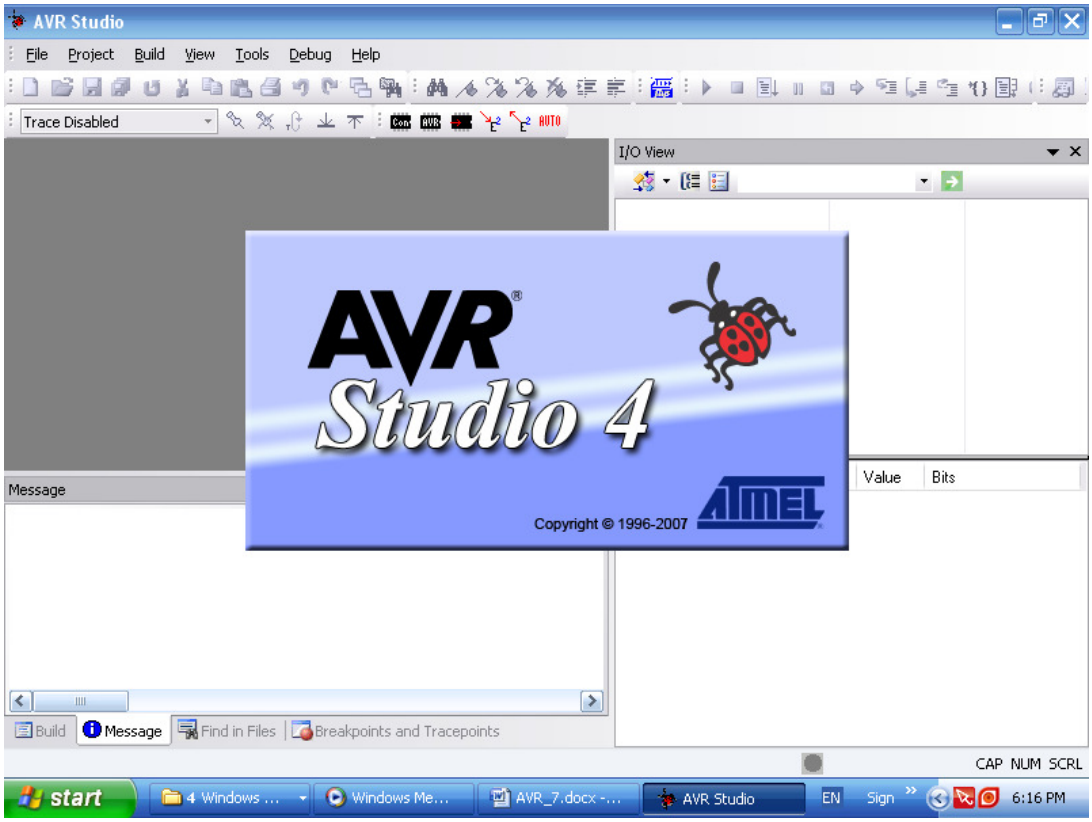
به نام خدا

(جلسه هفتم آموزش AVR)

(Assembly 2)



مقدمه:

در جلسه قبل با زبان اسمبلی و اسمبلر تحت DOS میکروکنترلرهای AVR آشنا شدیم. همچنین آموختیم چگونه یک برنامه ساده برای کنترل پورت و ارسال اطلاعات به پورت خروجی بنویسیم. در این جلسه با معرفی پروتکل نمایشگرهای کریستال مایع و نحوه ارتباط با کی پد ماتریسی و نیز نحوه معرفی متغیر ها و استفاده از فضای رم میکرو مبحث را قدری حرفه ای تر و کاربردی تر می کنیم. ضمناً از این جلسه بجای اسمبلر تحت داس از برنامه AVR Studio که به طور رایگان توسط سایت ATMEL.com ارائه شده است بهره می گیریم. این نرم افزار تحت ویندوز بوده و با داشتن یک محیط IDE و نیز شبیه ساز و توانایی ارتباط با کامپایلر GCC به عنوان ابزار مناسبی برای نوشتن برنامه های AVR به زبان های اسمبلی و C شناخته می شود. از لحاظ قدرت با اسمبلر تحت داس یکسان است و در واقع از همان اسمبلر استفاده می کند. **(نسخه ۵ نیز اخیراً ارائه شده)**

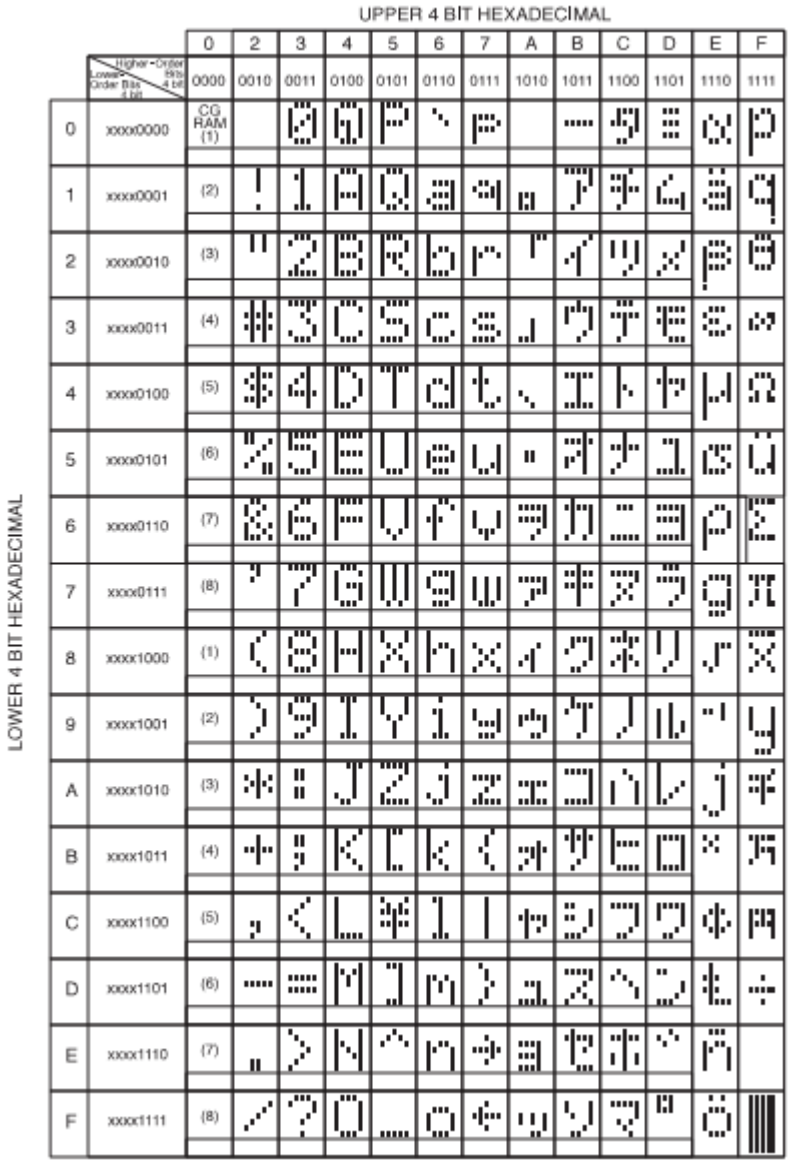
آشنایی با پروتکل LCD:

نمایشگرهای کریستال مایع ۲ در ۱۶ متداول ترین نوع نمایشگر برای خروجی میکرو می باشند. این نمایشگرها در نوع ۲ در ۱۶ شامل دو سطر ۱۶ کاراکتری بوده و قادر به نمایش تمام اعداد و حروف لاتین و نیز کاراکترهای قابل تعریف توسط کاربر هستند. نمایشگرهای LCD معمولا ۱۶ سیم ارتباطی دارند که عملکرد آنها به ترتیب پایه ها به شرح زیر است:

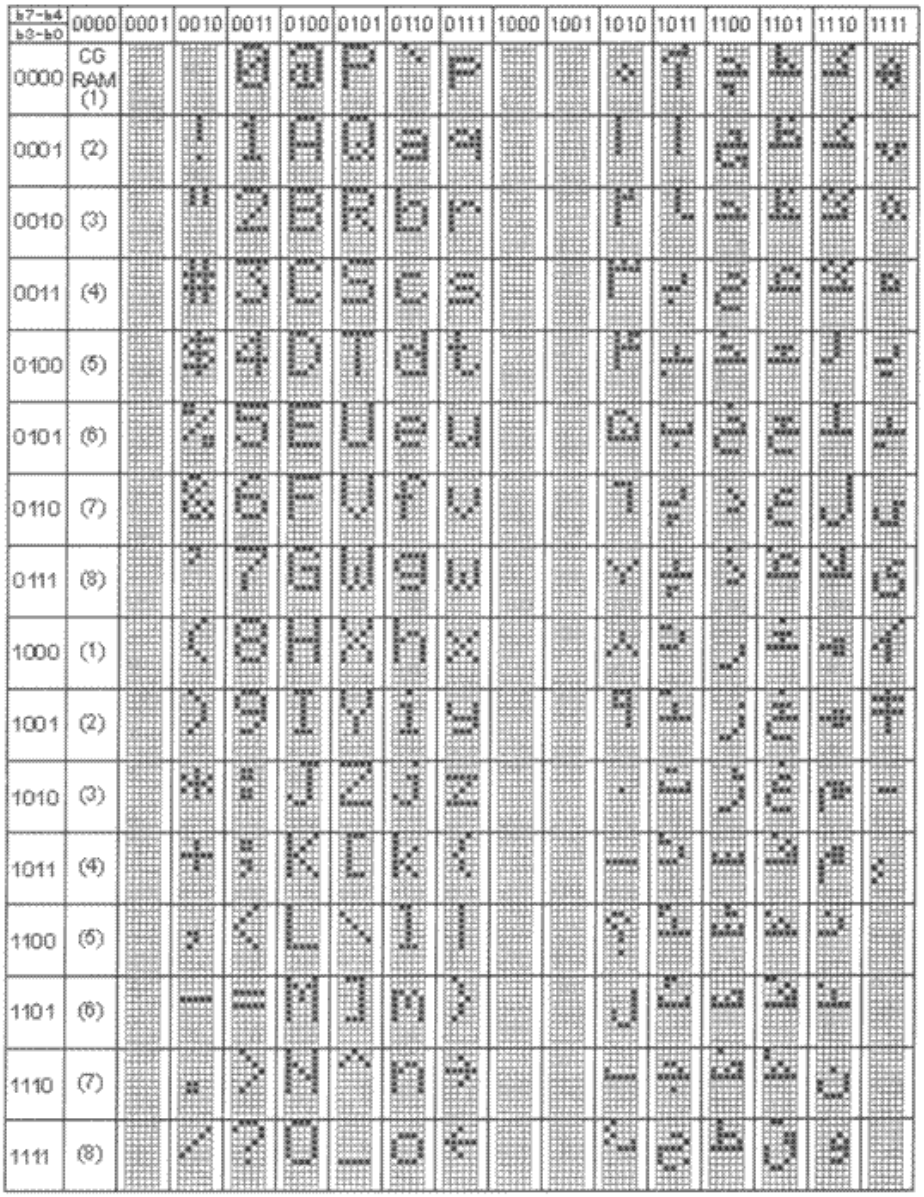
۱- زمین
۲- مثبت ۵+ ولت تغذیه چیپ های داخل LCD
۳- کنترل کنتراست (درخشندگی)
۴- اطلاعات/ دستور
۵- خواندن/ نوشتن
۶- فعال ساز
۷- بیت ۰ دیتا
۸- بیت ۱ دیتا
۹- بیت ۲ دیتا
۱۰- بیت ۳ دیتا
۱۱- بیت ۴ دیتا
۱۲- بیت ۵ دیتا
۱۳- بیت ۶ دیتا
۱۴- بیت ۷ دیتا
۱۵- مثبت ۵+ ولت بک لایت (چراغ پشت LCD)
۱۶- زمین بک لایت

در برخی از LCD های فاقد نور پس زمینه پایه های ۱۵ و ۱۶ وجود ندارند. پایه های ۷ تا ۱۴ مربوط به دیتای نمایشگر بوده و در کامپایلرهای پیشرفته معمولا فقط از ۴ بیت پر ارزش تر یعنی پایه های ۱۱ تا ۱۴ استفاده می شود و اطلاعات در قالب نیبل های ۴ بیتی به نمایشگر ارسال می شوند. ما در این مبحث آموزشی از مد ۸ بیتی استفاده کرده و هر ۱۶ پایه LCD را به کار می بریم تا کد ساده تری داشته باشیم.

قبل از هر چیز بد نیست بدانید که تمام کاراکترهای لاتین شامل اعداد و حروف و علامت های کاربردی در قالب استانداردی به نام کدهای آسکی (ASCII) گرد هم آمده اند. تعداد حروف در این استاندارد بدون درنظر گرفتن زبان محلی ۱۲۸ کاراکتر و با در نظر گرفتن زبان محلی ۲۵۶ کاراکتر می باشد (در استاندارد رسمی فقط ۱۲۸ کاراکتر مد نظر است). با توجه به کارخانه سازنده LCD ممکن است زبان محلی پشتیبانی شده چینی و یا عربی باشد:



LCD فارسی



آپدیت: نمونه ای از LCD فارسی در جام جم الکترونیک امجد تهران (جمهوری) مشاهده شده و عکس بالا مربوط به دیتا شیت های فروشگاه جهان کیت تهران می باشد و در کل این نوع LCD در حال حاضر احتمالاً از فروشگاه های یاد شده قابل تهیه است.

در هر صورت کد تمام کاراکترهای لاتین استاندارد و ثابت می باشد. به طور مثال کد کاراکتر A بزرگ به این صورت محاسبه می شود که در ردیف بالا ستون حاوی A را یافته و کد آن ستون را در بخش پر ارزش تر یادداشت می کنیم 0100xxxx و از ردیف سمت چپ سطر مربوطه به کاراکتر A را پیدا کرده و عدد آن سطر را در بخش کم ارزش تر می نویسیم تا یک عدد ۸ بیتی که همان کد اسکی کاراکتر A است پیدا شود:

01000001

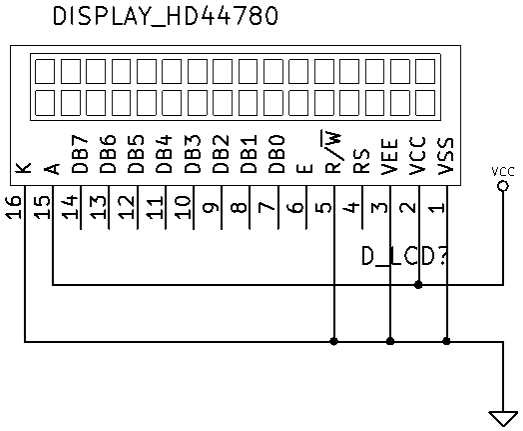
$1x2^0 + 0x2^1 + 0x2^2 + 0x2^3 + 0x2^4 + 0x2^5 + 1x2^6 = 1 + 0 + 64 = 65$

عدد ۶۵ کد اسکی کاراکتر A می باشد و کد هگز آن ۴۱ است. البته با کد Hex کاری نداریم و در مدارات الکترونیکی فقط با اعداد باینری کار می کنیم تا درک آن راحت تر باشد. برای کار با کدهای اسکی Notepad ویندوز را باز کرده و کلید Alt را نگه دارید و عدد ۶۵ را تایپ نمایید سپس کلید Alt را رها کنید، مشاهده می کنید که کاراکتر A تایپ می شود. همین کار را برای سایر اعداد انجام دهید و کدهای اسکی را بررسی کنید.

برای نوشتن کاراکتر A در LCD بایستی ۸ بیت بالا به ترتیب روی ۸ پایه دیتا یعنی پایه های ۷ تا ۱۴ قرار بگیرند. البته قبل از آن لازم است که دستورات کنترلی مثل پاک کردن LCD، بردن مکان نما به سطر و ستون اول و... نیز که در قالب کدهای باینری هستند به نمایشگر ارسال گردند. این دستورات نیز ۸ بیتی می باشند و برای اینکه نمایشگر آنها را با کاراکتر اشتباه نگیرد پایه ۴ در نظر گرفته شده است. در صورتی که این پایه ۰ باشد دیتای قرار داده شده روی پایه های دیتا به منزله دستور خواهد بود و در صوتی ۱ بودن پایه شماره ۴ این اطلاعات به منزله کاراکتر و یا هر چیزی که قرار است بر روی صفحه نشان داده شود تلقی می گردند. پایه ۵ نیز موقع نوشتن در نمایشگر بایستی ۰ باشد و موقع خواندن از آن باید در حالت ۱ قرار بگیرد. ما در اینجا آن را به طور دائم به زمین وصل می کنیم تا نمایشگر همیشه در حالت نوشتن باشد. (در کاربردهای حرفه ای این پایه به میکرو وصل شده و پس از نوشتن کاراکتر ۱ می شود تا LCD در حالت خواندن قرار بگیرد، سپس وضعیت پایه ۷ دیتای LCD که به عنوان Busy flag عمل می کند به طور مرتب اسکن می شود تا آماده بودن نمایشگر برای دریافت کاراکتر بعدی بررسی شود، ولی در روش های سنتی کمی تاخیر به صورت پیش فرض بین ارسال هر کاراکتر تا کاراکتر بعدی ایجاد می کنیم تا کد ساده تری داشته باشیم) در کدویژن از این پایه استفاده می شود و به همین دلیل سرعت رفرش صفحه از بسکام بیشتر است و در واقع LCD چشمک نمی زند. البته در بسکام هم می توان با ترفندی از چشمکزدن صفحه نمایش جلوگیری کرد و یا از کتابخانه

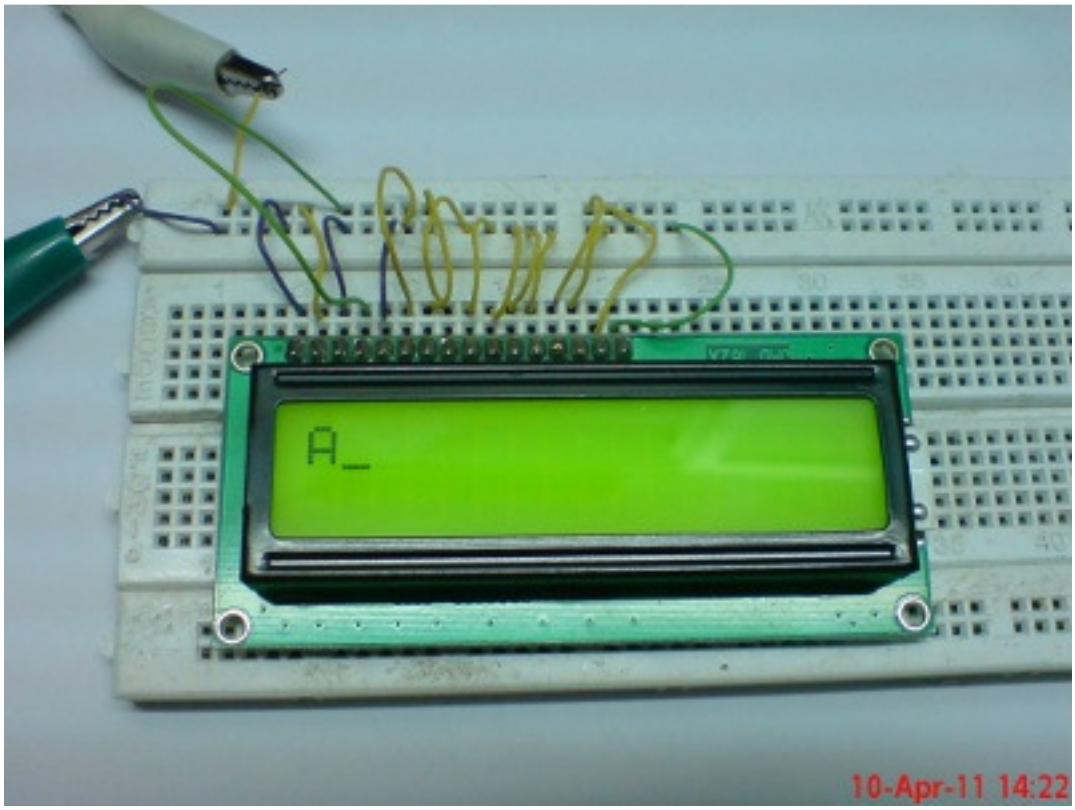
های Busy استفاده کرد که این پایه را به کار می برد. پایه ۶ نیز فعال ساز نام دارد و با سیگنالی از ۱ به ۰ که به آن اعمال می شود مقادیر روی پایه های دیتا در رجیستر داخلی نمایشگر ثبت می شوند. ما در اینجا تصمیم داریم یک LCD معمولی ۲ در ۱۶ را بدون میکرو و منحصراً با سیم راه اندازی نماییم و به صورت دستی کاراکتر A را در آن بنویسیم!

مداری مطابق زیر بر روی برد بورد ببندید و دستورالعمل های داده شده را مرحله به مرحله اجرا کنید تا کاراکتر A را به صورت دستی به نمایشگر ارسال نمایید. (در اینجا شما نقش یک میکروکنترلر را بازی می کنید، البته با سرعت چند میلیون بار کندتر!)



پس از وصل تغذیه ۵ ولت به Vcc و زمین، پایه شماره ۴ را به زمین وصل کنید تا ثبات دستور انتخاب شود (۰ : دستور). پایه ۶ هم که فعال ساز است باید در ابتدا به VCC متصل باشد (یک پالس از ۱ به ۰ باعث نوشته شدن وضعیت پایه های دیتا در رجیستر داخلی میکرو می شود. این پالس دست کم باید ۴۵۰ نانو ثانیه روی ۰ بماند تا نمایشگر آن را تشخیص دهد). برای پیکره بندی یک نمایشگر دستورات زیادی وجود دارند ولی مهمترین آنها تعیین مد و قابل روئیت کردن کاراکترها است. کد دستور اول 38H و دستور دوم 0CH می باشد. دستورات دیگر عبارتند از تعیین موقعیت مکان نما، پاک کردن صفحه، شیفت دادن کاراکترها، چشمکزدن مکان نما و نیز قابل روئیت کردن کاراکترها (بدون پاک کردن) {این دستور در Bascom به صورت Display On و Display Off موجود می باشد که کاراکترها را Hidden و آشکار می کند. (visible و invisible)}. کد دستور اول 00111000 می باشد. این کد را روی پایه های ۷ تا ۱۴ قرار دهید. (چپ ترین رقم روی پین ۱۴). بدین صورت که به ازای ۰ پایه را به زمین و به ازای ۱ پایه را به VCC متصل نمایید. سپس برای نوشتن این دستور پایه ۶ را یک لحظه به زمین وصل کردن و دوباره به

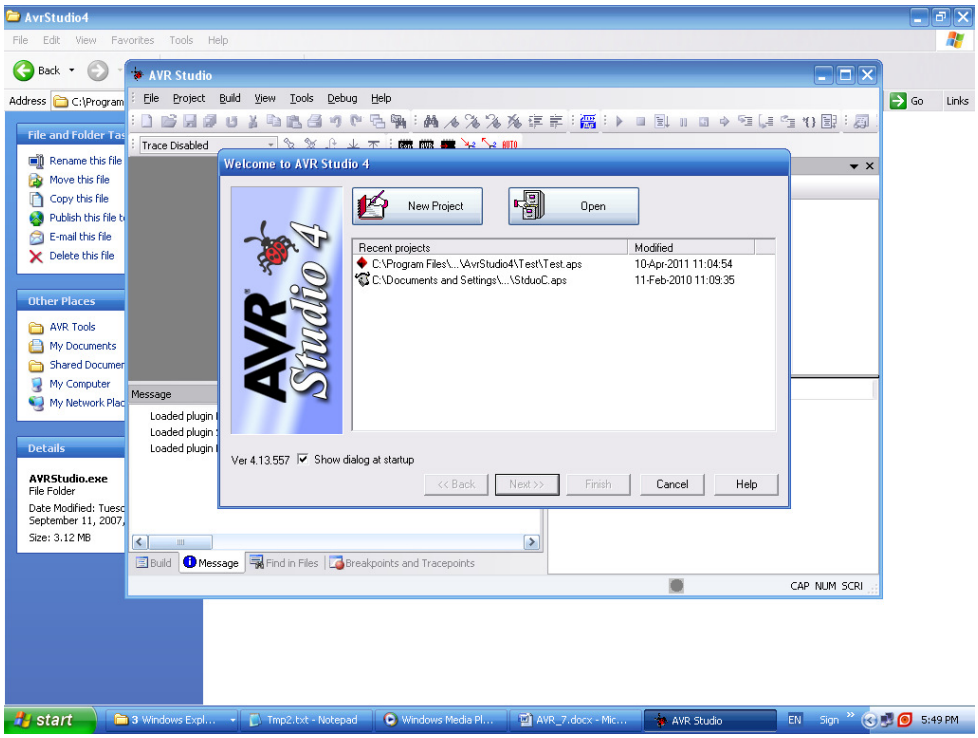
VCC متصل نمایید (هنوز پایه ۴ باید زمین باشد تا اطلاعات در رجیستر دستور نوشته شوند). مشاهده می کنید که خط بالای نمایشگر که تا الان تمام پیکسل های ۱۶ کاراکتر آن روشن بودند خاموش می شود. در واقع این دستور LCD را به مد ۱۶ در ۲ می برد و هر دو سطر را آماده برای نوشتن می نماید. پس از نوشتن دستور اول نوبت به دستور دوم می رسد. این دستور کاراکتر ها را آشکار می کند. کد این دستور C بوده که باینری آن 00001100 می شود. این کد را روی پایه های ۷ تا ۱۴ قرار داده و پایه ۶ را از VCC به زمین وصل کرده و دوباره به VCC متصل نمایید. اگر LCD شما برخی از پیشفرض ها را داشته باشد کاراکتر مکان نمای زیر خط (آندرلاین) _ را در سطر و ستون اول مشاهده می کنید. تا اینجا دستور دادن به LCD تمام شده است و نوبت نوشتن کاراکترهاست. پس سیم متصل به پایه ۴ را از زمین به VCC متصل کرده (آن را ۱ می کنیم) تا رجیستر دیتا را انتخاب کرده باشیم. سپس کد اسکی کاراکتر A که باینری آن به صورت 01000001 می باشد را روی پایه های ۶ تا ۱۴ قرار داده و سیم ۶ را از VCC به زمین وصل می کنیم؛ به محض وصل کردن این سیم به زمین یک پالس (لبه پایین رونده) روی پایه Enable نمایشگر ایجاد شده و اطلاعات روی پایه های دیتا به سرعت بر روی نمایشگر آشکار می شوند. تصویر زیر نتیجه نهایی این کارها را نشان می دهد:

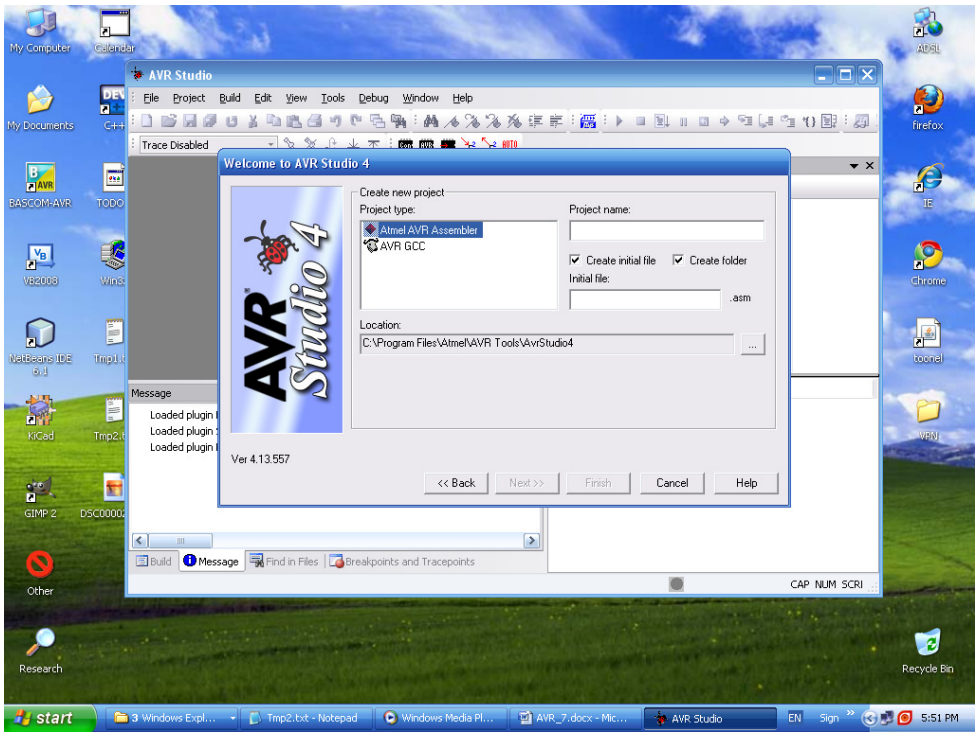


به همین ترتیب تصمیم داریم برنامه ای برای ۰ و ۱ کردن پایه های میکرو برای کنترل LCD و نوشتن جملات دلخواه بر روی آن طراحی و پیاده سازی نماییم. اصولاً اساس تمام ارتباطات دیجیتال بر مبنای ۰ و ۱ بوده و کار میکروکنترلر تولید ۰ و ۱ ها بر اساس برنامه نوشته شده می باشد. (ضمناً فراموش نکنید برنامه ای که در بالا اجرا کردید کاملاً به زبان ماشین بوده و ماشین آن خود شما بوده اید. در صورتی که بخواهید کامپایلری با زبان فارسی بنویسید می توانید با استفاده از یک محیط رایگان مثل VB Express تحت کامپیوتر برنامه ای بنویسید که جملات رشته ای مثل "نمایش بده" را دریافت کرده و به ازای کاراکتر جلوی آن رشته کد های باینری بالا را تولید و به فایل هگز میکرو تبدیل کند. نوشتن کامپایلر با زبان هایی غیر از C و Basic و Pascal هم ممکن است و نیازی نیست که تا ابد از زبان های موجود استفاده کنیم، با قدری خلاقیت می توان یک زبان برنامه نویسی جدید طراحی کرد).

همانطوریکه در جلسه قبل ذکر شد مجموعه ای شامل یک LCD و یک میکروی بزرگ ATmega16 بر روی برد بورد تهیه کرده ایم تا برخی از دستورات اسمبلی را روی آن اجرا کنیم. ضمناً برای راحتی کار از این جلسه به بعد اسمبلر تحت داس را کنار گذاشته و از AVR Studio نسخه ۴ استفاده می کنیم. این برنامه در بیس خود از همان اسمبلر تحت داس استفاده می کند (دقیقاً مانند کامپایلر FAST AVR). ویژگی این محیط اینست که دارای ادیتور و شبیه ساز بوده و به راحتی روی ویندوز اجرا می شود. ضمناً شامل شبیه ساز داخلی بوده و می توان وضعیت رجیسترها و پورت های میکرو را در موقع اجرای برنامه برای دیباگ کردن مشاهده کرد. (در جلسات بعدی نسخه ۵ که در سال ۲۰۱۱ وارد بازار شده را دانلود و نصب کرده و از آن استفاده خواهیم کرد). برای دانلود این نرم افزار به وبسایت اتمل به نشانی Atmel.com مراجعه نمایید.

پس از نصب و راه اندازی نرم افزار AVR Studio پنجره ای مطابق شکل زیر نشان داده می شود، در این پنجره روی کلید New Project کلیک نمایید و در پنجره بعدی گزینه Atmel AVR Assembler را انتخاب کرده و در بخش Project name نام دلخواهی به پروژه بدهید و مسیر ذخیره شدن آن را مشخص نمایید و در نهایت روی کلید Finish کلیک کنید. در پنجره باز شده می توان برنامه های اسمبلی را نوشت و با زدن دکمه F7 آن را اسمبل نمود. (در صورتی که پنجره های یاد شده در ابتدای کار مشاهده نشدند می توانید از منوی Project بر روی Project wizard کلیک نمایید).


AvrStudio4
AVR Studio
Welcome to AVR Studio 4
New Project
Open
Recent projects
Modified
C:\Program Files\...\AvrStudio4\Test\Test.aps
10-Apr-2011 11:04:54
C:\Documents and Settings\...\StduoC.aps
11-Feb-2010 11:09:35
Ver 4.13.557
Show dialog at startup
<< Back
Next >>
Finish
Cancel
Help
Message
Build
Find in Files
Breakpoints and Tracepoints
CAP NUM SCRI
start
5:49 PM

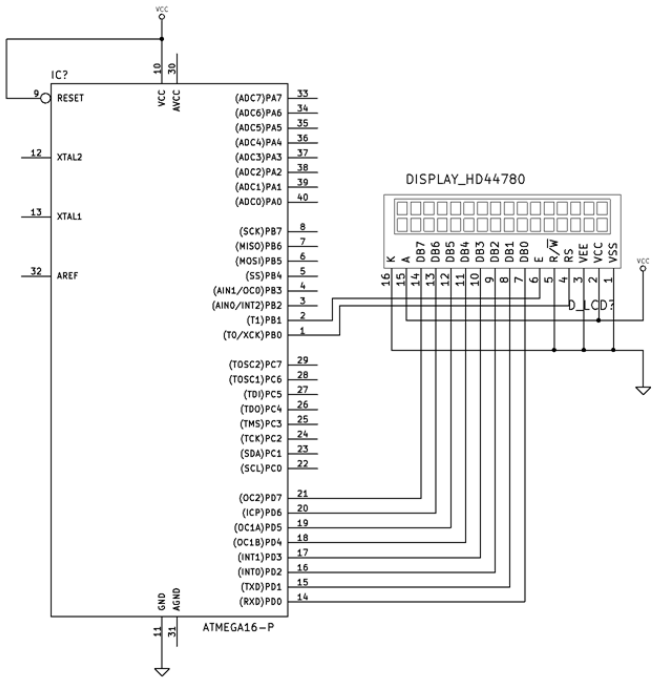

AVR Studio
Welcome to AVR Studio 4
Create new project
Project type:
Atmel AVR Assembler
AVR GCC
Project name:
Create initial file
Create folder
Initial file:
.asm
Location:
C:\Program Files\Atmel\AVR Tools\AvrStudio4
Ver 4.13.557
<< Back
Next >>
Finish
Cancel
Help
Message
Build
Find in Files
Breakpoints and Tracepoints
CAP NUM SCRI
start
5:51 PM

کدهای زیر را در پنجره باز شده وارد کرده و در نهایت کلید F7 را بزنید:

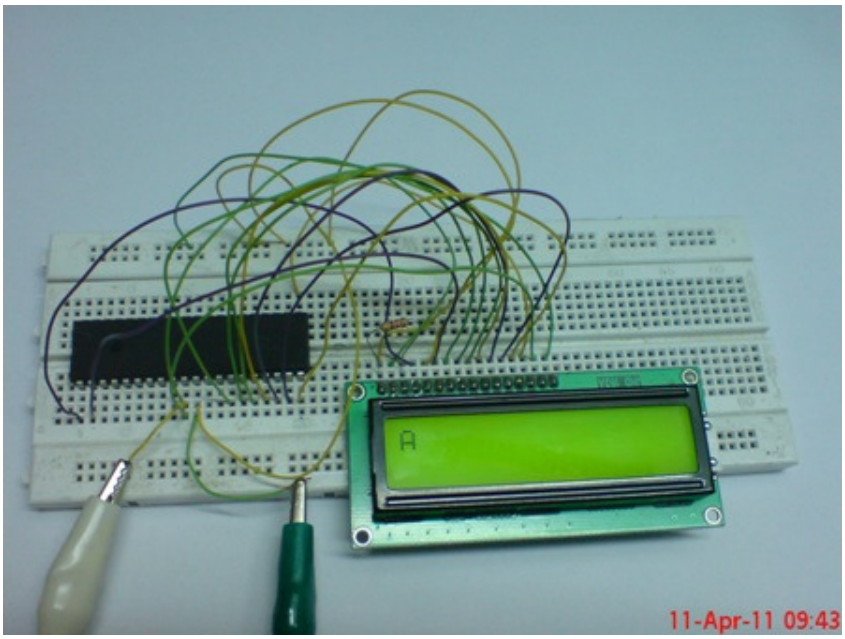
```
;LCD prog by Behnam Zakizadeh @ 10.April.2011 (AVR64.com)
.include "m16def.inc"
.cseg
.org 0000
rjmp start
;Interrupt vectors reserved area
.org 0040
start:
ldi   r20,byte1(RAMEND)
out   SPL,r20
ldi   r20,byte2(RAMEND)
out   SPH,r20
call delay
ldi r20,0xff
out ddrd, r20
sbi ddrb,0
sbi ddrb,1
cbi portb,0
sbi portb,1
ldi r20, 0x38
out portd, r20
cbi portb,1
sbi portb,1
call delay
ldi r20, 0x0c
out portd, r20
cbi portb,1
sbi portb,1
call delay
sbi portb,0
ldi r20, 'A'
out portd, r20
cbi portb,1
sbi portb,1
end:
rjmp end

delay:
ldi r20,0x40
loop2: ldi r21,0xff
loop3: dec r21
     brne loop3
     dec r20
     brne loop2
ret
```

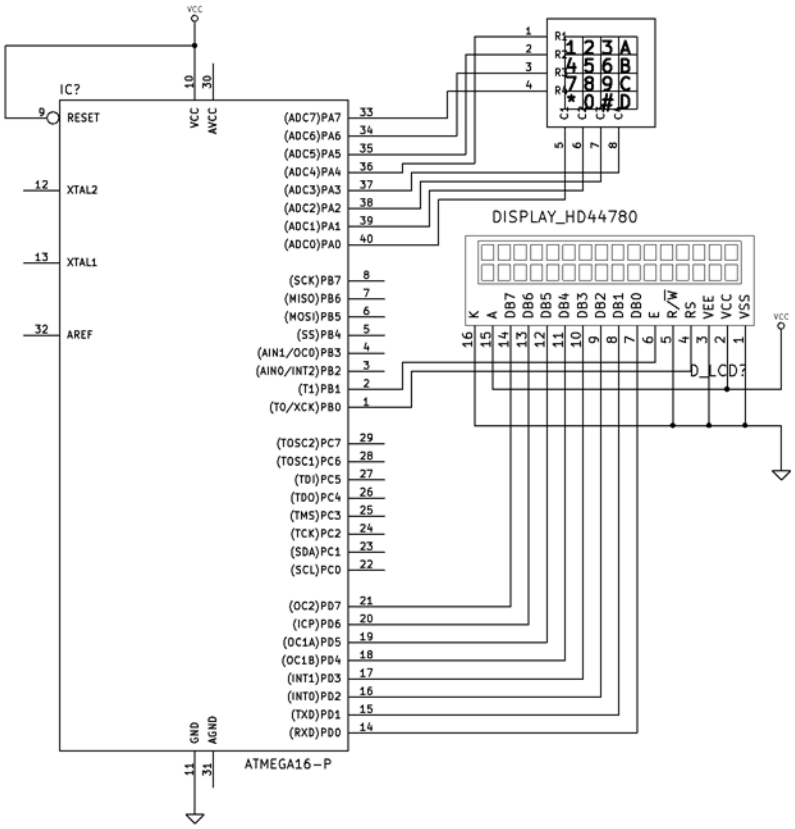
پس ایجاد فایل هگز و آپلود آن در میکروی ATmega16 با فرکانس داخلی ۱ مگاهرتز و بستن سخت افزار زیر خروجی تصویر پایین را مشاهده خواهید نمود. (برای پروگرم کردن فایل هگز می توانید یک فایل خالی در بسکام ایجاد کرده و کلید F4 را برای نمایش منوی پروگرمر کلیک کنید، سپس روی دکمه Chip identify (سمت راست نام آی سی) کلیک نمایید تا میکرو مجدداً شناسایی و تایید شود، سپس روی علامت پوشه کلیک کرده و در پنجره باز شده در قسمت File type گزینه Binary را به Intel Hex تغییر دهید و فایل هگز ایجاد شده توسط AVR Studio را بارگزاری و آپلود نمایید).



نتیجه :



در این برنامه اسمبلی مانند جلسه قبل فقط پایه های میکرو را ۰ و ۱ کرده ایم ولی اینبار به جای روشن و خاموش کردن یک دیود نورانی، هدفمندتر عمل کرده و با مطالعه در مورد پروتکل نمایشگر کریستال مایع مبادرت به راه اندازی آن نموده ایم. اصولاً با مطالعه در خصوص پروتکل ارتباطی هر نمایشگری مانند LCD موبایل و ... می توان به همین صورت آن را راه اندازی نمود. تا اینجا فقط نحوه نوشتن در پورت را یاد گرفتیم، در قسمت بعد تصمیم داریم با نحوه خواندن از پورت آشنا شده و برای کاربردی کردن این مبحث اعداد خوانده شده از کی پد ۴X۴ را روی نمایشگر نشان دهیم. در جلسات قبل (مباحث بسکام) با کی پد ماتریسی آشنا شدیم و از بیان مجدد ساختمان داخلی آن خودداری می کنیم. برای کار با این کی پدها بایستی سطرها را به ترتیب ۰ کرده و مقدار ستون ها را بخوانیم تا در صورتی که کلیدی فشرده شده بود، بیت متناظر با آن کلید ۰ شده و بر اساس آن تصمیم گیری کنیم. البته این کار به صورت برعکس نیز امکان پذیر است؛ یعنی می توان ستون ها را ۰ کرده و سطرها را خواند. (برای استفاده از مقاومتهای Pull-up از ۰ بجای ۱ استفاده می کنیم). برای شروع شماتیک زیر را ببندید (اگر خط بالا کاملاً سیاه بود پایه شماره ۳ نمایشگر را با مقاومت ۱ الی ۳ کیلو اهم به زمین وصل کنید) :



کدهای زیر را در Studio وارد کرده و F7 را فشار دهید:

```
;LCD prog by Behnam Zakizadeh @ 10.April.2011 (AVR64.com)
.include "m16def.inc"
.cseg
.org 0000
rjmp start

.org 0040
start:
ldi   r20,byte1(RAMEND)
out   SPL,r20
ldi   r20,byte2(RAMEND)
out   SPH,r20
call delay
ldi r20, 0b11110000 ;[7-4:Out-Rows] [3-0:In-Cols]
out ddra, r20
ldi r20, 0b11111111 ;Active Cols Pullup
out porta, r20
ldi r20,0xff
out ddrd, r20
sbi ddrb,0
sbi ddrb,1
cbi portb,0
sbi portb,1
ldi r20, 0x38
out portd, r20
cbi portb,1
sbi portb,1
call delay
ldi r20, 0x0c
out portd, r20
cbi portb,1
sbi portb,1
call delay
scan:
cbi portb,0
ldi r20, 0x01
out portd, r20
cbi portb,1
sbi portb,1
call delay
sbi portb,0
call keypad
;ldi r20, 'j'
out portd, r20
cbi portb,1
sbi portb,1
call delay_1s
call scan

end:
rjmp end

delay:
ldi r20,0x40
```

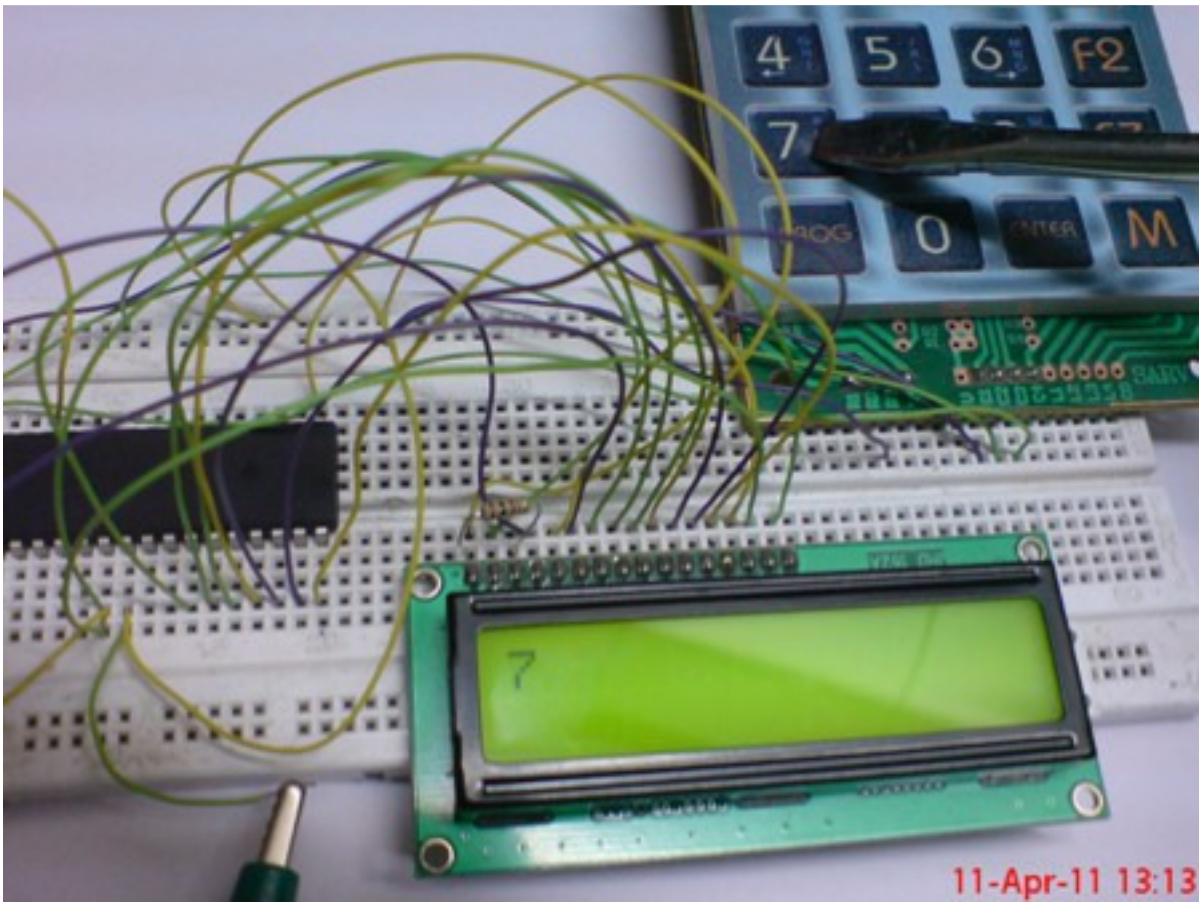
```
loop2: ldi r21,0xff
loop3: dec r21
     brne loop3
     dec r20
     brne loop2
ret

keypad:
ldi r20, 'c'
cbi porta,4
nop
     sbic pina,0
     jmp Next
     ldi r20, '1'
     Next:
     sbic pina,1
     jmp Next2
     ldi r20, '2'
     Next2:
     sbic pina,2
     jmp Next3
     ldi r20, '3'
     Next3:
     sbic pina,3
     jmp Next4
     ldi r20, 'A'
     Next4:
sbi porta,4
cbi porta,5
nop
     sbic pina,0
     jmp Next5
     ldi r20, '4'
     Next5:
     sbic pina,1
     jmp Next6
     ldi r20, '5'
     Next6:
     sbic pina,2
     jmp Next7
     ldi r20, '6'
     Next7:
     sbic pina,3
     jmp Next8
     ldi r20, 'B'
     Next8:
sbi porta,5
cbi porta,6
nop
     sbic pina,0
     jmp Next9
     ldi r20, '7'
     Next9:
     sbic pina,1
     jmp Next10
     ldi r20, '8'
     Next10:
```

```
     sbic pina,2
     jmp Next11
     ldi r20, '9'
     Next11:
     sbic pina,3
     jmp Next12
     ldi r20, 'C'
     Next12:
sbi porta,6
cbi porta,7
nop
     sbic pina,0
     jmp Next13
     ldi r20, '*'
     Next13:
     sbic pina,1
     jmp Next14
     ldi r20, '0'
     Next14:
     sbic pina,2
     jmp Next15
     ldi r20, '#'
     Next15:
     sbic pina,3
     jmp Next16
     ldi r20, 'D'
     Next16:
sbi porta,7
ret

delay_1s:
ldi r16, 1
loop11: ldi r17,0xff
loop12: ldi r18,0xff
loop13: dec r18
     brne loop13
     dec r17
     brne loop12
     dec r16
     brne loop11
ret
```

اگر همه چیز به درستی پیش رفته باشد پس از پروگرم کردن میکرو و نگه داشتن کلید شماره ۷ خروجی زیر را خواهید داشت، با رها کردن کلید کاراکتر C نمایش داده می شود. می توان با استفاده از این روش همانند کیبرد کامپیوتر و رویدادهای VB به کی برد حالت Keydown و Keypress و Keyup داد.



در این برنامه روتین keypad به ترتیب پین های متصل به ستونهای کی پد را صفر کرده و کل سطر مربوط به ستون جاری را بررسی می کند، اگر یکی از کلیدهای ستون جاری وصل شده باشد پایه سطر متصل به این ستون ۰ شده و عدد مربوطه به آن کلید در رجیستر r20 ذخیره می شود. به محض برگشت از روتین کی پد رجیستر r20 در نمایشگر نشان داده شده و پس از یک ثانیه تاخیر صفحه با دستور 0x01 پاک می شود و مجدداً روتین کی پد فراخوانی می گردد. در این برنامه برای خواندن و مقایسه پین ها از دستور sbic استفاده شده و معنای آن "اگر بیت مورد نظر clear یا ۰ بود از دستور بعدی صرفنظر کن و بپر" می باشد. اگر هم ۰ نبود دستور بعدی اجرا می شود که پرش به لیبل بعدی است.

تا اینجا آموختیم که چگونه با پورت ها ارتباط برقرار کنیم و آنها را کنترل کرده و مقادیر موجود در آنها را بخوانیم. در این قسمت در مورد متغیرها و نحوه اعلان آنها در اسمبلی صحبت می کنیم تا بتوانیم اعدادی را که از کی پد بدست می آوریم در فضای SRAM میکرو ذخیره نماییم. در حالت عادی در محیط اسمبلی با دستور .byte فقط قادر به تعریف بایت هستیم. برای تعریف متغیرهای حجیم تر می توانیم از .byte 2 یا بالاتر استفاده کنیم. مثلا دستور x: .byte 4 متغیر x را از نوع ۴ بایتی یا Long تعریف می کند. برای تعریف متغیرهای بیتی باید از رجیسترها استفاده کنیم (یا از بیت های بایت در SRAM). سایر متغیرهای مختلف مانند Int و String و آرایه ها را نیز بایستی به طریقی با ترکیب همین بایت ها بسازیم. در مورد متغیرهای اعشاری و ممیز شناور با دقت بالا مثل Single، Double، Float و نحوه اعلان و محاسبات آنها و نیز نحوه نمایش علامت منفی مقالات و کتابهای فراوانی وجود دارد که مطالعه کتاب مدارهای منطقی موریس مانو در این زمینه پیشنهاد می شود. ما فعلا فقط به صورت بالا از دستور .byte برای اعلان یک ویا رشته ای از بایت ها استفاده می کنیم تا بتوانیم متغیرهای خود را در فضای SRAM میکرو به صورت موقتی ذخیره نماییم.

در این قسمت برنامه ای روی سخت افزار قبل می نویسیم که عدد ۷ را در یک متغیر ذخیره کرده و آن را نمایش می دهد:

```
;LCD prog by Behnam Zakizadeh @ 10.April.2011 (AVR64.com)
.include "m16def.inc"
.dseg
n: .byte 1
.cseg
.org 0000
rjmp start

.org 0040
start:
ldi   r20,byte1(RAMEND)
out   SPL,r20
ldi   r20,byte2(RAMEND)
out   SPH,r20
call delay
ldi r20,0xff
out ddrd, r20
sbi ddrb,0
sbi ddrb,1
cbi portb,0
sbi portb,1
ldi r20, 0x38
```

```
out portd, r20
cbi portb,1
sbi portb,1
call delay
ldi r20, 0x0c
out portd, r20
cbi portb,1
sbi portb,1
call delay
cbi portb,0
ldi r20, 0x01
out portd, r20
cbi portb,1
sbi portb,1
call delay
sbi portb,0
ldi r21,'7'
sts n,r21
lds r22,n
mov r20,r22
out portd, r20
cbi portb,1
sbi portb,1

end:
rjmp end

delay:
ldi r20,0x40
loop2: ldi r21,0xff
loop3: dec r21
     brne loop3
     dec r20
     brne loop2
ret
```

در این برنامه متغیری به نام n تعریف کرده و با دستور sts کاراکتر ۷ را در آن ذخیره کرده ایم (در واقع در آدرسی از فضای SRAM که لیبل n به آن اشاره می کند). سپس با دستور lds آن را بازیابی کرده و با دستور mov به رجیستر r20 پاس داده ایم تا بر روی نمایشگر نوشته شود. به همین ترتیب می توان اعداد و کاراکتر ها را در فضای SRAM ذخیره نمود. در جلسه بعد در مورد وقفه ها بحث می کنیم تا مبحث بردار وقفه را قدری شفاف تر کنیم و منظور از خطوط .org ابتدای برنامه مشخص شود.

منابع:

[1] مکنزی اسکات، میکروکنترلر 8051، انتشارات ناقوس – انتشارات میرزایی، ۱۳۸۴

[2] شهریاری شیرزاد، مرجع علمی کاربردی میکروکنترلرهای AVR, PIC, MCS-51 انتشارات پرتونگار، ۱۳۸۵

[3] *www.**avr-asm-tutorial**.net*

ادامه دارد...

پایان جلسه دوم اسمبلی (جلسه هفتم آموزش AVR)

مولف: بهنام زکی زاده

www.avr64.com

۲۳ فروردین ۱۳۹۰

آخرین ویرایش: ۱۴ مهر ۱۳۹۲ ✓