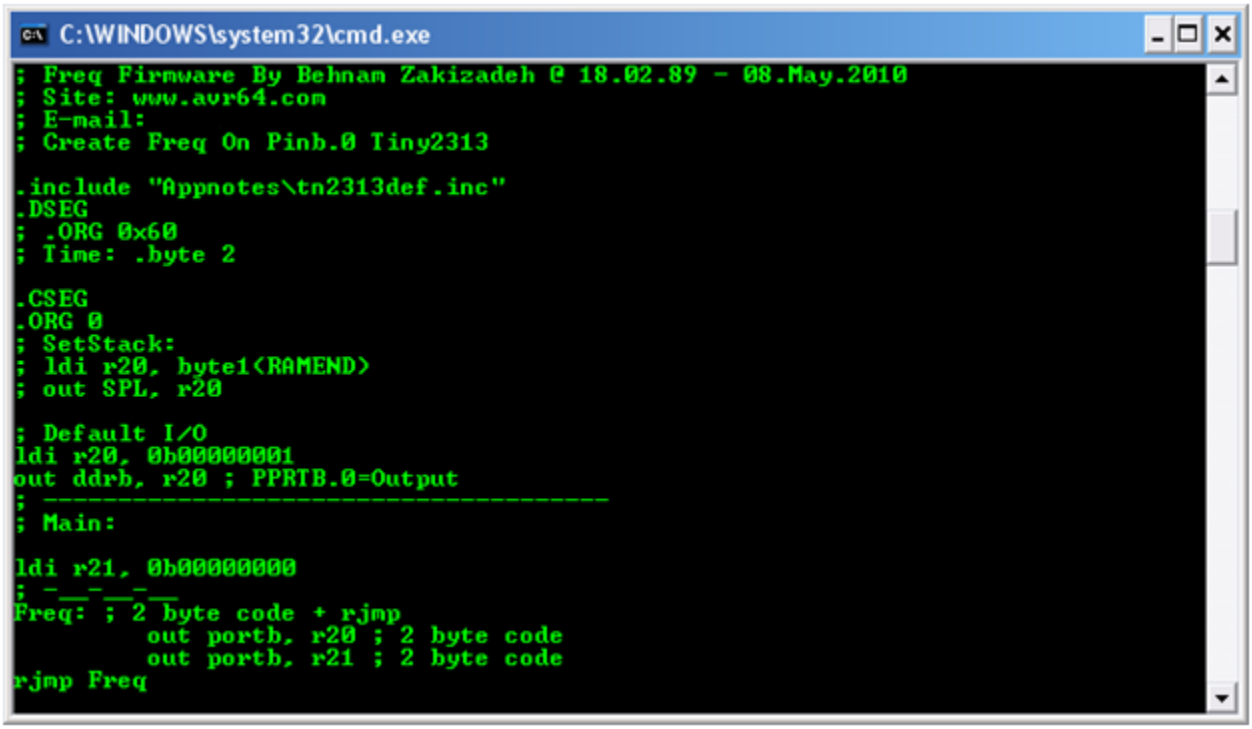
به نام خدا

(آموزش AVR جلسه ششم)

(Assembly 1)



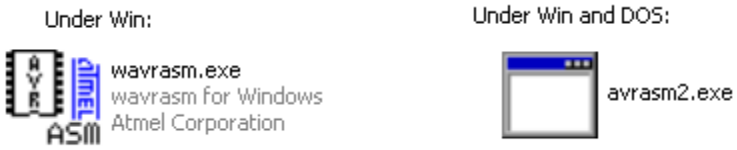
مقدمه:

پس از ۵ جلسه آشنایی با میکروها و راه اندازی بخش های مختلف آن ها توسط توابع پیش ساخته بسکام نوبت به آموزش زبان مادری میکرو یعنی اسمبلی می رسد. زبان اصلی میکرو همان زبان ماشین است که کار با آن برای افراد فوق حرفه ای که قصد نوشتن کامپایلرهای مختلفی مانند Bascom و یا کامپایلرهایی با زبان انحصاری خود (حتی فارسی) را دارند پیشنهاد می شود. زبان ماشین از کدهای ۱۶ بیتی شبیه به کد زیر تشکیل شده است که معادل اسمبلی آن نیز در بالای آن آمده است:

```
asm:
MOV Rd,Rr

16-bit Opcode:
0010 11rd dddd rrrr
```

ما از این جلسه کار با اسمبلی را شروع می کنیم. زبان اسمبلی دقیقاً همان کد زبان ماشین است با این تفاوت که به جای ۰ و ۱ ها از نام های بامعنی مثل MOV استفاده می شود. وقتی که برنامه ای به زبان اسمبلی نوشته می شود برای تبدیل دوباره اسمهای رمزی به کد زبان ماشین از برنامه کوچکی به نام اسمبلر استفاده می شود. با توجه به اینکه میکروکنترلرهای AVR محصول کمپانی ATMEL می باشند اسمبلر رایگان avrasm نیز برای همین منظور توسط اتمل طراحی شده است. این اسمبلر در دو مدل تحت DOS و تحت ویندوز موجود است و احتمالاً تحت Linux و MAC نیز طراحی شده است.



هر دو اسمبلر بالا حجمی در حد چند صد کیلوبایت دارند و در کنار هر یک پوشه ای شامل فایل های Include تمام میکروهای AVR قرار دارد. فایل های INC فایل ای متنی هستند که برای بخش های مختلف میکرو مثل تایمر، پورتها، مبدل آنالوگ به دیجیتال و... نامهای رمزی در نظر می گیرند. البته نیازی به استفاده از این فایلهای INC نمی باشد ولی در غیر این صورت مجبورید مثلاً در میکروی ATmega32 به جای استفاده از نام PORTB از آدرس سخت افزاری آن که شماره 0x18 استفاده کنید و یا خط زیر را از فایل Inc کپی گرفته و در ابتدای فایل Asm خود Paste نمایید:

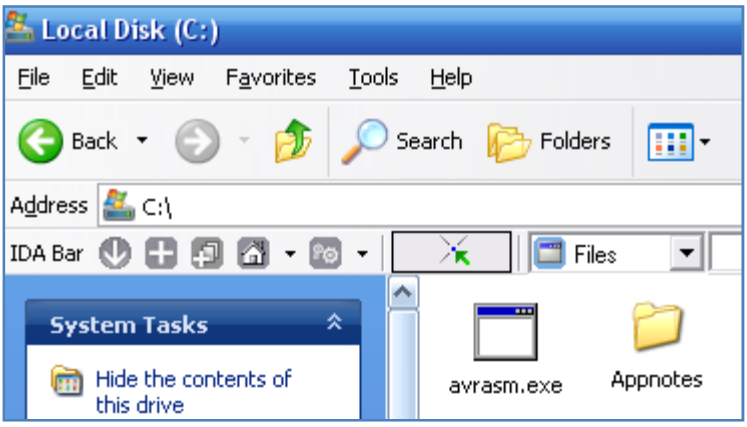
```
.equ PORTB = 0x18
```

ما در تمام برنامه های این سری از آموزش ها از فایلهای Inc استفاده می کنیم و فایلهای لازم به همراه اسمبلر تحت DOS را پوشه مقاله جاری قرار داده ایم.

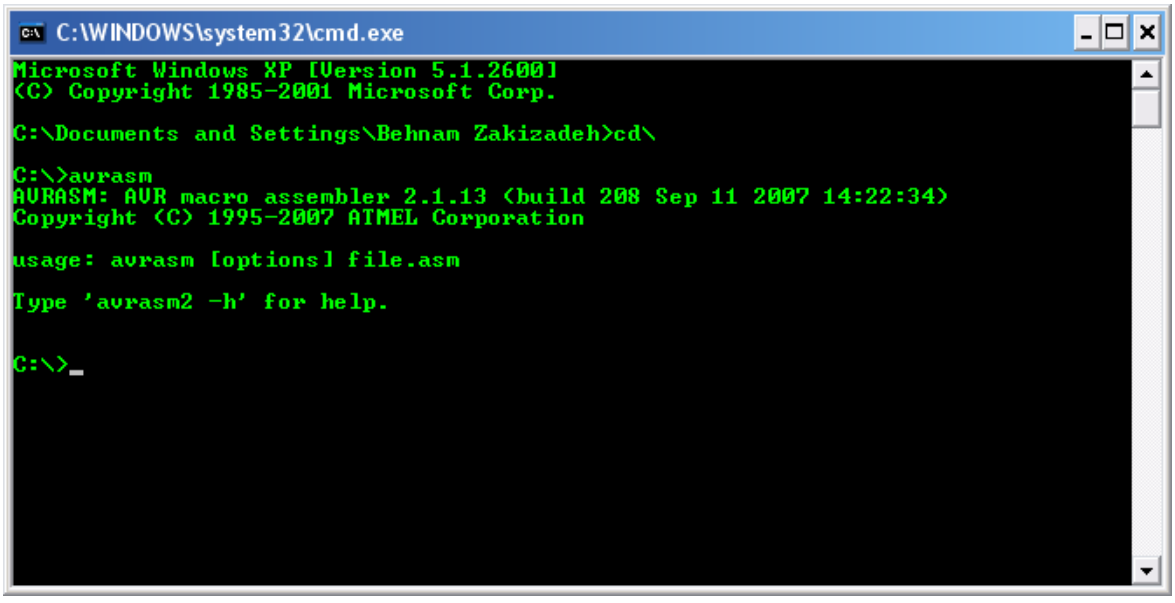
یکی دیگر از نرم افزار های رایگان که توسط ATMEL پشتیبانی می شود AVR Studio نام دارد. این برنامه به صورت پیشفرض به عنوان یک اسمبلر همراه با شبیه ساز تمام رجیسترها و اجرای برنامه به کار میرود و در صوتی که کامپایلر رایگان WinAVR نیز روی سیستم شما نصب شده باشد از هسته GCC این کامپایلر استفاده می کند و می توانید از آن به عنوان یک کامپایلر رایگان C نیز استفاده نمایید. (البته دوستانی که WinAVR را نصب می کنند معمولا با Editor خودِ WinAVR کار می کنند. ضمناً در خصوص WinAVR و GCC پس از اتمام جلسات اسمبلی بحث خواهیم کرد). ما در اینجا از اسمبلر تحت DOS استفاده می کنیم.

راه اندازی اسمبلر و نوشتن اولین برنامه اسمبلی:

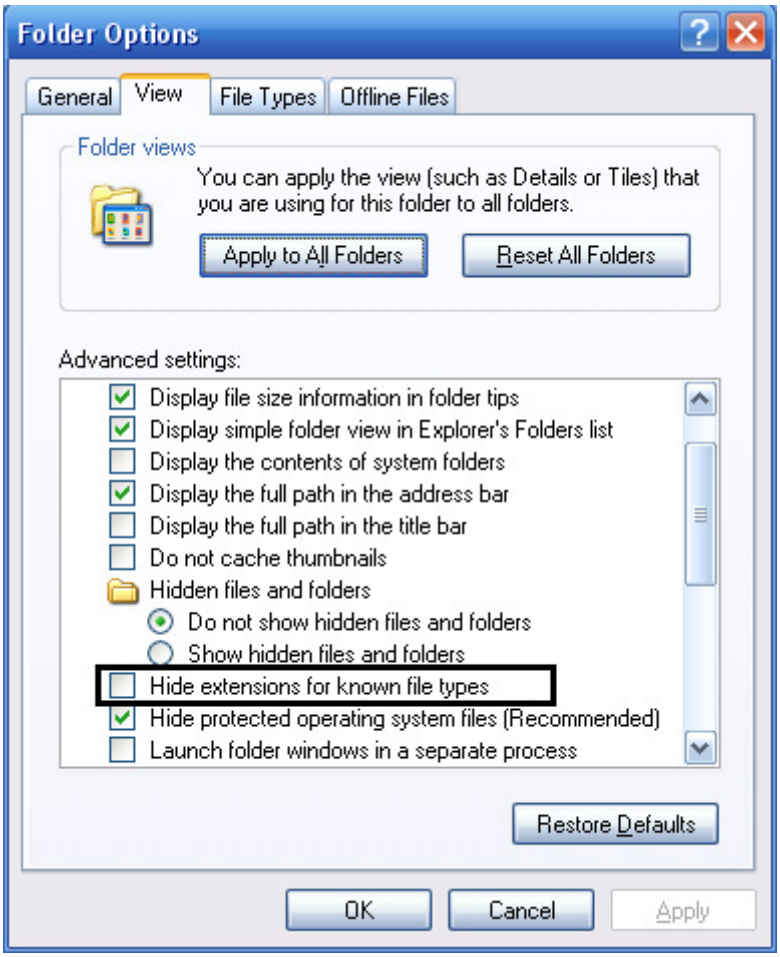
پوشه AVR64.com-avr-assembler.zip را که در کنار این مقاله وجود دارد Unzip کرده و برنامه avrasm و پوشه Appnotes را به طور مستقیم در درایو C کامپیوتر کپی نمایید.



سپس از منوی Start روی Run کلیک کرده و عبارت CMD را تایپ نمایید و OK کنید تا پنجره خط فرمان در داخل ویندوز XP باز شود. در داخل این پنجره برای رفتن به Root عبارت cd\ را تایپ و Enter کنید و در صورتی که در درایو دیگری غیر از C بودید عبارت C: را تایپ و Enter نمایید. (توجه: منظور ما از درایو C همان درایوی است که فایل اسمبلر و پوشه اپلیکیشن نوتز ها را در داخل آن کپی کرده اید و این درایو می تواند هر درایوی باشد). در نهایت برای تست اسمبلر و نمایش ورژن آن عبارت avrasm را در داخل خط فرمان تایپ و Enter نمایید:



پنجره مذکور را به صورت Open نگه داشته و آن را Minimize کنید. سپس به درایو C باز گشته و یک فایل خالی Notepad ایجاد نمایید. (راست کلیک در فضای خالی درایو، انتخاب گزینه New و در نهایت انتخاب گزینه Text file) نام فایل جدید را به prog1.asm تغییر دهید. توجه داشته باشد که اگر قبل از تغییر نام پسوند فایل که به صورت txt است نشان داده نمی شود و نام فایل فقط به صورت New text file نماش داده می شود بایستی سیستم خود را طوری تنظیم کنید که پسوند ها را نمایش دهد. برای این منظور در ویندوز XP از داخل همان درایو C وارد منوی Tools و سپس Folder Options... شوید، از سربرگ View تیک گزینه Hide extensions for known file types را بردارید و OK کنید. در این صورت نام فایل ایجاد شده توسط Notepad به صورت New Text file.txt درمی آید. البته اگر با تغییر این گزینه و OK کردن باز هم پسوند فایل ظهر نشد و با مراجعه مجدد به این پنجره مشاهده کردید که تنظیمات دوباره به حالت اولیه بازگشته اند نشان از ویروسی بودن سیستم دارد. همچنین اگر Folder Options در منوی Tools وجود نداشت باز هم مشکل از ویروس می باشد. البته جای هیچ نگرانی نیست و باز هم قادر به دیدن پسوند فایلها خواهید بود. برای این منظور هر دوفایل



showhiddenexts.vbs

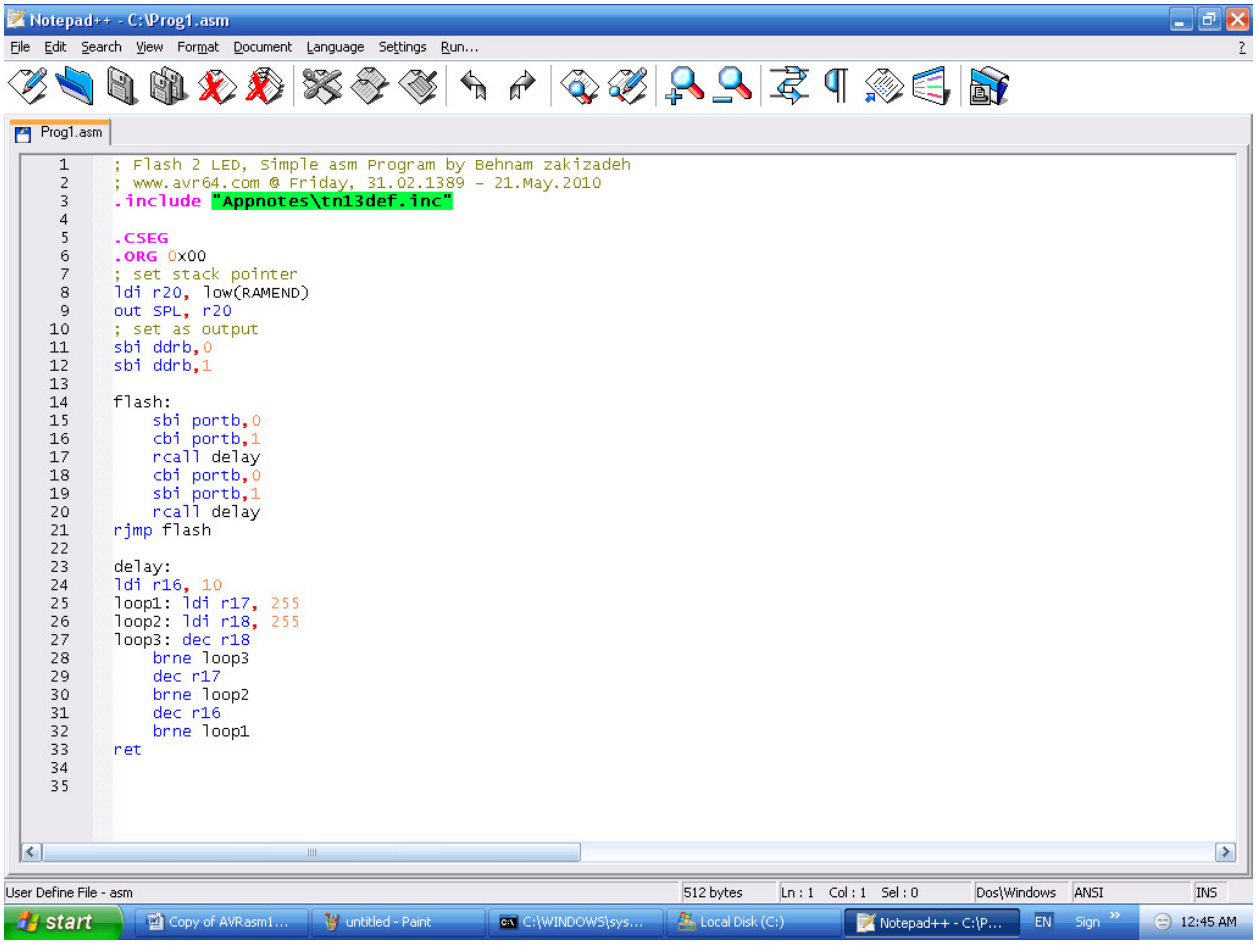
نمایش پسوند فایلها

showhiddenextsundo.vbs

عدم نمایش پسوند فایلها

که در پوشه مقاله جاری با نام AVR64.com-showhiddenexts.zip موجود می باشد Unzip کرده و روی اولی کلیک نمایید و سیستم را Restart کنید. در این صورت تمامی پسوندها نمایش داده می شوند و برای عدم نمایش نیز روی دومی کلیک کنید. (این فایلها کارKelly هستند).

پس از موفقیت در ایجاد فایل Prog1.asm روی آن کلیک کرده و عبارات زیر را داخل آن تایپ نمایید:

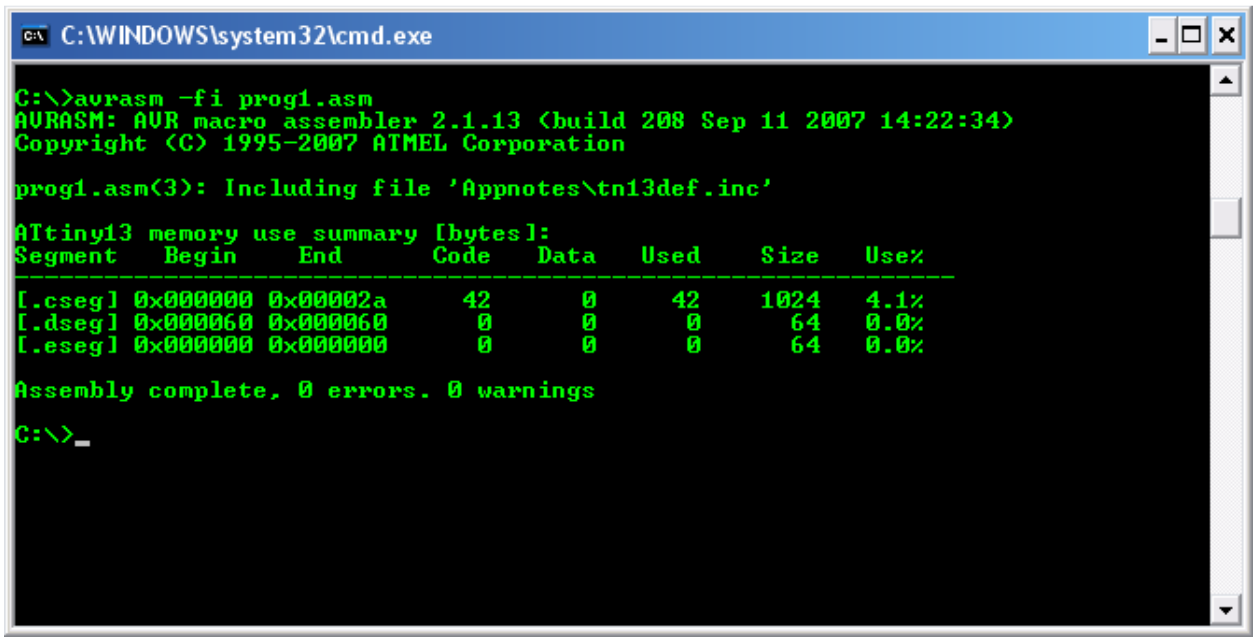


برنامه بالا یک چشمکزن دولامپی ساده با میکروکنترلر ATtiny13 می باشد که با Notepad++ باز شده است. البته Notepad++ قابلیت نمایش کدهای اسمبلی AVR به صورت رنگی را ندارد و تمامی حروف ذخیره شده و پسوند asm به صورت دستی در آن وارد شده اند. در مورد دستورات داخل این فایل بعداً به طور مفصل بحث خواهیم کرد ولی فعلاً نحوه اسمبل کردن و ایجاد فایل Hex و نیز نحوه پروگرم کردن میکروکنترلر را آموزش خواهیم داد.

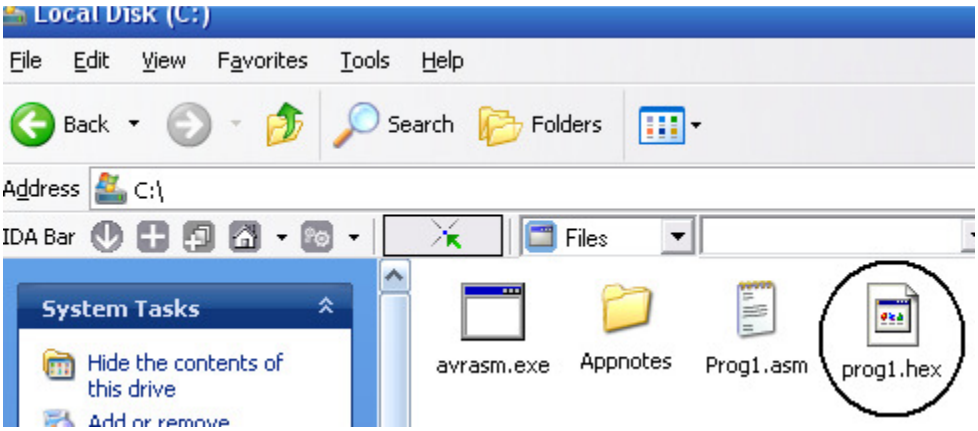
پس از ذخیره فایل Prog1.asm در داخل Root (ریشه) درایو C، پنجره DOS مربوط به اسمبلر را که قبلاً به حالت Minimize برده بودید باز کنید و در آن عبارت زیر را تایپ و Enter نمایید:

```
avrasm -fI Prog1.asm
```

توجه داشته باشید که بین avrasm و fI- یک فاصله و بین دو عبارت بعدی یعنی fI- و Prog1.asm نیز یک فاصله وجود دارد. عبارت fI- باید به صورت پشت سر هم نوشته شود. این Option به معنی فرمت فایل خروجی Intel Hex می باشد. پس از تایپ عبارت بالا و زدن کلید Enter جدولی نمایش داده می شود که اطلاعاتی در مورد آدرس های قرار گیری کد و دیتا و نیز حجم استفاده شده و باقیمانده از حافظه Flash(کد) و SRAM(دیتا) و نیز EEPROM(دیتای دائمی) را نشان می دهد. به علاوه خطاها و هشدارها نیز در انتهای صفحه نشان داده می شوند:

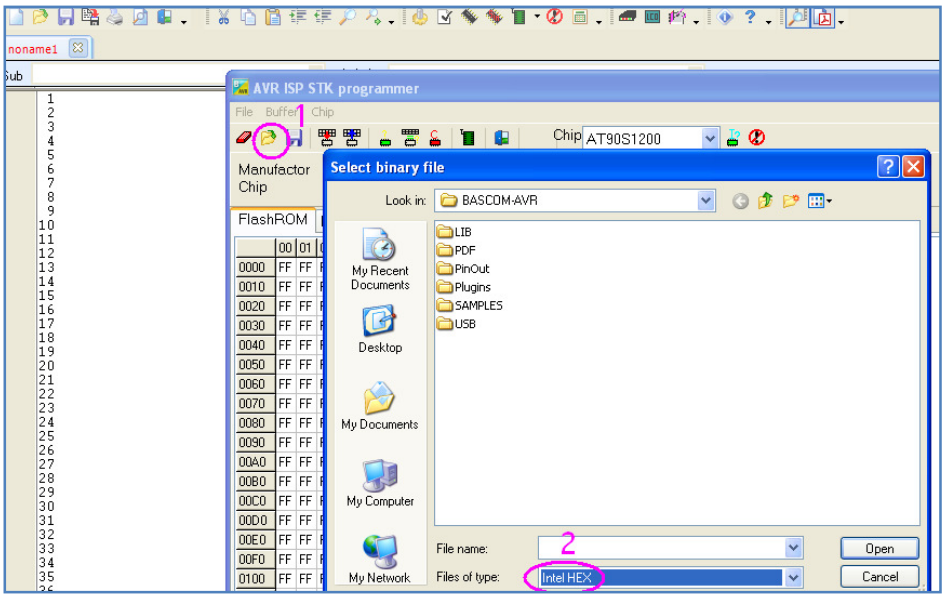


در صورتی که هیچ Error ی وجود نداشته باشد فایل Hex در کنار فایل Asm ایجاد می شود که می توانیم آن را در میکرو پروگرم نماییم.

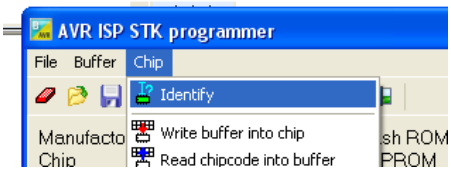


پروگرم کردن میکرو:

در این قسمت نوبت به پروگرم کردن میکرو (انتقال فایل هگز به حافظه فلش میکرو) می رسد. . لازم به ذکر است برای پروگرم کردن میکرو از نسخه دموی کامپایلر Bascom استفاده می کنیم. برای دانلود نسخه آزمایش این کامپایلر که در اینجا فقط از بخش پروگرمر آن استفاده می کنیم می توانیدبه سایت www.mcselec.com مراجعه کنید. در صورتی که پروگرمر شما نرم افزاری خاصی برای پروگرم کردن میکرو داشته باشد نیازی به دانلود استفاده از بسکام نیست و می توانید طبق توضیحات دفترچه پروگرمر خود برای انتقال فایل Hex تولید شده به میکرو استفاده نمایید. در صورت نیاز به پروگرمر Bascom کامپایلر بسکام را باز کرده، روی File > New کلیک کنید تا یک فایل خالی باز شود، سپس دکمه F4 را بزنید محیط پروگرمر STK نمایش داده شود. مطابق شکل صفحه بعد روی شکل پوشه Open واقع در نوار ابزار پنجره پروگرمر کلیک کنید تا پنجره انتخاب فایل هگز ظاهر شود. نظر به اینکه این پنجره به صورت پیش فرض به دنبال فایل باینری میگردد و فایل های با پسوند Hex را نشان نمی دهد بایستی از قسمت پایین Files of type گزینه Intel HEX را انتخاب نمایید. با استفاده از این روش به راحتی می توان توسط کامپایلر بسکام هر نوع فایل Hex را در میکرو پروگرم نمود. دلیل استفاده از این روش عدم عملکرد مناسب پروگرمر داخلی WinAVR می باشد که معمولاً قادر به شناسایی پورت LPT نیست.

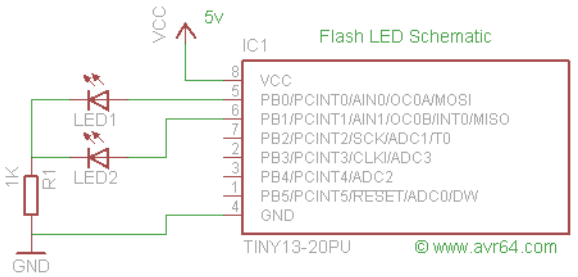


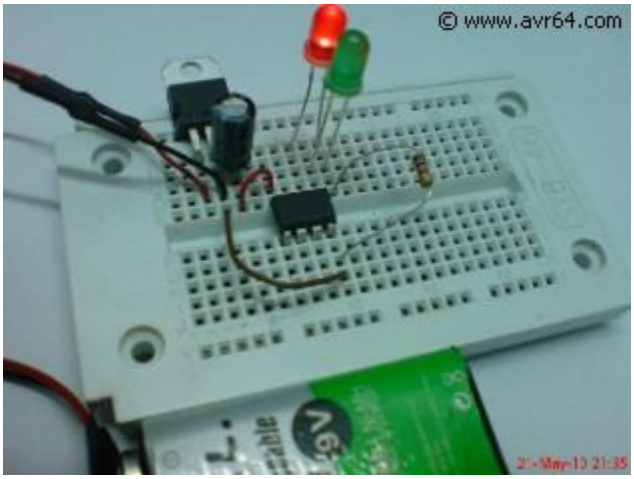
نکته: پس از انتخاب فایل هگز بایستی از منوی Chip یک بار بر روی Identify کلیک کنیم تا دکمه های این برنامه فعال شوند. (در این برنامه نیازی به تنظیم فیوزبیت ها نمی باشد، نحوه تنظیمات در مقالات آموزش بیسکام شرح داده شد).



آزمایش مدار:

برای تست این پروژه، سخت افزاری مطابق با شماتیک زیر بر روی بِرِد بُرد ببندید. با قرار دادن آی سی پروگرم شده در مکان مناسب و اتصال تغذیه ۵ ولت شاهد چشمکزدن دیود های نورانی خواهید بود. (البته این مدار ساده ترین مدار ممکن با اسمبلی است و در آینده به راه اندازی LCD و کی پد و ... خواهیم پرداخت).





با نگاهی به کد Hex تولید شده توسط اسمبلر مشاهده می کنیم که هیچ کد اضافی تولید نشده و تمام برنامه در ۵ خط جمع آوری شده است. برنامه های اسمبلی همیشه سرعت اجرای بالایی دارند و خطا در آنها صفر است. البته راه اندازی سایر دستگاه ها از قبیل LCD نیاز به داشتن اطلاعات کافی در مورد نحوه سیگنالینگ و طرز کار آنهاست. در بخش بعد به تحلیل برنامه چشمکزن دولامپی می پردازیم.

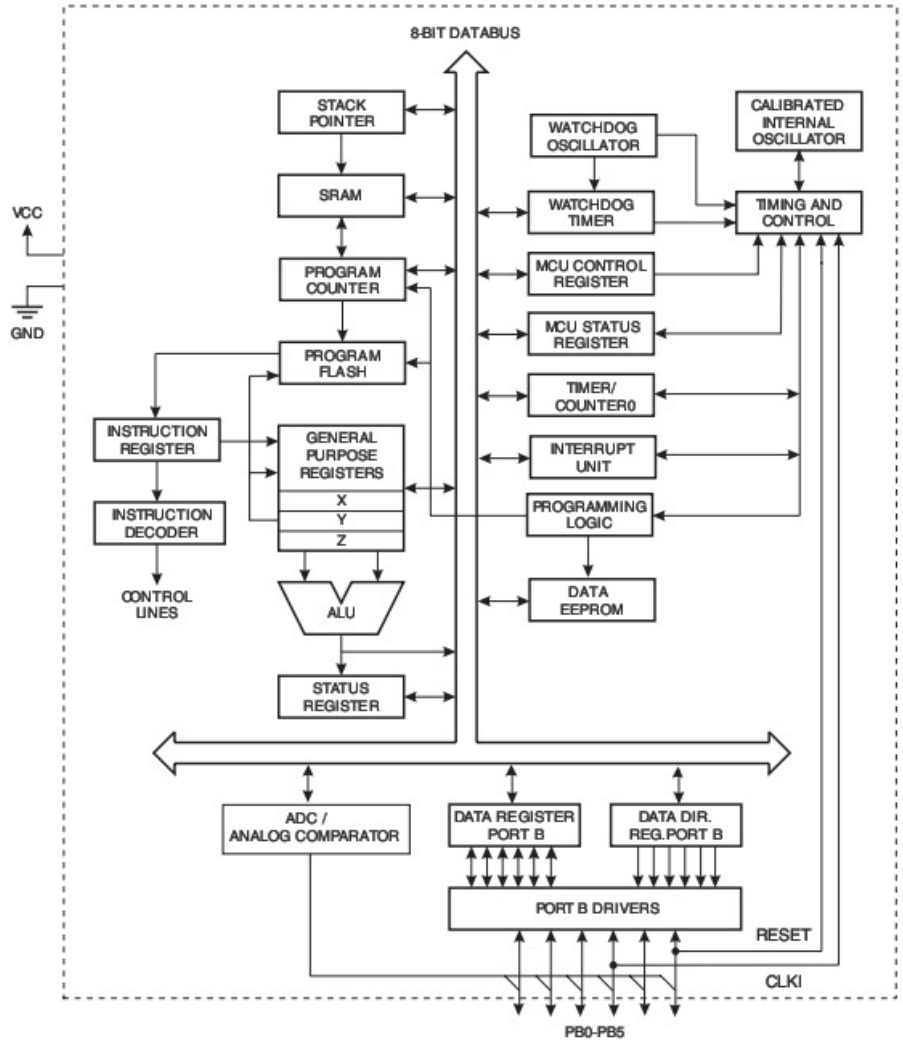
prog1.hex

```
1  :020000020000FC
2  :100000004FE94DBFB89AB99AC09AC19804D0C09828
3  :10001000C19A01D0F9CF0AE01FEF2FEF2A95F1F72F
4  :0A0020001A95D9F70A95C1F7089563
5  :00000001FF
```

تحلیل برنامه نوشته شده با اسمبلی:

هم اکنون نوبت به آموزش زبان اسمبلی رسیده است. تا اینجا با نحوه نوشتن یک کد اسمبلی، ذخیره و اسمبل کردن و نیز پروگرم کردن آن در میکرو آشنا شدیم. در این قسمت در مورد دستورات زبان اسمبلی و بخش های داخلی میکرو صحبت می کنیم. همانطوریکه می دانیم یک میکروکنترلر از بخش هایی مانند CPU، حافظه Flash (محل ذخیره سازی ثابت ها و Hex)، SRAM (محل ذخیره سازی موقت متغیرها)، EEPROM (محل ذخیره سازی دائمی متغیرها)، مبدل آنالوگ به دیجیتال (در برخی میکروها)، واحد تشخیص وقفه ها (Interrupts)، تایمرها و... تشکیل شده است. از طرفی تنها راه ارتباط میکرو با دنیای خارج پورت ها هستند. در میکروی نمونه ATtiny13 فقط یک پورت B وجود دارد که به جای ۸ پین فقط ۶ پین دارد و دلیل آن هم محدودیت پایه های این نوع آی سی است. (DIP8). هر یک از بخش های داخلی میکرو آدرس خاصی دارند و همگی روی یک DATABUS هشت بیتی قرار گرفته اند. به طور مثال در این برنامه ساده فقط پورت B مورد استفاده قرار می گیرد و بیت های این پورت ۰ و ۱ می شوند. در این برنامه از فضای SRAM و سایر بخش های میکرو هیچ استفاده ای به عمل نیامده و فقط از رجیستر

Figure 2-1. Block Diagram



های همه منظوره CPU برای ایجاد حلقه ها و تولید تاخیر (Delay) بهره گرفته شده است. در محیط اسمبلی خطوطی که با علامت سمی کالن شروع می شوند به عنوان توضیح به حساب آمده و در عملکرد اسمبلر نقشی ندارند. در این برنامه ۲ خط اول توضیح هستند. خط شماره ۳ که با .include شروع شده است برای ضمیمه کردن فایل inc میکروکنترلر مربوطه به کار می رود (تقریباً شبیه به **$regfile="ATtiny13.dat"** در Bascom).

```
1   ; Flash 2 LED, Simple asm Program by Behnam zakizadeh
2   ; www.avr64.com @ Friday, 31.02.1389 - 21.May.2010
3   .include "Appnotes\tn13def.inc"
4
```

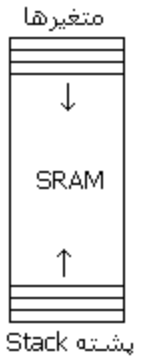
دو خط بعدی مربوط به تعیین محل قرار گیری کدهای Hex در حافظه فلش میکرو هستند. اصولاً دستوراتی که با نقطه شروع می شوند به رهنمودها یا راهنماهای پیش پردازنده معروفند و برای راهنمایی اسمبلر به کار می روند (رهنمودهای پیش پردازنده در اسمبلی با نقطه، در C با # و در بیسیک (بسکام) با $ شروع می شوند). رهنمود .CSEG مخفف Code Segment است و خط بعد از آن آدرس شروع سگمنت کد برنامه را مشخص می کند. ما در اینجا آدرس ۰ ر انتخاب کرده ایم. البته همیشه بعد از .CSEG بلافاصله .ORG 0x00 نوشته می شود که به معنای خانه شماره ۰ حافظه کدی میکرو است و در آن فقط دستور call main یا rcall main یا rjmp main نوشته شده و بلافاصله پس از آن دستورات پرش به روتین های وقفه آورده می شود (در جلسات بعدی با این مفاهیم در داخل کدهای موبوط به وقفه ها بیشتر آشنا خواهیم شد) در این برنامه چون این موارد رعایت نشده اگر وقفه ای روی دهد خط موجود درآدرس بردار وقفه اجرا می شود و ترتیب اجرای برنامه به هم می ریزد.

به غیر از .CSEG دو رهنمود دیگر نیز به نام های .DSEG برای حافظه دیتا یا همان SRAM و .ESEG برای حافظه EEPROM وجود دارند. آدرس .DSEG معمولاً از 0x60 شروع می شود و در مقادیر بالای این آدرس برای آدرس دهی SFR ها، پورتها و سایر امکانات داخلی میکرو به کار می روند و دلیل همپوشانی این آدرس ها استفاده از باس مشترک برای رم و سایر بخش های داخلی میکرو است. از .ESEG آدرس ۰ شروع می شود و این نشان دهنده جدا بودن باس آن از سایر بخش هاست. البته تمام این موارد ممکن است برای میکروهای مختلف متفاوت باشد و در دیتا شیت میکروی مورد نظر آورده شده است.

```
5   .CSEG
6   .ORG 0x00
```

پس از ORG مربوط به CSEG نوبت به نوشتن اولین خطوط از کدهای اسمبلی فرا می رسد. معمولاً اولین کاری در یک میکرو بعد از

Startup انجام می شود تعیین آدرس پشته یا همان Stack است. همانطوریکه در جلسات قبلی (بسکام) نیز ذکر شده بود با فراخوانی هر زیر برنامه که با دستور Call، rcall یا... صورت می گیرد آدرس جاری PC که مخفف پروگرم کانتر یا همان شمارنده برنامه است در مکان جاری پشته ذخیره می شود و پس از اجرا و اتمام زیر برنامه به رسیدن به ret یا reti (در زیر برنامه های مربوط به اینتراپت یا وقفه) آدرس مورد نظر از پشته بازخوانی شود و ادامه برنامه از PC+1 یعنی دستور بعد از Call از سر گرفته می شود. (محاسبات PC+1 ممکن است قبل از جایگذاری آدرس در پشته صورت بگیرد و یا بعد از بازخوانی آن و اهمیتی ندارد). به همین دلیل تنظیم سایز پشته اهمیت فراوانی دارد و در صورت عدم تنظیم آن هیچ یک از فراخوانی های برنامه کار نخواهند کرد. همیچنین نکته قابل توجه استفاده درست از فضای رم میکرو است. در صورتی که از حافظه SRAM تا نزدیک آخر استفاده شود و نیز به دلیل فراخوانی های تو در توی توابع و زیر برنامه ها فضای پشته ها به اندازه کافی به سمت بالا رشد کند مشکلی به نام Stack overflow ایجاد می شود که در هسته AVR می توان آن را Overlap یا همپوشانی نامید، چرا که چیزی به نام سرریزی پشته وجود ندارد ولی اطلاعات پشته با رم دچار اختلاط شده و باعث از بین رفتن اطلاعت داخل رم می شود. این مشکل ممکن است باعث ایجاد حلقه های نامحدود شود و کل برنامه مانند کامپیوتری که به حالت Freeze رفته هنگ کند که این مشکل با Watch dog قابل حل است ولی در صورتی که فقط باعث خراب کردن دیتا شود و فقط مقدار آن را تغییر دهد مشکل جدی تر خواهد بود چرا که در ظاهر، میکرو به درستی کار می کند ولی اطلاعات غلط می دهد. به همین دلیل بهترین راه استفاده کمتر از فضای رم و تخمین حداکثر فضای لازم برای فراخوانی های تو در تو می باشد.



برای تعیین سایز پشته بایستی آدرس انتهای SRAM در رجیسترهای SPL و SPH قرار بگیرند. این دو رجیستر ۸ بیتی که مخفف Stack Pointer می باشند برای تعیین محل شروع پشته به کار می روند. نظر به اینکه پشته به صورت معکوس رشد می کند بهترین مکان برای شروع پشته برای حداکثر استفاده از فضای رم انتهای آن است. در داخل فایل Inc میکرو متغیری به نام RAMEND تعریف شده است که مقدار آن برابر با آدرس پایان رم میکرو است. با توجه به اینکه در میکروهای با حافظه رم بالاتر از ۲۵۶ بایت این متغیر دو بایتی می باشد با استفاده از دستور ()Low مقدار LSB یا بیت های کم ارزش تر آن را جدا کرده و توسط دستور LDI که به معنای Load Immediate می باشد مقدار آن را در رجیستر R20 میکرو به طور موقت کپی می کنیم. توجه داشته باشید که AVR ها ۳۲ رجیستر همه منظوره با شماره های ۰ تا ۳۱ دارند که معمولاً از

رجیستر های ۱۶ به بالا برای مصارف عمومی استفاده می شود و تمام این رجیستر ها ۸ بیتی اند. البته ۶ رجیستر آخر دو به دو تشکیل رجیستر های ۱۶ بیتی X و Y و Z را می دهند که در جای خود بحث می شود. در خط بعدی با دستور Out که برای ارسال مقدار به یک آدرس خاص مثل پورت یا یکی از SFR های داخلی میکرو استفاده می شود، مقدار داخل رجیستر R20 را در رجیستر SPL کپی می کنیم. نظر به اینکه میکروی Tiny13 فقط 64 بایت SRAM دارد کاری با رجیستر SPH نداریم ولی در غیر این صورت می بایست با دستور High(RAMEND) مقدار بالای آدرس انتهایی رم را نیز در رجیستر SPH کپی می کردیم.

```
7     ; set stack pointer
8     ldi r20, low(RAMEND)
9     out SPL, r20
```

در میکروی ATmega32 بایستی اینگونه عمل کنیم: (دستور byte1 معادل low و دستور byte2 معادل high است).

```
.include "Appnotes\m32def.inc"
ldi r20, byte1(RAMEND)
out SPL, r20
ldi r20, byte2(RAMEND)
out SPH, r20
```

با مراجعه به دیتا شیت میکرو در صورتی که مشاهده کردید فضای SRAM از ۲۵۶ بایت بیشتر است از روش دوم استفاده کنید در غیر این صورت فقط متغیر SPL را مقدار دهی نمایید. (ضمناً استفاده از رجیستر R20 اجباری نیست و از هر کدام از رجیسترهای R16 تا R31 می توانید بهره ببرید).

در خطوط ۱۱ و ۱۲ با مقدار دهی رجیستر DDR که مخفف Data Direction Register به معنای رجیستر تعیین جهت داده است، ورودی یا خروجی بودن پورت مورد نظر را تعیین می نماییم. در این پروژه میکروی ما فقط یک پورت B داشته و فقط DDRB موجود است. نظر به اینکه یک پورت استاندارد دارای ۸ پایه می باشد رجیستر DDR نیز از ۰ تا ۷ به صورت DDRB,0 قابل آدرس دهی است. هر چند به طور کلی نیز می توان DDRB را با یک عدد ۸ بیتی آدرس دهی کرد. برای تعیین پایه مورد نظر به صورت خروجی بایستی DDR مربوطه را با ۱ مقدار دهی کرده و برای تنظیم به صورت ورودی با ۰ مقدار دهی نماییم. دستور sbi برای ست کردن یک بیت و cbi برای ریست کردن آن به کار می رود. در دو خط زیر با یک کردن دو بیت ۰ و ۱ رجیستر پورت B این دو پایه را به صورت خروجی تعریف می نماییم.

```
10      ; set as output
11      sbi ddrb,0
12      sbi ddrb,1
```

same: ddrb,1 = 1 (but '=' not allowed in asm!)

قسمت بعدی بدنه اصلی برنامه چشمکزن دولامپی است. این برنامه از یک لیبل با علامت دو نقطه پس از آن شروع شده و با rjmp خاتمه می یابد. در واقع دو دستور یاد شده تشکیل یک حلقه نامحدود مانند Do:Loop بیسیک یا While(1){}; زبان C را می دهند. در داخل این حلقه با دستورات sbi و cbi رجیسترهای portb را روشن خاموش کرده ایم. سپس با فراخوانی زیر برنامه delay کمی تاخیر ایجاد کرده و مجدداً دیود های LED را به صورت عکس روشن و خاموش نموده ایم.

```
14      flash:
15          sbi portb,0
16          cbi portb,1
17          rcall delay
18          cbi portb,0
19          sbi portb,1
20          rcall delay
21      rjmp flash
```

توجه داشته باشید که دستورات rcall و rjmp به صورت استاندارد call و jmp نیز وجود دارند و قادر به پرش به نقاط دور تری می باشند ولی به دلیل اینکه حافظه زیادی از میکرو می گیرند بهتر است در میکروهای کم حافظه از معادل آنها استفاده کنید. ضمناً برخی از دستورات استاندارد در میکروهای کوچک تعریف نشده اند.

قسمت آخر برنامه مربوط به زیربرنامه delay یا تاخیر است. در این زیر برنامه با پر و خالی کردن رجیستر های ۱۶ تا ۱۸ وقت CPU گرفته می شود و اندکی تاخیر در چشمکزدن LED ها ایجاد می شود. در این برنامه با دستور dec یک واحد از رجیستر کم می شود و با دستور brne بررسی می کنیم که اگر مقدار رجیستری که در سیکل قبلی روی آن عملیات صورت گرفت 0 نشده بود به دستور جلوی brne پرش کند در غیر این صورت به خط بعد برود. (چیزی شبیه به دستور شرطی IF). در این برنامه حلقه داخلی Loop3 به تداد 255 سیکل ماشین وقت CPU را هدر می دهد و حلقه Loop2 به تعداد 255x255 بار و حلقه اصلی Loop1 به میزان 10x255x255 بار. با تغییر عدد ۱۰ می توان زمان این تاخیر را تغییر داد. اصولا در کامپایلرهایی مانند بسکام، کدویژن یا WinAVR از تابعی به نام Waitms x یا wait_ms(x) استفاده می

شود که در درون آن چنین کدی نهفته است و مقدار متغیر ورودی با محاسباتی داخل رجیستر r16 ذخیره می شود (r16 در این برنامه). دستور ret نیز به معنای بازگشت می باشد و میکرو با رسیدن به آن، آدرس جایی را که از آنجا پرش کرده بود از Stack فرا می خواند و به PC تحویل می دهد.

```
23    delay:
24    ldi r16, 10
25    loop1: ldi r17, 255
26    loop2: ldi r18, 255
27    loop3: dec r18
28        brne loop3
29        dec r17
30        brne loop2
31        dec r16
32        brne loop1
33    ret
```

در این برنامه از کد سگمنت فقط برای ذخیره فایل اجرایی یعنی Hex میکرو استفاده شد و از دیتا سگمنت و ESEG نیز استفاده ای به عمل نیامد. ازکد سگمنت می توان برای تعریف جدول دیتا و یا ثابت ها استفاده کرد و از دیتا سگمنت نیز برای تعریف متغیرها استفاده می شود. متغیرهایی که در بسکام با دستور Dim و در C با ساختار دستوری int x; تعریف می شوند در فضای دیتا سگمنت قرار می گیرند. با تعریف متغیرها می توان اطلاعات را به جای رجیستر های محدود در داخل آنها کپی کرد و برنامه را وسعت داد. البته رجیستر ها کاربرد خود را دارند و به دلیل سرعت دسترسی بالا و اینکه به طور مستقیم در یک خط قابل دسترسی هستند قابل مقایسه با فضای رم نمی باشند. فضای ایپرام نیز برای ذخیره متغیرهای دائمی به کار می رود که در اسمبلی کار با آن ساده است. برای راه اندازی سایر بخش های داخل میکرو از قبیل تایمرها و ... دستورات خاصی وجود دارد و همانند پورت ها که قبل از استفاده احتیاج به مقدار دهی رجیستر خاص DDR دارند، ادوات دیگر نیز بایستی قبل از استفاده توسط رجیستر تنظیمات خود پیکره بندی شوند و سپس مورد استفاده قرار گیرند. در جلسه بعدی از میکروی بزرگتر Mega16/32 برای راه اندازی LCD و کی پد بهره می گیریم و پس از آشنایی با روش تعریف متغیرها نحوه کار با تایمر را شرح می دهیم. بیس اصلی اسمبلی در این جلسه نهفته است و در صورتی که تسلط کافی در تعریف پورت ها، تشخیص سگمنت ها، اسمبل کردن و اشکال زدایی برنامه این جلسه پیدا کنید مطالعه جلسات بعدی برای شما به مراتب آسان تر و لذت بخش تر خواهد بود. ضمناً از جلسه بعد کنسول LCDو کی پد و میکروی Mega16/32 را تا چند جلسه ثابت نگه می داریم تا بتوانیم توسط این ورودی و خروجی های استاندارد با سایر بخش های داخلی میکرو از قبلی تایمرها، وقفه ها، مبدل آنالوگ به دیجیتال و ... ارتباط برقرار کنیم. شایان ذکر است تمام میکرو ها همه بلوک

های موجود را در داخل خود ندارند ولی با زبان اسمبلی (و حتی بیسیک و C) می توان بلوک هایی را که فقط با ۰ و ۱ ها سر و کار دارند (نه بلوک های آنالوگ) مانند ارتباط سریال، ارتباط با MMC، ارتباط با TCP/IP بدون نیاز به چیپ جدا، ارتباط با انواع سنسورها از قبیل DS1820، SMT160، فرستنده و گیرنده مادون قرمز و... را به صورت نرم افزاری پیاده کرد. از اسمبلی برای نوشتن هدر فایل های C و Lib های بسکام نیز استفاده می شود و با مطالعه آن و تسلط کافی و تحقیق در مورد فرمت های وسایل ارتباطی مختلف (مثل ماوس) می توانید هدر ها و کتابخانه های شخصی و تجاری برای کامپایلرهای مختلف بنویسید.

پایان جلسه اول

ادامه دارد...

مولف: بهنام زکی زاده

www.avr64.com

۲۵ آذر ۱۳۸۹

آخرین ویرایش ۲۰ فروردین ۱۳۹۳ ✓