعایت شود:

۱. نخست این که باید در متن نوشتاری داخل بلوک ها خلاصه نویسی رعایت شود به طوری که آن جمله کوتاه بتواند مفهوم عملکرد لازم را برساند.
۲. بلوک های مرتبط با هم در کنار یکدیگر قرار گیرند تا از پیچیدگی فلوچارت ها جلوگیری شود.
۳. برای برنامه های بزرگ بهتر است اول قسمتی از برنامه را طراحی کرده و سپس قسمت های دیگر را بعد از موفقیت در قسمت قبل دنبال کنید.

۳ . نوشتن برنامه نرم افزاری و طراحی سخت افزار پروژه

در این مرحله کاربر می بایست انتخاب کند که اول سخت افزار را طراحی کند یا این که برنامه نرم افزاری را بنویسد. برای این مرحله از کار، سه حالت وجود دارد:

۱.سخت افزار پیچیده ولی نرم افزار آسان

در چنین وضعیتی بهتر است که اول سخت افزار را طراحی کنید زیرا اگر نتوانید از عهده سخت افزار آن برآیید حتما نوشتن نرم افزار کمکی نخواهد کرد.

۲.سخت افزار آسان ولی نرم افزار پیچیده

در این حالت نیز راه پیچیده را انتخاب می کنیم چرا که اگر نتوانیم نرم افزار را بنویسیم طراحی سخت افزار راهی اشتباه است.

۳.سخت افزار و نرم افزار پیچیده

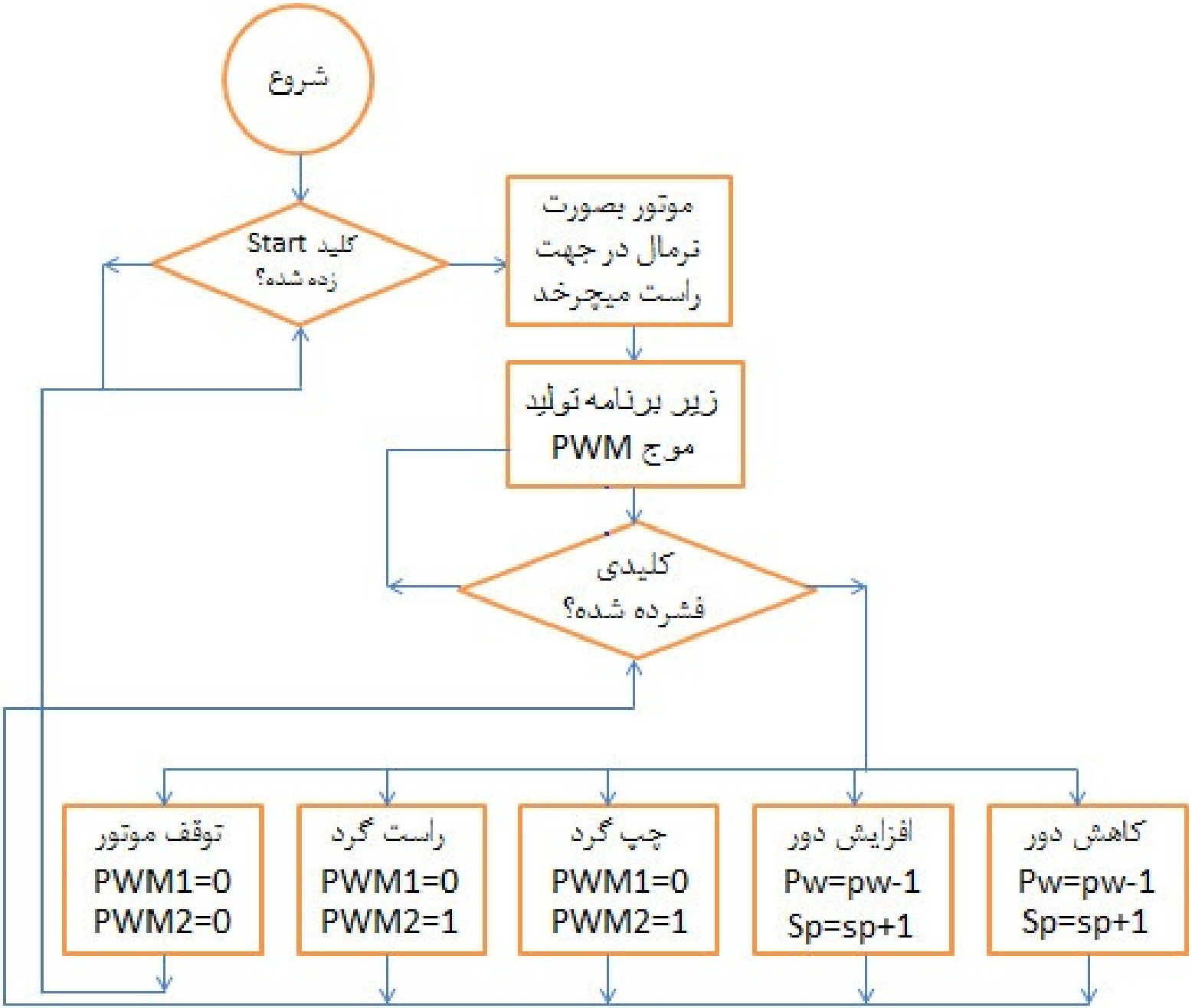
در این حالت شما ابتدا به دنبال نرم افزار پروژه بروید و سخت افزار را یک بلوک با یک سری ورودی و خروجی در نظر بگیرید زیرا شما در این شرایط می توانید برای طراحی سخت افزار از دیگران کمک بگیرید اما در مورد برنامه نویسی فقط خودتان باید تلاش کنید.

خصوصیاتی که سخت افزار پروژه باید داشته باشد:

۱. تا حد امکان سخت افزار ساده و کم حجم باشد.
۲. در انتخاب ورودی و خروجی های میکروکنترلر بهترین حالت را انتخاب کنیم.
۳. PCB پروژه را خوب طراحی کینم.
۴. مدار را تا می توانیم از نویز پذیری مصون کنیم (این کار را با گذاشتن فیلترها و خازن های حذف نویز و عایق بندی با فلز متصل به زمین و سیم های شیلد دار و .. انجام می دهیم)

خصوصیاتی که نرم افزار پروژه باید داشته باشد:

۱. تا آن جایی که ممکن است کم ظرفیت باشد.
۲. توابع و دستورات اضافی در آن وجود نداشته باشد.
۳. برنامه را به زبانی که تسلط دارید بنویسید.
۴. قبل از نوشتن نرم افزار، ورودی و خروجی ها را تعریف کنید.



الگوریتم ساده راه اندازی موتور DC

نکاتی که از الگوریتم شکل پیشین در طراحی کنترل موتور DC باید رعایت شود:

۱. برنامه باید دارای مقادیر پیش فرض باشد و موتور ابتدا نرم راه اندازی شود.

۲. به جز کلید start بقیه کلیدها می بایست از نوع وقفه بیرونی میکروکنترلر استفاده شوند تا عملکرد بسیار سریعی داشته باشند.

۳. ماکزیمم و مینیمم درصد PWM باید مشخص شود.

۲-۲ آموزش زبان C

در زبان C کاربر می بایست دستورات، توابع و سخت افزار تراشه ای که با آن کار می کند را بشناسد. زمانی که ما یک برنامه را به زبان C می نویسیم پس از کامپایل شدن به فایل های گوناگونی مانند فایل هگزاد و اسمبلی تبدیل می شود که در فصل ۳ توضیح خواهم داد.

مفاهیم اولیه زبان C

- در زبان C بین حروف کوچک و بزرگ تفاوت وجود دارد. در این زبان دستورات و کلمات کلیدی با حروف کوچک نوشته می شوند. به عنوان مثال: کلمه کلیدی void با کلمه VOID فرق دارد و اگر دستورات و کلمات کلیدی را با حروف بزرگ بنویسیم کامپایلر پیغام خطا می دهد.

- هر دستور به ; ختم می شود که نشان دهنده آن است که خط دستوری پایان یافته است.

- حداکثر طول یک دستور، ۲۵۵ کاراکتر است.

- هر دستور را می توان در یک سطر یا چند سطر نوشت.
- در هر سطر میتوان چند دستور نوشت.
- توضیحات در یک سطر می توانند بعد از // قرار بگیرند و همچنین توضیحات گروهی با /* شروع میشوند و به */ ختم می شوند.

ساختار کلی زبان C

```
#includ  <نام تراشه.h>                    // معرفی فایل های سرآمد یا الحاقی
#include  <نام کتابخانه.h>
#define  out1   PORTC.0                   // معرفی شناسه ها
 .       .
 .       .
 .       .
unsigned   char   X,Y;                   // معرفی متغیرهای ۸ بیتی کلی(همگانی)
unsigned   int   n, count;               // معرفی متغیرهای ۱۶ بیتی کلی(همگانی)
 .       .
 .       .
 .       .
void   function1 ( ) {                   // توابع فرعی
unsigned   char   k, J;                  // معرفی متغیرهای ۸ بیتی محلی
 دستورالعمل ها;
}
 .       .
 .       .
 .       .
void main ( ) {                          // تابع اصلی برنامه
دستورالعمل ها;
function1 ( ) ;                          // فراخوانی توابع فرعی
 while (1) {                             // حلقه بی نهایت برنامه
    };
  }
```

پیش پردازنده ها

فایل های سرآمد توسط دستورات پیش پردازنده در ابتدای برنامه معرفی (گنجانده) می شوند. در زبان C می بایست میکروکنترلری که استفاده می کنید را معرفی نمایید و بعد از آن کتابخانه هایی را که به آن ها در برنامه نویسی نیاز دارید، باید معرفی کنید. توجه داشته باشید که کتابخانه ها برای میکروکنترلرها و کامپایلرهای مختلف، متفاوت می باشند.

در واقع فایل های سرآمد، فایل های از قبل نوشته شده ای می باشند که در پوشه INC در مسیر نصب نرم افزار قرار دارند. حداکثر فایل های سرآمد در برنامه، بستگی به نوع کامپایلر دارد. مثلا نرم افزار Code vision AVR، ۱۶ فایل سرآمد در یک برنامه را می پذیرد.

فایل های سرآمد را با دستور پیش پردازنده #include در بین دو علامت < > قرار می دهیم و پسوند این گونه فایل ها *.h می باشد. بعضی از کامپایلرها اجازه ساخت کتابخانه و فایل الحاقی را می دهند که در فصل سوم در این مورد بحث خواهیم کرد.

تعریف شناسه ها یا ثوابت

شناسه (ماکرو) برای تعریف برچسب هایی معادل یک نام رشته ای یا هر مقداری می باشند و ضمنا می توانند برای تعریف ماکروهای شبه تابع استفاده شوند که در واقع شبیه دستور EQU در زبان اسمبلی عمل می کند. برای درک بیشتر به مثال های زیر توجه فرمایید.

مثال ۲-۳ :

```
#define    name        "micro“
#define    number      125.456
#define     out1       PORTA. 5
```

در این تعاریف، کلمه name با مقدار رشته ای "micro" برابر است و number نیز با عدد 125.456 برابر است.

فراموش نکنید که شما ثوابت را تعریف می کنید بنابراین مقدارهای number یا name نمی توانند در برنامه تغییر کنند. کلمه out1 معادل PORTA. 5 می باشد و می تواند صفر یا یک شود.

مثال ۲-۴ :

```
#define   sum ( x , y)    x + y
i =sum(4,6);
```

در این تعاریف ماکرو، عمل جمع انجام می شود. بنابراین اعداد ۴ و ۶ با هم جمع می شوند و حاصل آن یعنی عدد ۱۰ در متغیر از قبل تعریف شده، یعنی متغیر i قرار می گیرد.

متغیر ها

متغیر نامی است که ما برای حافظه موقت یا دائمی میکروکنترلر تعریف می کنیم. متغیر یا داده می تواند در توابع دارای مقدار باشد و اعمال منطقی و ریاضیاتی بر روی آن در قالب یک تابع انجام پذیرد. در واقع یک کاربر با متغیر کار دارد و نیازی نیست که ثبات ها و آدرس مکانی حافظه را به خاطر بسپارد هر چند که می توان نحوه ذخیره سازی یک متغیر را تعیین کرد. برای نامگذاری متغیرها از حروف کوچک و بزرگ لاتین (a...z or A...Z) و همچنین از علائمی نظیر آندرلاین (_) استفاده می شود. توجه داشته باشید که نمی توانید از کلمات کلیدی و دستورات به عنوان نام متغیر استفاده کنید. همچنین نباید متغر با عدد شروع شود ولی می تواند به عدد ختم شود. متغیرها در زبان C می توانند ۸ بیتی، ۱۶بیتی، ۳۲ بیتی و اعداد اعشاری باشند. این که متغیر را از چه نوع داده ای تعریف کنیم بستگی به مقداری است که می خواهیم در آن قرار دهیم. در جدول ۲-۱ محدوده مقدار متغیرها آورده شده است.

توجه: هر چقدر شما داده ها را از نوع ۸ بیتی تعریف کنید و کم تر از متغیر های کلی استفاده کنید حجم کد برنامه کاهش می یابد.

انواع داده ها در زبان C

جدول ۲-۱

نوع متغیر	اندازه بر حسب بیت	محدوده تغییرات
bit	1	0 or 1
char یا signed char	8	-128 to 127
signed char	8	0 to 255
int یا signed int	16	-32768 to 32767
signed int	16	0 to 65535
signed short int	16	-32768 to 32767
unsigned short int	16	0 to 65535
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
float	32	±1.175e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

نحوه تعریف متغیر

متغیرها در زبان C دو گونه اند؛ یکی کلی و دیگری محلی که نحوه تعریف آن ها مشابه می باشد و تنها محل تعریف آن ها در برنامه نویسی متفاوت است.

انواع متغیر در زبان C

- **متغیر های کلی:** در ابتدای برنامه و خارج از توابع می آید و مقدار آن ها در تمامی توابع قابل دسترس است.

- **متغیر های محلی:** در داخل بدنه توابع تعریف می شوند و مقدار آن ها با خارج شدن از بدنه توابع از بین می رود و صفر می شود.

معرفی متغیر به شکل زیر است: نام متغیر نوع داده ;

مثال ۲-۵: دو متغیر ۸ بیتی بی علامت x و y تعریف کنید و به متغیر x مقدار اولیه ۱۲ بدهید و یک متغیر دیگر با نام k از همین نوع در آدرس ۲۰۰ هگزاد SRAM و همچنین یک متغیر از نوع اعشاری با نام fv تعریف کنید.

```
Unsigned char x=12,z;
Unsigned char k@0x200;
float fv=0.0;
```

همان گونه که ملاحظه می کنید متغیرهای x,z چون از یک نوع داده بودند در یک خط تعریف شدند و با علامت کاما (،) از هم جدا شدند. این کار برای تعریف سریع تر متغیر در یک خط به کاربر کمک می کند. می توانسیتم متغیرهای x و z را به شکل زیر نیز تعریف کنیم و مقدار x را در ابتدا قرار ندهیم.

```
Unsigned char x;
Unsigned char z;
X=12;
```

فرمت یا مبنای متغیر ها

⇐ ( ) : اگر شما قبل از عدد علامتی قرار ندهید به معنای دسیمال است.

```
unsigned char X=15;   // به معنی مقدار دادن ۱۵ دسیمال می‌باشد
unsigned char X=15U;  // قرار دادن حرف U بعد از دسیمال اختیاری است
float X=3.14F;        // قرار دادن حرف F بعد از یک داده اعشاری اختیاری است
```

⇐ (0x) : اگر قبل از یک عدد بیاید به معنی هگزاد است.

```
unsigned char X=0x12;  // به معنی قرار دادن مقدار ۱۲ هگزاد می‌باشد
```

⇐ (0b) : اگر قبل از عدد بیاید به معنی باینری است.

```
unsigned char X=0b11001111;// به معنی قرار دادن مقدار بر حسب باینری می‌باشد
```

⇐ (" ") : اگر کلمه‌ای در بین دو علامت دابل کوتیشن قرار گیرد یعنی مقدار رشته‌ای است.

```
char X[]="alvandi";
```

⇐ (' ') : اگر یک کاراکتر در بین دو علامت کوتیشن قرار گیرد مقدار بر حسب اسکی است.

```
unsigned char X='s';
```

کلاس ذخیره سازی متغیر ها

در معرفی یک متغیر می توان نحوه ذخیره سازی آن را تعریف کرد. برای تعیین کلاس ذخیره سازی می بایست پیش از تعریف نوع داده متغیر، کلاس آن را تعیین کنیم. فرم کلی:

<کلاس ذخیره سازی حافظه> <نوع داده> نام متغیر;

انواع کلاس ذخیره سازی:

auto , static , extern , register

auto: متغیرهایی که در داخل بدنه یک تابع تعریف می شوند، متغیرهای محلی نامیده می شوند. این متغیرها با فراخوانی یک تابع در درون آن شکل می گیرند و با برگشتن از تابع از بین می روند. به این نوع کلاس ذخیره سازی، کلاس حافظه اتوماتیک گفته می شود. این حالت برای متغیرهای محلی پیش فرض کامپایلر می باشد و نیاز نیست که پیش از نوع داده واژه auto بیاید به عبارت دیگر متغیرهای تعریف شده در یک تابع از نوع اتوماتیک هستند.

static: متغیرهای محلی که از نوع ثابت تعریف می شوند با خارج شدن از یک تابع از بین نمی روند و تنها در بدنه همان تابعی که تعریف شده اند قابل دسترسی هستند.
متغیرهای کلی در واقع متغیرهای ثابتی هستند که در همگی توابع قابل دسترسی اند و محتوای آن ها از بین نمی روند.

extern: این گونه کلاس برای متغیرهایی است که در یک فایل دیگر (مثل فایل سرآمد) معرفی و مقدار دهی اولیه شده است و در فایل جاری برنامه، می توان از آن ها استفاده کرد.

register: این گونه کلاس نحوه ذخیره شدن یک متغیر را در یکی از رجیسترهای میکروکنترلر تعیین می کند و تنها در قالب متغیرهای محلی کاربرد دارند.

جدول ۲-۲ عملگرهای محاسباتی

مثال	مفهوم	عملگر	تقدم عملگر
X++	افزایش به اندازه یک واحد	++	اولویت اول
x--	کاهش به اندازه یک واحد	--	
-x	علامت منفی	-	اولویت دوم
x * y	ضرب	*	اولویت سوم
X/10	تقسیم (خارج قسمت)	/	
X%10	تقسیم (باقی مانده)	%	
x-y	تفریق	-	اولویت چهارم
X+y	جمع	+	

مثال ۲-۶ : مقدار متغیر a=14 را پس از یک واحد افزایش در متغیر b قرار دهید؟

```
Unsigned char a=14,b;
b=a++;    // b=14
```

(بخش نخست)

```
Unsigned char a=14,b;
b=++a;    //b=15
```

(بخش دوم)

در مثال ۲-۶ بخش نخست اشتباه می باشد. چون عملگر ++ در اولی پس از متغیر a آمده است بنابراین ابتدا مقدار a در متغیر b قرار می گیرد و سپس یک واحد به متغیر a افزوده می شود.

مثال ۲-۷ چه مقداری در متغیر z پس از انجام عبارت زیر قرار می گیرد؟

```
unsigned int x=7, y=6 , z;
z=x+y*6/2
```

مثال ۲-۸ : چه مقداری در متغیر z پس از انجام عبارت زیر قرار می گیرد؟

```
unsigned int x=7, y=6 , z;
z=(x+y)*(6/2);
```

جدول ۲-۳ **عملگر های مقایسه ای (رابطه ای)**

مثال	مفهوم	عملگر	تقدم عملگر
a>b	بزرگتر	>	اولویت اول
a>=b	بزرگتر یا مساوی	>=	
a<b	کوچکتر	<	
a<=b	کوچکتر یا مساوی	<=	
a==b	متساوی	==	اولویت دوم
a!=b	نامساوی	!=	

جدول ۲-۴ عملگرهای منطقی

مثال	مفهوم	عملگر	تقدم عملگر
!x	نقیض (not)	!	اولویت اول
a && b	وَ (and)	&&	اولویت دوم
a || b	یا (or)	||	اولویت سوم

جدول ۲-۵ عملگرهای بیتی و منطقی

عملگر	مفهوم	مثال	معادل
=	مساوی یا mov	x=12	
*=	انتساب ضرب	x*=y	x=x*y
/=	انتساب تقسیم	x/=10	x=x/10
%=	انتساب باقی مانده تقسیم	x%=10	x=x%10
+=	انتساب جمع	x+=y	x=x+y
-=	انتساب تفریق	x-=y	x=x-y
&=	انتساب AND بیتی	x&=z	x=x&z
^=	انتساب XOR بیتی	x^=z	x=x^z
|=	انتساب OR بیتی	x|=z	x=x|z
<<=	انتساب شیفت به چپ	x<<=5	x=x<<5
>>=	انتساب شیفت به راست	x>>=3	x=x>>3

جدول ۲-۶ عملگرهای انتسابی یا ترکیبی

تقدم عملگر	عملگر	مفهوم	مثال	نتیجه
اولویت اول	~	مکمل ۱	~ (0x66)	0x9A
اولویت دوم	>>	شیفت به راست	0b11001011 >> 4	0b00001100
	<<	شیفت به چپ	0b11001011 << 4	0b10110000
اولویت سوم	&	AND بیتی	0x66 & 0xff	0x66
اولویت چهارم	^	XOR بیتی	0x12 ^ 0x12	0
اولویت پنجم	|	OR بیتی	0x05 | 0x30	0x35

عملگر شرطی ؟

فرم کلی استفاده از این عملگر به گونه زیر است:

<عبارت ۳> : <عبارت ۲> ؟ <عبارت۱> = متغیر

در این فرم دستوری، عبارت شرطی اول مورد ارزیابی قرار می گیرد که اگر نتیجه آن درست باشد، عبارت دوم در متغیر قرار می گیرد و در غیر این صورت عبارت سوم در متغیر قرار خواهد گرفت.

مثال ۲-۹:

```
Y=(x>z) ?   X : z
```

در مثال ۲-۹ مقدار x و z با هم مقایسه می شود که اگر x بزرگ تر باشد، عبارت دوم یعنی x در متغیر y قرار می گیرد و گرنه مقدار z در متغیر y قرار خواهد گرفت.

عملگرهای & و *

با استفاده از عملگر & می توانیم به آدرس یک متغیر و با استفاده از عملگر * می توانیم به محتوای آدرس یک متغیر دسترسی داشته باشیم.

مثال ۲-۱۰ :

```
unsigned char y@0x400;     //  معرفی متغیر در مکان ۴۰۰ هگزاد در SRAM
unsigned char *x,I;        //  معرفی متغیر اشاره گر x و متغیر دلخواه i
y=15;                      //  مقداردهی متغیر با عدد ۱۵ بر حسب دسیمال
x=&y;                      //  متغیر x برابر آدرس متغیر y یعنی ۴۰۰ هگزاد می شود
i=*x                       //  محتوای مکانی که x به آن اشاره می کند را درون i کپی می کند
```

در مثال ۲-۱۰ آدرس متغیر y، ۴۰۰ هگزاد و مقدار آن ۱۵ دسیمال می باشد و در سطر چهارم مقدار x برابر آدرس y می شود و در سطر پنجم، x به مکان ۴۰۰ هگزاد از SRAM داخلی اشاره می کند و محتوای آن مکان یعنی ۱۵ دسیمال را به متغیر i کپی می کند.

توجه: اگر از اشاره گر * استفاده می کنید باید توجه داشته باشید که نوع داده اشاره گر، باید با نوع داده آرایه یا رشته مورد نظر، یکی باشد. این را نیز به یاد داشته باشید که عملگرهای & و * را با عملگرهای مشابه انتسابی اشتباه نگیرید.

عملگر کاما (,): این عملگر برای جداسازی چند متغیر، عبارت و عمل دستوری می باشد.

عملگر ( ) size of : این عملگر طول یک متغیر یا نوع داده را بر حسب بایت بر می گرداند.

```
unsigned int y,x;
x=sizeof () ;          // طول نوع داده int بر حسب بایت ۲ می باشد، پس مقدار x=۲ می شود
```

عملگر ( ): پرانتز ها عملگرهایی هستند که تقدم عملگر های داخل خود را بالا می برند و در تعریف توابع نیز به کار می روند. (جدول ۲-۷)

تقدم عملگر ها در حالت کلی

عملگر	تقدم	عملگر	تقدم
( )	۱	&	۸
! ~ ++ -- sizeof	۲	^	۹
* / %	۳	|	۱۰
+ -	۴	&&	۱۱
<< >>	۵	||	۱۲
< <= > >=	۶	?	۱۳
== !=	۷	= += -= *= /= %=	۱۴

دستورات زبان C

دستورات زبان C در قالب بدنه یک تابع می آیند و یک عمل خاصی را بر روی متغیرهای کلی و محلی انجام می دهند. هر دستور در زبان C با علامت } باز می شود و با علامت { بسته می شود.

۱. دستور شرطی if-else : از این دستور برای ارزیابی یک شرط کلی در برنامه استفاده می شود.

ساختار کلی دستور به این گونه است:

```
if ( شرط ) {
دستورالعمل‌های ۱ ;
}
else{
دستورالعمل‌های ۲ ;
}
```

در این دستور، نخست شرط if بررسی می گردد که اگر درست باشد دستورالعمل های ۱ انجام می شوند و در غیر این صورت دستورالعمل های ۲ اجرا می شوند.

مثال ۲-۱۲ :

```
unsigned  char  a,b;
if (a > b){
  a++ ;
  b+=10;}
else{
  a-- ;
  b-=10;
}
```

در مثال ۲-۱۲ اگر متغیر a از متغیر b بزرگ تر باشد، a یک واحد افزایش پیدا کرده و به متغیر b، عدد ۱۰ افزوده می شود و در غیر این صورت متغیرa یک واحد کم شده و از متغیر b، عدد ۱۰ کم می شود.

```
unsigned  char  zx,sd;
if (zx==sd) zx++;
else sd++;
```

مثال ۲-۱۳:

در این مثال اگر هر دو متغیر با هم برابر باشند، به متغیر zx یک واحد افزوده و در غیر این صورت یک واحد به متغیر sd افزوده خواهد شد. همچنین می بینید که علامت های {} وجود ندارند. زمانی که یک دستورالعمل وجود داشته باشد نیازی به علامت } و { نمی باشد.

مثال ۲-۱۴ :

```
unsigned  char  data,i;
bit x,y;
if ((x==1)&&(y==0))
{
data*=10;
i=data;
}
```

در مثال ۲-۱۴ متغیرهای تک بیتی x و y با هم مقایسه می شوند که اگر x=1 باشد و y=0 باشد data در عدد ۱۰ ضرب و در متیغر i کپی می شود. همچنین در این مثال مشاهده کردید که علامت } پایین دستور if آمده است و هیچ فرقی ندارد. ضمنا همیشه نیاز نیست که با هر if یک else وجود داشته باشد ولی عکس این قضیه صادق نیست.

مثال ۲-۱۵:

```
unsigned  int  x,y,z;
if ((x==y) && (z==15)) x++;
else if ((x>=y) && (z==15)) y++;
else if ((x<=y) && (z==15)) z++;
```

همان جوری که می بینید از دستور else if برای شرط های متوالی استفاده شده است. اگر شرط سطر دوم درست باشد متغیر x یک واحد افزایش پیدا میکند در غیر این صورت شرط سطر سوم مورد ارزیابی قرار می گیرد و به همین گونه همگی دستورها پیاپی تست میشوند.

۲. دستور حلقه for: از این دستور برای ساختار حلقه شرطی در برنامه و زمانی که نیاز به کانتر حلقه باشد، استفاده می شود. فرم کلی این دستور به صورت زیر است:

```
for ( مقداردهی اولیه ; شرط حلقه ; شمارنده حلقه ) {
دستورالعمل‌ها;
}
```

ابتدا متغیر حلقه، مقداردهی اولیه می شود و در هر مرحله از اجـرای حلقـه، یـک واحـد شمارنده حلقه افزایش پیدا می کند تا زمانی که شرط حلقه درست باشد، این حلقه تکرار می شود.

مثال ۲-۱۶:

```
unsigned  char i;
unsigned int w;
for ( i=0 ; i < 25 ; i++){
w+=5;
}
```

در مثال ۲-۱۶ متغیر حلقه، یعنی i در آغاز حقله صفر است و در هر مرحله که حلقه تکرار می شود یک واحد افزایش پیدا می کند تا زمانی که مساوی ۲۵ شده و شرط حلقه نقض می شود و خط برنامه از حلقه خارج می شود. بنابراین این حلقه ۲۵ بار تکرار می شود و مقدار ۵، در هر مرحله از تکرار حلقه به متغیر W افزوده می شود. همچنین در این دستور به جهت وجود یک دستورالعمل می توانستیم علامت های {} را حذف کنیم. توجه کنید که مقداردهی اولیه یک بار در آغاز ورود به حلقه انجام می شود و در هر بار تکرار حلقه، مقداردهی اولیه انجام نمی شود.

حلقه در حلقه با دستور for:

در بسیاری از موارد نظیر تاخیر های طولانی، برگرداندن کدهای آرایه مانند برنامه تابلو روان و ... ما نیاز به حلقه های تو در تو داریم که آن ها را با دستور for به وجود می آوریم.

مثال ۲-۱۷:

```
unsigned  char i,j,z=2;
for ( i=10 ;i>0 ;i--){
      for ( j=0 ;j <=50;j++){
        z*=10;
        if (z ==140) z=2;
     }
 }
```

در مثال ۲-۱۷ در آغاز، برنامه وارد حلقه نخست شده و متغیر i=10 می شود و چون بزرگ تر از عدد صفر (شرط درست) است وارد حلقه دوم شده و متغیر حلقه دوم یعنی j=0 می شود و چون j کوچک تر مساوی عدد ۵۰ می باشد، این حلقه ادامه پیدا می کند و متغیر z در عدد ۱۰ ضرب شده و با دستور if مورد ارزیابی قرار می گیرد که اگر مقدار آن برابر عدد ۱۴۰ شود، متغیر z برابر ۲ می گردد. بنابراین حلقه دوم ۵۱ مرتبه تکرار می شود و سپس برنامه از حلقه دوم خارج می شود و یک واحد از شمارنده حلقه اول کم شده و چون شرط همچنان برقرار است دوباره وارد حلقه دوم می شود. بنابراین نتیجه می گیریم که حلقه دوم ۱۰ مرتبه در حلقه اول تکرار می شود و در هر مرحله خود حلقه دوم ۵۱ بار تکرار می شود.

ایجاد کردن حلقه بی نهایت با استفاده از دستور for به گونه زیر است:

```
for(; ;) {
; دستورالعمل ها
}
```

۳. دستور حلقه شرطی while :

این دستور نیز برای تکرار اجرای دستور ها در یک حلقه برنامه می باشد با این تفاوت که فاقد شمارنده حلقه است. این دستور نخست شرط حلقه را تست می کند، اگر درست باشد خط برنامه وارد حلقه می شود و در غیر این صورت این حلقه هرگز اجرا نمی شود. این دستور پس از هر بار تکرار حلقه، شرط را بررسی می کند. اگر نتیجه شرط حلقه درست باشد (غیر صفر باشد) حلقه را ادامه می دهد و در غیر این صورت از حلقه خارج می شود.

فرم کلی این دستور ها به گونه زیر است:

```
while (شرط)  {
دستورالعمل ها
}
```

مثال ۲-۱۸:

```
unsigned char a,b;
while (a > b){
 a=(a/b*2);
}
```

در این مثال، نخست متغیر a با b مقایسه شده و اگر a بزرگ تر از b باشد، برنامه وارد حلقه شده و در هر بار تکرار حلقه شرط مورد تست قرار می گیرد. همان گونه که پیش تر گفته شد در زمانی که یک دستورالعمل وجود دارد می توانیم علامت های {} را قرار ندهیم ولی گذاشتن آن ها خطا محسوب نمی شود

ایجاد کردن حلقه بی نهایت با استفاده از دستور while به گونه زیر است:

```
While (1) {
دستورالعمل ها ;
}
```

۴. دستور حلقه شرطی do - while :

این دستور نیز مانند دستور پیشین است با این این تفاوت که نخست اجازه تکرار یک بار حلقه را می دهد و در پایان حلقه، شرط را بررسی می کند. فرم این دستور به گونه زیر است:

```
do {
دستورالعمل ها ;
} while (شرط) ;
```

مثال ۲-۱۹ :

```
unsigned char i,sw,y ;
do {
i ++ ;
y=sw*i ;
} while (i<10) ;
```

در این مثال، نخست برنامه وارد حلقه شده و متغیر شرط حلقه یک واحد افزایش پیدا می کند و متغیر sw در i ضرب شده و حاصل آن در y قرار می گیرد و شرط در پایان حلقه تست می شود. اگر شرط درست بود دوباره حلقه تکرار می شود و در غیر این صورت از حلقه خارج می گردد.

ایجاد کردن حلقه بی نهایت با استفاده از دستور do – while به گونه زیر است:

```
do {
دستورالعمل ها ;
} while (1) ;
```

۵. دستور break :

این دستور برای شکستن و خارج شدن بدون شرط از حلقه می باشد. هرگاه برنامه به این دستور برسد از ساختار حلقه (منظور علامت {}) خارج شده و برنامه از نخستین دستور پس از ساختار حلقه ادامه می یابد. معمولا دستور break همراه با دستور switch استفاده می شود.

۶. دستور switch :

یکی از دستور های مهم زبان c دستور مقایسه ای switch می باشد. این دستور یک عبارت (متغیر) را با یک سری از اعداد ثابت مقایسه می کند و عمل خاصی را مطابق مقایسه انجام شده صورت می دهد. در این دستور عبارت (متغیر) با مقدارهای ثابت قرار داده شده مقایسه می شود و اگر با مقداری که جلوی دستور case قرار می گیرد برابر بود، پس از انجام دستورالعمل آن، با استفاده از دستور break سریعا از ساختار دستور switch خارج می گردد.

فرم کلی این دستور بدین گونه است:

```
switch(عبارت) {
case مقدار ۱ :
دستورالعمل‌های اول;
break;
case مقدار ۲ :
دستورالعمل‌های دوم;
break;
default:
دستورالعمل‌های پیش فرض;
}
```

در دستور switch، عبارت تنها با مقادیر ثابت مقایسه می شود، بنابراین نمی توان در جلوی دستور case از متغیر استفاده کرد. همچنین در این دستور می توانیم default را قرار ندهیم که در این صورت اگر عبارت با هیچ یک از مقادیر ثابت جلوی دستورات case برابر نباشد از ساختار دستور switch خارج می شود. اگر در یک case از دستور break استفاده نشود، با مقدار بخش بعدی or می شود.

مثال ۲-۲۰

```
unsigned char x,z;
switch(x){
case '1' : z=10;
break;
case 26 : z=20;
break;
default : z=50;
}
```

در این مثال، مقدار x با عدد اسکی ۱ مقایسه می شود که اگر درست باشد z=10 می شود و از دستور switch خارج می شود ولی اگر برابر نبود با عدد ۲۶ دسیمال مقایسه می شود که اگر درست باشد z=20 خواهد شد و از دستور switch خارج می شود ولی اگر با هیچ یک از مقادیر case ها برابر نبود z به گونه پیش فرض برابر ۵۰ می شود و از switch خارج می شود.

۷. دستور goto :

این دستور برای پرش به یک برچسب محلی درون یک تابع می باشد. فرم آن به گونه زیر است:

goto برچسب ;

مثال ۲-۲۱ :

```
Loop :
دستورالعمل ها ;
goto Loop ;
```

۸. دستور continue :

هر گاه خط برنامه به این دستور برسد برنامه به آغاز حلقه تکرار پرش می کند و شرط بررسی می شود که اگر نادرست باشد حلقه خاتمه می یابد. فرم این دستور:

```
continue ;
```

مثال ۲-۲۲ :

```
unsigned char x=1,i,n;
while(x){
   i++;
   if(n==10){
    x=0;
    continue;
  }
}
```

در مثال ۲-۲۲ برنامه وارد حلقه می شود که شرط آن صفر نبودن x است. متغیر i در هر مرحله تکرار حلقه، یک واحد افزایش پیدا می کند و متغیر n نیز که در بخش دیگری مانند وقفه ها مقداردهی می شود مورد تست قرار می گیرد که اگر برابر با عدد ۱۰ بود x=0 شده و به دستور continue رسیده و خط برنامه به آغاز حلقه پرش کرده و شرط مورد ارزیابی قرار می گیرد و چون شرط برقرار نیست برنامه از حلقه خارج می گردد.

۹. دستور typedef :

با استفاده از این دستور ما می توانیم برای داده ها در زبان C یک نام جدید تعریف کنیم و نوع داده متغیرها را با نام جدید تعریف کنیم. فرم استفاده از این دستورها به گونه زیر است:

typedef نام قدیمی نوع داده نام جدید نوع داده ;

مثال ۲-۲۳ :

```
typedef unsigned int 2byte ;
```

اگر به مانند مثال ۲-۲۳ در آغاز برنامه، نامی برای نوع داده در نظر گرفته شود می توان یک متغیر ۱۶ بیتی بدون علامت را به این شکل تعریف کرد:

```
2byte x ;
```

توابع در زبان c

در حقیقت یک برنامه به زبان C از یک تاب اصلی به نام main و چند تابع فرعی با هر نام مجازی تشکیل شده است. همگی دستورهایی که به شما آموختیم در قالب بدنه یک تابع می توانند استفاده شوند و استفاده از دستور ها در خارج از یک تابع، معنی و مفهومی ندارند. تابع، زیر برنامه ای است که دستورها و فراخوانی توابع دیگر در آن وجود داشته و عمل خاصی را انجام می دهد.

هر برنامه ای در زبان c دارای یک تابع اصلی است. منظور از تابع اصلی، تابع main یک برنامه می باشد. در هنگامی که یک برنامه آغاز می شود نخستین تابعی که اجرا می شود تابع main می باشد. در صورتی که تابع اصلی با نام main نداشته باشیم نرم افزار کامپایلر اعلام خطا می کند. به یاد داشته باشید که تابع اصلی مانند ریشه یک درخت می ماند بنابراین این تابع می تواند انشعاب داشته باشد و دیگر توابع را فراخوانی کند ولی نمی توان از توابع فرعی تابع اصلی را فراخوانی کرد.

توجه:

توابع فرعی می توانند همدیگر را فراخوانی کنند به شرط آن که تابعی که می خواهد فراخوانی شود از پیش به کامپایلر معرفی شده باشد که برای این کار دو راه وجود دارد یکی این که تابع فرعی که می خواهد فراخوانی شود در برنامه نوشتاری بالاتر از تابعی که فراخوانی می خواهد در آن انجام شود بیاید و دیگر این که توابع در آغاز برنامه معرفی شوند.

فرم کلی تعریف تابع به گونه زیر است:

کلاس ذخیره سازی تابع نوع برگشتن تابع نام تابع (آرگومان های تابع) {

دستورالعمل ها ;

return متغیر یا عدد ;

}

همان گونه که پیش تر در مورد تعریف کلاس ذخیره سازی متغیرها بحث کردیم در تعریف توابع نیز می توان کلاس ذخیره سازی را تعریف کرد ولی الزامی نیست. همچنین می توان نوع داده برگشتی به وسیله یک تابع را مشخص کرد. برگشت یک تابع یعنی متغیر یا عددی که با دستور return تعیین می گردد. در واقع توابع مانند یک فرمول ریاضیاتی می باشند که در نهایت پس از انجام اعمال خاصی در برنامه یک حاصلی را نیز به همراه خواهند داشت.

اگر تابعی نوع داده برگشتی آن تعریف نشود، نوع داده پیش فرض کامپایلر می باشد که معمولا از نوع int است. اگر تابعی که استفاده می کنیم دارای برگشت نباشد، نوع برگشتن آن را با اشاره گر void تهی می کنیم. نام تابع نیز اختیاری است و هر نامی که مجاز باشد (حروف لاتین a تا z، A تا Z و اعداد ۰ تا ۹ به شرط آن که در آغاز نام نباشند و یا استفاده از علایمی مانند آندر لاین) را می توان قرار داد. اگر یک تابع درون پرانتز خود متغیرهایی داشته باشد، آرگومان یا پارامتر تابع نامیده می شود و تابع می تواند ضمن فراخوانی، مقادیری را نیز به عنوان پارامتر ورودی به همراه داشته باشد. اگر یک تابع دارای آرگومان نباشد درون پرانتز تابع، واژه کلیدی void یا خالی گذاشته می شود. برای فراخوانی یک تابع نیز از نام تابع با علامت () که به ; ختم می شود استفاده می کنیم.

مثال ۲-۲۴ : یک تابع فرعی با نام display تعریف کرده و آن را فراخوانی کنید.

```
void   display( ){
دستورالعمل‌ها;
}
void  main( ){
display( );
دستورالعمل‌ها;
}
```

همان طور که گفته شد توابع قبل از فراخوانی می‌بایست تعریف شوند حال اگر شما از فایل‌های کتابخانه‌ای استفاده کنید در واقع در داخل این فایل‌ها توابعی که عمل خاصی را انجام می‌دهند تعریف شده است و ما بعد از معرفی فایل کتابخانه‌ای در برنامه، می‌توانیم توابع آن را نیز فراخوانی کنیم که در ادامه مطالب همین فصل به توابع کتابخانه‌ای نیز خواهیم پرداخت.

مثال ۲-۲۵: برنامه یک شمارنده 0 تا 9 با سون سگمنت کاتد مشترک را بنویسید؟

```
#include <mega16.h>                    //معرفی میکروکنترلر استفاده شده
#include <delay.h>                     //معرفی کتابخانه تأخیر زمانی
unsigned char i;                       //تعریف متغیر ۸ بیتی برای کانتر حلقه for
unsigned char mask(unsigned  char num) {   //تابع برگشتی کد سون سگمنت
switch(num){                           //مقایسه مقدار متغیر با اعداد زیر
 case 0 : return 0x3f;                 //کد معادل عدد صفر برای سون سگمنت کاتد مشترک
```

```
 case 1 : return 0x06;     //کد معادل عدد یک برای سون سگمنت کاتد مشترک
 case 2 : return 0x5b;     //کد معادل عدد دو برای سون سگمنت کاتد مشترک
 case 3 : return 0x4f;     //کد معادل عدد سه برای سون سگمنت کاتد مشترک
 case 4 : return 0x66;     //کد معادل عدد چهار برای سون سگمنت کاتد مشترک
 case 5 : return 0x6d;     //کد معادل عدد پنج برای سون سگمنت کاتد مشترک
 case 6 : return 0x7d;     //کد معادل عدد شش برای سون سگمنت کاتد مشترک
 case 7 : return 0x07;     //کد معادل عدد هفت برای سون سگمنت کاتد مشترک
 case 8 : return 0x7f;     //کد معادل عدد هشت برای سون سگمنت کاتد مشترک
 case 9 : return 0x6f;     //کد معادل عدد نه برای سون سگمنت کاتد مشترک
 default: return 0x00;     //کد پیش فرض
  }
}
void main(){                          //تابع اصلی برنامه
PORTA=0x00;                           //مقدار اولیه پورت متصل به سون سگمنت را صفر قرار می‌دهیم
DDRA=0xff;                            //تعیین پورت A به عنوان خروجی
while(1){                             //ایجاد کردن حلقه بی نهایت برای تکرار شمارش
for(i=0 ; i<=9 ; i++){                //ایجاد یک حلقه که ۱۰ مرتبه تکرار می‌شود
PORTA=mask(i);   //برگشت تابع کد معادل سون سگمنت کانتر حلقه بوده و به خروجی ارسال می‌شود
delay_ms(1000);                       //تأخیر زمانی ۱ ثانیه برای شمارش
  }
 };
}
```

در مثال ۲-۲۵ در آغاز، برنامه وارد تابع اصلی main می شود و پس از تنظیم پورت متصل به سون سگمنت، وارد یک حلقه بی نهایت می شود و یک حلقه for جهت شمارش و تغییر مقدار آرگومان تابع mask که فراخوانی می شود، استفاده شده است. در واقع مقدار i در موقع فراخوانی در متغیر num کپی می شود. متغیر num به عنوان پارامتر یا آرگومان تابع معرفی شده است. مقدار num در دستور switch قرار می گیرد و مطابق با هر یک از مقادیر ثابت که برابر بود یک کد معادل سون سگمنت کاتد مشترک را بر می گرداند. این مقدار برگشتی در PORTA که به سون سگمنت کاتد مشترک وصل شده است فرستاده می شود. مقدار داده برگشتی ۸ بیتی بدون علامت می باشد.

آرایه ها

به متغیرهای به هم پیوسته که یک جا تعریف می شوند، آرایه گفته می شود و نحوه تعریف کردن آن ها مشابه متغیرها می باشد. آرایه ها می توانند یک بعدی یا چند بعدی به گونه ماتریسی باشند. فرم کلی تعریف آرایه یک بعدی به گونه زیر است:

```
نوع داده  نام آرایه[اندازه متغیرها];
```

نوع داده می تواند یکی از انواع داده باشد و نام آرایه نیز می تواند یک نام مجاز از حروف لاتین بوده و شمار عضوهای موجود در آرایه نیز می تواند تعریف شوند. ضمنا برای دسترسی به اعضای آرایه می بایست نام آرایه و همچنین شماره عضو در بین دو علامت [] قرار بگیرد.

مثال ۲-۲۶: یک آرایه تک بعدی با چهار عضو تعریف کنید و به آنها مقدار بدهید؟

```
unsigned char code[4];          //تعریف یک آرایه یک بعدی با ۴ عضو
code[0]=0x10;                   //مقداردهی به عضو اول
code[1]=0xaf;                   //مقداردهی به عضو دوم
code[2]=0x19;                   //مقداردهی به عضو سوم
code[3]=0x66;                   //مقداردهی به عضو چهارم
```

مثال ۲-۲۷: آرایه قبلی را تعریف کنید و همان لحظه اول مقداردهی را انجام دهید؟

```
unsigned char code[4]={0x10,0xaf,0x19,0x66};
```

مثال ۲-۲۸: برنامه یک شمارنده 0 تا 9 را این بار با استفاده از آرایه ذخیره شده در حافظه ثابت میکروکنترلر بنویسید؟

```
#include <mega16.h>                 //معرفی کتابخانه میکروکنترلر استفاده شده
#include <delay.h>                  //معرفی کتابخانه تأخیر زمانی
flash unsigned char display[]={     //کدهای سون سگمنت کاتد مشترک در حافظه ثابت
0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
void main(){                        //تابع اصلی برنامه
unsigned char i;                    //تعریف متغیر ۸ بیتی برای کانتر حلقه for
PORTA=0x00;                         //مقدار اولیه پورت متصل به سون سگمنت را صفر قرار می‌دهیم
DDRA=0xff;                          //تعیین پورت A به عنوان خروجی
while(1){                           //ایجاد کردن حلقه بی نهایت برای تکرار شمارش
for(i=0 ; i<=9 ; i++){              //ایجاد یک حلقه که ۱۰ مرتبه تکرار می‌شود
PORTA=display[i];                   //کانتر حلقه اندیس عضو آرایه را تعیین کرده و به خروجی ارسال می‌کنیم
delay_ms(1000);                     //تأخیر زمانی ۱ ثانیه برای شمارش
  }
 };
}
```

در مثال ۲-۲۸ نخست در تابع main متغیر محلی i تعریف می شود و پورت A به عنوان خروجی تنظیم می گردد و برنامه وارد حلقه بی نهایت while می شود و حلقه for تشکیل می گردد. در هر بار که حلقه for تکرار می شود شمارنده حلقه یعنی i افزایش می یابد و به ترتیب اعضای آرایه display را به پورت A ارسال می کند و در این جا تاخیر حلقه یک ثانیه است. هر وقت i=10 شود، شرط حلقه for برقرار نخواهد بود و وارد حلقه بی نهایت می شود و دوباره وارد حلقه for می شود.

فرم کلی تعریف ارایه دو بعدی به گونه زیر است:

```
نوع داده    نام آرایه[اندازه بُعد ۱][ اندازه بُعد ۲ ];
```

اندازه بعد نخست؛ شمار سطرها و اندازه بعد دوم؛ شمار ستون های آرایه را تعیین می کند که در واقع یک ماتریس را به وجود می آورد. ترتیب نوشتن مقدار اولیه آرایه های دو بعدی باید رعایت شود.

مثال ۲-۲۹ : یک آرایه دو بعدی تعریف کنید و به اعضای آن دسترسی پیدا کنید.

```
unsigned char x,j;
unsigned char matrix[2][3]={{'0','1','2'},
                            {'3','4','5'}};
x= matrix[1][2];
j= matrix[0][0];
```

در مثال ۲-۲۹ مقدار قرار گرفته در متغیر x='5' خواهد بود. چون سطر دوم و ستون سوم انتخاب شده است و در متغیر j نیز عدد اسکی '0' قرار می گیرد. اعداد می توانند در آرایه ها با هر مبنایی باشند و لزومی ندارد که حتما اعداد اسکی باشند و دقت کنید اندیس اعضا از صفر آغاز می شود.

رشته ها

در واقع رشته ها همان آرایه هایی هستند که از کاراکترهای به هم پیوسته تشکیل شده اند و بین دوم علامت " " تعریف می گردند. رشته ها در برنامه نویسی تنها به گونه تک بعدی تعریف می شوند.

مثال ۲-۳۰ : یک رشته بدون تعیین اعضا تعریف کنید.

```
char name [] = “alvandi” ;
```

هر رشته به یک کاراکتر تهی null (برابر 0 دسیمال یا ’\0‘ اسکی) ختم می شود.
کاراکتر تهی نشان دهنده پایان رشته می باشد بنابراین اگر آرایه هایی داشته باشیم که اندازه آن ۸ باشد، تنها از ۷ مکان آن استفاده می شود که آخری یک کاراکتر تهی می باشد.

ساختمان (ساختارها)

ساختارها مجموعه ای از اعضای متغیرها با نوع داده های گوناگون، تحت عنوان یک نام می باشند. تاکنون هر متغیری را که تعریف می کردیم می بایست نوع داده آن نیز ذکر شود و اگر متغیری با نوع داده دیگری بخواهد تعریف شود باید به گونه جداگانه تعریف شود. ولی به وسیله ساختارها ما می توانیم برای اعضای ساختمان خود یک نام کلی در نظر بگیریم. فرم کلی تعریف ساختار به گونه زیر است:

```
Struct <نام ساختمان>
{
اعضای ساختمان ;
} اسامی متغیرهای ساختمان ;
```

واژه کلیدی struct نشان دهنده تعریف ساختمان می باشد و نام ساختمان می تواند از حروف لاتین باشد و اعضای ساختمان نیز می توانند متغیرهایی با هر نوع داده ای باشند و اسامی متغیرهای ساختمان نیز می توانند با هر نامی از حروف لاتین تعریف شوند. در واقع این اسامی متغیرها هست که به اعضای داخل ساختمان دسترسی دارند.

مثال ۲-۳۱:

```
struct time {
char contrl;
unsigned int hour,minute,seconds;
}par;
...
par.contrl=0x12;
par.minute=0x60;
```

برای دسترسی به اعضای ساختار از علامت "." استفاده می کنیم و همچنین اگر اشاره گری از نوع ساختمان تعریف شده باشد با استفاده از علامت "->" قابل دسترسی است.

مثال ۲-۳۲:

```
struct time *ptr;
ptr->seconds=30;
```

یونیون ها (اتحادها)

اتحاد ها نیز مانند ساختارها یک نوع داده گروهی می باشند. تفاوت آن ها با ساختارها در این است که از مکان های حافظه اختصاص یافته به اتحادها، بین اعضای گروه به گونه مشترک استفاده می شود. فرم کلی تعریف اتحاد ها به گونه زیر است:

```
union   نام اتحاد {
اعضای اتحادها ;
اسامی متغیرهای اتحادها } ;
```

مثال ۲-۳۳:

```
union save_R{
char x,y;
int z;
}s_data;
```

نحوه دسترسی به اعضای اتحادها نیز مانند ساختارها می باشد. همان گونه که گفتیم میزان فضای اشغال شده به وسیله اتحادها کم تر از ساختارها می باشد. در مثال ۲-۳۳ دو متغیر ۸ بیتی x و y و یک متغیر ۱۶ بیتی داریم. پس اگر از نوع ساختمان باشد در مجموع ۳۲ بیت حافظه مورد نیاز است. ولی چون از اتحادها استفاده کردیم، تنها به ۱۶ بیت نیاز داریم و از سویی چون برنامه یک مکان ۱۶ بیتی را برای اعضای گروه به اشتراک می گیرد، در هر لحظه می توان تنها از یکی از اعضای گروه استفاده کرد. در اتحاد ها بزرگ ترین متغیر، تعیین کننده میزان فضای حافظه برای یک اتحاد می باشد.

شمارش ها

نوع داده شمارشی، برای تعریف یک مجموعه متناهی جهت نام گذاری یا شماره گذاری است. فرم کلی تعریف نوع داده شمارشی به گونه زیر است:

فرم کلی تعریف نوع داده شمارشی بصورت زیر است:

```
enum نام شمارش{
اعضای شمارش ;
اسامی متغیرهای شمارش ;
```

مثال ۲-۳۴:

```
enum color{
red;
bulo;
green;
}color1,color2;
```

در این مثال به عنصر اولی یعنی red عدد ۱ و به دومی یعنی blue عدد ۲ و به سومی یعنی green عدد ۳ نسبت داده می شود و همین گونه اگر ما شماری عنصر در یک شمارشگر قرار دهیم، اعداد صحیح به آن ها نسبت داده می شود، مگر این که کاربر شماره عنصر را تعیین کند. مثلا green=6 ، آن گاه به عنصر سوم عدد ۶ نسبت داده می شود.

کتابخانه های Ctype ،stdlib،string،math جزء کتابخانه های استاندارد در زبان C می باشند و کامپایلر CodeVisionAvr نیز دارای این کتابخانه ها می باشد. توابع کتابخانه ای در مسیر نصب نرم افزار در پوشه INC,LIB قرار دارند دارند که ما در صورت نیاز، می بایست آن ها را در ابتدای برنامه معرفی کنیم و از توابع آماده آن ها استفاده کنیم. برخی از این توابع عبارتند از:

•STDIO , STDARG,SPI, SLEEP, SETJMP, PCF8583, MEM, LM75, LCD4C40, LCD, io, GRAY, BCD, 43USB355, 86RF401, DS1307, MEGA16, DELAY, 90S8535,TINY13

تذکرات مهم:

۱- قبل از استفاده از هر تابع، باید با دستور #include<name.h> آن کتابخانه را در ابتدای برنامه معرفی کنیم.

۲- توابعی که قبل از آن ها کلمه کلیدی viod وجود دارد، موقع فراخوانی نباید کلمه void را تایپ کنیم و همچنین برای توابع برگشتی نیز، نباید نوع داده برگشتی موقع فراخوانی تایپ گردد.

۳- توابعی که برگشت پذیر هستند و مقداری که بر می گردانند ممکن است علامت دار یا بدون علامت باشد که باید در نظر گرفته شود.

۴- توابعی که پارامتر ورودی دریافت می کنند خواه برگشت پذیر یا برگشت ناپذیر باشند باید به آن ها متناسب با پذیرش مقدار پارامتر ورودی، مقداری را ارسال کنیم.

۵- توابعی که پارامتر ورودی آن ها رشته یا آرایه می باشد، موقع ارسال رشته یا آرایه نباید علامت کروشه [ ] تایپ گردد و فقط نام رشته یا آرایه را می نویسیم.

۶- توجه کنید موقع فراخوانی باید در انتهای هر تابع ; را قرار دهید.

۷- تمامی توابع باید با حروف کوچک فراخوانی شوند.

۸- منظور از علامت * در برخی از توابع، اشارگر رشته ای می باشد.

کتابخانه ctype.h

unsigned char isalnum (char c)

اگر c یک کاراکتر پارمتر ورودی، یکی از حروف الفبای – رقمی لاتین (شامل حروف و ارقام) باشد، مقدار ۱ را بر می گرداند و در غیر این صورت مقدار 0 را بر می گرداند.

Unsigned char isalpha (char c)

اگر c یک کاراکتر پارمتر ورودی، یکی از حروف الفبای لاتین (شامل حروف) باشد، مقدار ۱ را بر می گرداند و در غیر این صورت مقدار 0 را بر می گرداند.

Unsigned char isascii (char c)

اگر c یک کاراکتر پارمتر ورودی باشد. اگر این کاراکتر یکی کارکترهای اسکی در محدوده اعداد (۰تا ۱۲۷) باشد، مقدار ۱ و در غیر این صورت مقدار ۰ را بر میگرداند.

Unsigned char iscntr1(char c)

اگر c یک کاراکتر کنترلی به عنوان پارامتر ورودی تابع باشد. اگر این کاراکتر ورودی در محدوده اعداد (۰ تا ۳۱ یا ۱۲۷) باشد، مقدار ۱ و در غیر این صورت مقدار 0 را بر می گرداند.

Unsigned char isdigit (char c)

اگر c یک کاراکتر پارامتر ورودی یکی از اعداد دسیمال ۰ تا ۹ باشد این تابع مقدار ۱ و در غیر این صورت مقدار ۰ را بر می گرداند.

Unsigned char islower (char c)

اگر c یک کاراکتر پارامتر ورودی یکی از حروف کوچک لاتین a تا z باشد این تابع مقدار ۱ و در غیر این صورت مقدار 0 را بر می گرداند.

Unsigned char isprint (char c)

اگر c یک کاراکتر پرامتر ورودی یکی از کاراکترهای قابل چاپ در محدوده (۳۲ تا ۱۲۷) باشد این تابع مقدار ۱و در غیر این صورت مقدار ۰ را بر می گرداند.

Unsigned char ispunct (char c)

اگر c یک کاراکتر پارامتر ورودی یکی از کاراکترهای علائمی مانند (؟ ، ! و ...) باشد این تابع مقدار ۱و در غیر این صورت مقدار۰ رابر می گرداند.

Unsigned char isspace (char c)

اگر C یک کاراکتر پارامتر ورودی، یکی از کاراکترهای فضای خالی " 0/" و یا carriage return(" \r\n") باشد این تابع مقدار ۱ و در غیر این صورت مقدار0 را بر می گرداند.

Unsigned char isupper (char c)

اگر c یک کاراکتر پارامتر ورودی، یکی از حروف بزرگ لاتین A تا Z باشد این تابع مقدار ۱و در غیر این صورت مقدار ۰ را بر می گرداند.

Unsigned char isxdigit (char c)

اگر c یک کاراکتر پارامتر ورودی، یکی از اعداد هگزاد دسیمال 0 تا F باشد این تابع مقدار ۱و در غیر این صورت مقدار ۰را بر می گرداند.

Char toascii (char c)

این تابع کاراکتر c را به عنوان پارامتر ورودی میگیرد و معادل اسکی آن را برمی گرداند.

unsigned char toint (char c)

این تابع کاراکتر c را به عنوان پارامتر ورودی بر حسب هگزاد دسیمال می گیرد و معادل رقمی بر حسب دسیمال اعداد ۰تا ۵ را بر می گرداند.

char tolower (char c)

این تابع کاراکتر c را به عنوان پارامتر ورودی می گیرد و اگر این کاراکتر حرف بزرگ باشد آن را به حرف کوچک تبدیل می کند و آن را بر می گرداند، در غیر این صورت خود کاراکتر c را بر می گرداند.

char toupper (char c)

این تابع کاراکتر c را به عنوان پارامتر ورودی می گیرد و اگر این کاراکتر حرف کوچک باشد آن را به حرف بزرگ تبدیل می کند و آن را بر می گرداند، در غیر این صورت خود کاراکتر c را بر می گرداند.

کتابخانه stdlib.h

Int atoi (char *str)

یک متغیر رشته ای به عنوان پارامتر ورودی می پذیرد و آن را به عدد صحیح تبدیل می کند.

Long int atol (char *str)

این تابع یک متغیر رشته ای به عنوان پارامتر ورودی می پذیرد و آن را به عدد صحیح بلند تبدیل می کند.

Void itoa (int n , char *str)

این تابع یک متغیرعدد صحیح n و یک متغییر رشته ای str را به عنوان پارامتر ورودی می گیرد و عدد صحیح را تبدیل به کاراکترهای اسکی کرده و در متغیر رشته ای str ذخیره می کند.

Void ltoa (long int n, char *str)

این تابع یک متغیرعدد صحیح بلند n و یک متغییر رشته ای str را به عنوان پارامتر ورودی می گیرد و عدد صحیح بلند را تبدیل به کاراکترهای اسکی کرده و در متغیر رشته ای str ذخیره می کند.

Void ftoa (float n, unsigned char decimals, char *str)

این تابع یک متغیر اعشاری n و یک متغیر عدد صحیح دسیمال و یک متغیر رشته ای str را به عنوان پارامتر ورودی می گیرد و متغیر اعشاری را تبدیل به کاراکترهای اسکی کرده و در متغیر رشته ایstr ذخیره می کند. مقدار متغیر decimals دقت ارقام اعشار برای تبدیل به کاراکتر اسکی را تعیین می کند.

Void ftoe (float n, unsigned char decimals, char *str)

این تابع یک متغیر اعشاری n و یک متغیر عدد صحیح دسیمال و یک متغیر رشته ای str را به عنوان پارامتر ورودی می گیرد و متغیر اعشاری را تبدیل به نماد علمی معادل کاراکتر های اسکی کرده و در متغیر رشته ای str ذخیره می کند. مقدار متغیر decimals دقت ارقام اعشار را تعیین می کند. برای نمونه با ۲رقم دقت اعشار، عدد $۱۲,۳۵۹۸ * ۱۰^{-۵}$ را تبدیل به رشته "۱۲,۳۵e -۵"می نماید.

Float atof (char *str)

یک متغیر رشته عددی را به عنوان پارامتر ورودی می پذیرد و آن را به عدد اعشاری تبدیل می کند.

Int rand (void)

این تابع در لحظه فراخوانی یک عدد صحیح از بین ارقام ۰ تا ۳۲۷۶۷ را به طور کلی تصادفی بر می گرداند.

Void srand (int seed)

این تابع مانند تابع قبلی عمل می کند با این تفاوت که رقم شروع برای انتخاب رقم تصادفی را به عنوان پارامتر ورودی می گیرد و سرانجام یک رقم تصادفی بر می گرداند.

کتابخانه math.h

Unsined int abs (int x)

این تابع یک متغیر عدد صحیح به عنوان پارامتر ورودی می گیرد و قدر مطلق آن را بر می گرداند.

Unsigned long labs (long int x)

یک متغیر عدد صحیح بلند به عنوان پارامتر ورودی می گیرد و قدر مطلق آن را بر می گرداند.

Float fabs (float x)

این تابع یک متغیر عدد اعشاری به عنوان پارامتر وردی می گیرد و قدر مطلق آن را بر می گرداند.

Int max (int a, int b)

تابع دو متغیر ۱۶ بیتی a و b را به عنوان پارامتر ورودی می گیرد و مقدار بزرگ تر را بر می گرداند.

Int min (int a, int b)

تابع دو متغیر ۱۶ بیتی a و b را به عنوان پارامتر ورودی می گیرد و مقدار کوچک تر را بر می گرداند.

Signed char sign (int x)

این تابع یک متغیر عدد صحیح را به عنوان پارامتر ورودی می گیرد و اگر متغیر منفی باشد ۱- ، صفر باشد 0 و مثبت باشد ۱+ را برمی گرداند.

Signed char fsign (float x)

این تابع یک متغیر عدد اعشاری را به عنوان پارامتر ورودی می گیرد و اگر متغیر منفی باشد ۱- ، صفر باشد 0 و مثبت باشد ۱+ را برمی گرداند.

Float sqrt (float x)

یک متغیر عدد اعشاری را به عنوان پارامتر ورودی می گیرد و جذر آن را بر می گرداند.

Float exp (float x)

یک متغیر عدد اعشاری x را به عنوان پارامتر ورودی میگیرد و حاصل e^x را بر میگرداند.

Float log (float x)

یک متغیر عدد اعشاری را به عنوان پارامتر وردی می گیرد و حاصل$\ln(x)$را بر میگرداند.

Float log10 (float x)

یک متغیر عدد اعشاری را به عنوان پارامتر ورودی می گیرد و لگاریتم در مبنای ۱۰ را بر می گرداند.

Float pow (float x, float y)

این تابع دو متغیر اعشاری x و y را به عنوان پارامتر ورودی می گیرد و مقدار x^y را بر می گرداند.

Float sin (float x)

متغیر اعشاری را به عنوان پارامتر ورودی میگیرد و سینوس آن را بر حسب رادیان بر می گرداند.

Float cos (float x)

متغیر اعشاری را به عنوان پارامتر ورودی میگیرد و کسینوس آن را بر حسب رادیان بر میگرداند.

Float tan(float x)

متغیر اعشاری را به عنوان پارامتر ورودی میگیرد وتانژانت آن را بر حسب رادیان بر میگرداند.

Float asin (float x)

یک متغیر اعشاری را به عنوان پارامتر ورودی میگیرد و معکوس سینوس آن را بر حسب رادیان بر میگرداند.

Float acos (float x)

یک متغیر اعشاری را به عنوان پارامتر ورودی میگیرد و معکوس کسینوس آن عدد را بر حسب رادیان بر میگرداند.

Float atan (float x)

یک متغیر اعشاری را به عنوان پارامتر ورودی میگیرد و معکوس تانژانت آن عدد را بر حسب رادیان بر میگرداند.

کتابخانه string.h

Char * strcat (char * str1, char *str2)

این تابع رشته str2 را به انتهای رشته str1 متصل می کند.

Char *strcatf (char *str1, char flash *str2)

این تابع رشته str2 که در حافظه ثابت قرار دارد را به انتهای رشته str1 متصل میکند.

Char *strncat (char *str1, char * str2, unsigned char n)

این تابع حداکثر n کاراکتر از رشته str2 را به انتهای رشته str1 متصل می کند.

Char *strncatf (char *str1, char flash *str2, unsigned char n)

حداکثر n کاراکتر از رشته sr2 که در حافظه ثابت قرار دارد را به انتهای رشته str1 متصل می کند.

Char *strchr (char *str, char c)

این تابع رشته str را برای محل اولین وقوع کاراکتر ورودی c جستجو می کند و آن را به یک اشاره گر نسبت می دهد و در غیر این صورت اشاره گر برابر کاراکتر null (تهی) می شود.

Char *strrchr (char *str, char c)

این تابع رشته str را برای محل آخرین وقوع کاراکتر ورودی c جستجو می کند و آن را به یک اشاره گر نسبت می دهد و در غیر این صورت اشاره گر برابر کاراکتر null (تهی) می شود.

Signed char strcmp (char *str1, char *str2)

این تابع رشته str1 را با رشته str2 مقایسه می کند. اگر str1 > str2 مقدار بزرگ تر از صفر و اگر str=str2 مقدار صفر و اگر str1<str2 مقدار کوچک تر از صفر (عدد منفی) را بر می گرداند.

Signed char strcmpf (char *str1, char flash *str2)

این تابع رشته str1 را واقع در حافظه SRAM با رشته str2 واقع در flash مقایسه می کند. اگر str1>str2 ، مقدار بزرگ تر از صفر و اگر str=str2 مقدار صفر و اگر str1<str2 مقدار کوچک تر از صفر را بر می گرداند.

Char * strcpy (char * dest, char * src)

این تابع رشته src را به رشته dest کپی می کند.

Char *strcpyf (char *dest, char flash *src)

این تابع رشته src واقع در حافظه flash را به رشته dest واقع در حافظه SRAM کپی می کند.

کتابخانه bcd.h

Unsigned char bcd2bin (unsigned char n)

این تابع یک پارامتر ورودی n بر حسب کد BCD می گیرد و کد باینری معادل آن را بر می گرداند.

Unsigned char bin2bcd (unsigned char n)

این تابع یک پارامتر ورودی n که می تواند در محدوده اعداد ۰ تا۹۹ باشد را می گیرد و کد BCD معادل آن را بر می گرداند.

کتابخانه delay.h

توجه داشته باشید مقدار تاخیر توابع این کتابخانه به مقدار فرکانس کاری برنامه در نرم افزار کامپایلر که از مسیر project/configure/c compiler تنظیم می شود وابسته است.

Void delay_us (unsigned int n)

این تابع یک عدد ثابت به عنوان پارامتر ورودی در محدوده اعداد ۰ تا ۶۵۵۳۵ میگیرد و بر حسب میکروثانیه تاخیر زمانی ایجاد میکند. برای نمونه برای تاخیر 100µs داریم:

Delay_us (100);

Void delay_ms (unsigned int n)

این تابع یک متغیر متغیر عدد صحیح به عنوان پارامتر ورودی در محدوده اعداد ۰ تا ۶۵۵۳۵ می گیرد و بر حسب میلی ثانیه تاخیر زمانی ایجاد می کند.

به طور مثال برای تاخیر یک ثانیه 1000ms داریم:

```
Delay_ms(1000);
```

همچنین فرض کنید یک متغیر مانند TD=250 مفروض باشد:

```
Delay_ms(TD);
```

در تابع فوق به مدت ۲۵۰ میلی ثانیه تاخیر ایجاد می شود.

نکته دیگری که در مورد تابع delay_ms() باید به آن اشاره کنیم این است که این تابع به طور اتوماتیک تایمر watchdog را هر 1ms توسط دستور اسمبلی wdr، reset می کند. این ویژگی زمانی که در برنامه، تایمر watchdog را فعال کرده باشیم یک مزیت محسوب می شود.

فصل سوم ۳

آموزش نرم افزارCodevisionAVR

اهداف

۱-آشنایی با محیط برنامه نویسی به زبان c نرم افزار codevisionAVR
۲-آشنایی با ابزار codewizardAVR نرم افزار کامپایلر C
۳-معرفی پروگرامرISP و نحوه برنامه ریزی میکروکنترلر
۴-آشنایی با محیط اشکال زدایی نرم افزار AVR Studio

۳-۱ نرم افزار CodeVisionAVR

برنامه ای که ما با دستورات می نویسیم صرفا ارزش یک متن معمولی را دارد و ارزشی برای میکروکنترلر ندارد. برای این منظور هر شرکت سازنده می بایست کامپایلر مخصوص تراشه خود را ارائه دهد و یا این که شرکت های طراح کامپایلر باید از قطعه شرکت سازنده حمایت کنند. در واقع کامپایلر به معنی تفسیر کننده می باشد و فایل نوشته شده ما را به کدهای قابل فهم برای میکروکنترلر تبدیل می کند و توسط پروگرامر فایل ایجاد شده بر روی تراشه load می شود.

در بین کامپایلر های زبان c که برای میکروکنترلرهایAVR از شرکت ATMEL توسط شرکت های سازنده نرم افزار ارائه شده است می توان CodeVisionAVR را یکی از بهترین و قوی ترین کامپایلرهای زبان C برای میکروکنترلرهای AVR دانست. این نرم افزار طوری طراحی شده است که سرعت اجرای دستورات را بهینه و خیلی خوب از فضای حافظه استفاده می کند. ما در این کتاب به معرفی و کار با این نرم افزار خواهیم پرداخت اما همه ما ملتفت هستیم که آن طور که باید به خوبی یک نرم افزار را در کتاب به طور کامل نمی توان آموزش داد. اما در این کتاب روش مدرن و بهتری برای آموزش انتخاب گردیده است. در این فصل نرم افزار کدویژن را به طور کلی قسمت های مهم آن را آموزش داده و نکات تکمیلی و جزئی را به صورت صوتی و تصویری در CD همراه کتاب آموزش می دهیم.

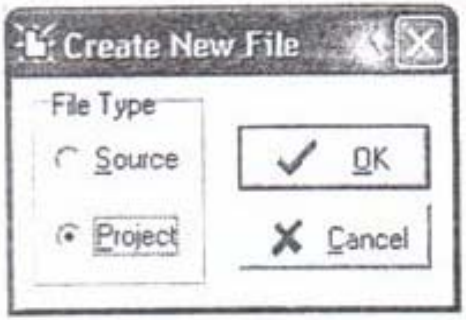
نرم افزار code vision AVR نسخه۱,۲۵,۸ سال ۲۰۰۷ میلا دی برای این کتاب انتخاب شده است این نسخه کامل و بدون محدودیت است البته نسخه سال ۲۰۰۸ میلادی را نیز در CD همراه کتاب قرار داده ایم. شما می توانید آخرین نسخه نرم افزار code vision AVR را از سایت سازنده آن به نشانی www.hpinfotech.com دانلود کنید اما توجه کنید که به صورت رایگان نسخه آزمایشی این نرم افزار را در اختیار شما قرار می دهد که دارای محدودیت در حجم فایل ایجاد شده است.

نرم افزار code vision AVR واقع در CD همراه کتاب را نصب کنید پس از اتمام مراحل نصب محیط نرم افزار را باز کنید و از پیام داده شده گزینه cancel را انتخاب کنید و فایل keygen را دابل کلیک نمائید تا پنجره شکل ۳-۱ باز در قسمت serial number کلیک راست کرده و گزینه paste را انتخاب کنید سپس گزینه make را کلیک کرده و مسیر خواسته شده برای ذخیره فایل license را پوشه BIN واقع در مسیر نصب نرم افزار تعیین کنید. سپس با اجرای نرم افزار وارد محیط برنامه نویسی شوید. اگر پروژه ای از قبل باز شده بود از منوی file گزینه close project را انتخاب کنید تا آن پروژه بسته شود.

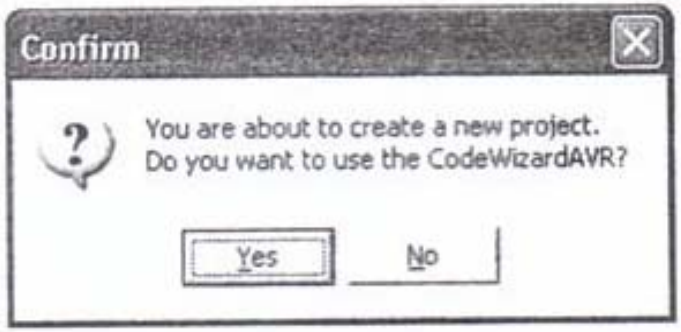
ایجاد پروژه جدید

ابتدا از منوی File گزینه New را کلیک کنید تا پنجره شکل ۳-۲ نمایان شود.

شکل ۳-۱ پنجره نرم‌افزار KeyGen



شکل ۳-۲ ایجاد فایل جدید



شکل ۳-۳ پنجره Confirm

از پنجره شکل ۳-۲ گزینه Project را انتخاب کرده و ok را تایید کنید تا پنجره شکل ۳-۳ نمایان شود. در این کادر باز شده سوال می شود که شما برای ایجاد پروژه می خواهید از کد ویزارد استفاده کنید؟ شما گزین Yes را برگزینید.

استفاده از کد ویزارد

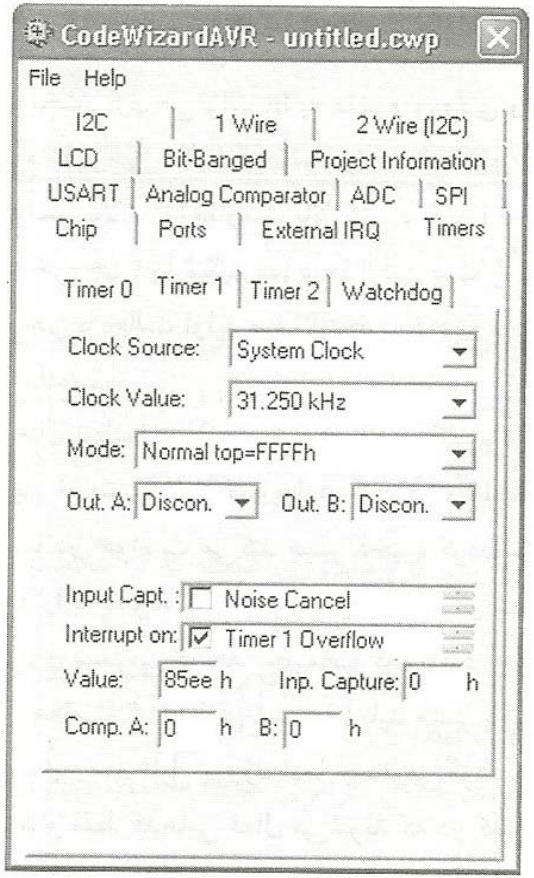
پس از انتخاب گزینه yes پنجره ای مطابق شکل ۳-۴ با زبانه های متعددی باز میشود.

از ویژگی های مهم کد ویزارد این است که ریجستر ها را برای کاربر با سرعت تنظیم میکند و کدهایی را در اختیار کاربر قرار میدهد و کاربر با کپی کردن آنها به برنامه اصلی به راحتی میتواند از آنها بهره بگیرد.

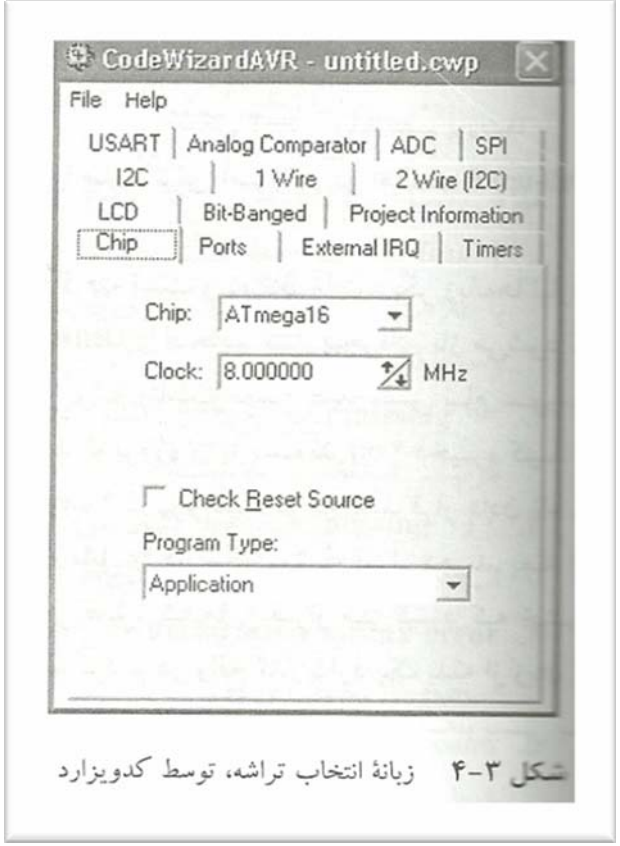
در زبانه chip شما باید نوع میکروکنترلر خانواده AVR و کلاک فرکانسی که میخواهد با آن کار کند را تعیین کنید زیرا خیلی از توابع وابسته به زمان هستند مانند توابع تاخیر زمانی،توابع تبادل سریال و ...

در قسمت Program type شما باید نوع حافظه را تعیین کنید اگر قصد استفاده از Boot loader حافظه را نداشته باشید آن را بر روی Application قرار دهید.

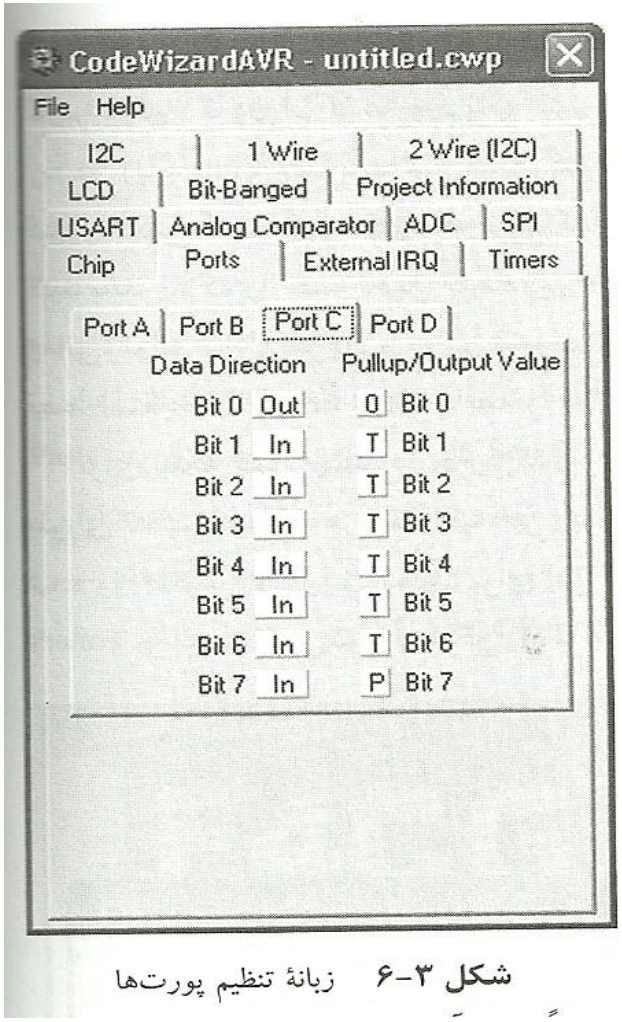
برای اینکه کمی بیشتر در مورد کد ویزارد مطلع شوید فرض کنیم که می خواهیم تایمر یک را طوری تنظیم کنیم که هر ۱ ثانیه سر ریز کند و یک LED که با یک مقاومت ۲۲۰ اهم به PC.0 متصل شده را خاموش و روشن نمایید. برای این منظور بعد از تنظیم تراشه و کلاک در مرحله قبل حال زبانه timers را انتخاب میکنیم تا شکل ۳-۵ نمایان شود.



شکل ۳-۵ زبانهٔ تنظیمات تایمرها



شکل ۳-۴ زبانهٔ انتخاب تراشه، توسط کدویزارد



شکل ۳-۶ زبانهٔ تنظیم پورت‌ها

در زبانه Timers تایمر ۱ را انتخاب میکنیم تا گزینه های تنظیم آن نماین شود. حال چون میخواهیم تایمر یک عمل تایمر را انجام دهد در قسمت Clock source آن را بر روی System clock قرار می دهیم تا از کلاک سیستم پالس دریافت کند. در قسمت Clock value تقسیم فرکانسی تایمر را طوری قرار میدهیم که تایمر با سرعت ۳۲ میکرو ثانیه شمارش کند. در قسمت Mode تایمر را در حالت نرمال قرار میدهیم. قسمت های Out.B و Out.A برای تنظیم خروجی حالت مقایسه ای تایمر است که درحالت قطع قرار میدهیم و چون از capture تایمر نمی خواهیم استفاده کنیم قسمت Input capt را رها میکنیم و در قسمت Interrupt on گزینه Timer 1 overflowرا تیک میزنیم و در قسمت value مقدار اولیه تایمر را تعیین میکنیم. حال تنظیمات پایان یافته است و زبانه Ports را انتخاب میکنیم تا شکل ۳-۶ نمایان شود.

چون میخواهیم LED را به Pc.0 وصل کنیم پس Port c را انتخاب می کنیم. هر بیت دو بخش تنظیم دارد بخش اول می تواند In به مفهوم ورودی و out به مفهوم خروجی تنظیم گردد و در بخش دوم می توان (امپدانس بالا)، p (مقاومت بالاکش)، ۰ یا ۱ (حالت اولیه خروجی) را تنظیم نمود. ما اولین بیت را در حالت خروجی با حالت اولیه صفر انتخاب کردیم و بیت هفتم را تنها برای آموزش در حالت p قرار داده ایم.

حال تنظیمات لازم و مورد نیاز در کد ویزارد صورت گرفته است و به تنظیمات دیگر زبانه ها نیازی نداریم. از منوی فایل کد ویزارد گزینه Generate ,save and exit را انتخاب کنید پنجره ای باز می شود که از شما درخواست میکند مسیر ذخیره کردن پروژه در پوشه مناسب را تعیین کنید سپس نام سورس برنامه را با پسوند *.c ذخیره کنید، مجددا از شما میخواهد تا پروژه را با پسوند *.prj خیره کنید و سپس از شما میخواهد تنظیمات کد ویزارد را با پسوند *.cwp ذخیره کنید. توجه کنید قرار دادن نام به جای * اختیاری است ولی باید مسیر ذخیره این سه فایل را در یک پوشه قرار دهید. بعد از ذخیره سه فایل پروژه شما ایجاد میشود و در سورس فایل، کدهایی قرار می گیرد که خنثی هستند و فقط کدهایی فعال می شوند که در کد ویزارد تنظیم کردیم.