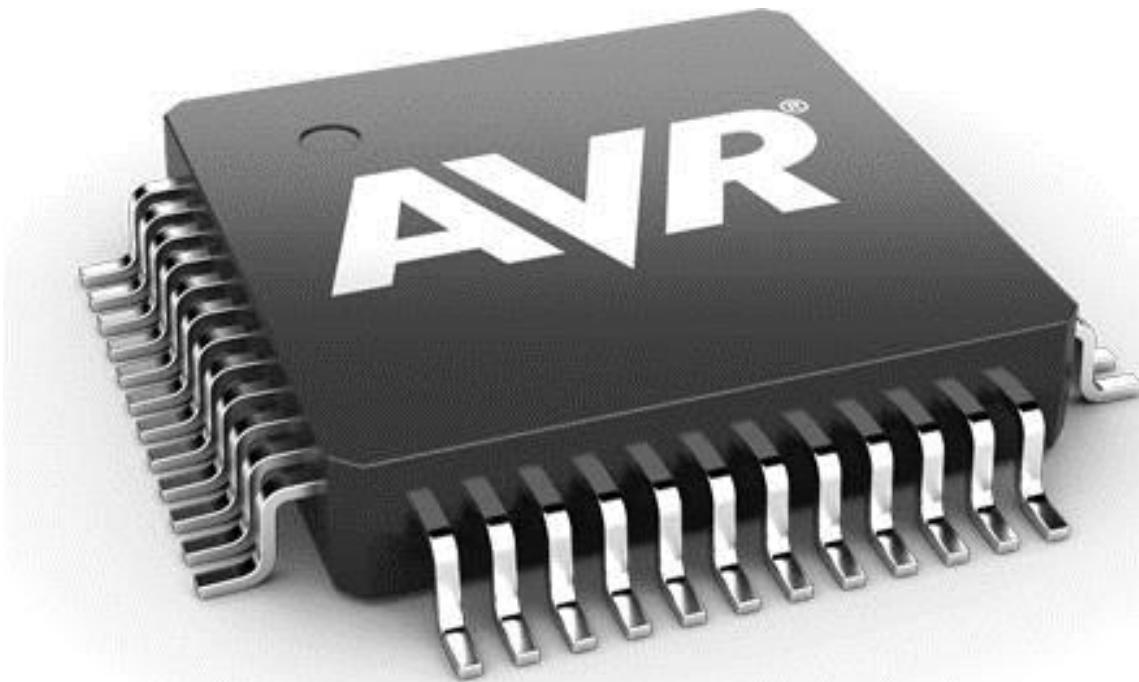
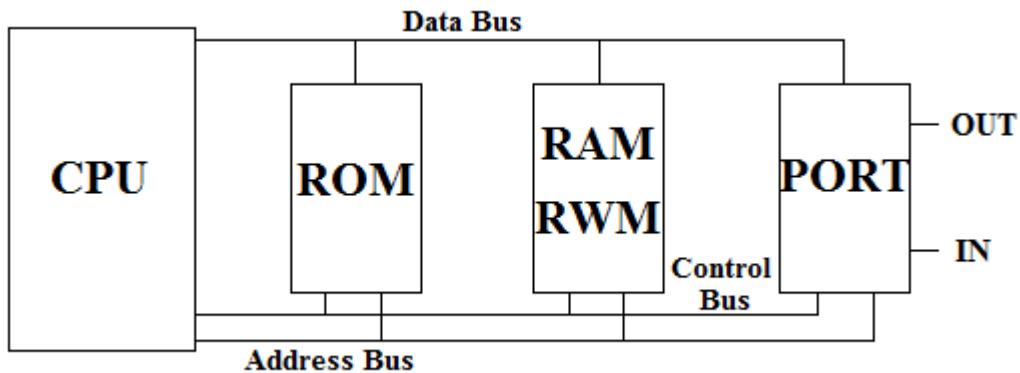


موقوفات



مدرس: رضا فتاحی

بلوک دیاگرام یک سیستم میکروپروسسوری می:



Central Processing Unit : CPU

در هر سیستم یک CPU وجود دارد که کار کنترل و پردازش Data را انجام می دهد.

Read Only Memory : ROM

حافظه ای است فقط خواندنی که اطلاعات راه اندازی سیستم که به آن سیستم عامل OS نیز گفته می شود درون آن قرار دارد. (Operating System)

Read Write Memory (RWM): RAM

حافظه ای با قابلیت خواندن و نوشتن که اطلاعات میانی سیستم درون آن قرار می گیرد.

PORT : در هر سیستم تعدادی PORT وجود دارد که وظیفه آنها ورود و خروج اطلاعات است.

Bus : در هر سیستم سه نوع Bus وجود دارد: Data, Address, Control

• مشخص می کند با کدام حافظه یا پورت کار داریم . Address

- مشخص می کند چه کاری داریم Control.
- توسط این خطوط اطلاعات بین CPU و بقیه مدار منتقل می شود Data.

توضیح و ترجمه بخش هایی از سه صفحه اول دیتا شیت AVR (ATMEGA 32)

نکته: هنگامی که گفته می شود یک میکرو کنترلر ۸ بیتی است؛ یعنی می تواند عملی را روی دو عدد ۸ بیتی انجام دهد. حال اگر مثلا دو عدد ۱۶ بیتی داشته باشیم، باید آن را در دو مرحله انجام دهیم.

به این پارامتر طول کلمه CPU نیز گفته می شود. (Word length)

این میکرو دارای ساختار RISC است:

RISC: Reduced Instruction Set Computer

CISC: Complex Instruction Set Computer

RISC: دستورهای ساده تر، برنامه نویسی مشکل، سرعت بالا (AVR)

CISC: دستورهای پیچیده، برنامه نویسی ساده تر، سرعت پایین (8051)

رجیسترها: در این میکرو ۳۲ رجیستر همه منظوره (General - purpose register) ۸ بیتی (R_{31}) تا

۶۴ رجیستر تک منظوره (special purpose register) (R_0) وجود دارد؛ وظیفه رجیسترها IO

تنظیم قسمت های مختلف میکرو است.

برای مثال:

A : رجیستر تنظیم پورت Data Direction Register : DDRA

A : رجیستر خروجی پورت PORTA

A : رجیستر ورودی پورت PINA

Timer Counter Control Register 0 : TCCR0 تنظیم کننده تایмер کانتر صفر

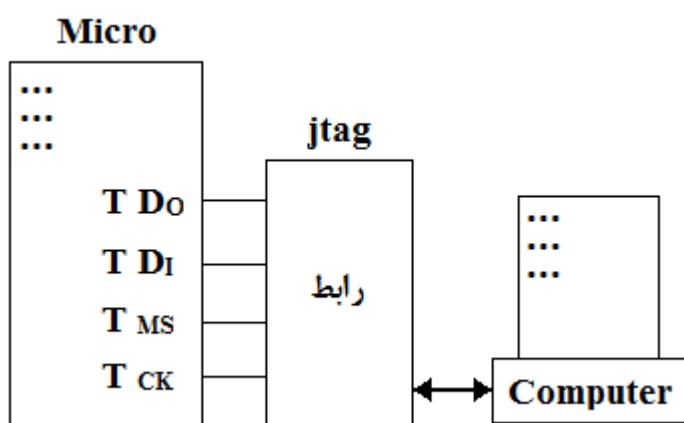
سرعت: این IC می‌تواند تحت فرکانس 16 MHZ MIPS 16 با سرعت 16 MIPS عملیات را انجام دهد؛ یعنی در هر ثانیه 16 میلیون دستور را اجرا نماید.

MIPS = Million Instruction Per Second

حافظه: در این میکرو سه نوع حافظه به شرح زیر وجود دارد:



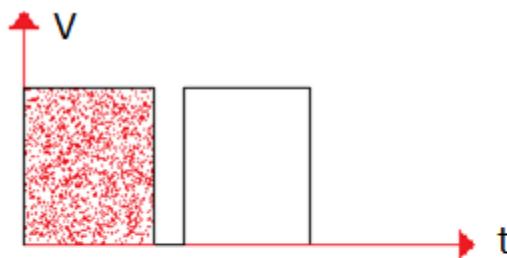
JTAG: یک واسطه بین میکرو و کامپیوتر است که می‌توان از طریق آن علاوه بر برنامه ریزی میکرو، برنامه داخل میکرو را خط به خط اجرا و در صورت نیاز آن را تصحیح کرد. **Joint Test Action Group**.



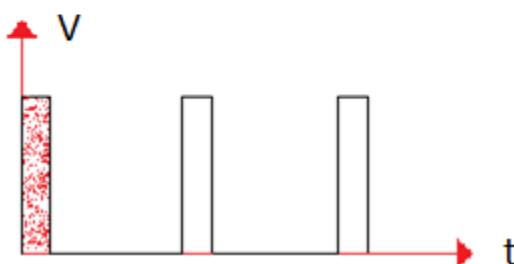
تایمر و کانتر: در این میکرو دو تایмер کانتر ۸ بیتی و یک ۱۶ بیتی وجود دارد.

(Real Time Counter) RTC : اگر بخواهیم زمان سنجی واقعی داشته باشیم، لازم است که بتوانیم پالس یک ثانیه را تولید کنیم؛ در این میکرو واحدی به نام RTC وجود دارد که می‌توان توسط آن پالس یک ثانیه و به تبع آن دقیقه و ساعت را تولید کرد.

(Pulse Width Modulator) PWM : یکی از راه‌های کنترل بعضی از دستگاه‌ها، کنترل انرژی است که به آن‌ها می‌رسد. توسط تغییر در عرض پالس، می‌توانیم انرژی و در نتیجه، آن دستگاه را کنترل کنیم. برای مثال اگر دو موج هم دامنه و هم فرکانس زیر را به دو موتور مشابه بدهیم:



سرعت بیشتر به خاطر انرژی بیشتر



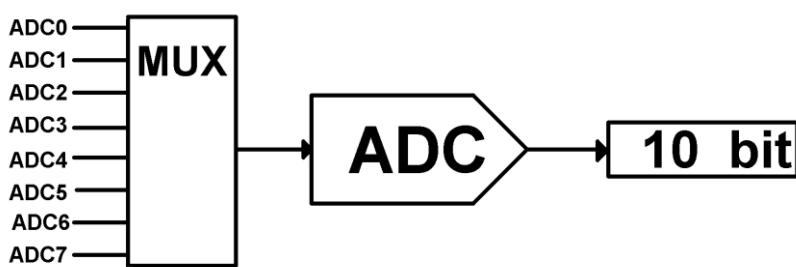
سرعت کمتر به خاطر انرژی کمتر

در این میکرو چهار کanal PWM پیش‌بینی شده است، پایه‌های OC2, OC1B, OC1A, OC0.

Analog to Digital Converter :ADC تمام کمیت‌های اطراف ما مقادیری آنالوگ هستند، مانند

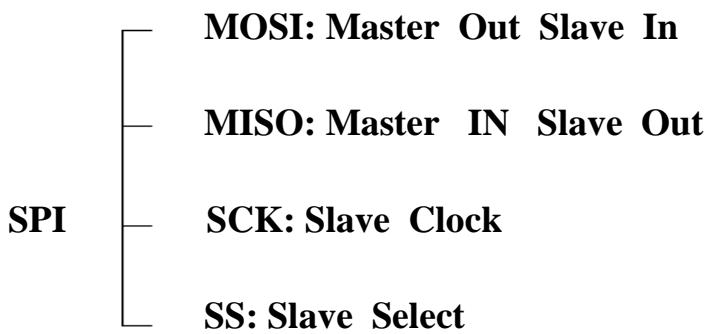
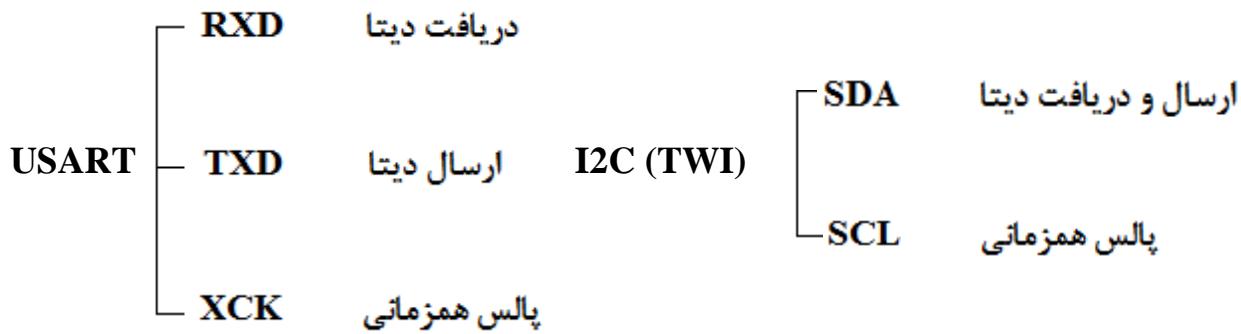
دما، فشار، رطوبت و... اگر بخواهیم آن‌ها را اندازه‌گیری و پردازش کنیم، لازم است که ابتدا به یک

مقدار دیجیتال تبدیل شوند و سپس پردازش گردند. در این میکرو یک مبدل ۸ کاناله و ۱۰ بیتی در



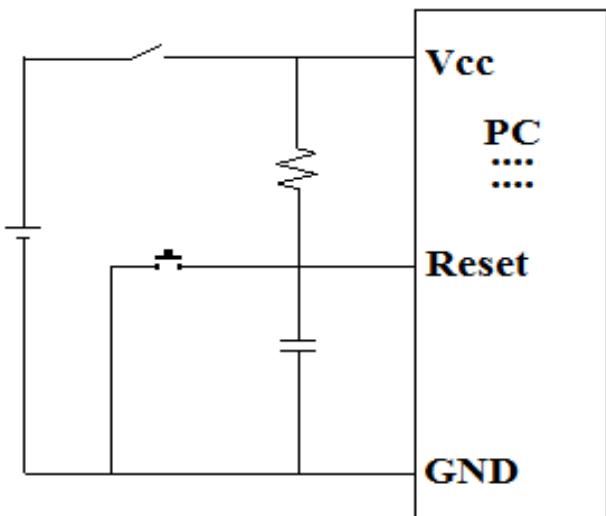
نظر گرفته شده است.

ارتباط: همواره یکی از مسائلی که با آن درگیر خواهد بود، ارتباط دو یا چند دستگاه است تا بتوان اطلاعات را از یک نقطه به نقطه دیگر منتقل کرد. در این میکرو سه نوع ارتباط پیش بینی شده است:



: هرگاه سیستمی هنگ کند برای اینکه دوباره به کار بیفتند لازم است که آن را ریست کنیم. (WD) وظیفه دارد که Cpu را تحت نظر داشته باشد، تا در صورتی که به هر علتی هنگ کرد آن را ریست نماید. این بخش یک تایмер است که حداقل تا ۲ ثانیه تنظیم می شود اگر به هر علتی میکرو هنگ کند و نتواند WD را ریست کند بعد از زمان تنظیم شده WD میکرو را ریست می کند.

Reset : تمام میکروها لازم است که پس از روشن شدن ریست شوند، تا کار محول شده را از خط اول برنامه شروع کنند. این میکرو طوری طراحی شده که هر گاه آن را روشن کنید، میکرو را ریست می کند، که به آن Power On Reset گفته می شود.



هر گاه ریست اتفاق می افتد، شمارنده برنامه

برابر صفر PC (Program Counter)

می شود؛ در نتیجه برنامه حتماً از خط اول

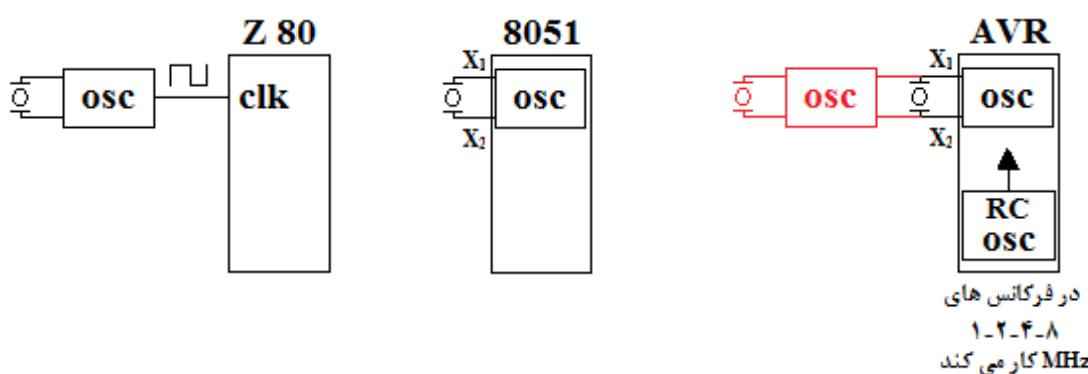
شروع خواهد شد.

Brown-Out Detection : ساز و کاری است

که ولتاژ تغذیه میکرو را پایش می کند، تا در

صورتی که از مقدار مشخصی کمتر شد، آن را ریست کند.

Oscillator : می دانیم که پردازنده ها از تعداد زیادی مدارهای ترتیبی و ترکیبی ساخته شده اند. برای هماهنگی بین قسمت های مختلف نیاز به یک پالس ساعت داریم که همه قسمت ها را با هم هماهنگ کنند. برای این کار روش های زیر وجود دارد.



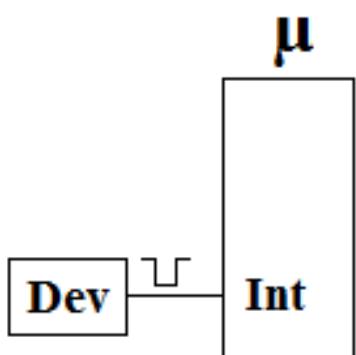
همانطور که در شکل ها دیده می شود، برای تهیه این پالس سه روش وجود دارد:

- اسیلاتور خارجی
- اسیلاتور داخلی با کریستال خارجی
- اسیلاتور RC داخلی (در میکروی نو روی فرکانس داخلی 1mhz تنظیم است)

وقفه (Interrupt): برای سرویس دهی به هر دستگاه جانبی دو روش وجود دارد:

- سرکشی Polling
- وقفه Interrupt

در روش اول Cpu موظف است در فاصله زمانی های مشخص به دستگاه جانبی سرکشی کند، تا در صورت نیاز به آن دستگاه سرویس لازم را ارائه نماید. در این روش وقت زیادی از Cpu تلف می شود.



اما در روش وقفه دستگاه هر موقع نیاز داشت با ارسال سیگنال وقفه در خواست سرویس می کند؛ میکرو کار خود را قطع و به آن دستگاه سرویس می دهد، در نتیجه وقتی تلف نمی شود. در این میکرو سه وقفه خارجی وجود دارد؛ پایه های Int2, Int1, Int0 .

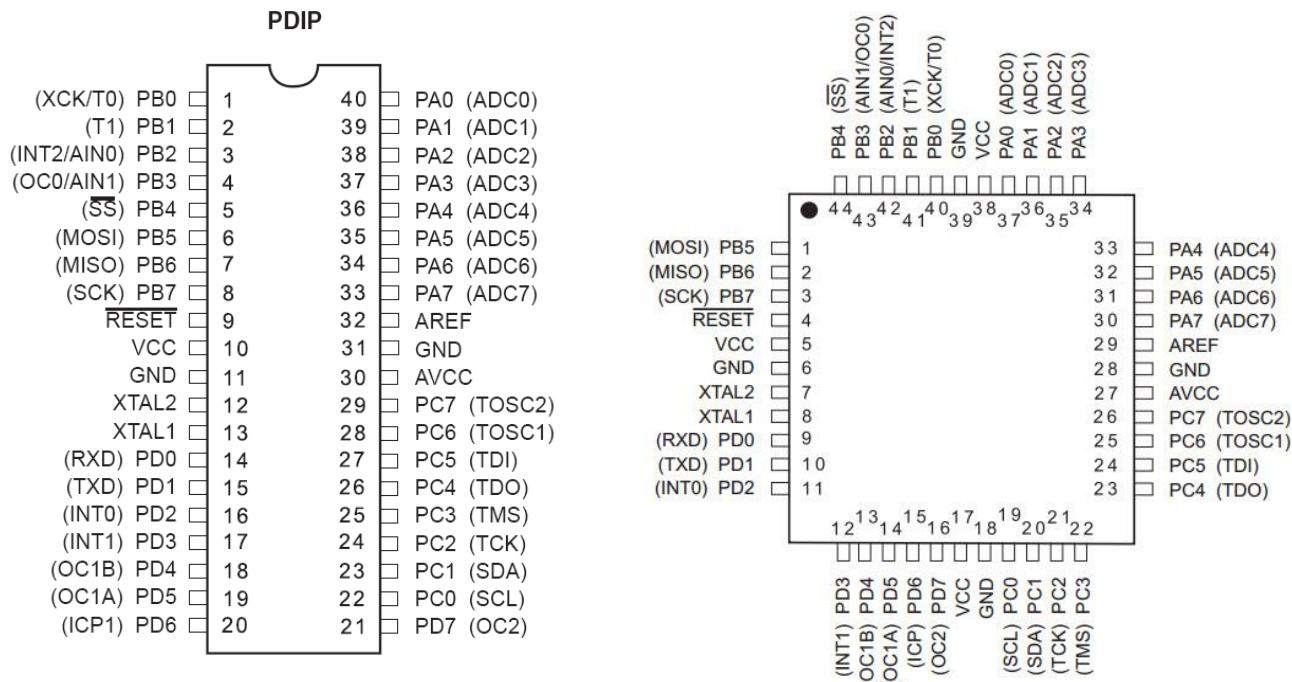
Sleep : برای ساخت دستگاه های پرتاپل لازم است که تا حد امکان انرژی کمتری مصرف شود تا طول عمر باتری بیشتر گردد؛ لازم است که میکرو بتواند در هنگام بیکاری به حالت (Sleep) برود تا انرژی کمتری مصرف کند و طول عمر باتری افزایش یابد. در این میکرو شش روش sleep پیش بینی شده است

:Packing بسته بندی

این میکرو در دو بسته بندی TQFP و PDIP عرضه می شود.

PDIP: Plastic Dual Inline Package

TQFP: Thin Quad Flat Package



نام محصول : این میکرو در ابتدای تولید با دو نام ATmega32 و ATmega32L به بازار عرضه شد.

مزیت نوع L ولتاژ کار 2.7v تا 5.5v و اشکال آن حداقلر کانس کار 8Mhz بود.

مزیت نوع معمولی حداقلر کانس کار 16Mhz و اشکال آن ولتاژ کار 4.5v تا 5.5v بود.

کارخانه سازنده در ATmega32A هر دو مزیت را قرار داده است. هم اکنون ، این محصول در بازار

وجود دارد.

پایه ها:

پایه های این میکرو تک وظیفه ای ، دو یا سه وظیفه ای می باشد.

PA0~PA7 PB0~PB7 PC0~PC7 PD0~PD7 : پایه های پورت ها

T1 و T0 : پالس ورودی به تایمر/کانتر صفر و یک.

AIN0~AIN1 : پایه های ورودی مقایسه کننده آنالوگ.

INT0,1,2 : پایه های ورودی اینترپت خارجی .

OC0,OC1A,OC1B,OC2 : چهار کanal خروجی PWM

SS,MOSI,MISO,SCK : چهار پایه مربوط به ارتباط SPI

slave فعال کننده SS: slave select

دیتای خارج شده از **slave** را به **master** وارد می کند. **MISO: master in slave out**

دیتای خارج شده از **master** را به **slave** وارد می کند. **MOSI: master out slave in**

پالس همزمانی را از **master** به **slave** منتقل می کند. **SCK: slave clock**

هر گاه این پایه صفر شود میکرو باز نشانی و برنامه از ابتدا اجرا می شود. **RESET**

با اعمال ولتاژ بین 2.7v تا 5.5v به این دو پایه میکرو تغذیه و کار می کند. **VCC,GND**

پایه های اتصال کریستال خارجی تا فرکانس **16mhz** . **XTAL1,2**

این سه پایه مربوط به واحد **USART RXD,TXD,XCK** می باشند.

. **uart** پایه دریافت دیتا به واحد **RXD: Receive data**

. **uart** پایه ارسال دیتا از واحد **TXD: Transmit data**

پایه ارسال و دریافت پالس همزمانی واحد **XCK: Clock** . **uart**

ICP1: (Timer/Counter1 Input Capture Pin) با تحریک این پایه عدد داخل تایمر1 در رجیستر **ICR1** ذخیره می شود.

(Two Wire Interface /Inter-Integrated Circuit) **TWI** یا **I2C** : پایه های ارتباط **SCL,SDA**

: خط ارسال و دریافت دیتا **Serial Data Line (SDA)**

: خط انتقال پالس همزمانی **Serial Clock Line (SCL)**

JTAG : چهار پایه مربوط به ارتباط **TDI,TDO,TMS,TCK**

نکته: در میکرو نو **JTAG** فعال است در نتیجه چهار پایه از **PORTC** در اختیار پورت نیست.

TOSC1,2 : برای استفاده از **RTC** داخلی میکرو ، باید یک کریستال با فرکانس **32768hz** به این دو پایه وصل شود.

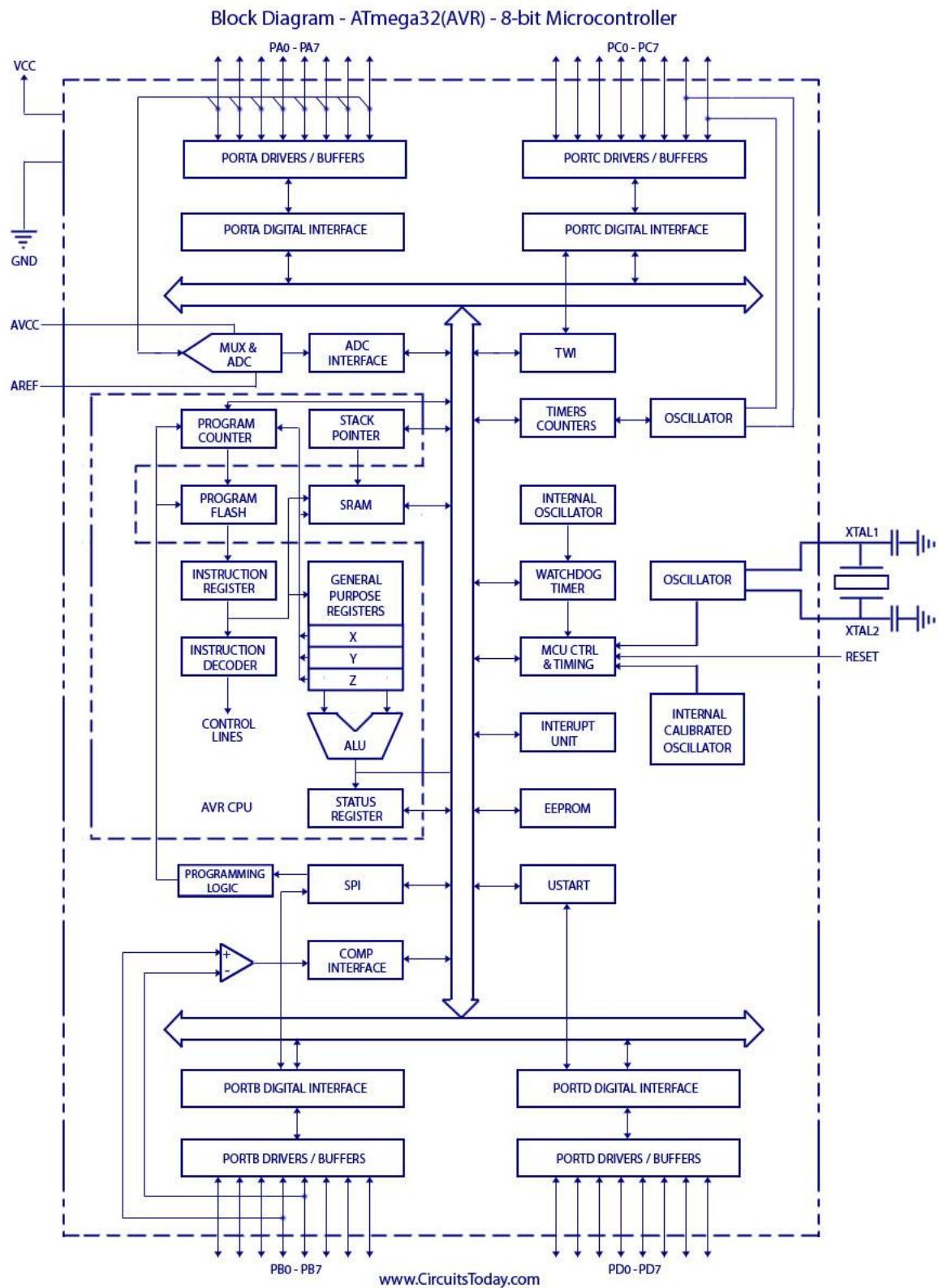
ADC : پایه های مربوط به مبدل آنالوگ به دیجیتال **AVCC,GND,AREF**

: تغذیه واحد **ADC** را تامین می کنند. **AVCC,GND**

AREF : ولتاژ مرجع خارجی بخش **ADC** که بین **0** تا **5v** است به این پایه وصل می شود.

ADC0~ADC7 : هشت کanal ورودی آنالوگ به بخش **ADC**.

بلوک دیاگرام داخلی AVR



نرم افزار: پس از طراحی و ساخت سخت افزار لازم است برنامه ای نوشته شود تا آن را کنترل کند. تنها زبان قابل فهم برای پردازنده ها زبان ماشین می باشد، ولی نوشتن و رفع اشکال برنامه به زبان ماشین سخت و مشکل است؛ لذا از زبان های سطح بالا مانند C استفاده می کنیم.

ساختار زبان C : همانطور که قبلا گفته شد، لازم است که برنامه ای برای میکرو نوشته شود و زبان C مناسب و ساختار آن به شکل زیر است:

```
#include (هدرفایل)
```

ماکروها

متغیرهای عمومی

توابع

```
void main (void)
```

```
{
```

```
....
```

```
....
```

```
.... برنامه
```

```
....
```

```
....
```

```
}
```

میکروی مورد نظر ما دارای ۴ پورت است و هر پورت دارای سه رجیستر به شرح زیر می باشد:

DDRX Data Direction Register	مشخص می کند پورت ورودی باشد یا خروجی برای خروجی ۰ برای ورودی ۱
PINX	رجیستر برای ورود دیتا
PORTX	رجیستر برای خروج دیتا

نوشتن اعداد در میناهای مختلف در زبان C :

در کد ویژن می توانید اعداد را در مبنای ۲، ۸، ۱۶ و ۱۰ بنویسید.

در جدول زیر اعداد ۰ تا ۱۵ در میناهای ۸,۱۶,۲,۱۰ جهت پادآوری نوشته شده است.

۱۰ مبنای	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵
۲ مبنای	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
۱۶ مبنای	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	A	B	C	D	E	F
۸ مبنای	۰	۱	۲	۳	۴	۵	۶	۷	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵	۱۶	۱۷

فرض کنید می خواهیم عدد 12 را در مبنا های مختلف به PORTA ارسال کنیم.

$$(12)_{10} = (1100)_2 = (14)_8 = (c)_{16}$$

PORTA=12; مبنای ۱۰

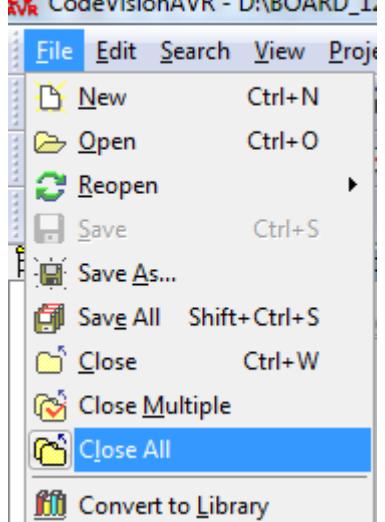
PORTA=0b1100; مینای ۲

منای، ۸ PORTA=014;

PORTA=0xC; **منای ۱۶**

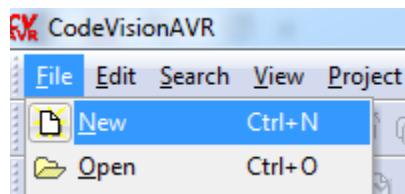
مراحل نوشتن برنامه در : Code Vision

۱. معمولاً پس از باز کردن برنامه آخرین پروژه ای که کار کرده اید



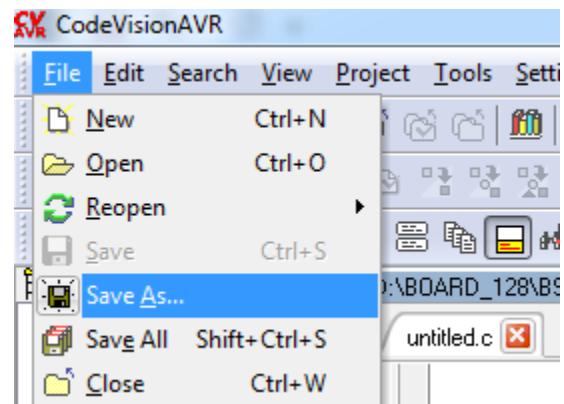
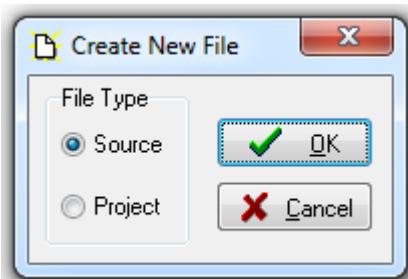
باز می شود؛ از مسیر زیر آن را بیندید. File\Close All

۲. از مسیر زیر یک فایل برای نوشتن برنامه باز و آن را با نامی مناسب و در محلی مشخص ذخیره

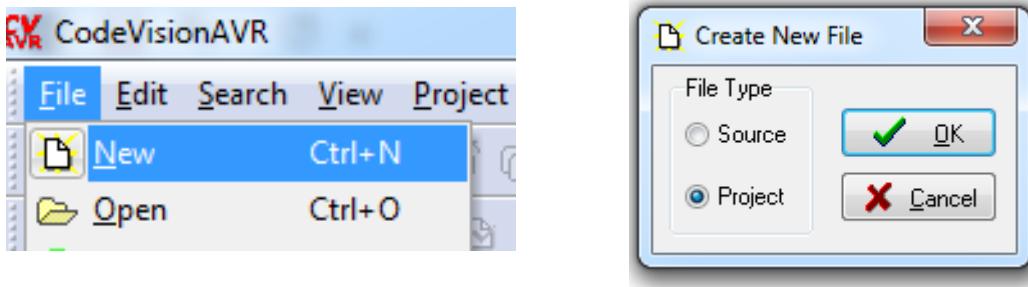


File\New\Source\Ok\File\Save As. نمایید.

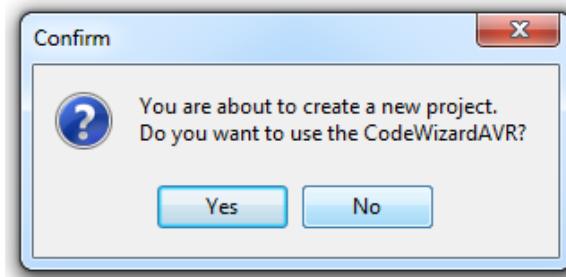
نام main برای سورس مناسب است.



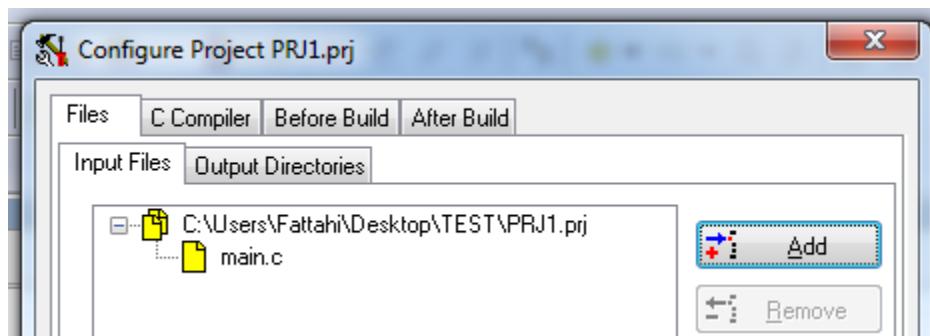
۳. از مسیر زیر یک پروژه باز کرده و ذخیره می کنیم.



۴. در پاسخ به سوال استفاده از Wizard دکمه No را می زنیم.

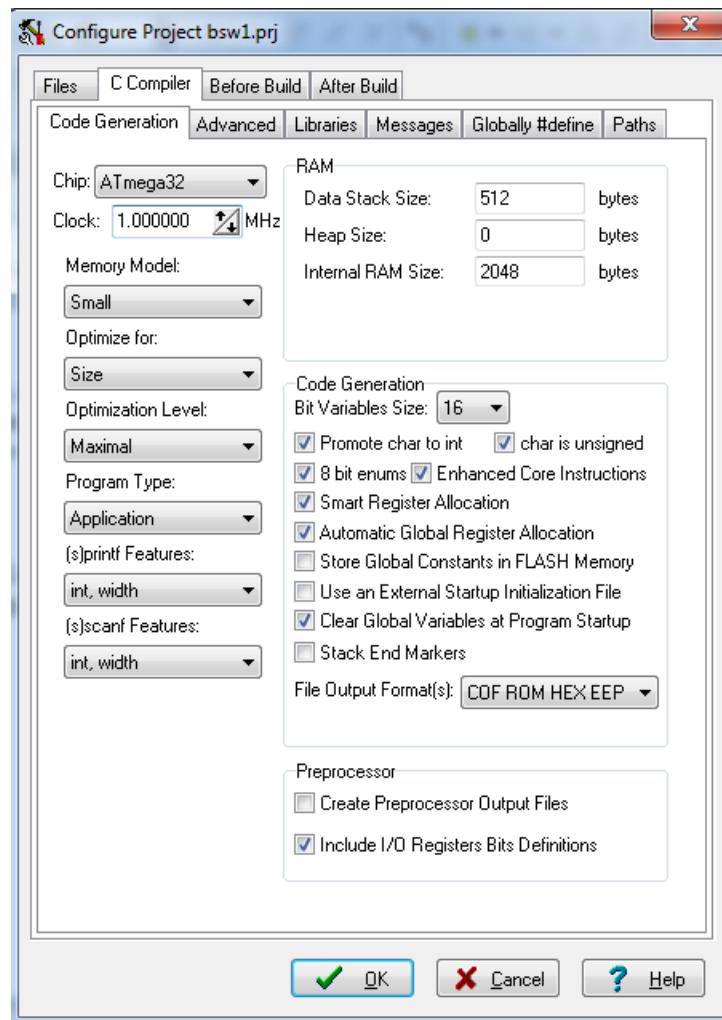


۵. پس از انتخاب نام و ذخیره کردن فایل پروژه، پنجره‌ی Configure Project باز می شود. با زدن کلید Add فایل Source مرحله قبل را به پروژه اضافه کنید.

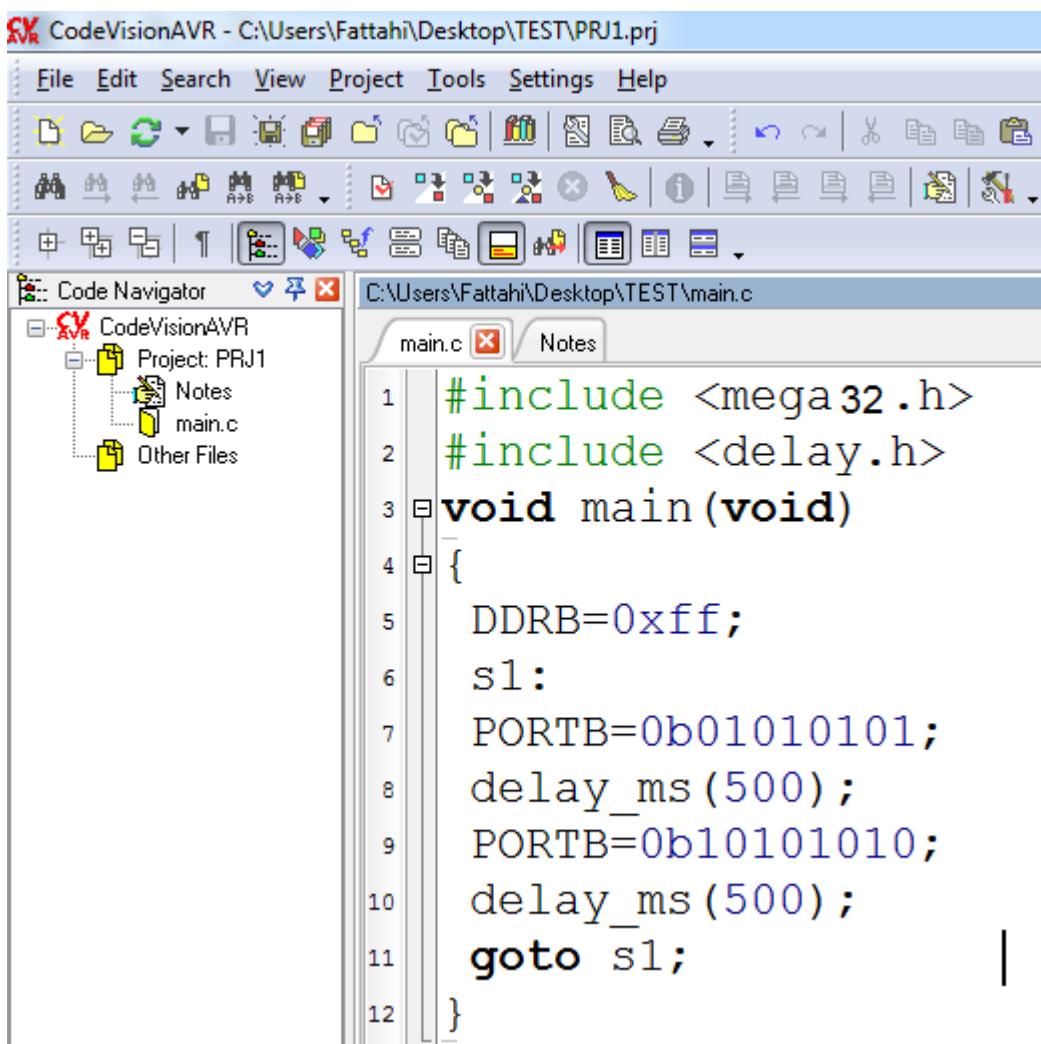


۶. در همین پنجره برگه C Compiler را باز کرده و نوع میکرو و فرکانس آن را تعیین می کنیم

؛ با زدن کلید Ok مقدمات ساختن پروژه به اتمام می رسد.



۷. متن برنامه را در فایل C می نویسیم.



The screenshot shows the CodeVisionAVR IDE interface. The title bar reads "CodeVisionAVR - C:\Users\Fattahi\Desktop\TEST\PRJ1.prj". The menu bar includes File, Edit, Search, View, Project, Tools, Settings, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The left sidebar is the "Code Navigator" showing a project structure with "Project: PRJ1" containing "main.c" and "Notes". The main editor window displays the following C code:

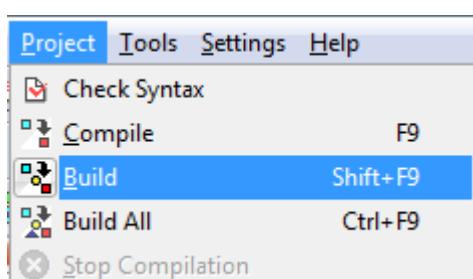
```

1 #include <mega32.h>
2 #include <delay.h>
3 void main(void)
4 {
5     DDRB=0xff;
6     s1:
7     PORTB=0b01010101;
8     delay_ms(500);
9     PORTB=0b10101010;
10    delay_ms(500);
11    goto s1;
12 }

```

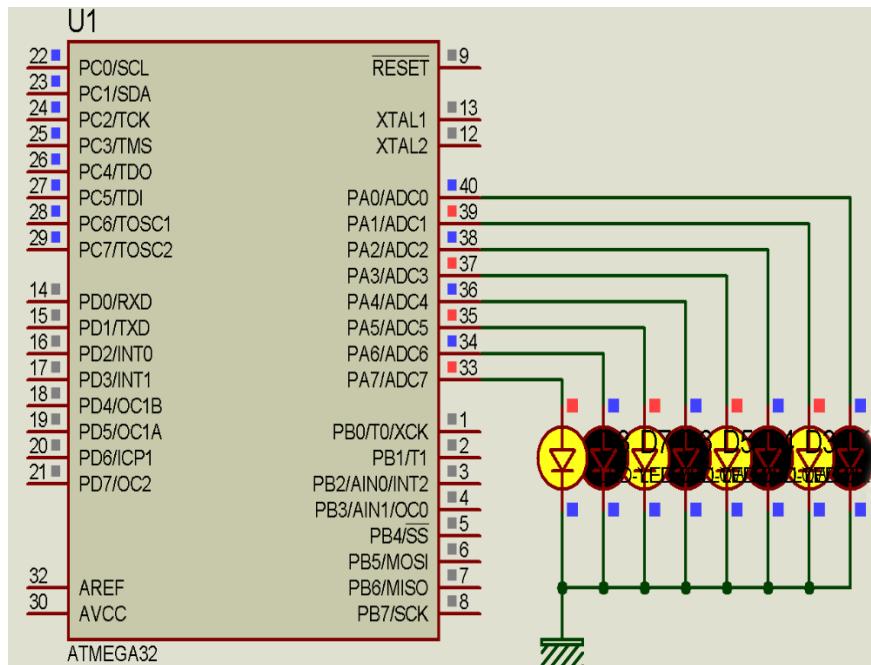
۸. از منوی Project گزینهی Build را انتخاب می کنیم

تا برنامه به زبان ماشین تبدیل و فایل HEX ساخته شود.



مثال: هشت عدد LED را به پورت A متصل کنید برنامه ای بنویسید که آن ها را یکی در میان روشن کند و نیم ثانیه در این حالت نگه دارد. سپس LED های خاموش و روشن عوض شوند و نیم ثانیه نیز در این حالت بماند و در ادامه همین روند تکرار شود.

```
#include <mega32.h>
#include <delay.h>
void main (void)
{
    DDRA=0xFF;
    S1:
    PORTA=0b01010101;
    delay_ms(500);
    PORTA=0b10101010;
    delay_ms(500);
    goto S1;
}
```

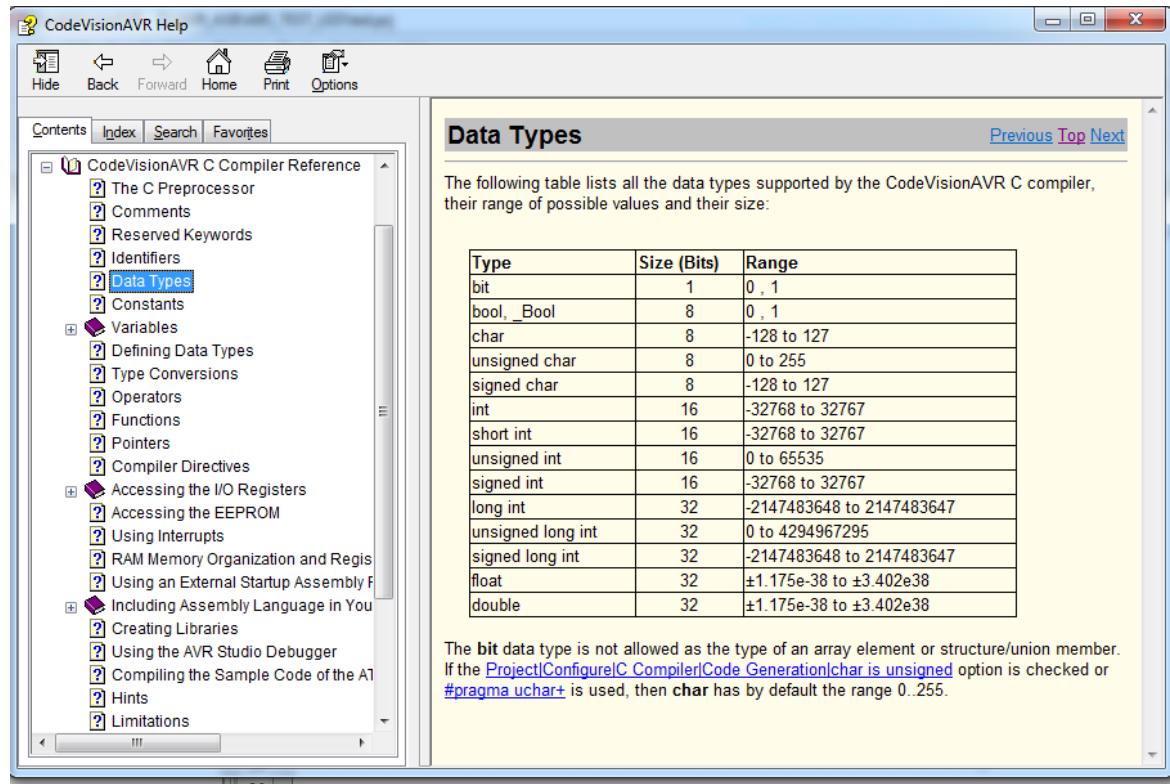


متغیر Variable : در هر زبان برنامه نویسی لازم است محل هایی از حافظه برای نگه داری اعداد، کاراکترها و رشته ها تعین گردد تا در هنگام اجرای برنامه بتوان آن ها را خواند یا نوشت. در زبان C برای تعریف یک متغیر از ساختار زیر استفاده می شود.

نوع متغير = مقدار نام متغير

```
unsigned char    a=5 ;  
int   b,c,d=2000;  
float  h=3.14;  
char   t='R';  
char   s[ ]="REZA";
```

انواع متغیرها و رنج اعداد قابل نمایش توسط آن ها در Help برنامه و در بخش Data Types در دسترس می باشد.



مثال : هشت عدد LED به PORTA متصل کنید و یک شمارنده بالا شمار بر روی آن ایجاد نمایید.

```
#include <mega32.h>
#include <delay.h>
unsigned char i=0;
void main(void)
{
    DDRA=0xFF;
    s1:
    PORTA=i;
    delay_ms(500);
    i++;
    goto s1;
}
```

دستور شرطی if : هرگاه قرار باشد دستوراتی بنابر شرایط خاص انجام شود، از دستورهای شرطی

استفاده می کنیم.

If (شرط) دستور

If (شرط)

{

....

....

.... دستورها

If (شرط)

{

.... دستورها

}

else

{

.... دستورها

}

}

مثال: بر روی LED های PORTA یک شمارنده بالا شمار ۰ تا ۹ ایجاد کنید.

```
#include <mega32.h>
#include <delay.h>
unsigned char i=0;
void main(void)
{
DDRA=0xFF;
s1:
PORTA=i;
delay_ms(500);
i++;
if (i==10) i=0;
goto s1;
}
```

مثال: بر روی LED های PORTA یک شمارنده پایین شمار ۹ تا ۰ ایجاد کنید.

```
#include <mega32.h>
#include <delay.h>
unsigned char i=9;
void main(void)
{
DDRA=0xFF;
s1:
PORTA=i;
delay_ms(500);
i--;
if (i==255) i=9;
goto s1;
}
```

```
#include <mega32.h>
#include <delay.h>
signed char i=9;
void main(void)
{
DDRA=0xFF;
s1:
PORTA=i;
delay_ms(500);
i--;
if (i== -1) i=9;
goto s1;
}
```

دستور switch :

هر گاه لازم باشد در برنامه به ازای مقادیر مختلف یک متغیر دستورهای متفاوتی اجرا شود می توان از دستور switch با ساختار زیر استفاده کرد.

عبارتی که باید مورد بررسی قرار گیرد)

{

case ثابت ۱:

مجموعه دستورات ۱

break;

case ثابت ۲:

مجموعه دستورات ۲

break;

.

.

case n ثابت:

مجموعه دستورات n

break;

default :

مجموعه دستورات حالت پیش فرض

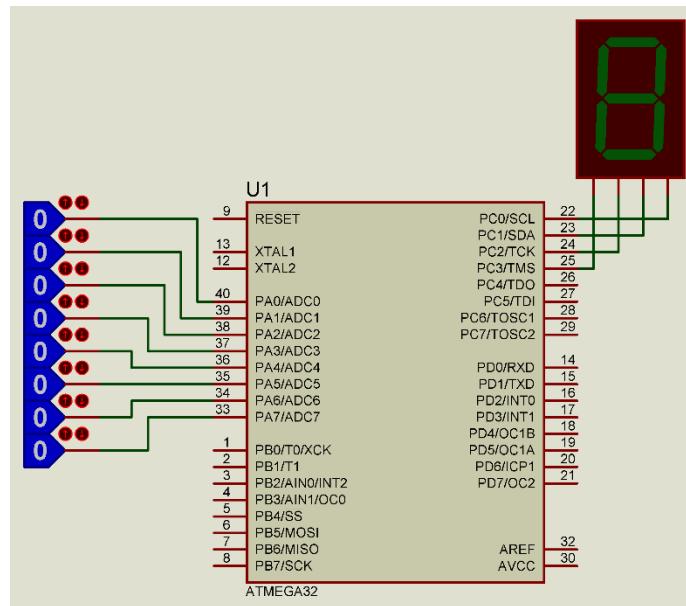
}

مثال: عددی را از **PORTA** بخوانید ، اگر عدد خوانده شده ۱ بود روی **PORTC** عدد ۵ ، اگر ۲ بود عدد ۷ ، اگر ۳ بود عدد ۱۳ دیده شود و اگر غیر از ۱ و ۲ و ۳ بود عدد ۱۵ دیده شود.

```
#include <mega32.h>

unsigned char a;

void main(void)
{
    DDRA=0x00;
    DDRC=0xFF;
    s1:
    a=PINA;
    switch(a)
    {
        case 1:
            PORTC=5;
            break;
        case 2:
            PORTC=7;
            break;
        case 3:
            PORTC=13;
            break;
        default:
            PORTC=15;
    }
    goto s1;
}
```

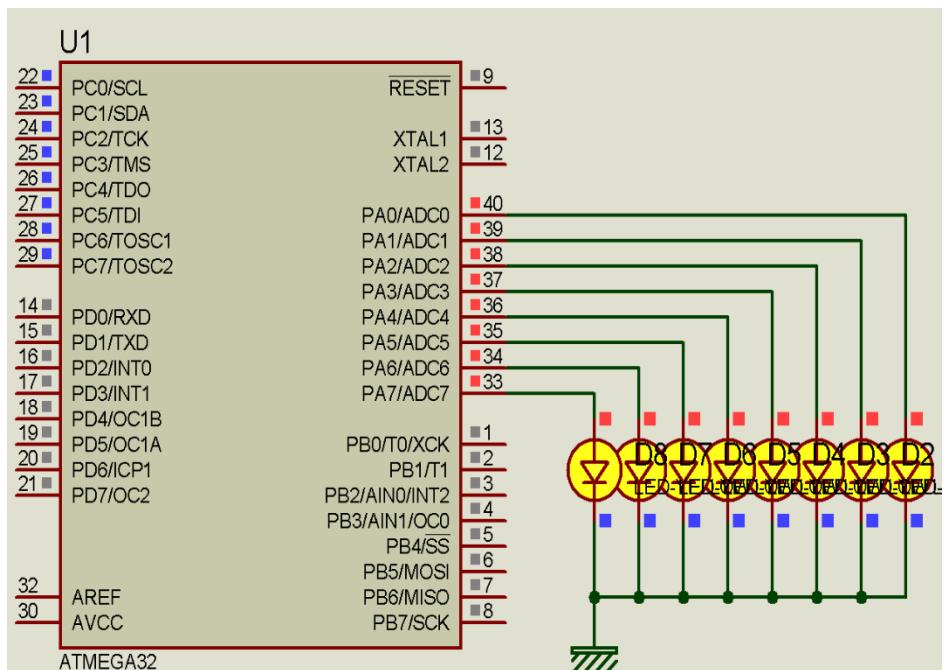


حلقه : Loop در برنامه نویسی بسیار پیش می آید که لازم است دستور یا دستورهایی چندین بار

اجرا گردد. راه مناسب این است که آن ها را درون یک حلقه قرار دهیم تا به تعداد دفعات لازم تکرار گردد. در هر حلقه یک کنتور وجود دارد؛ آن را با عدد حلقه پر می کنیم و با هر بار اجرا یک واحد از آن کم می کنیم؛ هرگاه صفر شد، از حلقه خارج می شویم.

مثال: هشت عدد LED به پورت A متصل کنید . برنامه ای بنویسید که آن ها را پنج بار خاموش و روشن کند.

```
#include <mega32.h>
#include <delay.h>
unsigned char i;
void main (void)
{
DDRA=0xFF;
i=5;
S1:
PORTA=0xFF;
delay_ms(500);
PORTA=0x00;
delay_ms(500);
i--;
if(i!=0) goto S1;
S2:goto S2;
}
```



دستور While : روش دیگر ایجاد حلقه دستور **while** با ساختار زیر می باشد.

شرط اجرای حلقه) while

{

....

....

}

دستورها

```
while (1)
{
    .....
    .....
}
```

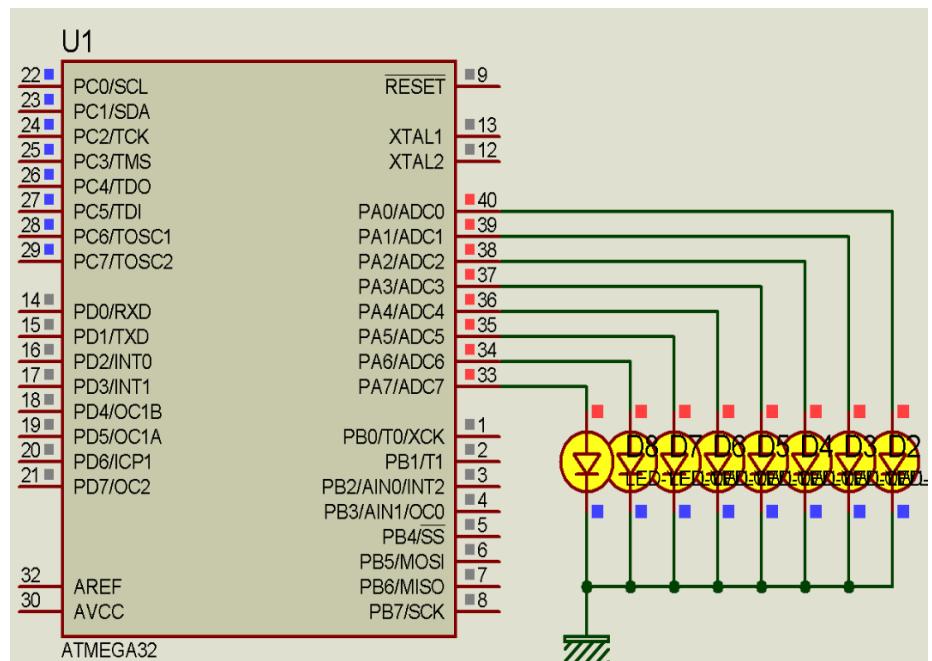
```
while (1);
```

نکته: برای ایجاد یک حلقه بی نهایت می توان به شکل زیر عمل کرد.

نکته: اگر می خواهید برنامه روی خطی متوقف شود دستور زیر را بنویسید.

مثال: مثال قبل را با دستور while اجرا نمایید.

```
#include <mega32.h>
#include <delay.h>
unsigned char i;
void main (void)
{
    DDRA=0xFF;
    i=5;
    while(i>0)
    {
        PORTA=0xFF;
        delay_ms(500);
        PORTA=0x00;
        delay_ms(500);
        i--;
    }
    S2:goto S2;
}
```



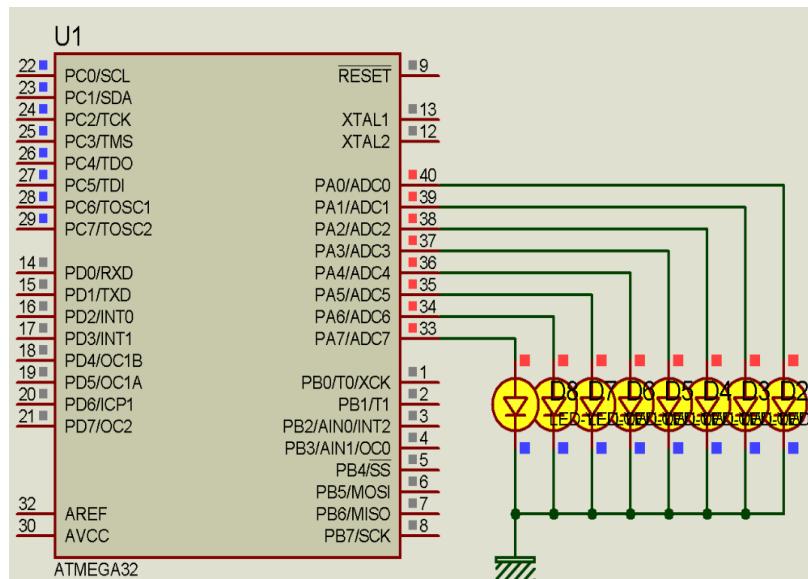
حلقه for: هرگاه تعداد دفعات تکرار حلقه مشخص باشد، می‌توان از دستور `for` استفاده کرد.

(گام حلقه ; شرط اجرای حلقه ; مقدار اولیه = متغیر حلقه)

```
{
.....
.... دستورها
}
```

مثال: مثال قبل را با دستور `for` اجرا نمایید.

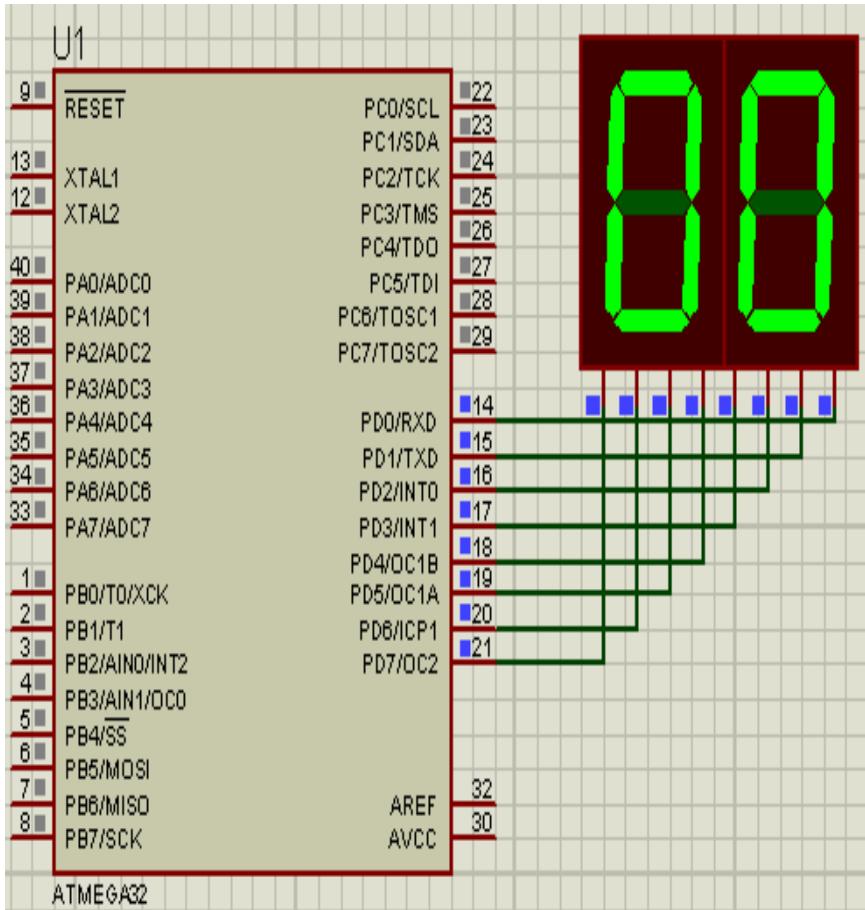
```
#include <mega32.h>
#include <delay.h>
unsigned char i;
void main (void)
{
DDRA=0xFF;
for(i=0;i<5;i++)
{
PORTA=0xFF;
delay_ms(500);
PORTA=0x00;
delay_ms(500);
}
while(1);
}
```



مثال: دو عدد سون سگمنت BCD به پورت D متصل کنید . برنامه ای بنویسید که یک شمارنده "

" تا "۲۰" بر روی پورت D داشته باشیم.

```
#include <mega32.h>
#include <delay.h>
unsigned char i;
void main (void)
{
    DDRD=0xFF;
    while(1)
    {
        for(i=0;i<21;i++)
        {
            PORTD=i;
            delay_ms(500);
        }
    }
}
```



مثال: مثال قبل را طوری تغییر دهید که یک شمارنده زوج شمار داشته باشیم.

برای این کار کافیست خط ۹ را به صورت زیر تغییر دهیم:

```
for(i=0;i<21;i=i+2)
```

مثال: مثال قبل را فرد شمار کنید.

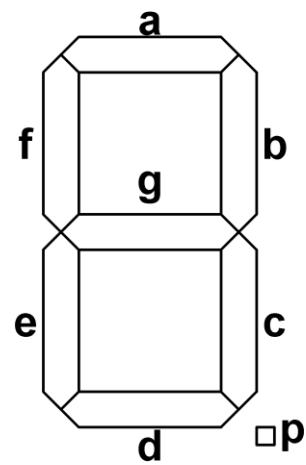
برای این کار کافیست خط ۹ را به صورت زیر تغییر دهیم:

```
for(i=1;i<21;i=i+2)
```

مثال: یک سون سگمنت کاتد مشترک را به پورت C متصل کنید و شمارنده ۰ تا ۹ بر روی آن بسازید.

برای این کار باید از کد اعداد برای سون سگمنت استفاده نمود که به شرح زیر است.

عدد	p	g	f	e	d	c	b	a	کد
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	1	1	1	1	0x6F

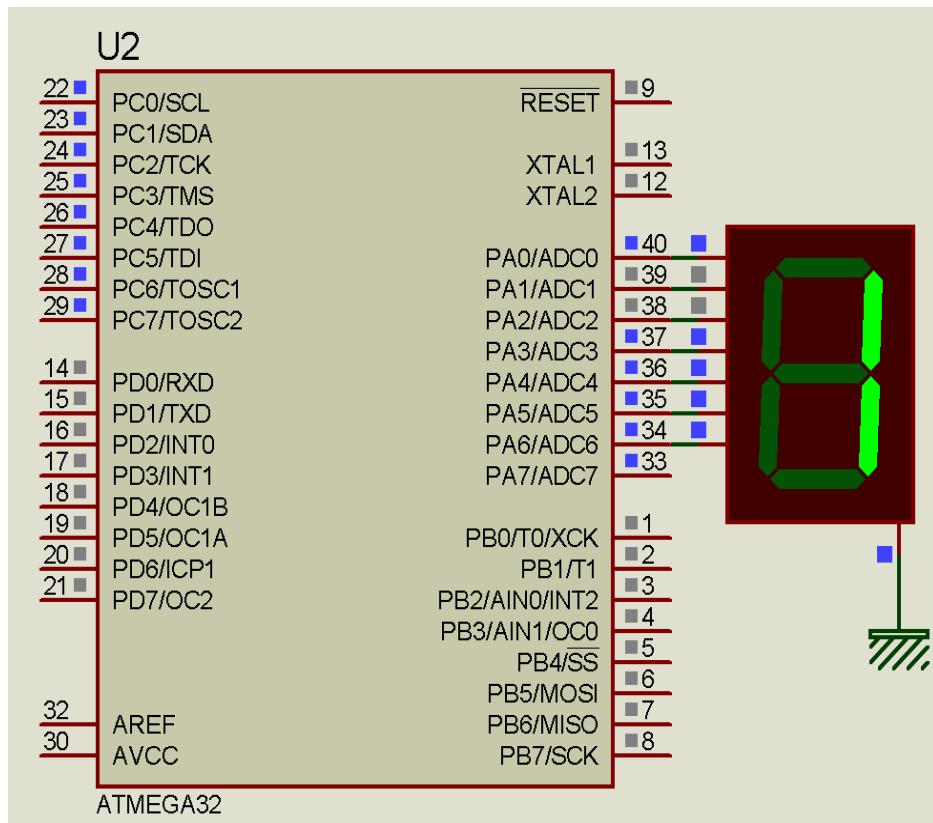


Number	0	1	2	3	4	5	6	7	8	9
Code	0x3F	0x06	0x5B	0x4F	0x66	0x6D	0x7D	0x07	0x7F	0x6F

```
#include <mega32.h>
#include <delay.h>
void main (void)
{
DDRA=0xFF;
while(1)
{
PORTA=0x3F;
delay_ms(500);
PORTA=0x06;
delay_ms(500);

.
.
.

PORTA=0x6F;
delay_ms(500);
}
}
```



آرایه (ARRAY) : اگر به تعداد زیادی متغیر از یک نوع نیاز پاشد بهتر است برای معرفی آن ها از

آرایه یا متغیر های اندیس، دار استفاده نماییم. که به شکل زیر تعریف می، شوند.

نوع متغیر ، مقدار دوم ، مقدار اول }=[تعداد خانه ها] نام آرایه }

```
int d[5] = { 7, 12, 0, 99, 20};
```

d[0] d[1] d[2] d[3] d[4]

7	12	0	99	20
---	----	---	----	----

آرایه	d[0]	d[1]	d[2]	d[3]	d[4]
مقدار	7	12	0	99	20
بعد از عملیات	8	11	60	100	12

```
d[4]= d[0]+5;
```

```
d[3]++;
```

d[2]= d[1]^*5;

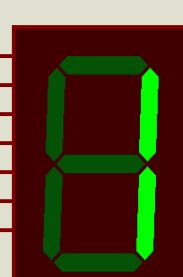
```
d[1]--;
```

d[0]=8;

مثال: مثال قبل را به کمک آرایه بنویسید.

```

#include <mega32.h>
#include <delay.h>
unsigned char i;
unsigned char bts[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F ,0x6F};
void main (void)
{
DDRA=0xFF;
while(1)
{
for(i=0;i<10;i++)
{
PORTA=bts[i];
delay_ms(500);
}
}
}

U2
 22 PC0/SCL
 23 PC1/SDA
 24 PC2/TCK
 25 PC3/TMS
 26 PC4/TDO
 27 PC5/TDI
 28 PC6/TOSC1
 29 PC7/TOSC2
 14 PD0/RXD
 15 PD1/TXD
 16 PD2/INT0
 17 PD3/INT1
 18 PD4/OC1B
 19 PD5/OC1A
 20 PD6/ICP1
 21 PD7/OC2
 32 AREF
 30 AVCC
RESET 9
XTAL1 13
XTAL2 12
PA0/ADC0 40
PA1/ADC1 39
PA2/ADC2 38
PA3/ADC3 37
PA4/ADC4 36
PA5/ADC5 35
PA6/ADC6 34
PA7/ADC7 33
PB0/T0/XCK 1
PB1/T1 2
PB2/AIN0/INT2 3
PB3/AIN1/OC0 4
PB4/SS 5
PB5/MOSI 6
PB6/MISO 7
PB7/SCK 8


```

آرایه های چند بعدی: به شکل زیر تعریف می شوند.

نوع متغیر ، {عناصر سطر دوم} ، {عناصر سطر اول} } = [ستون] [سطر] نام آرایه

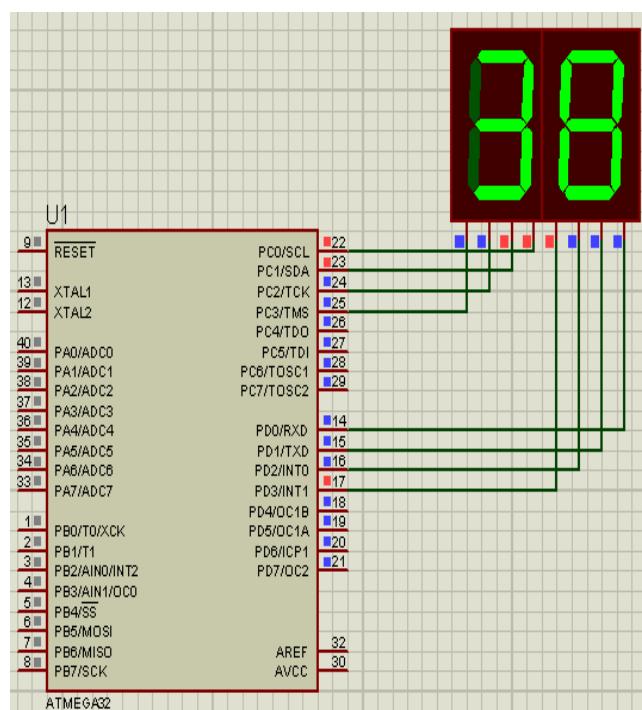
```
int a[3][4]={{1,5,3,4},{7,2,1,0},{1,2,3,4}};  ↴  int a[3][4]={{1,5,3,4,7,2,1,0,1,2,3,4}};
```

	ستون 0	ستون 1	ستون 2	ستون 3
سطر 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
سطر 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
سطر 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

نام آرایه ← → اندیس ستون
اندیس سطر

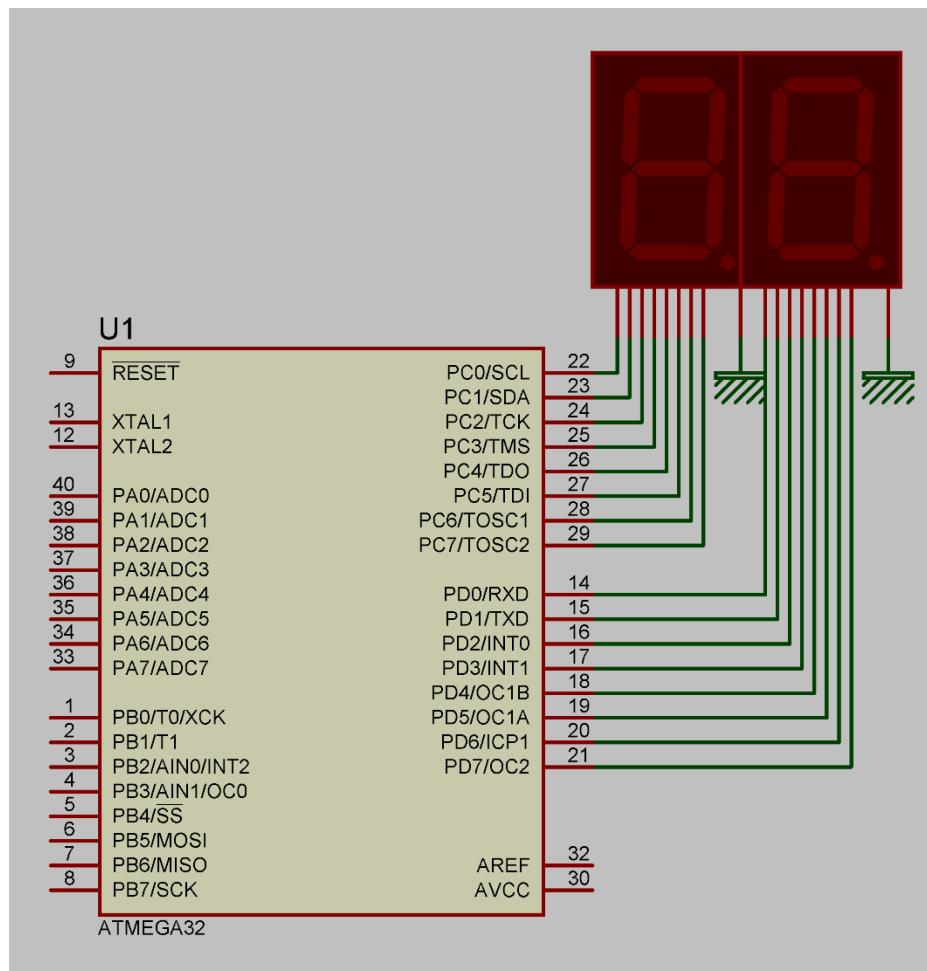
مثال: دو عدد سون سگمنت BCD را به پورت A و B متصل کنید و یک شمارنده ۰ تا ۹۹ دسیمال بسازید.

<pre>#include <mega32.h> #include <delay.h> unsigned char y=0,d=0; void main (void) { DDRD=0xFF; DDRC=0xFF; while(1) { PORTC=d; PORTD=y; delay_ms(500); y++; if(y==10) { y=0; d++; if(d==10) d=0; } } }</pre>	<pre>#include <mega32.h> #include <delay.h> unsigned char i=0; void main (void) { DDRD=0xFF; DDRC=0xFF; while(1) { For(i=0;i<100;i++) { PORTC=i/10; PORTD=i%10; delay_ms(500); } } }</pre>
---	---

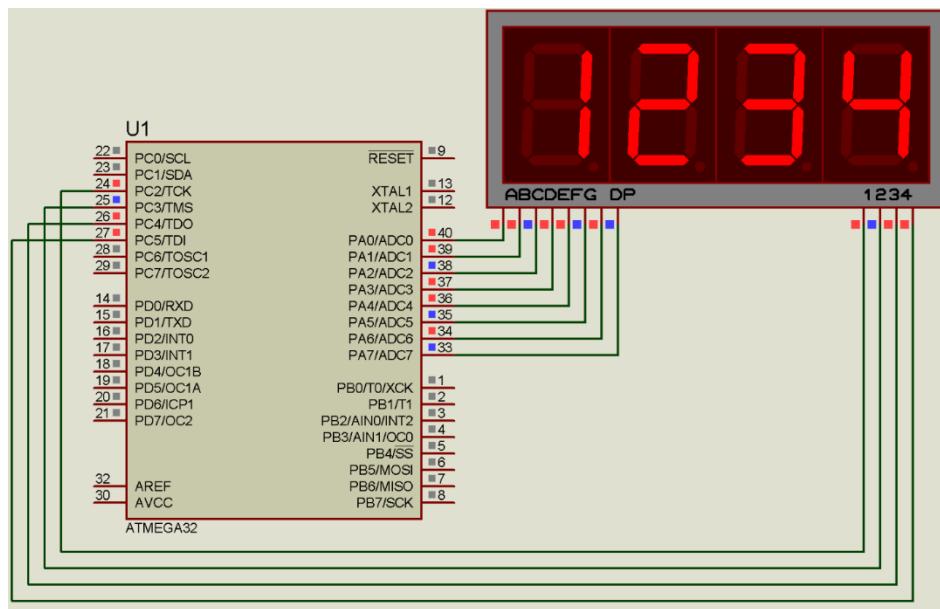


مثال: به جای سون سگمنت کاتد مشترک قرار دهید و مثال قبل را انجام دهید.

```
#include <mega32.h>
#include <delay.h>
unsigned char y=0,d=0;
unsigned char bts[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f ,0x6f};
void main (void)
{
    DDRD=0xFF;
    DDRC=0xFF;
    while(1)
    {
        PORTD=bts[y];
        PORTC=bts[d];
        delay_ms(500);
        y++;
        if(y==10)
        {
            y=0;
            d++;
            if(d==10) d=0;
        }
    }
}
```

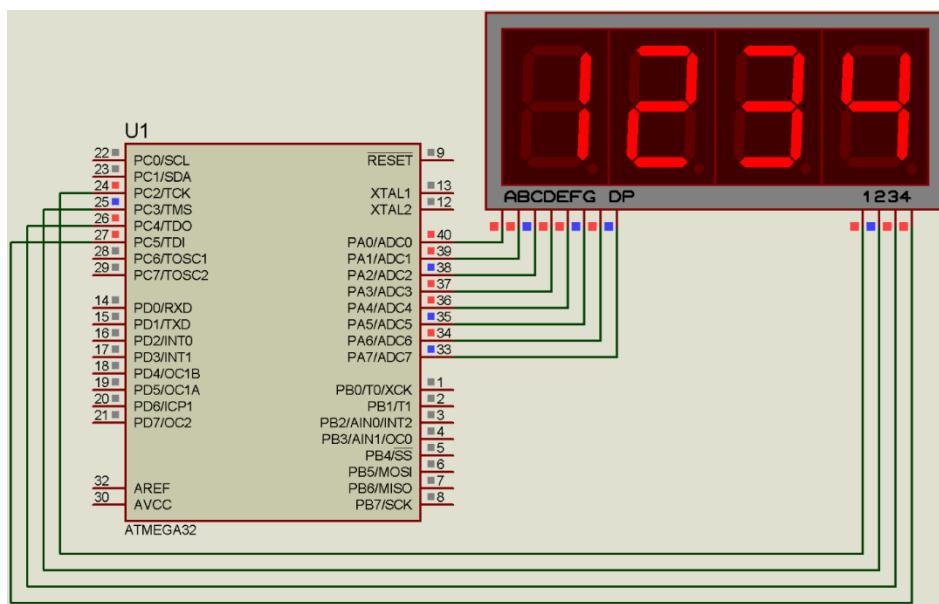


نمایشگر (Display): در اکثر مدارها برای نمایش دیتای خروجی نیاز به یک نمایشگر داریم که تعداد رقم های آن برای خروجی ما مناسب باشد. فرض کنید در یک پروژه نیاز به چهار رقم در خروجی داریم؛ اگر قرار باشد که هر سون سگمنت را به یک پورت متصل کنیم، هر چهار پورت میکرو مشغول می شود و برای دستگاه های دیگر پورتی باقی نمی ماند. راه حل این مشکل این است که سون سگمنت را به شکل زیر بیندیم و آن ها را به روش مالتی پلکس روشن کنیم.



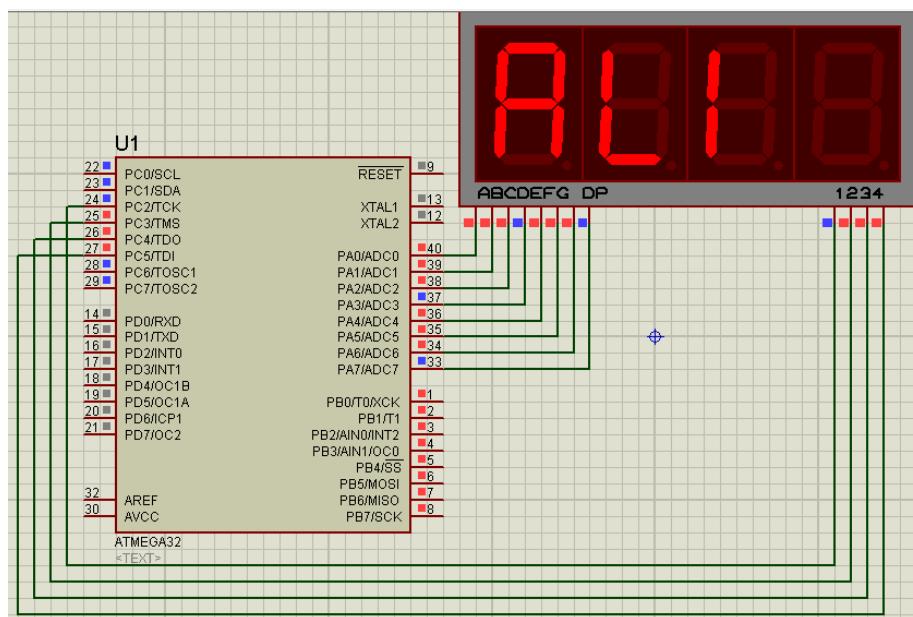
در مثال زیر عدد 1234 بر روی نمایشگر، نمایش داده شده است.

```
#include <mega32.h>
#include <delay.h>
unsigned char bts[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F ,0x6F};
void main (void)
{
    DDRA=0xFF;
    DDRC=0x3C; //0b00111100
    while(1)
    {
        PORTC=0x3C; // خاموش کردن تمام رقم ها
        //----Digit1-----
        PORTA=bts[1]; // قرار دادن کد عدد یک روی پورت A
        PORTC.2=0; // روشن کردن رقم اول
        delay_ms(5); // تاخیر برای دیدن رقم اول
        PORTC.2=1; // خاموش کردن رقم اول
        //----Digit2-----
        PORTA=bts[2];
        PORTC.3=0;
        delay_ms(5);
        PORTC.3=1;
        //----Digit3-----
        PORTA=bts[3];
        PORTC.4=0;
        delay_ms(5);
        PORTC.4=1;
        //----Digit4-----
        PORTA=bts[4];
        PORTC.5=0;
        delay_ms(5);
        PORTC.5=1;
    }
}
```



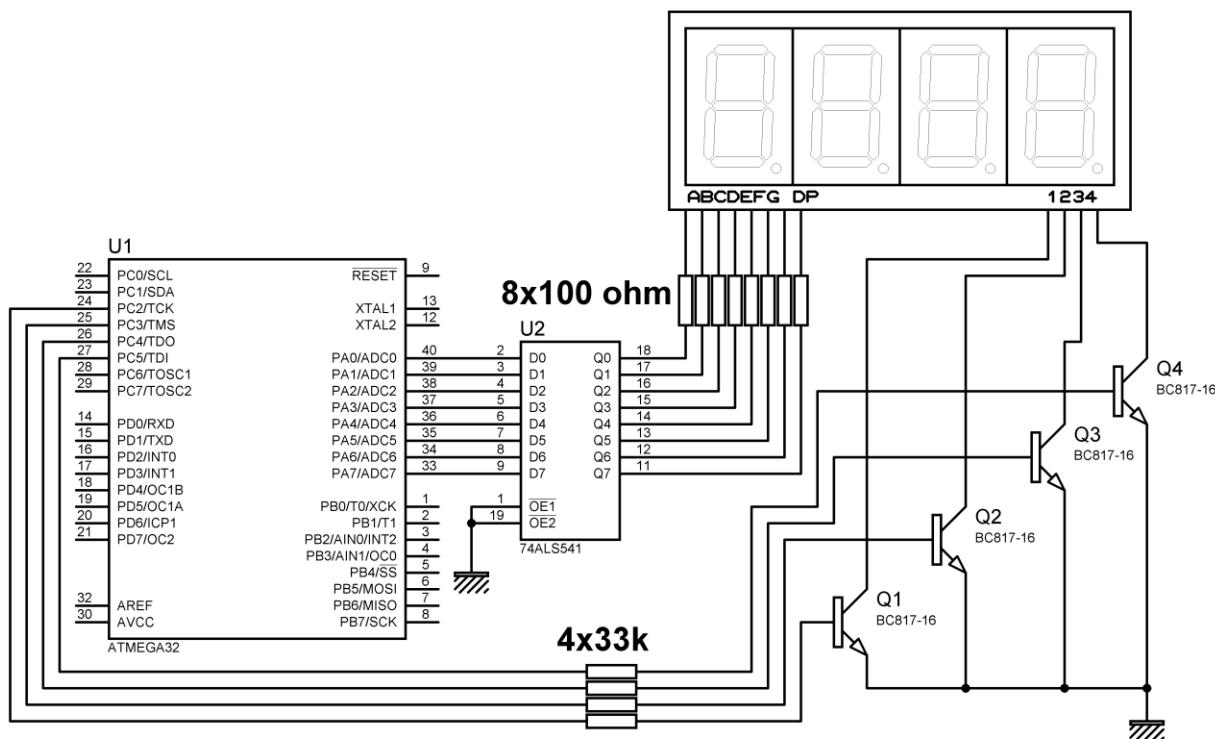
مثال: کلمه ALI را بر روی سون سگمنت ۴ تایی نمایش دهید.

```
#include <mega32.h>
#include <delay.h>
void main(void)
{
    DDRC=0x3C;
    DDRA=0xff;
    while(1)
    {
        PORTC=0x3C;
        //---DIGIT1---
        PORTA=0x77;
        PORTC.2=0;
        delay_ms(5);
        PORTC.2=1;
        //---DIGIT2---
        PORTA=0x38;
        PORTC.3=0;
        delay_ms(5);
        PORTC.3=1;
        //---DIGIT3---
        PORTA=0x30;
        PORTC.4=0;
        delay_ms(5);
        PORTC.4=1;
        //---DIGIT4---
        PORTA=0;
        PORTC.5=0;
        delay_ms(5);
        PORTC.5=1;
    }
}
```



درایور نمایشگر :

با توجه به این که در مدار نمایشگر بالا ، کاتد هر رقم به یکی از بیت های PORTC متصل شده ، و جریان عبوری از کاتد در حالتی که تمام سگمنت ها و نقطه اعشار روشن باشد ، در حدود 80ma خواهد شد و با توجه به این که جریان قابل تحمل هر بیت از پورت حداکثر 20ma است پس در عمل نمی توان کاتد را مستقیم به پورت متصل کرد، و نیاز به یک درایور داریم مدار زیر که در برآ آزمایش نیز استفاده شده است می تواند یک نمونه درایور برای نمایشگرهای سون سگمنتی باشد.



لازم به ذکر است که مقدار مقاومت های 33k را می توان با توجه به نور سون سگمنت ها با مقادیر مختلف جایگزین کرد تا نور مناسب حاصل شود.

تمرين: برنامه اي بنويسيد که يك شمارنده ۰ تا ۹۹۹۹ بر روی نمایشگر داشته باشيم.

تمرين: برنامه اي بنويسيد که به طور هم زمان دو شمارنده بالا شمار ۰ تا ۹۹ و شمارنده پایین شمار

۹۹ تا ۰ بر روی نمایشگر داشته باشيم .

تمرين: برنامه اي بنويسيد که ثانيه و دقيقه شمار را بر روی نمایشگر داشته باشيم . در ضمن اعداد

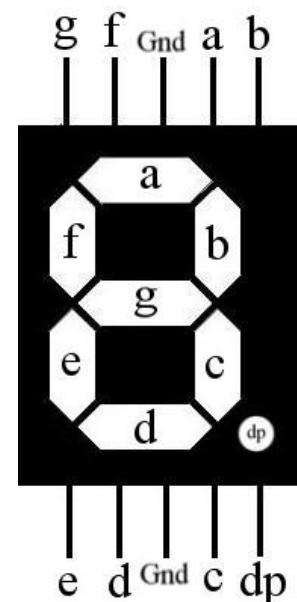
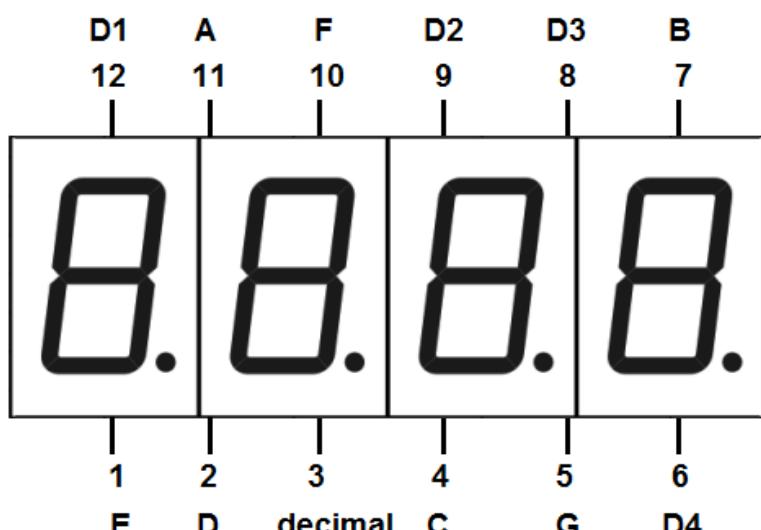
دقيقه و ثانие را با روشن کردن نقطه اعشار يکان دقيقه از هم جدا کنيد.

تمرين: برنامه اي بنويسيد که کلمه ALI روی نمایشگر حرکت کند(تابلوی روان) .

تمرين: در نرم افزار پروتئوس يك نمایشگر هشت رقمی را به میکرو متصل کنيد و کلمه ALI را حرکت

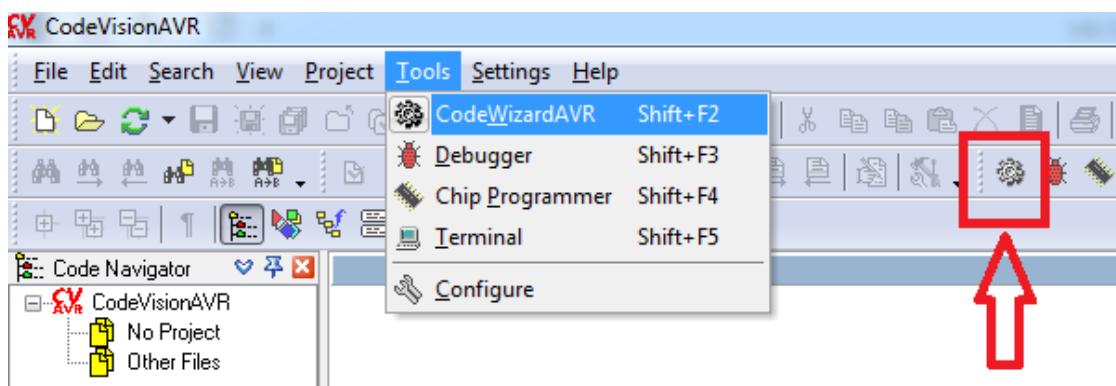
دهيد . برنامه را طوري بنويسيد که حرکت بصورت رفت و برگشت باشد.

* شماره پایه ها و نحوه قرارگیری آن ها در قطعات واقعی به شکل زیر است :



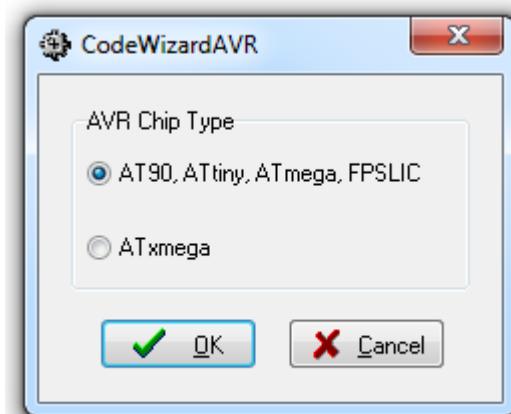
مراحل ایجاد پروژه با استفاده از ویزارد : WIZARD

۱- از منوی Tools یا نوار ابزار **Code Wizard AVR** را انتخاب کنید.

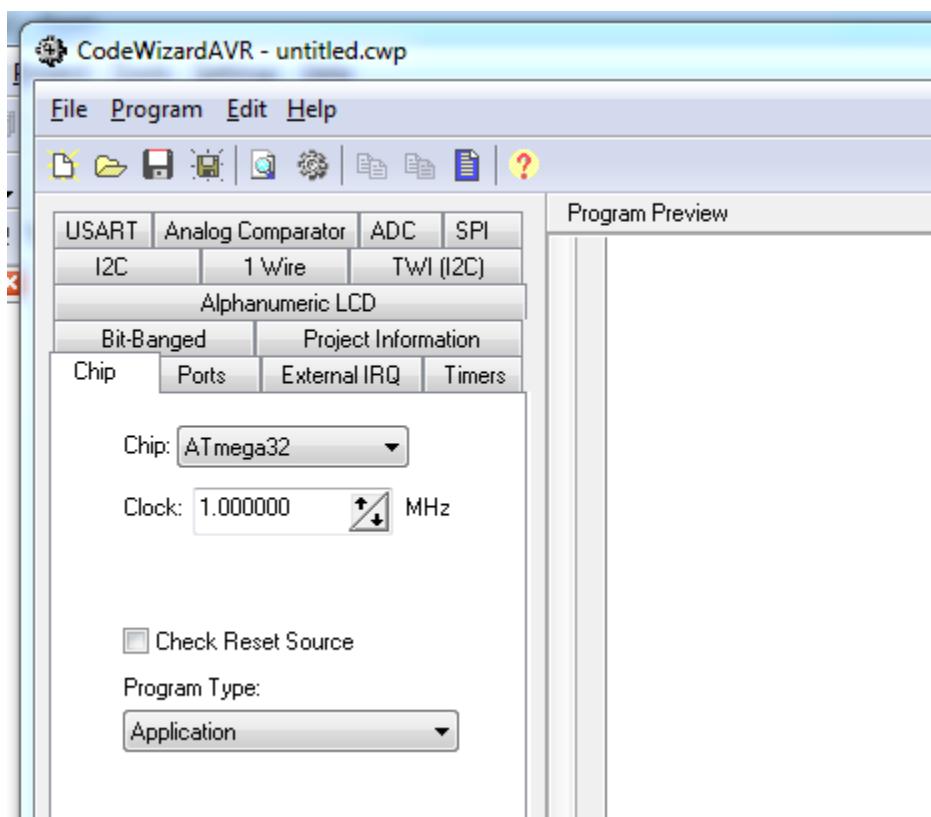


۲- در پنجره **CodeWizardAVR** که باز می شود **ATmega** انتخاب شده است با کلیک روی

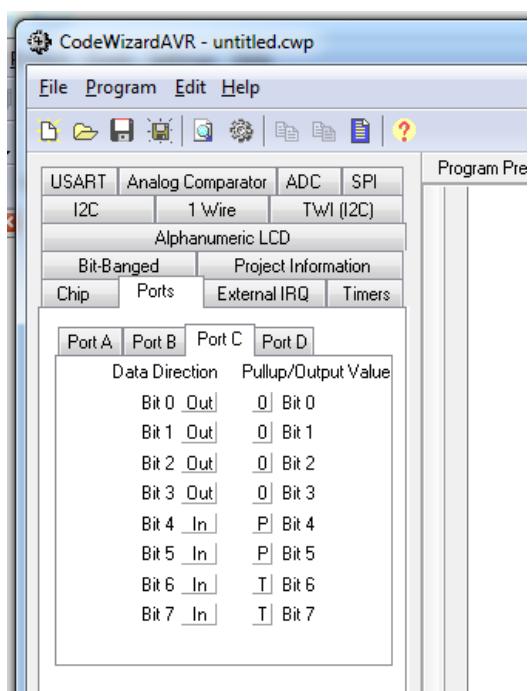
OK به مرحله بعدی می رویم.



۳- پنجره Wizard در حالی که برگه Chip انتخاب شده باز می شود دراین برگه شماره میکرو و فرکانس کاری آن را مشخص می کنیم.



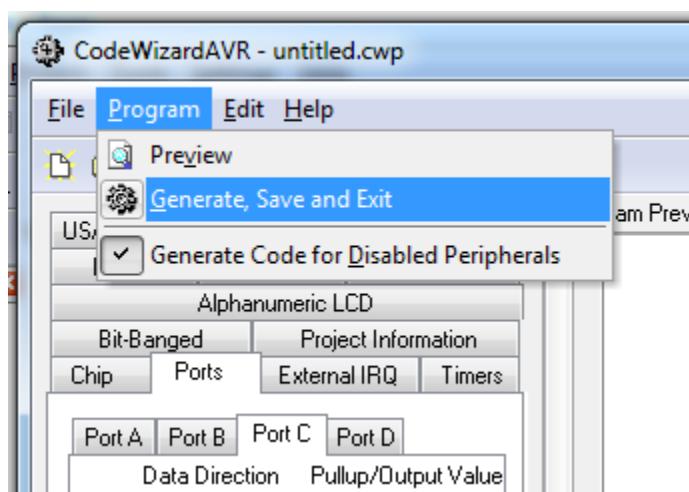
۴- با توجه به نیاز پروژه ، برگه های دیگر را باز و تنظیم های لازم را انجام می دهیم.



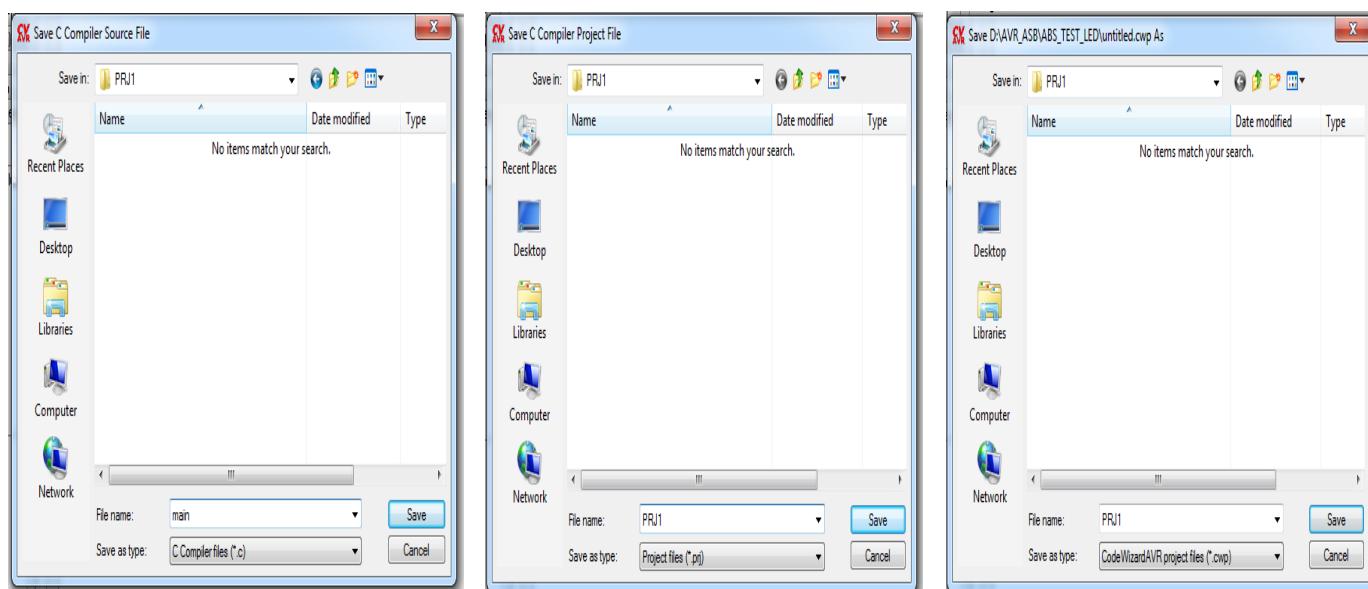
*نکته: در برگه پورت ها اگر پورتی ورودی تنظیم شود و در قسمت pull_up حرف T باشد یعنی این پایه HI_Z است و منطق لاجیکی آن بستگی به ورودی دارد .
اگر حرف T را با کلیک به P تبدیل کنید یعنی این پایه از داخل میکرو pull_up شده و منطق یک دارد.

۵- پس از اتمام تنظیمات ، از مسیر زیر پروژه و فایل های مربوط به آن را با نام مناسب و در محلی

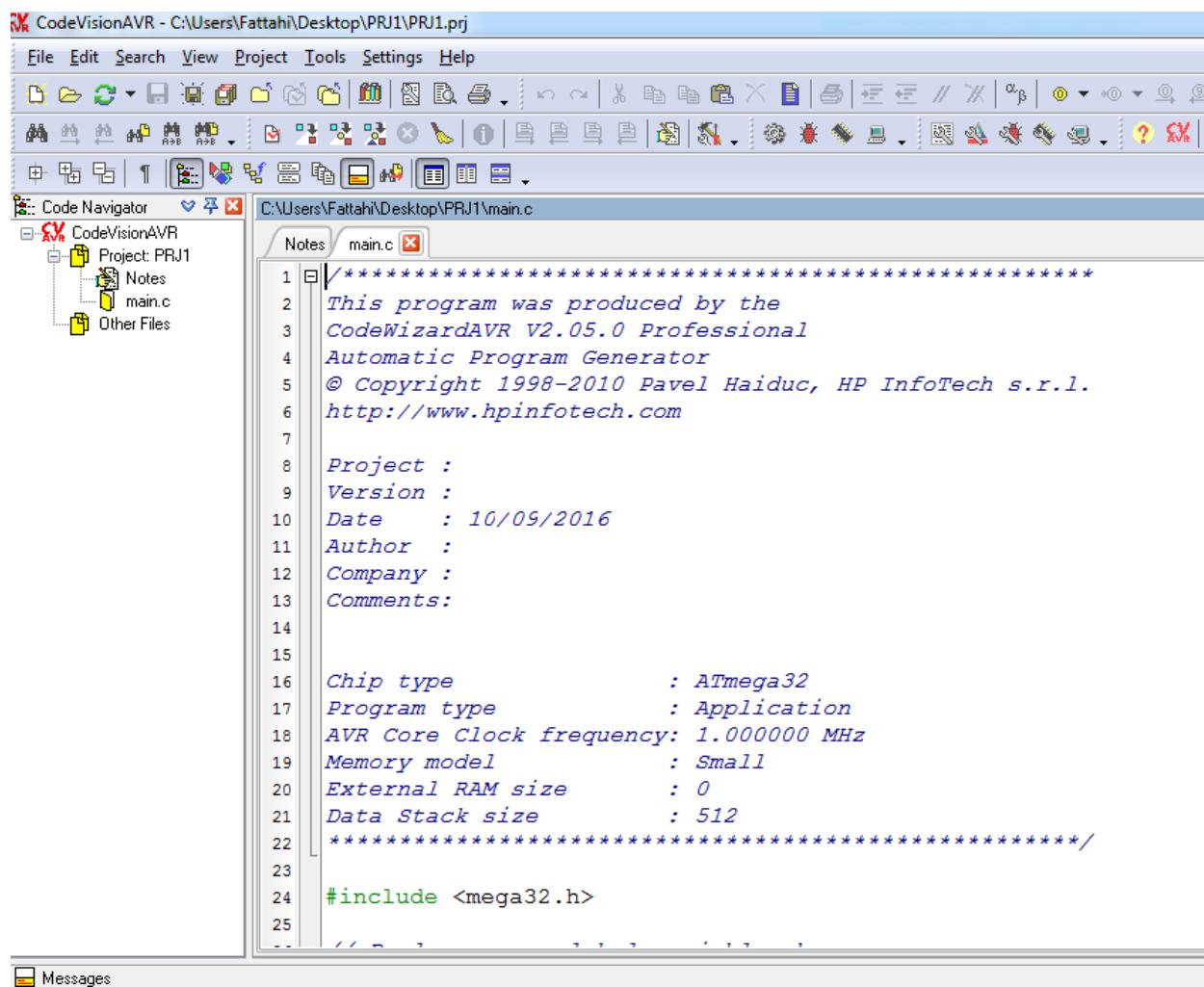
Program/Generate, Save and Exit مشخص ذخیره می کنیم.



***نکته:** در این مرحله لازم است سه بار فایل ذخیره کنیم ، Source --- Project --- CWP



۶- پس از ذخیره فایل ها ، کد برنامه با توجه به تنظیمات ویزارد باز می شود حال می توان برنامه مورد نظر را به آن اضافه کرد.



```

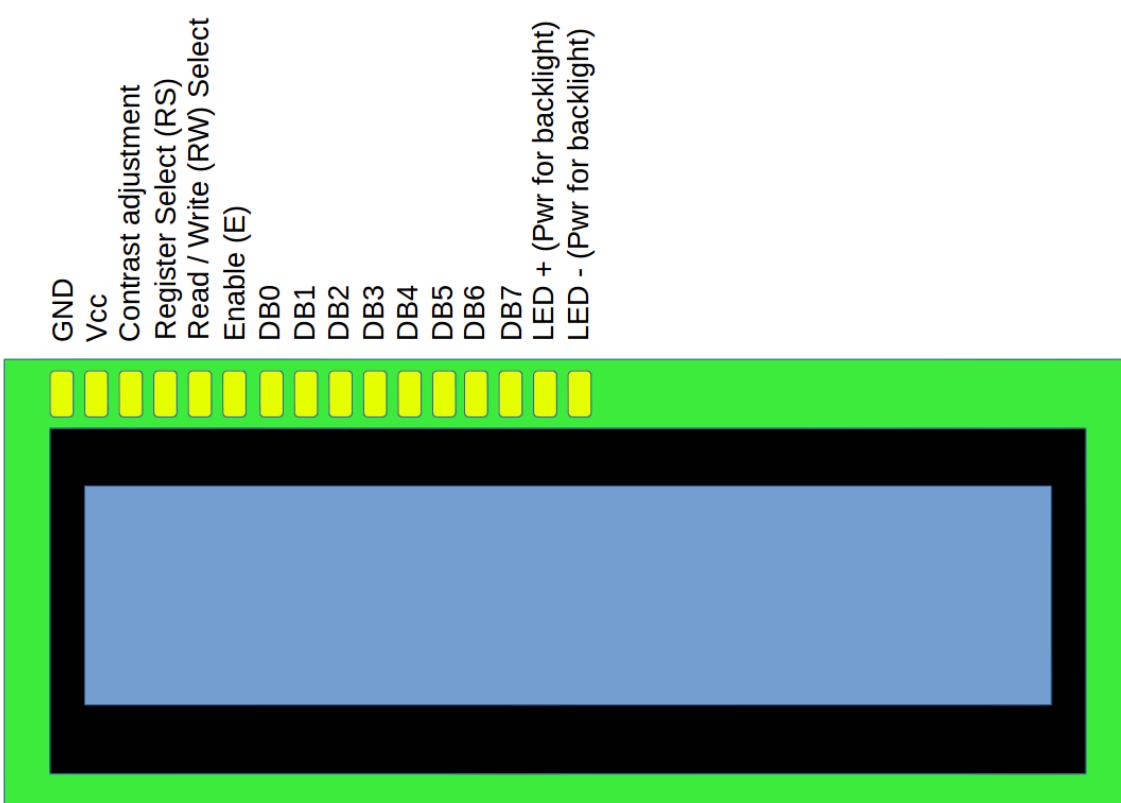
CodeVisionAVR - C:\Users\Fattahi\Desktop\PRJ1\PRJ1.pj
File Edit Search View Project Tools Settings Help
[File Explorer] [Project Navigator] [Code Navigator] [Search] [Editor] [Terminal]
Code Navigator Notes main.c
C:\Users\Fattahi\Desktop\PRJ1\main.c
Notes main.c
1  //*****
2  This program was produced by the
3  CodeWizardAVR V2.05.0 Professional
4  Automatic Program Generator
5  @ Copyright 1998-2010 Pavel Haiduc, HP InfoTech s.r.l.
6  http://www.hpinfotech.com
7
8  Project :
9  Version :
10 Date : 10/09/2016
11 Author :
12 Company :
13 Comments:
14
15
16  Chip type : ATmega32
17  Program type : Application
18  AVR Core Clock frequency: 1.000000 MHz
19  Memory model : Small
20  External RAM size : 0
21  Data Stack size : 512
22  *****/
23
24  #include <mega32.h>
25

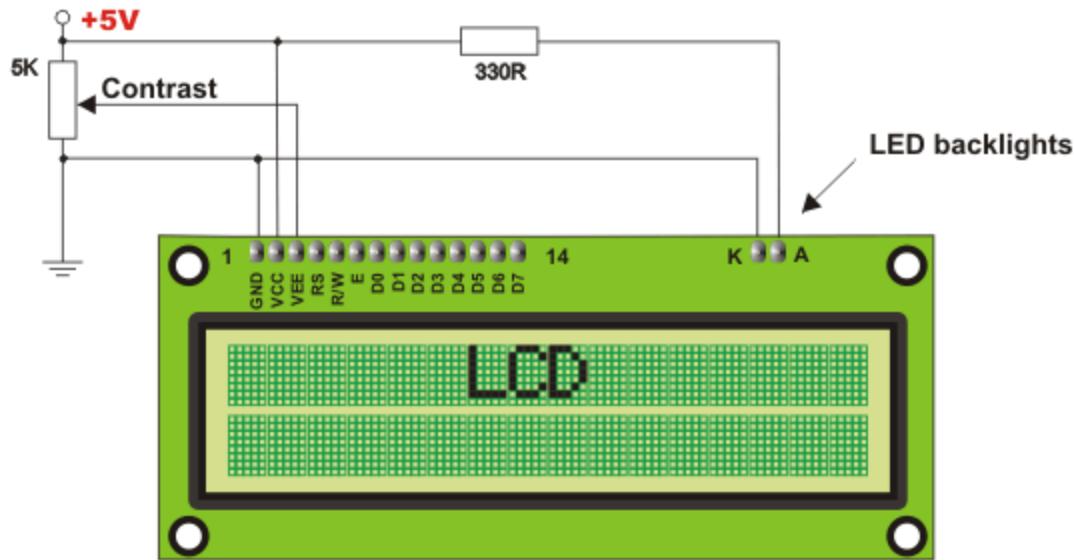
```

LCD: یکی دیگر از دستگاه های خروجی LCD کاراکتری می باشد که می توانید بر روی آن متن و اعداد را نمایش دهید. در LCD کاراکتری پارامتر های مهم تعداد خط و تعداد کاراکتر در هر خط است. نوع ۲*۱۶ آن پر کاربردتر است.



پایه های LCD: معمولاً ۱۶ پایه به شرح زیر دارد.





A(16), K(15), D7(14),...,D0(7), E(6), RW(5), RS(4), VEE(3), VDD(2), VSS(1)

K : برای کنترل نور پس زمینه (Back Light) می باشد.

D7,...,D0 : دستور و دیتا را دریافت می کند.

RS : اگر صفر باشد خطوط انتقال دیتا، دستور و اگر یک باشد دیتا دریافت می کنند.

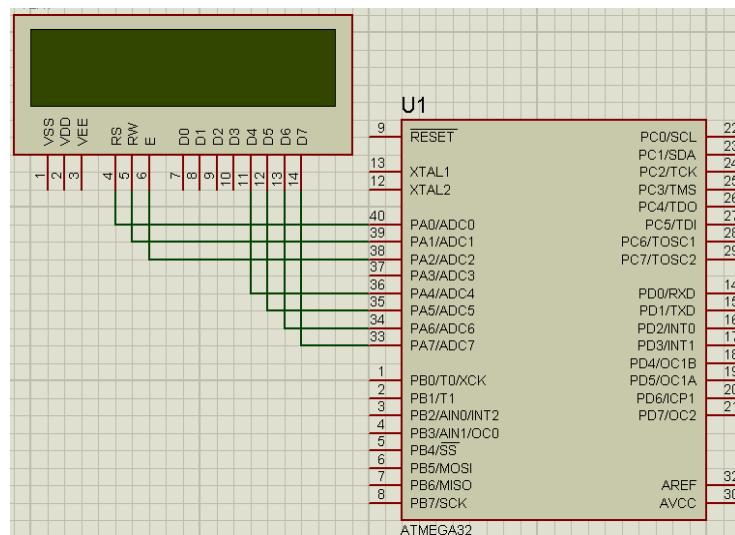
RW : اگر یک باشد یعنی Read و اگر صفر باشد یعنی Write.

E : با اعمال یک لبه ی پایین رونده می توان اجازه ورود دیتا یا دستور را صادر کرد.

VEE : با متصل کردن یک پتانسیومتر 10k به این پایه می توان نور پشت صفحه Back Light را کنترل کرد.

VSS , VDD : تغذیه LCD از این پایه ها تامین می شود.

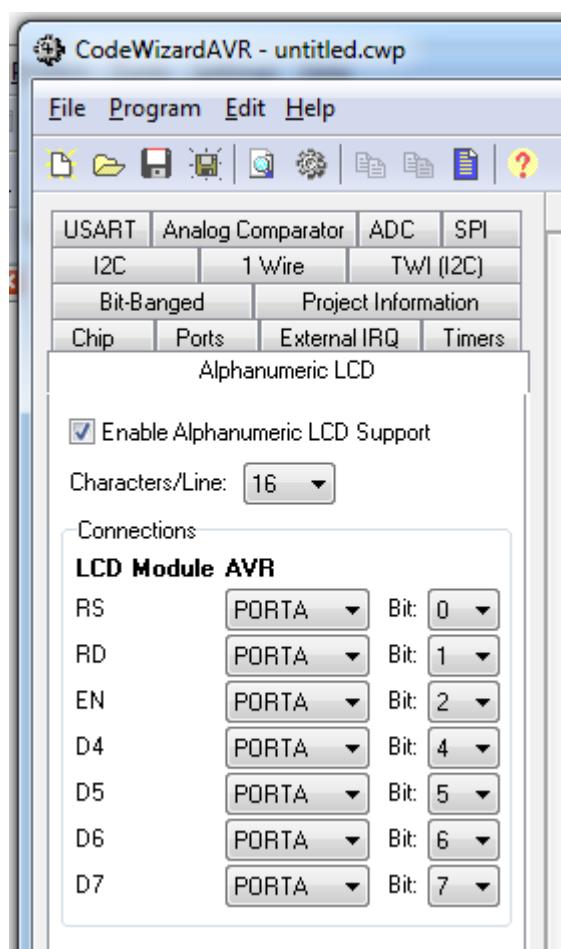
نکته: LCD های کاراکتری را می توان با ۴ خط دیتا راه اندازی کرد (خطوط D4 تا D7). برای مثال در شکل زیر اتصال یک LCD به PORTA را مشاهده می کنید.



کد ویژن از این نوع اتصال

پشتیبانی می کند.

مثال: یک LCD را به پورت A متصل کنید. برنامه ای بنویسید که وسط خط اول کلمه IRAN و وسط خط دوم 1394 نوشته شود.



ابتدا در ویزارد برگه Alphanumeric LCD را باز
و آن را فعال می کنیم. در قسمت Character/Line مشخص می کنیم که در هر خط چند کاراکتر وجود دارد.
در قسمت Connections نحوه اتصال LCD به میکرو مشخص شده که قابل ویرایش است.

پس از تولید کد، کتابخانه `<alcd.h>` به برنامه اضافه می شود که توابع کار با LCD در این کتابخانه قرار دارد.

`lcd_clear();` پاک کردن صفحه نمایش

`lcd_gotoxy(x,y);` مشخص کردن ستون و سطری که نوشتگی از آنجا شروع می شود

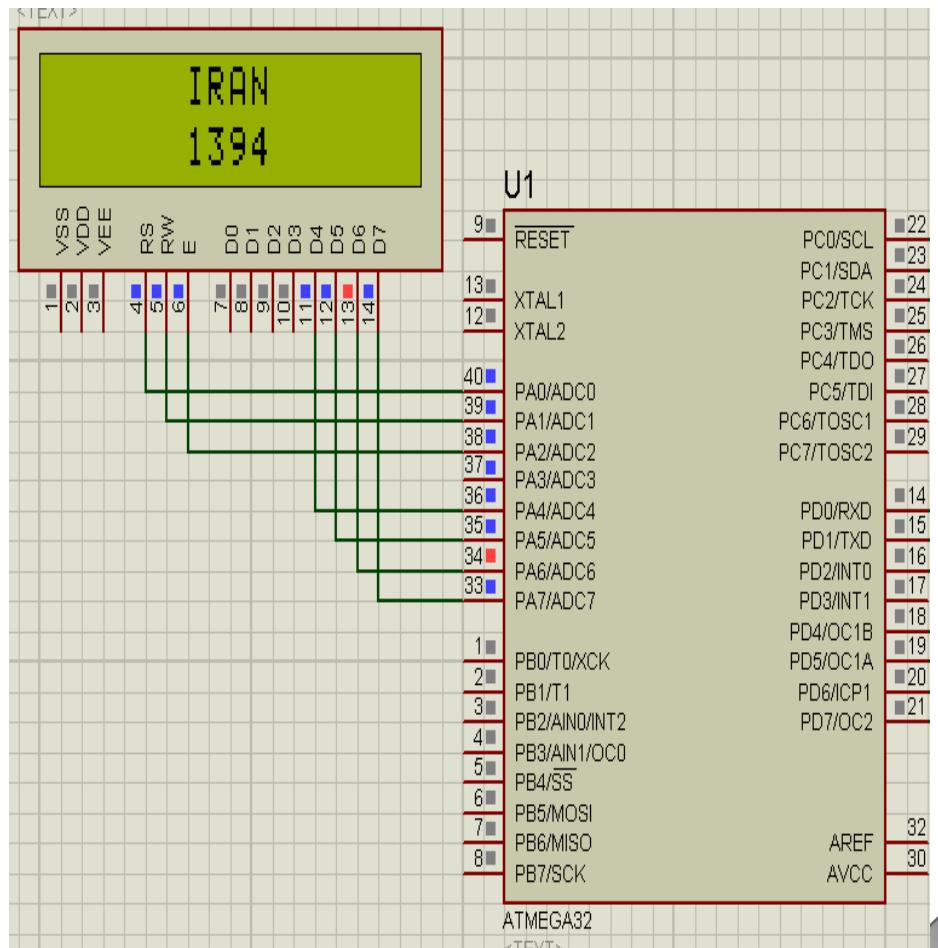
$Y=0 \longrightarrow$	X=0	X=1	X=15
$Y=1 \longrightarrow$	X=0	X=1		X=15

`Lcd_putchar('کاراکتر');`

`Lcd_putsf("رشته");`

`Lcd_puts(متغیر رشته ای);`

```
char s[ ]="IRAN";
void main(void)
{
    lcd_init(16);
    lcd_clear();
    lcd_gotoxy(6,0);
    lcd_puts(s);
    lcd_gotoxy(6,1);
    lcd_putsf("1394");
    while (1)
    {
    }
}
```



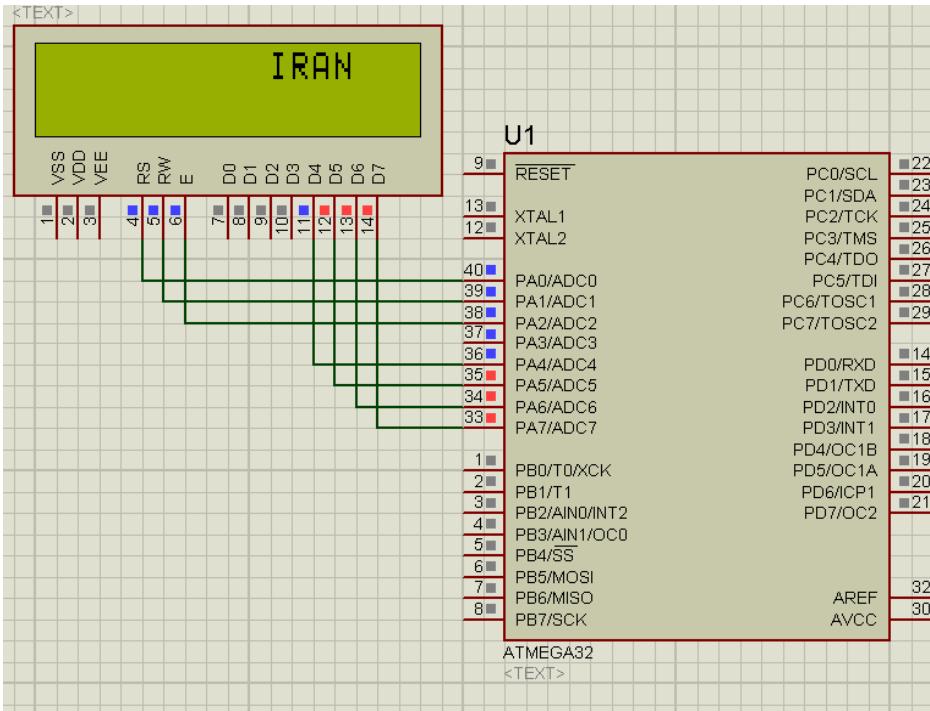
مثال: کلمه IRAN را از ابتدای خط اول تا انتهای خط اول حرکت دهید.

```
#include <delay.h>

unsigned char x;

void main(void)

{
    while (1)
    {
        for(x=0;x<13;x++)
        {
            lcd_clear();
            lcd_gotoxy(x,0);
            lcd_putsf("IRAN");
            delay_ms(500);
        }
    }
}
```



کد اسکی American Standard Code Information Interchange : (ASCII)

با توجه به اینکه کامپیوتر از بخش های مختلفی ساخته شده و لازم است که این بخش ها به یکدیگر اطلاعات ارسال و دریافت نمایند، باید کدهایی استاندارد و معروفی می شد تا تمام سازندگان کامپیوتر و دستگاه های جانبی از آن پیروی کنند. لذا انجمنی در آمریکا این کار را انجام داد و کد تولیدی را به

نام ASCII معرفی کرد. در LCD های کاراکتری نیز از کد ASCII استفاده می شود. در زیر کدهای ASCII برخی کاراکترها و جدول کدهای ASCII آمده است.

A=65 a=97 0=48 Enter=13 Space=32 Esc=27

The ASCII code

American Standard Code for Information Interchange

www.theasciicode.com.ar

ASCII control characters			ASCII printable characters									Extended ASCII characters											
DEC	HEX	Simbolo ASCII	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo			
00	00h	NULL (carácter nulo) (inicio encabezado)	32	20h	espacio	64	40h	@	96	60h	'	128	80h	Ç	160	A0h	á	192	C0h	l	224	E0h	Ó
01	01h	SOH (inicio texto)	33	21h	!	65	41h	A	97	61h	a	130	82h	ú	161	A1h	í	193	C1h	ł	225	E1h	þ
02	02h	STX (fin de texto)	34	22h	#	66	42h	B	98	62h	b	131	83h	â	162	A2h	ó	194	C2h	ł	226	E2h	Ö
03	03h	ETX (fin transmisión)	35	23h	\$	67	43h	C	99	63h	c	132	84h	ã	163	A3h	ú	195	C3h	ł	227	E3h	Ö
04	04h	EOT (enquiry)	36	24h	%	68	44h	D	100	64h	d	133	85h	ä	164	A4h	ñ	196	C4h	-	228	E4h	ö
05	05h	ENQ (acknowledgement)	37	25h	&	69	45h	E	101	65h	e	134	86h	å	165	A5h	N	197	C5h	+	229	E5h	ö
06	06h	ACK (timbre)	38	26h	*	70	46h	F	102	66h	f	135	87h	ç	166	A6h	=	198	C6h	ä	230	E6h	µ
07	07h	BEL (retroceso)	39	27h	(71	47h	G	103	67h	g	136	88h	é	167	A7h	º	199	C7h	À	231	E7h	þ
08	08h	BS (tab horizontal)	40	28h)	72	48h	H	104	68h	h	137	89h	ë	168	A8h	¿	200	C8h	Œ	232	E8h	þ
09	09h	HT (salto de linea)	41	29h	,	73	49h	I	105	69h	i	138	8Ah	è	169	A9h	Ø	201	C9h	Ù	233	E9h	Ú
10	0Ah	LF (tab vertical)	42	2Ah	.	74	4Ah	J	106	6Ah	j	139	8Bh	í	170	AAh	¬	202	CAh	Œ	234	EAh	Ù
11	0Bh	VT (form feed)	43	2Bh	+	75	4Bh	K	107	6Bh	k	140	8Ch	í	171	ABh	½	203	CBh	Œ	235	EBh	Ù
12	0Ch	FF (retorno de carro)	44	2Ch	,	76	4Ch	L	108	6Ch	l	141	8Dh	í	172	ACH	¼	204	CH	ý	236	EH	ý
13	0Dh	CR (shift Out)	45	2Dh	-	77	4Dh	M	109	6Dh	m	142	8Eh	Á	173	ADh	í	205	CDh	ý	237	EDh	ý
14	0Eh	SO (shift In)	46	2Eh	.	78	4Eh	N	110	6Eh	n	143	8Fh	Á	174	A Eh	«	206	CEh	»	238	EEh	»
15	0Fh	SI (shift In)	47	2Fh	/	79	4Fh	O	111	6Fh	o	144	90h	É	176	B0h	»	207	CFh	»	239	EFh	»
16	10h	DLE (data link escape)	48	30h	0	80	50h	P	112	70h	p	145	91h	æ	177	B1h	»	208	D0h	ð	240	F0h	ð
17	11h	DC1 (device control 1)	49	31h	1	81	51h	Q	113	71h	q	146	92h	Æ	178	B2h	»	209	D1h	ð	241	F1h	ð
18	12h	DC2 (device control 2)	50	32h	2	82	52h	R	114	72h	r	147	93h	ô	179	B3h	»	211	D3h	ð	243	F3h	ð
19	13h	DC3 (device control 3)	51	33h	3	83	53h	S	115	73h	s	148	94h	ò	180	B4h	»	212	D4h	»	244	F4h	»
20	14h	DC4 (device control 4)	52	34h	4	84	54h	T	116	74h	t	149	95h	ó	181	B5h	À	213	D5h	»	245	F5h	»
21	15h	NAK (negative acknowledge.)	53	35h	5	85	55h	U	117	75h	u	150	96h	ú	182	B6h	Â	214	D6h	»	246	F6h	»
22	16h	SYN (synchronous idle)	54	36h	6	86	56h	V	118	76h	v	151	97h	û	183	B7h	Ã	215	D7h	»	247	F7h	»
23	17h	ETB (end of trans. block)	55	37h	7	87	57h	W	119	77h	w	152	98h	ÿ	184	B8h	©	216	D8h	»	248	F8h	»
24	18h	CAN (cancel)	56	38h	8	88	58h	X	120	78h	x	153	99h	Ô	185	B9h	÷	217	D9h	»	249	F9h	÷
25	19h	EM (end of medium)	57	39h	9	89	59h	Y	121	79h	y	154	9Ah	Ü	186	BAh	»	218	DAh	»	250	FAh	»
26	1Ah	SUB (substitute)	58	3Ah	:	90	5Ah	Z	122	7Ah	z	155	9Bh	ø	187	BBh	»	219	DBh	»	251	FBh	»
27	1Bh	ESC (escape)	59	3Bh	:	91	5Bh	[123	7Bh	{	156	9Ch	£	188	BCh	»	220	DCh	»	252	FCh	»
28	1Ch	FS (file separator)	60	3Ch	<	92	5Ch	\	124	7Ch		157	9Dh	Ø	189	BDh	€	221	DDh	»	253	FDh	»
29	1Dh	GS (group separator)	61	3Dh	=	93	5Dh]	125	7Dh	}	158	9Eh	×	190	BEh	¥	222	DEh	»	254	FEh	»
30	1Eh	RS (record separator)	62	3Eh	>	94	5Eh	^	126	7Eh	~	159	9Fh	f	191	BFh	»	223	DFh	»	255	FFh	»
31	1Fh	US (unit separator)	63	3Fh	?	95	5Fh	-															
127	20h	DEL (delete)																					

اگر بخواهیم عددی را به LCD بفرستیم لازم است که ابتدا آن را توسط دستورهای موجود به کد ASCII تبدیل کنیم و سپس ارسال نماییم. برای این کار لازم است از دو تابع itoa و ftoa که در کتابخانه `<stdlib.h>` قرار دارند استفاده کنیم.

(متغیر رشته ای , متغیر عددی) itoa

برای اعداد صحیح -1

(متغیر رشته ای , تعداد اعشار , متغیر عددی) ftoa

برای اعداد اعشاری -2

char s[6]; [طول رشته ای ; نام رشته]

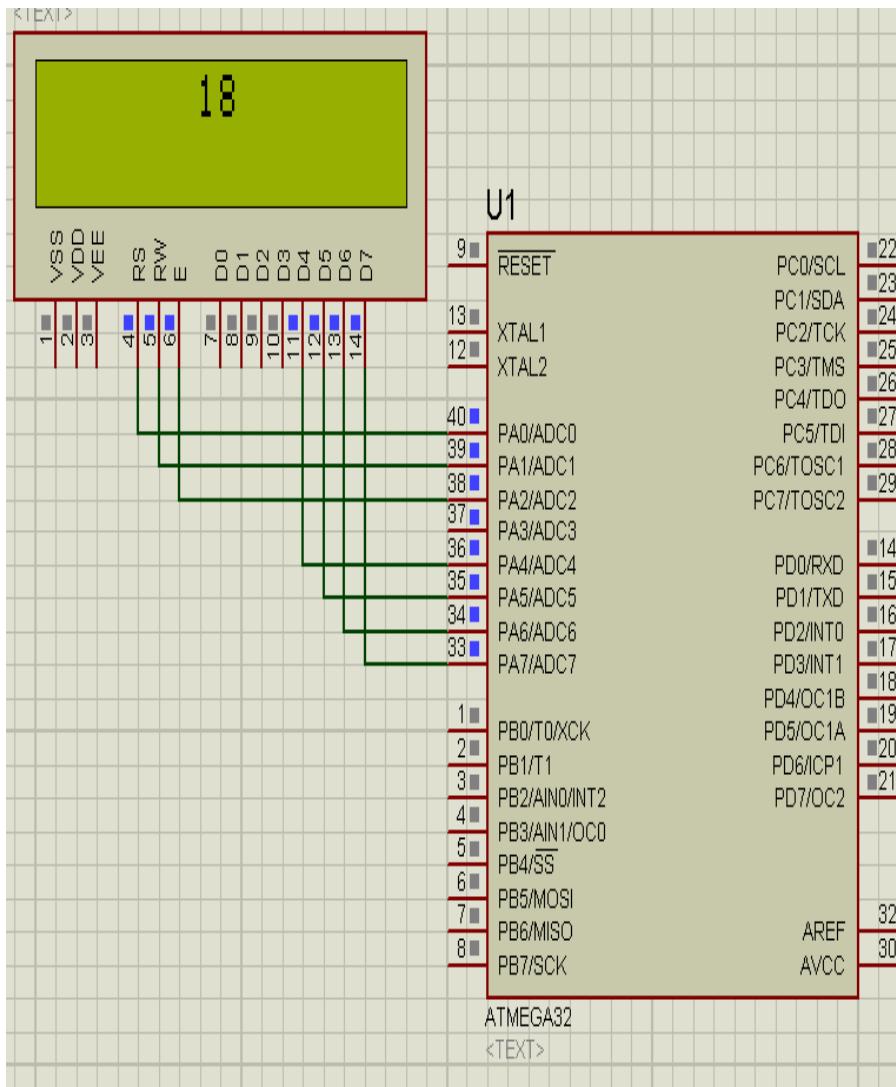
نکته: بهتر است طول رشته از تعداد ارقام بزرگترین عددی که می خواهیم نمایش دهیم ۲ خانه بیشتر

باشد . برای مثال اگر بزرگترین عدد 1023 (چهار رقم) است طول رشته را ۶ خانه وارد کنید.

char s[6];

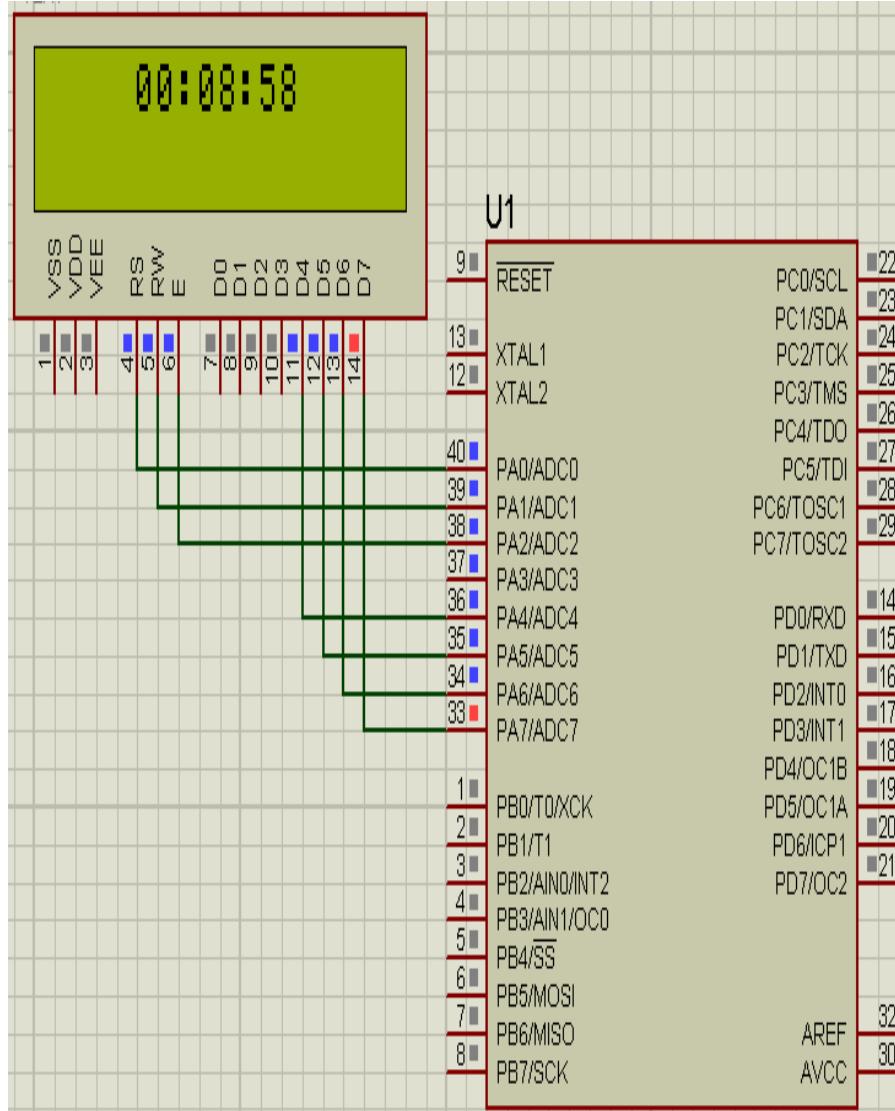
مثال: بر روی LCD یک شمارنده ۰ تا ۱۹ بسازید.

```
#include <delay.h>
#include <stdlib.h>
unsigned char i;
char s[4];
void main(void)
{
while (1)
{
for(i=0;i<20;i++)
{
itoa(i,s);
lcd_gotoxy(7,0);
lcd_puts(s);
lcd_putsf(" ");
delay_ms(500);
}}}
```



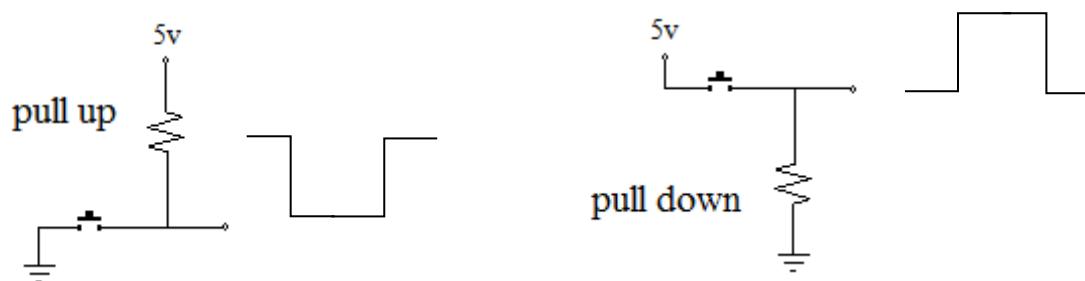
مثال: یک ساعت بر روی LCD طراحی کنید.

```
#include <stdlib.h>
#include <delay.h>
unsigned char h,m,s;
char hd[4],md[4],sd[4];
void main(void)
{
lcd_clear();
while (1)
{
s++;
if(s==60){s=0;m++;}
if(m==60){m=0;h++;}
if(h==24) h=0;
itoa(h,hd);
itoa(m,md);
itoa(s,sd);
lcd_gotoxy(4,0);
if(h<10)lcd_putchar('0');
lcd_puts(hd);
lcd_putchar(':');
if(m<10)lcd_putchar('0');
lcd_puts(md);
lcd_putchar(':');
if(s<10)lcd_putchar('0');
lcd_puts(sd);
delay_ms(50);
}
}
```

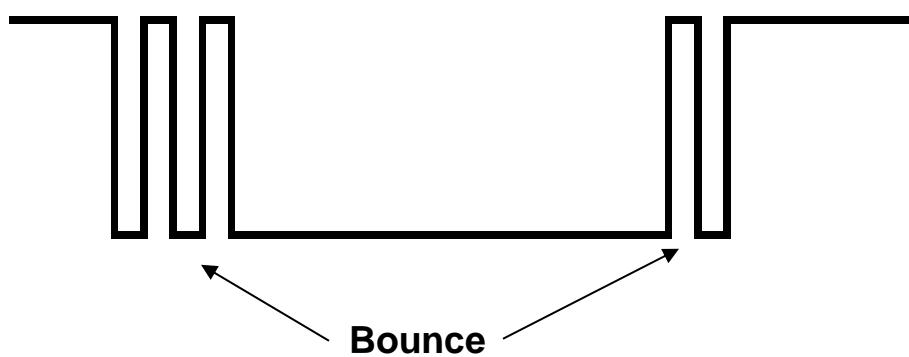


کلید (key): یکی از ساده ترین و پر کاربردترین دستگاه های ورودی کلید است. توسط کلید می توان

صفر یا یک را تولید و به مدار اعمال کرد. معمولاً دو مدار زیر برای کلیدها بسته می شوند.



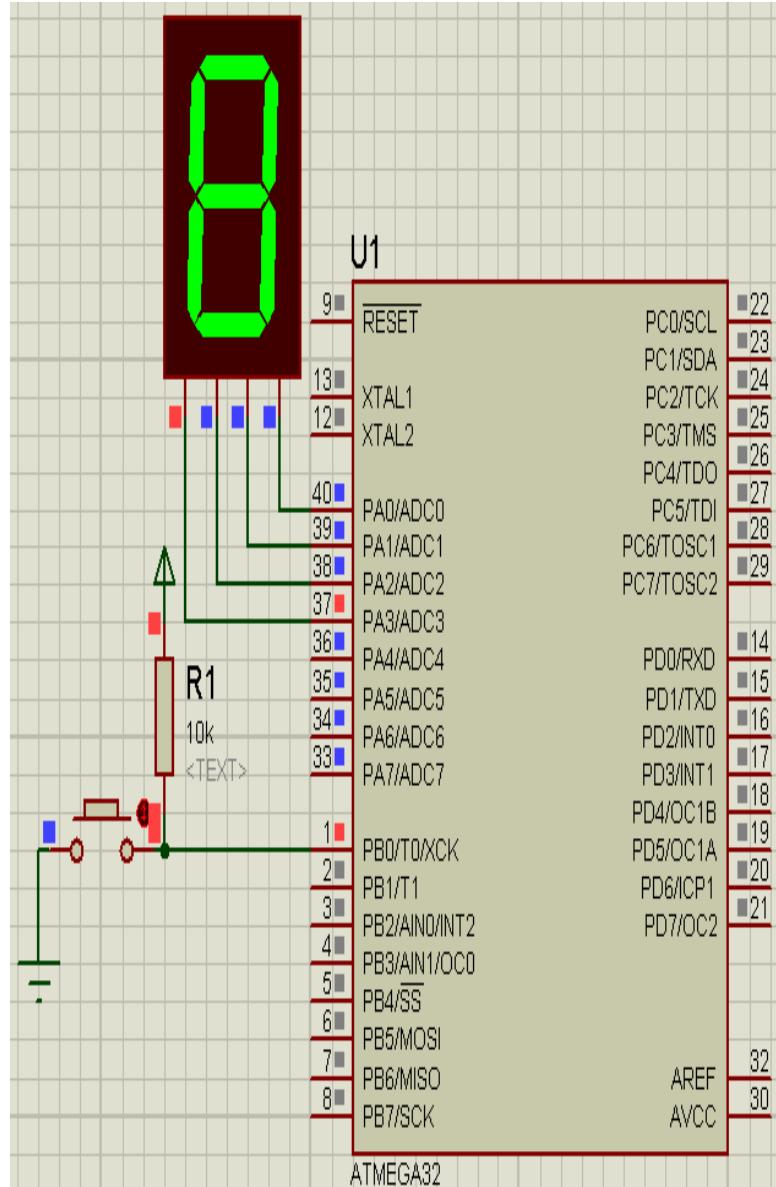
شکل های نشان داده شده بالا برای خروجی در حالت ایده آل است اما در عمل به علت لرزش یا هنگام قطع و وصل کلید پالس های اضافی خواهیم داشت که کار مدار را دچار مشکل خواهد کرد.



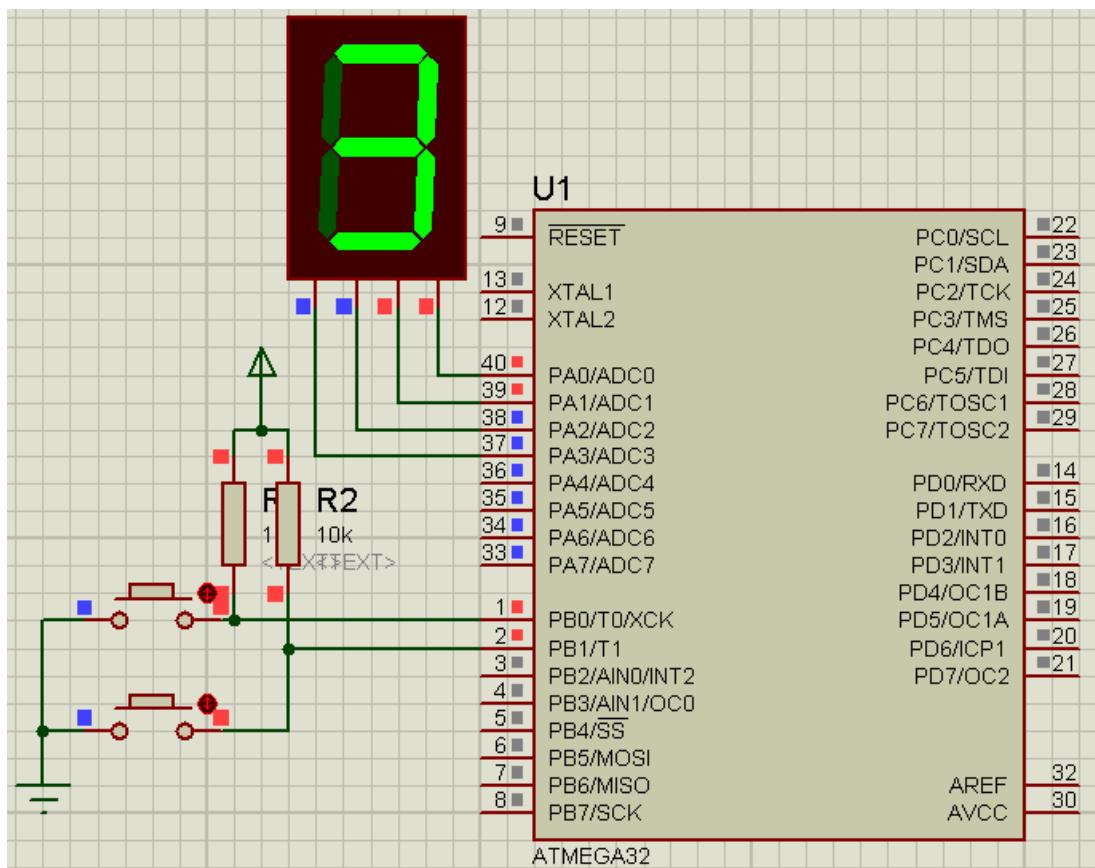
برای رفع این مشکل پس از هر تغییر در خروجی کلید حدود ۲۰ تا ۳۰ میلی ثانیه تاخیر ایجاد می کنیم تا لرزش ها تمام شود.

مثال: یک سون سگمنت BCD و یک کلید را به میکرو وصل می کنیم. برنامه ای بنویسید که با هر بار زدن کلید یک واحد به عدد خروجی اضافه شود. در ضمن حلقه شمارش ۰ تا ۹ باشد و بعد از ۹ صفر شود.

```
#include <mega32a.h>
#include <delay.h>
unsigned char i=0;
void main (void)
{
DDRA=0xFF;
PORTA=0x00;
DDRB=0x00;
while(1)
{
if(PINB.0==0)
{
delay_ms(25);
i++;
if( i==10) i=0 ;
PORTA=i;
while(PINB.0==0);
delay_ms(25);
}
}
}
```



مثال: یک کلید دیگر به مثال قبلی اضافه کنید و با آن عدد خروجی را کاهش دهید.



برای این کار کافی است برنامه زیر را به برنامه قبلی اضافه کنیم:

```
if(PINB.1==0)
{
    delay_ms(25);

    i--;
    if(i==255) i=9;

    PORTA=i;
    while(PINB.1==0);
    delay_ms(25);
}
```

مثال: مثال قبل را طوری تغییر دهید که هنگام زیاد شدن ، عدد به 9 که رسید متوقف شود و هنگام کم شدن به 0 که رسید، متوقف شود.
برای این کار کافی است خط پنجم هر دو if را به صورت زیر بنویسیم.

```
if(i==10) i=9;
```

```
if(i==255) i=0;
```

تمرين: يك کلید به **PB.0** و يك **Buzzer** به **PD.7** متصل است برنامه اي بنویسید که با نگه داشتن کلید ، **Buzzer** روشن و بوق بزند و با رها شدن **Buzzer** قطع شود.

تمرين: دو عدد کلید به **PD.2** و **PD.3** و يك **Buzzer** به **PD.7** متصل است برنامه اي بنویسید که با زدن کلید **PD.2** **Buzzer** روشن و بوق بزند و با کلید **PD.3** قطع شود.

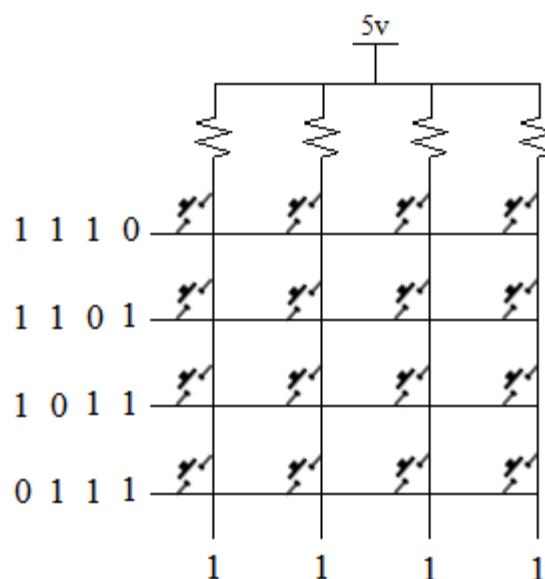
تمرين: هشت عدد **led** به **PORTA** و دو عدد کلید به **PD.2** و **PD.3** متصل کنید برنامه اي بنویسید که با نگه داشتن يکی از کلید ها روی **PORTA** شمارنده حلقوی و با کلید دیگر شمارنده جانسون ، نمایش داده شود .توجه داشته باشید با رها شدن کلید ، نمایش متوقف شود.

تمرين: يك **lcd** به **PORTA** و دو عدد کلید به **PD.2** و **PD.3** متصل کنید برنامه اي بنویسید که با نگه داشتن يکی از کلید ها عدد روی **lcd** با سرعت مناسب زیاد و با رسیدن به عدد **20** متوقف شود و دیگر زیاد نشود و با کلید دیگر کاهش یابد و با رسیدن به عدد **0** متوقف شود. در ضمن با هر بار افزایش يا کاهش عدد ، **Buzzer** متصل به **PD.7** نیز بوق کوتاهی بزند.

تمرين: يك **lcd** به **PORTA** و سه عدد کلید به **PD.2** و **PD.3** و **PB.0** و يك **Buzzer** به **PD.7** متصل کنید برنامه اي بنویسید که روی خط اول **lcd** ساعت و روی خط دوم **lcd** زمان آلام را داشته باشیم . کاربر بتواند با کلید **PB.0** مشخص کند کدام پارامتر باید تغییر کند و روی **lcd** آن عدد چشمک زن شود ، و با دو کلید **PD2,3** عدد کم يا زیاد شود . و هر گاه زمان تنظیم شده آلام با ساعت برابر شد **Buzzer** بطور مناسب بوق بزند.

صفحه کلید: یکی دیگر از وسایل ورودی صفحه کلید می باشد که در آن کلیدها به صورت ماتریسی

چیده شده اند.



برای خواندن صفحه کلید به شکل زیر عمل شود:

ابتدا یک سطر را صفر و بقیه را یک می کنیم، سپس ستون ها را می خوانیم. اگر همه ی آن ها یک باشند، یعنی در آن سطر کلیدی زده نشده است. آن سطر را یک و سطر بعدی را صفر می کنیم و مجددا ستون هارا می خوانیم. این کار را برای همه ی سطراها انجام می دهیم. اگر سطري را صفر کردیم و ستونی صفر شد با توجه به سطر و ستونی که صفر شده، کلید فعال مشخص می شود.

مثال: یک کیبورد تلفنی و یک سون سگمنت BCD را به میکرو متصل کنید. برنامه ای بنویسید که با زدن هر کلید عدد متناظر با آن در خروجی دیده شود. (برای * عدد ۱۰ و برای # عدد ۱۱ نمایش داده شود).

نکته: توجه داشته باشید که روی ستون ها از Pull-Up داخلی استفاده شده است.

```
#include <mega32a.h>
```

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

```
unsigned char k;
```

```
void main (void)
```

```
{
```

```
DDRC=0xFF;
```

```
DDRD=0xF0;
```

```
while(1)
```

```
{
```

```
k=12;
```

```
PORTD=0xFF;
```

```
//---ROW1---//
```

```
PORTD.4=0;
```

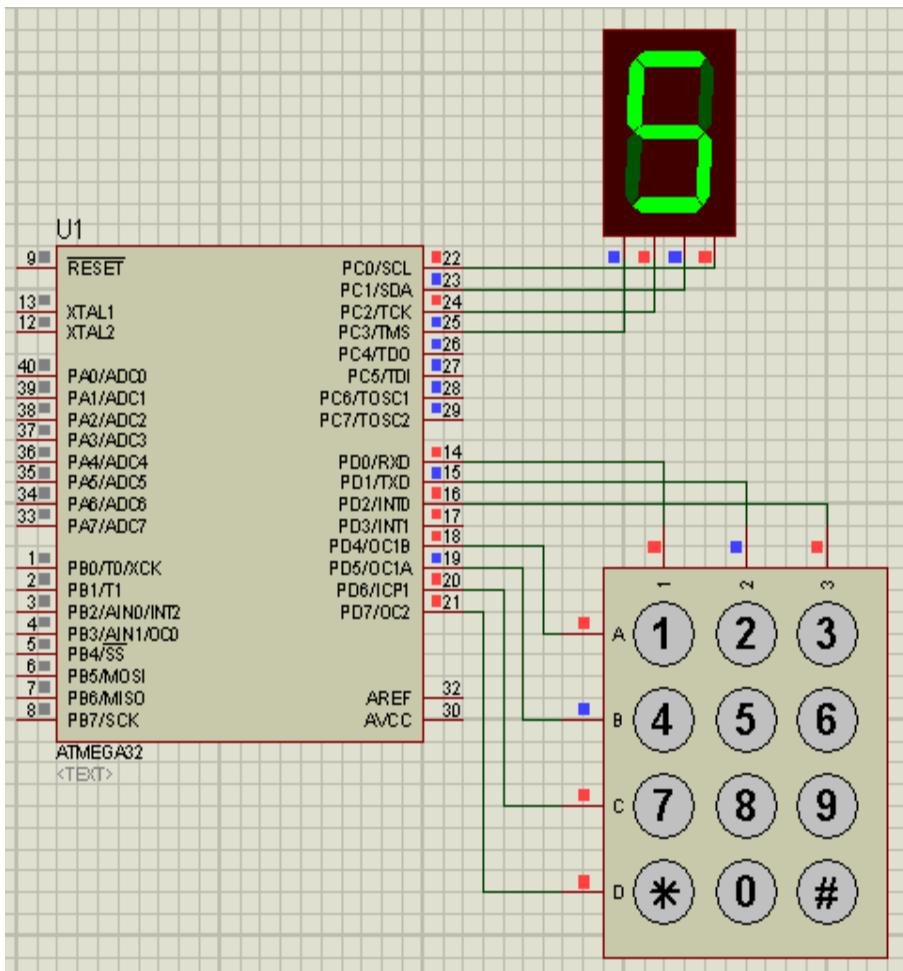
```
delay_ms(3);
```

```
if(PIND.0==0){k=1;while(PIND.0==0);}

if(PIND.1==0){k=2;while(PIND.1==0);}

if(PIND.2==0){k=3;while(PIND.2==0);}

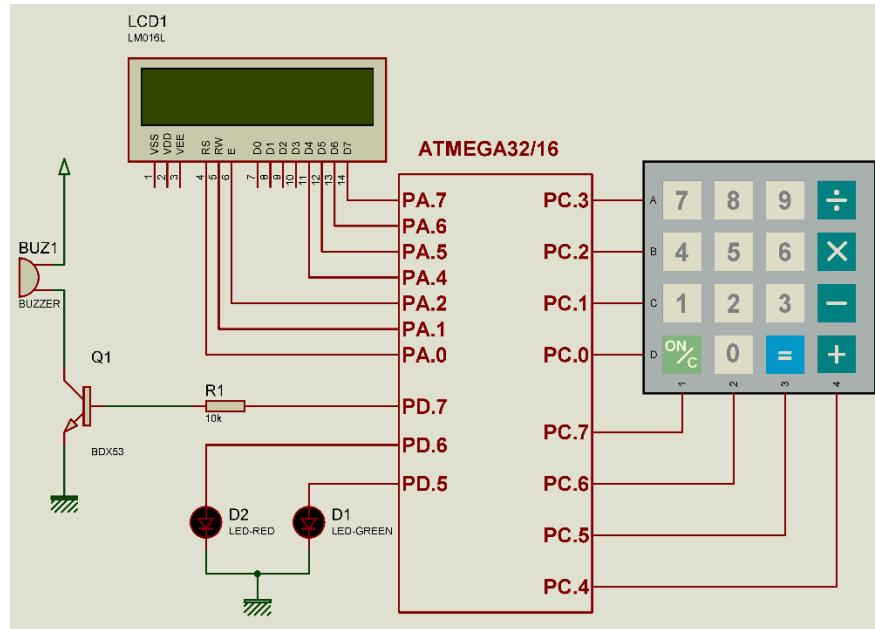
PORTD.4=1;
```



```
//---ROW2---//  
PORTD.5=0;  
delay_ms(3);  
if(PIND.0==0){k=4;while(PIND.0==0);}  
if(PIND.1==0){k=5;while(PIND.1==0);}  
if(PIND.2==0){k=6;while(PIND.2==0);}  
PORTD.5=1;  
//---ROW3---//  
PORTD.6=0;  
delay_ms(3);  
if(PIND.0==0){k=7;while(PIND.0==0);}  
if(PIND.1==0){k=8;while(PIND.1==0);}  
if(PIND.2==0){k=9;while(PIND.2==0);}  
PORTD.6=1;  
//---ROW4---//  
PORTD.7=0;  
delay_ms(3);  
if(PIND.0==0){k=10;while(PIND.0==0);}  
if(PIND.1==0){k=0;while(PIND.1==0);}  
if(PIND.2==0){k=11;while(PIND.2==0);}  
PORTD.7=1;  
if(k!=12)PORTC=k;  
}}}
```

تمرين: يك صفحه کلید ۴*۴ و يك LCD کاراكتري را مطابق شكل زير به ميكرو متصل کرده برنامه اي بنويسيد که با زدن هر کلید عدد يا علامت روی کلید بر روی LCD نمايش داده شود با زدن کلید

نيز LCD ON/C پاک شود.



تمرين: برنامه صفحه کلید را طوري تكميل کنيد که با زدن هر کلید علاوه بر نمايش عدد و علامت نيز بوق کوتاهي بزند.

تمرين: يك ماشين حساب بسازيد.

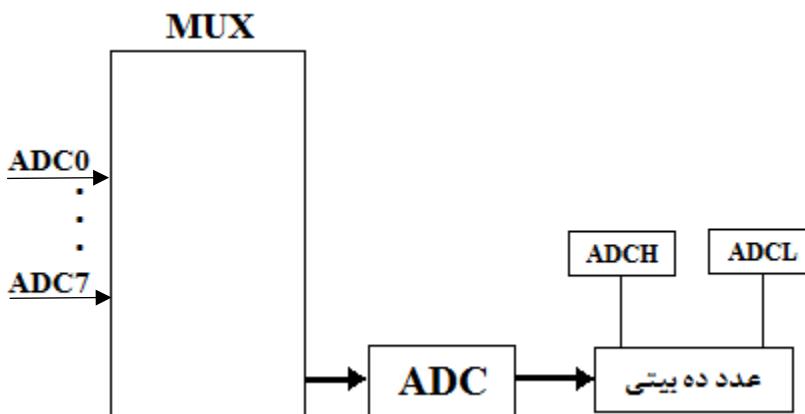
الف) يك رقمي

ب) چند رقمي

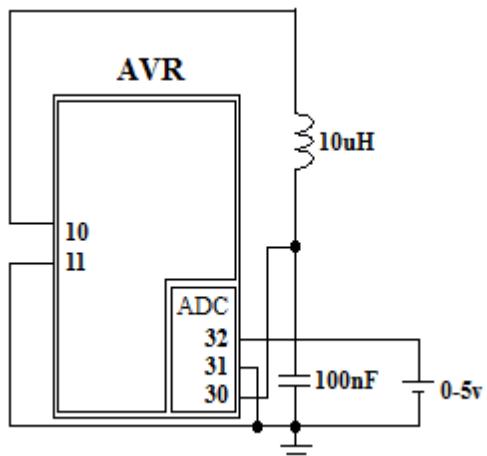
تمرين: يك قفل رمز بسازيد. که کاربر يك رمز چهار رقمي را وارد کند اگر رمز صحيح بود led سبز روشن شود و buzzer يك بار بوق بزند، و اگر غلط بود led قرمز روشن شود و buzzer سه بار بوق بزند .

تمرين: تمرين قبل را طوري كامل کنيد که مدیر بتواند رمز دستگاه را نيز عوض کند . در ضمن هنگامي که کاربران رمز را وارد مي کنند اعداد زده شده بصورت * روی lcd ديده شوند. همچنين اگر کاربر رمز ورودی را سه بار اشتباه وارد کرد ، کيبورد قفل ، و هر دو led روشن شوند.

مبدل آنالوگ به دیجیتال (ADC): یکی دیگر از قسمت های جانبی این میکرو واحد ADC می باشد. می دانیم که اکثر کمیت های اطراف ما مقادیری آنالوگ هستند؛ مانند دما، فشار و... که اگر بخواهیم آن ها را اندازه گیری و پردازش نماییم، لازم است که ابتدا به یک مقدار دیجیتال تبدیل و سپس پردازش گردد. در این میکرو یک ADC هشت کanal و ۱۰ بیتی پیش بینی شده است.



تغذیه ADC: برای بالا بردن دقت ADC ولتاژ تغذیه این قسمت را از بقیه میکرو جدا کرده اند که می توانید ولتاژ جدا یا با قطعات زیر به ولتاژ اصلی وصل کنید.



- $V_{CC} = 10$
- $GND = 11 \& 31$
- $Aref = 32$
- $AV_{CC} = 30$

ولتاژ مرجع: در این نوع مبدل نیاز به یک ولتاژ مرجع داریم که می توان آن را از سه طریق تامین نمود:

۱. ولتاژ تغذیه: یعنی AV_{CC} مرجع باشد. ($5V$)
۲. ولتاژ منبع خارجی: یعنی $Aref$ (۰ تا $5V$).
۳. ولتاژ منبع داخلی: یعنی $2.56V$ (یک خازن در حد $1\mu F$ را روی پایه $Aref$ قرار دهید)

ضریب تفکیک: پارامتری است که مشخص می کند حساسیت یا دقت ADC چقدر است و از رابطه زیر محاسبه می شود.

$$\text{ضریب تفکیک} = \frac{V_{ref}}{2^n - 1}$$

عدد خروجی: عدد خروجی مبدل را نیز می توان از رابطه زیر محاسبه کرد.

$$\text{عدد خروجی مبدل} = \frac{V_{in}}{\text{ضریب تفکیک}}$$

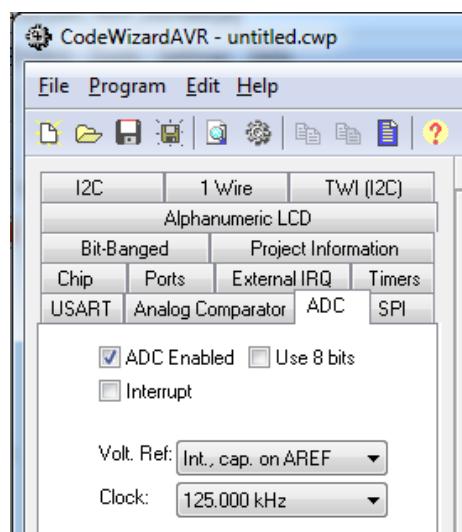
مثال: اگر ولتاژ ورودی به مبدل، یک ولت و ولتاژ مرجع ، داخلی و برابر 2.56V باشد، مطلوب است:

$$\frac{2.56}{2^{10} - 1} = 2.5mv$$

$$\frac{1}{2.5mv} = 400$$

۱. ضریب تفکیک

۲. عدد خروجی مبدل



نکته: برای تنظیم ADC در ویزارد برگه‌ی ADC را باز و آن را فعال می کنیم. در قسمت volt. ref مشخص می کنیم ولتاژ مرجع از کدام منبع تامین شود.

نکته: وقتی از مرجع داخلی استفاده می شود، باید یک خازن که طرف دیگر آن به زمین متصل است، در حد 1uF بر روی پایه Aref قرار گیرد.

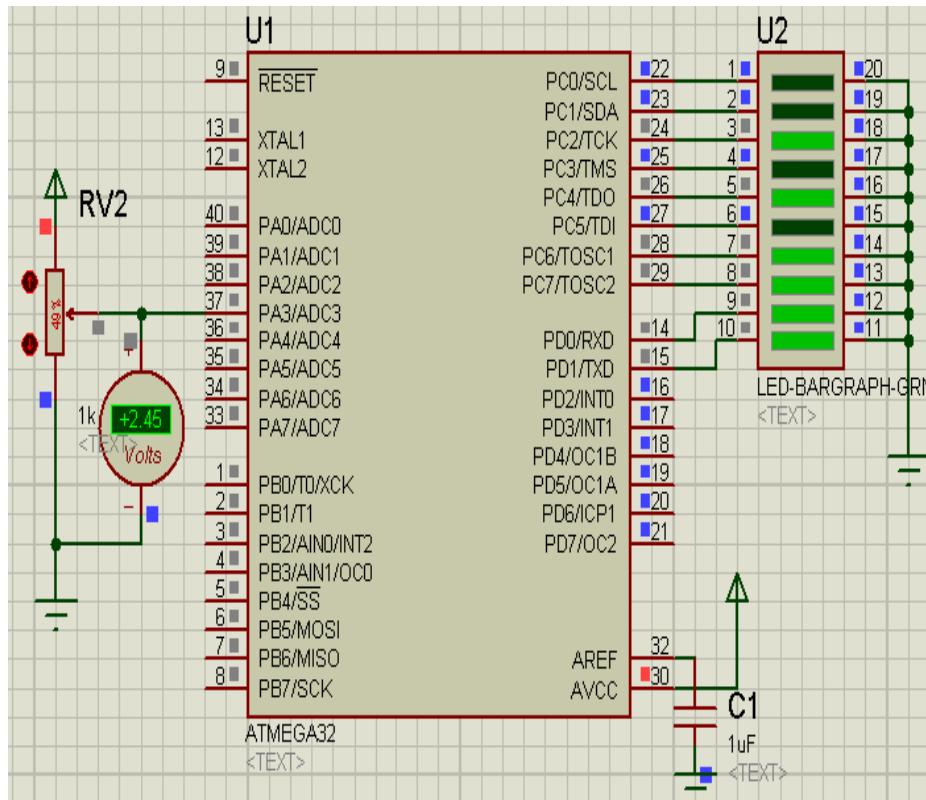
در قسمت Clock فرکانس ورودی به قسمت ADC را مشخص می کنیم. لازم است که این فرکانس بین 50k تا 200k هرتز باشد. در صورت وجود ، قسمت Trigger را نیز روی ADC Stopped قرار دهید.

مثال: ۱۰ عدد LED را به پورت های C و D و یک ولتاژ متغیر را به کanal (3) ADC متصل کنید.

برنامه ای بنویسید که ولتاژ کanal ۳ را به عدد دیجیتال تبدیل و روی LED ها نمایش دهد.

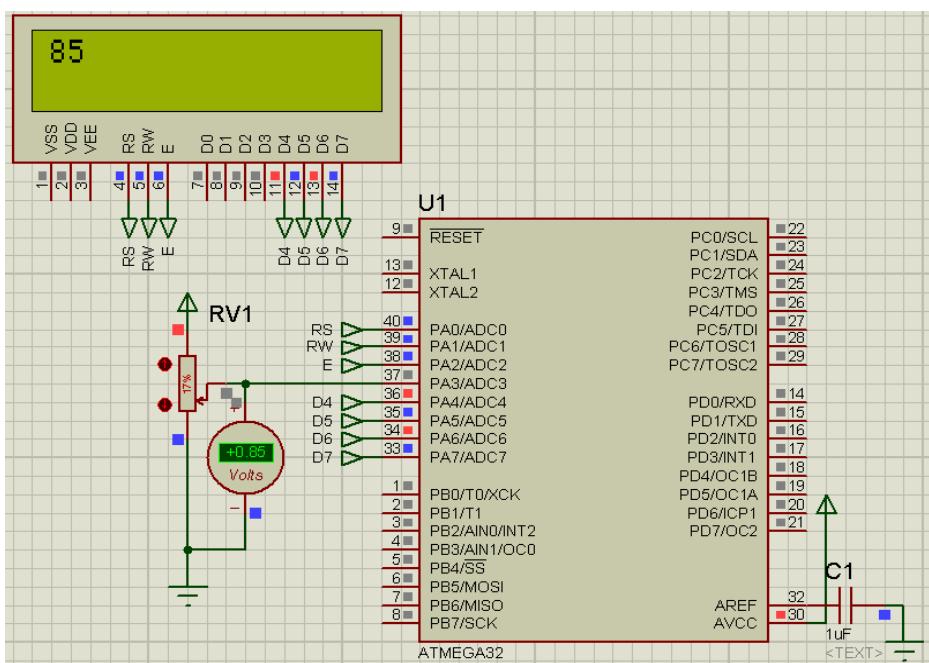
پس از تولید کد ملاحظه می شود تابع `read_adc` به برنامه اضافه شده است. این تابع مشخص می کند که از کدام کanal مقدار آنالوگ خوانده شود و خروجی آن عدد خروجی مبدل است.

```
void main(void)
{
    while (1)
    {
        read_adc(3);
        PORTC=ADCL;
        PORTD=ADCH;
    }
}
```



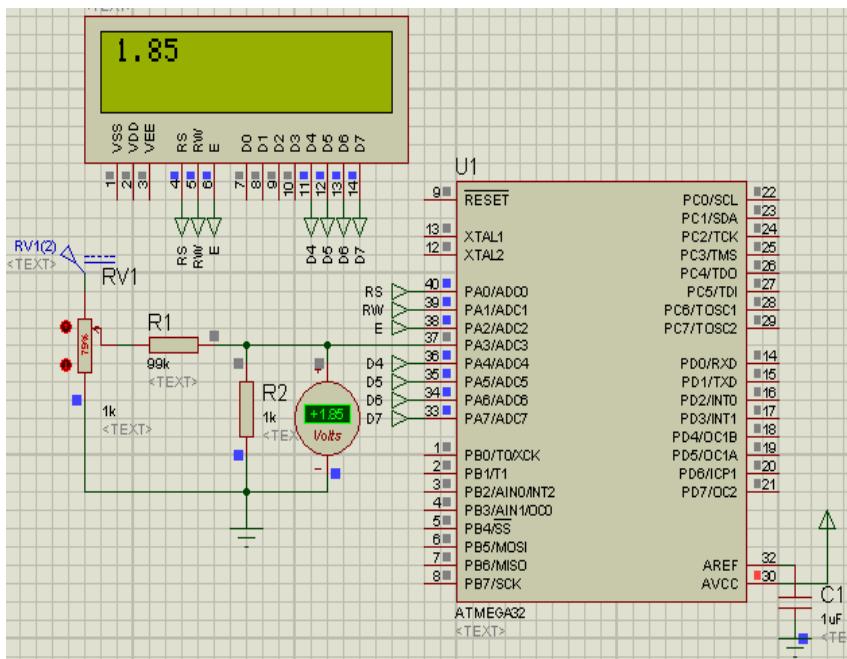
مثال: در مثال قبل یک LCD به پورت A متصل کنید و عدد خروجی مبدل را بر روی آن نمایش دهید.

```
#include <stdlib.h>
#include <delay.h>
unsigned int a;
char s[6];
void main(void)
{
while (1)
{
a=read_adc(3);
itoa(a,s);
lcd_gotoxy(0,0);
lcd_puts(s);
lcd_putsf("  ");
delay_ms(200);
}
}
```



مثال: یک عدد LCD به پورت A متصل کرده و با استفاده از کانال ۳ یک ولتمتر بسازید که مقدار ولتاژ را بر روی LCD نمایش دهد.

```
#include <stdlib.h>
unsigned int a;
float b;
char s[5];
void main(void)
{
while (1)
{
a=read_adc(3);
b=(float)a/400;
}
```



```

ftoa(b,2,s);

lcd_gotoxy(0,0);

lcd_puts(s);

lcd_putsf("  ");

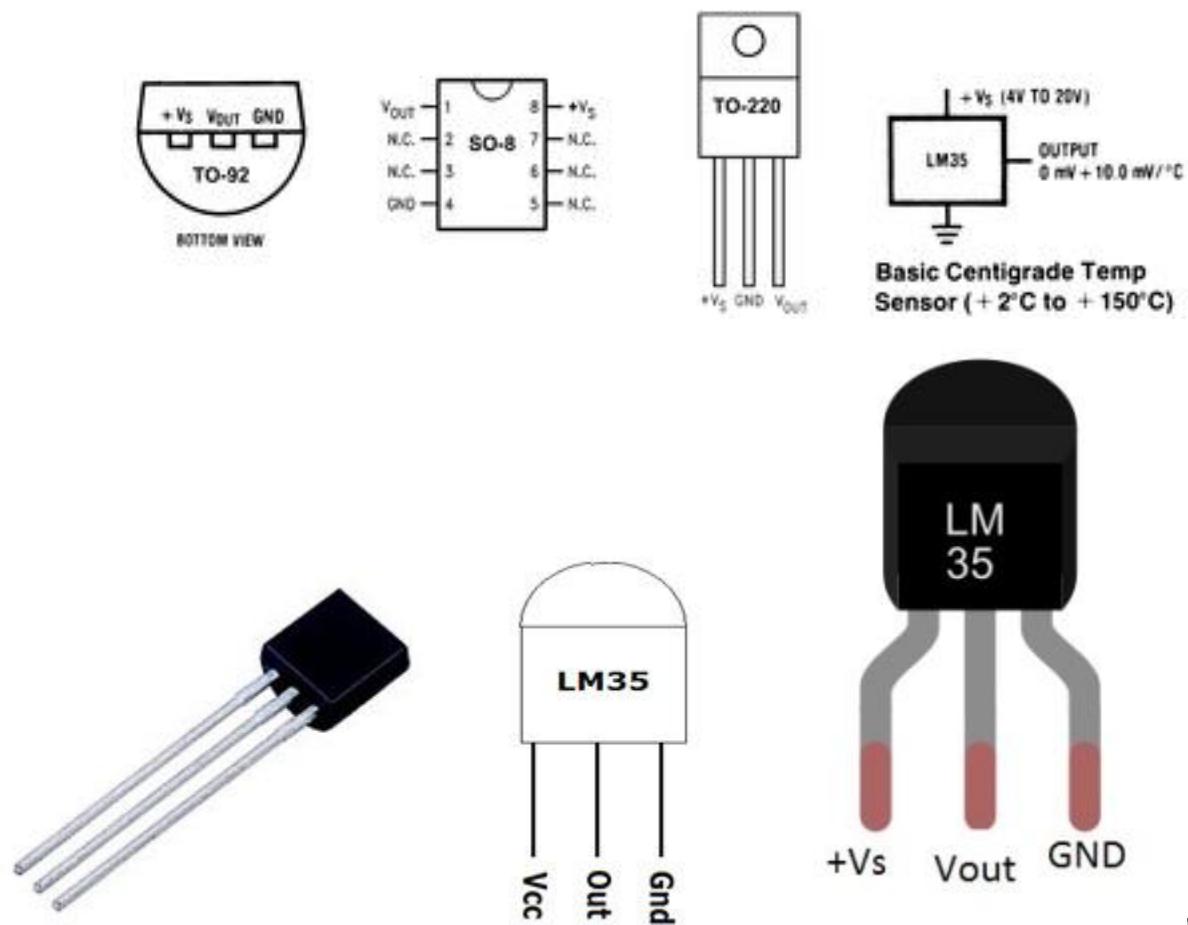
delay_ms(100);

}

}

```

سنسور LM35: این سنسور مبدل دما به ولتاژ است و به ازای هر درجه دما 10mv خروجی دارد؛
معنی خروجی آن دارای حساسیت 10mv بر درجه سانتیگراد است. برای مثال اگر دمای محیط ۱۵ درجه سانتیگراد باشد، خروجی سنسور 150mv خواهد بود.



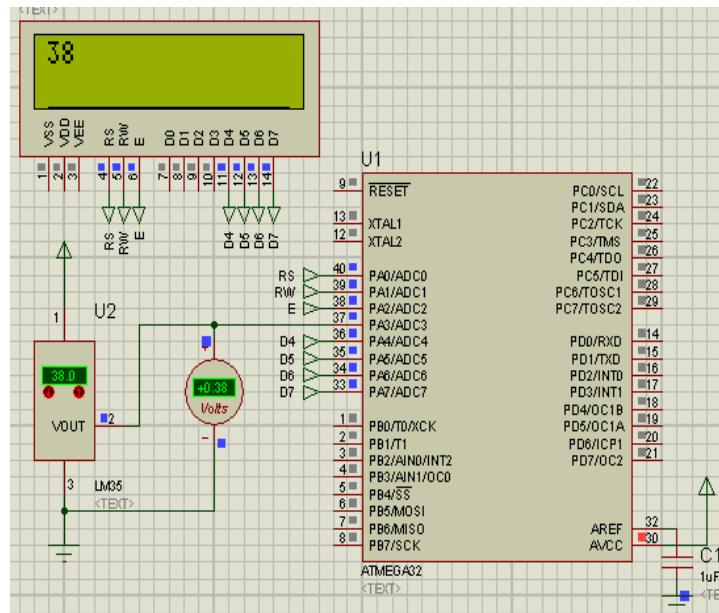
مثال: یک عدد LCD و سنسور LM35 به پورت A متصل کنید و برنامه‌ای بنویسید تا دمای محیط بر روی LCD نمایش داده شود. ولتاژ مرجع داخلی و برابر $V = 2.56$ فرض شود.

پاسخ : فرض می‌کنیم دما یک درجه سانتیگراد باشد با توجه به ضریب تفکیک $2.5\text{mV}/2.5\text{mV} = 4$ عدد خروجی مبدل 4 خواهد شد.

با توجه به محاسبه بالا هر عددی که خوانده شد را بر 4 تقسیم می‌کنیم تا دما بدست آید.

و برای برنامه کافی است در مثال بالا ولتمتر، بجائی 400 عدد خروجی مبدل را به 4 تقسیم کنیم.

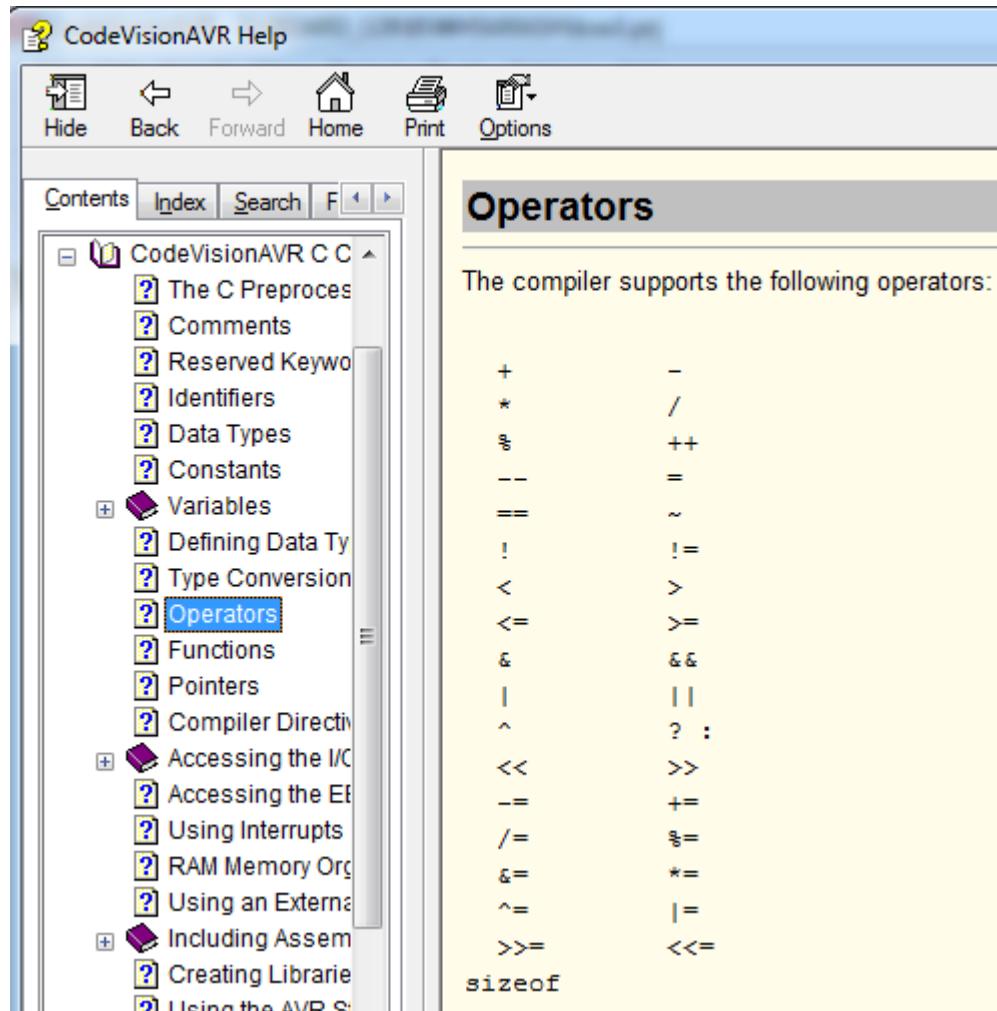
$$b = (\text{float}) a / 4;$$



OPERATORS

عملگرهایی که می توانید در زبان C استفاده کنید عبارتند از (جدول موجود در صفحه operators

در help برنامه):



+ ، - ، * : عملگر های جمع، تفریق و ضرب نکته خاصی ندارد.

/ ، % : عملگر های تقسیم معمولی و باقی مانده. در مثال زیر تفاوت و نتیجه هر یک را مشاهده می کنید.

$$13 / 5 = 2$$

$$13 \% 5 = 3$$

-- ، ++ : عملگر های افزایش و کاهش به مقدار یک واحد.

$$a=5 \quad a++; \longrightarrow \quad a=6$$

$$a=5 \quad a--; \longrightarrow \quad a=4$$

: عملگر های انتساب ، شرط مساوی و شرط نامساوی :

تک مساوی مقداری را به یک متغیر نسبت می دهد، و جفت مساوی بررسی می کند که آیا دو مقدار با هم مساوی هستند یا خیر، و علامت تعجب ! و مساوی بررسی می کند که آیا دو مقدار با هم نامساوی هستند یا خیر .

در دستور مقابله مقدار 5 در متغیر K قرار می گیرد.

در دستور زیر مقدار K با عدد 5 مقایسه می شود اگر **برابر باشند** مقدار یک یا TRUE و در صورتی که برابر نباشند مقدار صفر یا FALSE نتیجه دستور می باشد.

در دستور زیر مقدار K با عدد 5 مقایسه می شود اگر **برابر باشند** مقدار یک یا TRUE و در صورتی که برابر باشند مقدار صفر یا FALSE نتیجه دستور می باشد.

: عملگر های بیتی و منطقی :

عملگر	بیتی	منطقی
NOT	~	!
AND	&	&&
OR		
XOR	^	ندارد

در عملگرهای بیتی ابتدا عملوندها را بصورت باینری نوشته و سپس بیت به بیت عمل مورد نظر را بر روی بیت ها انجام می شود.

مثال: اگر a=0x56 و b=0x9C باشد، مقادیر زیر را به دست آورید.

PORTC=a & b; PORTC=a | b; PORTC=a ^ b; PORTC=~a;

a=0x56 01010110

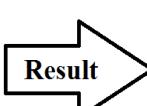
b=0x9C 10011100

a&b 00010100



PORTC=00010100=0x14

a=0x56 01010110
 b=0x9C 10011100  PORTC=11011110=0xDE
 a|b 11011110

a=0x56 01010110
 b=0x9C 10011100  PORTC=00010100=0xCA
 a^b 11001010

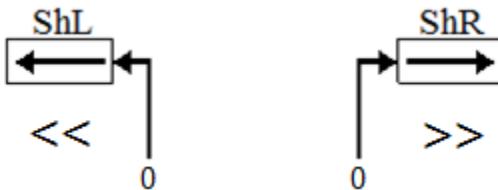
a=0x56 01010110
 ~a 10101001  PORTC=10101001=0xA9

مثال: اگر a=5 و b=8 باشد، پورت C چه عددی را نمایش می دهد؟

در این مثال به دلیل برقرار نبودن شرط اول else اجرا می شود و خروجی عدد 55 را نمایش می دهد.

```
if ((a>6)&&(b<10))      PORTC=99;
else
  PORTC=55;
```

شیفت به چپ و راست:



مثال: اگر b=0xCF و a=0xFE باشد. حاصل عبارت زیر را بدست آورید.

PORTC=((a>>3)&(b<<2)) a=11001111 b=11111110
 a>>3=00011001 & b<<2=11111000  PORTC=00011000=0x18

<، > عملگر های شرط بزرگتر و شرط کوچکتر :

مثال: اگر $a=5$ و $b=8$ باشد، پورت C چه عددی را نمایش می دهد؟

در این مثال به دلیل برقرار بودن شرط دستور PORTC=99 اجرا می شود.

```
if(a<b)
PORTC=99;
else
PORTC=55;
```

؟ عملگر شرطی:

این عملگر مانند دستور if else عمل می کند.

(دستور دوم) : (دستور اول) ? (عبارت شرطی)

اگر شرط برقرار باشد دستور اول و در غیر این صورت دستور دوم اجرا می شود.

(a>b) ? (PORTC=99) : (PORTC=55)

عملگرهای تخصیص مرکب:

توسط این روش می توان عبارات محاسبه ای را بصورت خلاصه نوشت.

```
a=a+5    a+=5
b=b*7    b*=7
c=c&8    c&=8
d=d<<4   d<<=4
```

: sizeof

خروجی این عملگر مقدار حافظه ای است که یک متغیر بر حسب بایت اشغال می کند.

char a; x=sizeof (a) نتیجه → x=1

int b; x=sizeof (b) نتیجه → x=2

float c; x=sizeof (c) نتیجه → x=4

تمرين: برنامه بنويسيد که عددی را از PORTA دریافت کند ، اگر فرد بود روی PORTC عدد 1 و اگر عدد زوج بود عدد 2 را نمایش دهد .

تمرين: برنامه بنويسيد که یک عددی باينری را از PORTA دریافت کند ، و دهدھی آن را روی پورت های b,c,d نمایش دهد. به شکل مثال زیر:

PORTA	PORTB	PORTC	PORTD
10110110	1	8	2

تمرين: برنامه بنويسيد که یک عددی باينری را از PORTA دریافت کند ، و تعداد صفرهای آن را روی PORTC و تعداد يك های آن را روی PORTD نمایش دهد. به شکل مثال زیر:

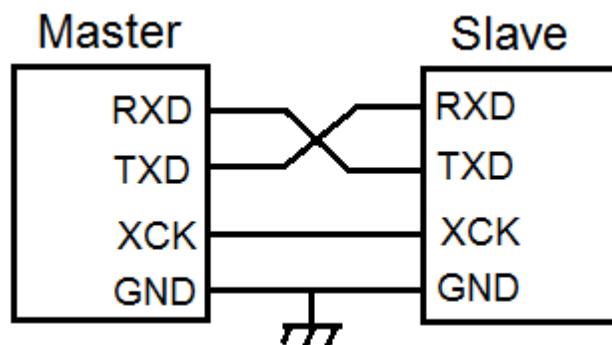
PORTA	PORTC	PORTD
10110110	3	5

تمرين: برنامه بنويسيد که یک عددی باينری را از PORTA دریافت کند ، فرض کنيد که خروجي سه سنسور به اين پورت متصل شده عدد ارسالی هر سنسور را جدا کرد و به پورت های b,c,d ارسال و نمایش دهيد. به شکل مثال زير:

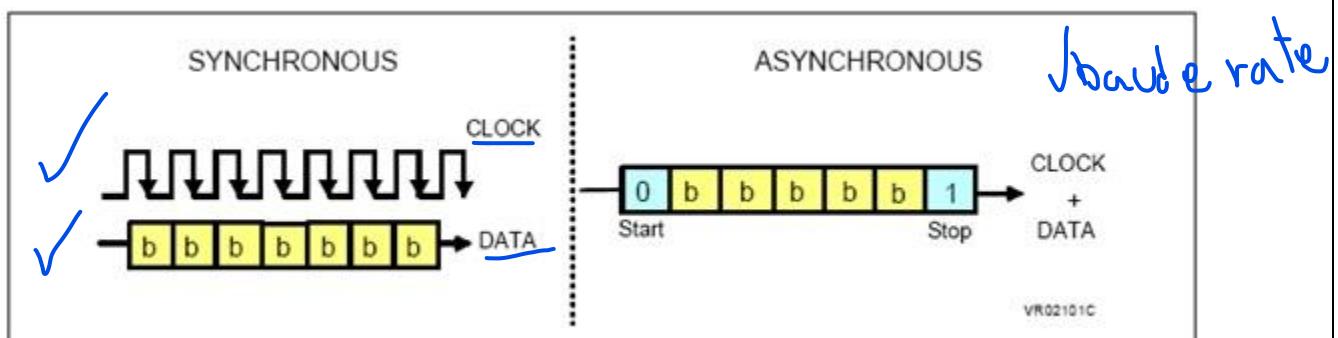
PORTA			PORTB	PORTC	PORTD
S1	S2	S3	S1	S2	S3
10	101	111	2	5	7

The Universal Synchronous and Asynchronous serial Receiver and Transmitter :USART

در این نوع ارتباط سیم بندی زیر را داریم .



در حالت سنکرون دیتا روی لبه بالا یا پایین رونده پالس ساعت خوانده می شود.



قاب دیتا : Frame Format

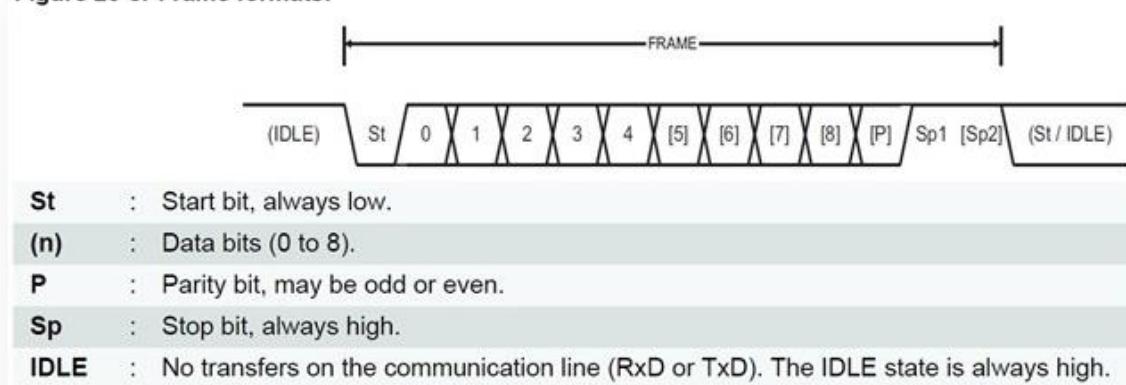
هنگام ارسال دیتا در حالت **Asynchronous** علاوه بر دیتا ، بیت های کنترلی نیز همراه با آن ارسال می شود به مجموعه بیت های کنترلی و دیتا فریم **Frame** گفته می شود. این بیت های کنترلی عبارتند از:

بیت شروع **Start bit** : این بیت که همواره صفر است به گیرنده شروع ارسال دیتا را خبر می دهد.

بیت پایان **Stop bit** : این بیت که همواره یک است به گیرنده اتمام ارسال دیتا را خبر می دهد. تعداد این بیت می تواند یک بیت ، دو بیت و در بعضی سیستم ها یک و نیم بیت باشد.

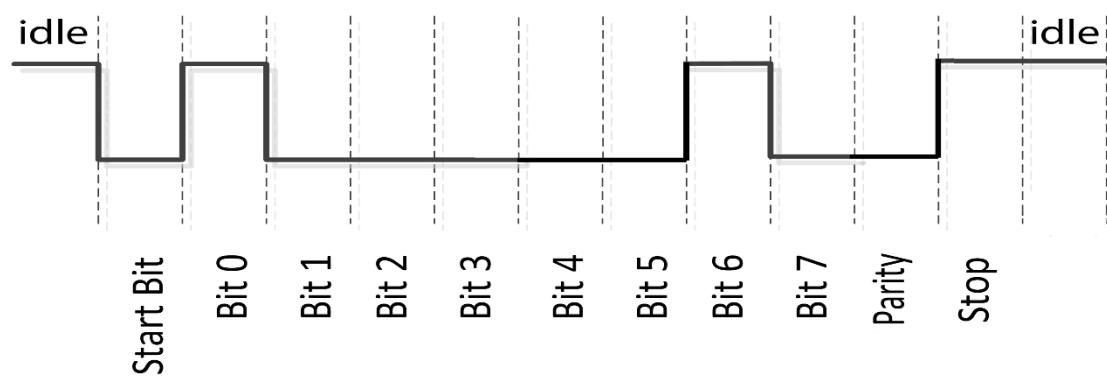
بیت توازن **Parity bit** : این بیت که برای آشکارسازی خط استفاده می شود می تواند توازن زوج یا فرد باشد.

Figure 23-5. Frame formats.



مثال: فرض کنید می خواهیم کد اسکی حرف A را با توازن زوج و یک بیت توقف ارسال کنیم قاب دیتای آن رارسم کنید . (کد اسکی $A = 65 = 01000001$)

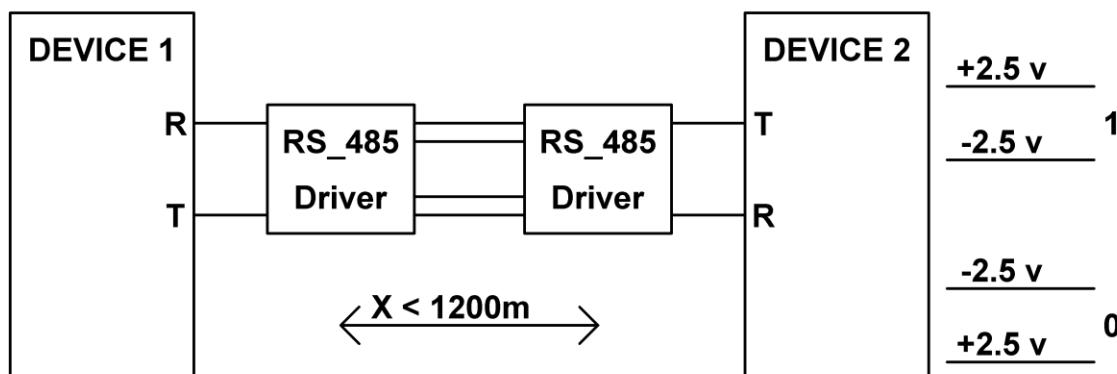
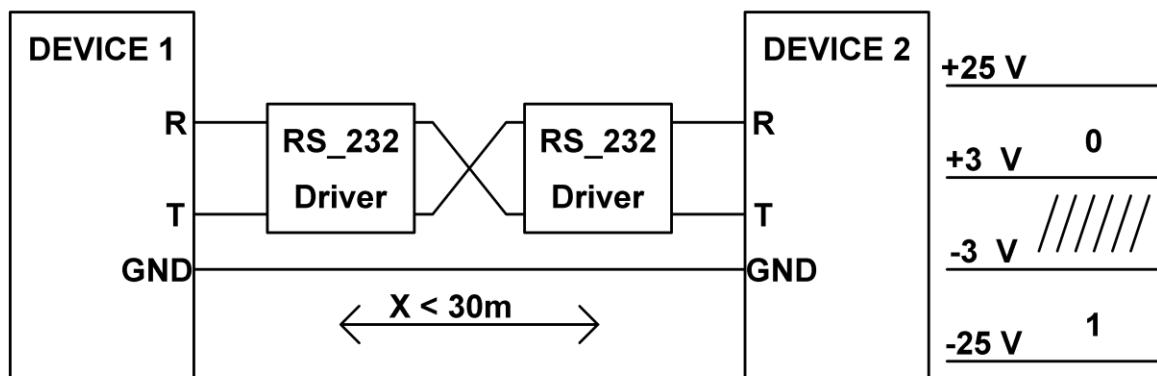
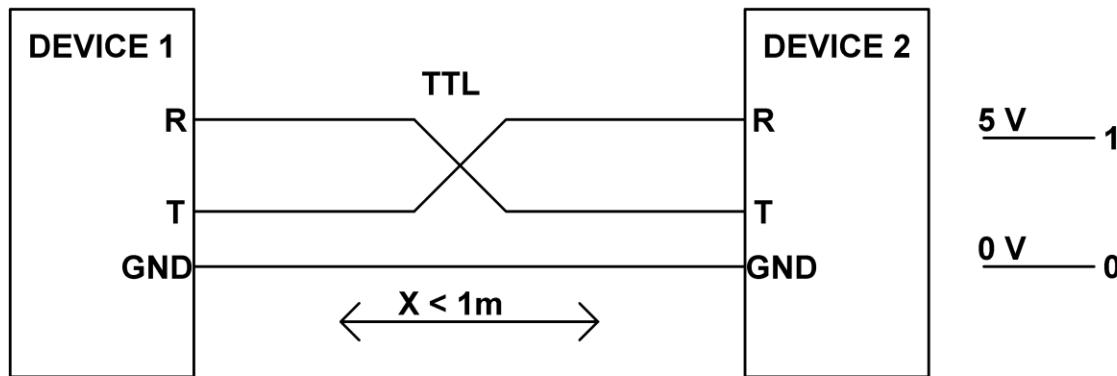
0x41, 801 (8 Data bits, Odd Parity, 1 Stop)



روش های ارتباط سخت افزاری بین دو دستگاه در **UART**

سه روش **TTL**، **RS485** و **RS232** پر کاربردتر می باشند.

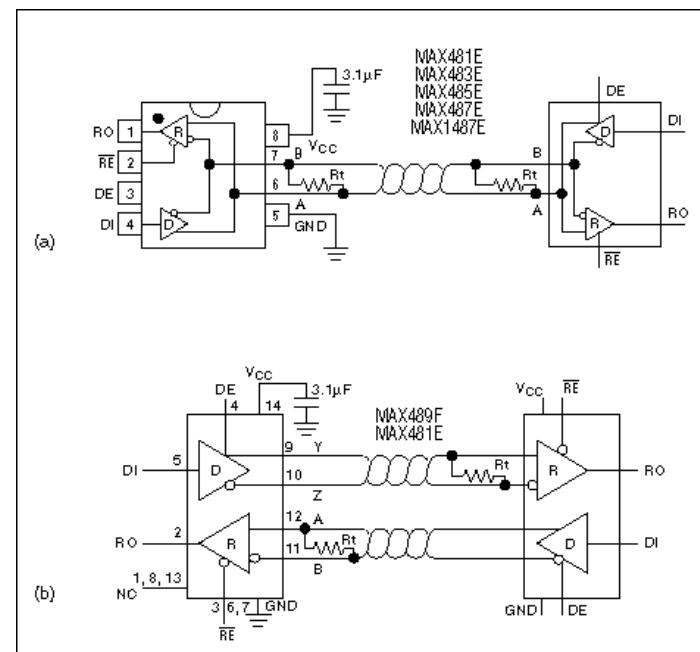
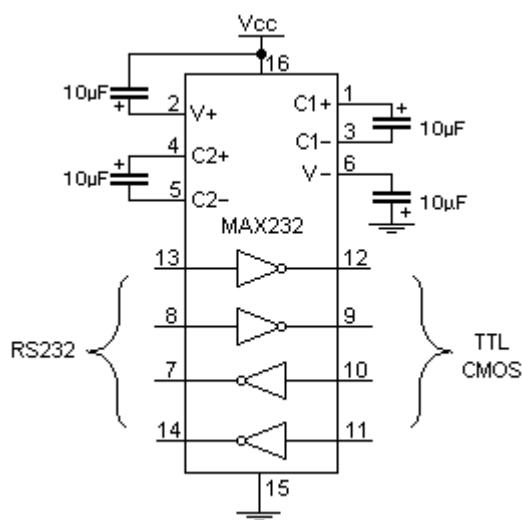
توجه داشته باشید فاصله های مشخص شده بین فرستنده و گیرنده حالت عمومی دارد و با توجه به نویز محیط ، جنس کابل و سرعت انتقال دیتا **Baud Rate** تغییر می کند.



Data rates and the maximum distances recommended in RS-232

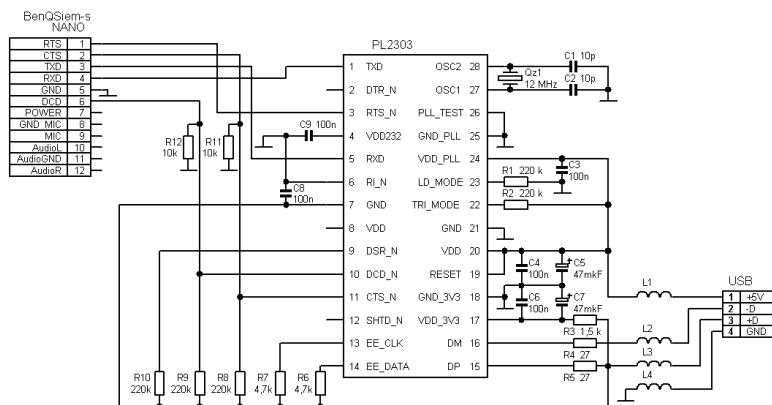
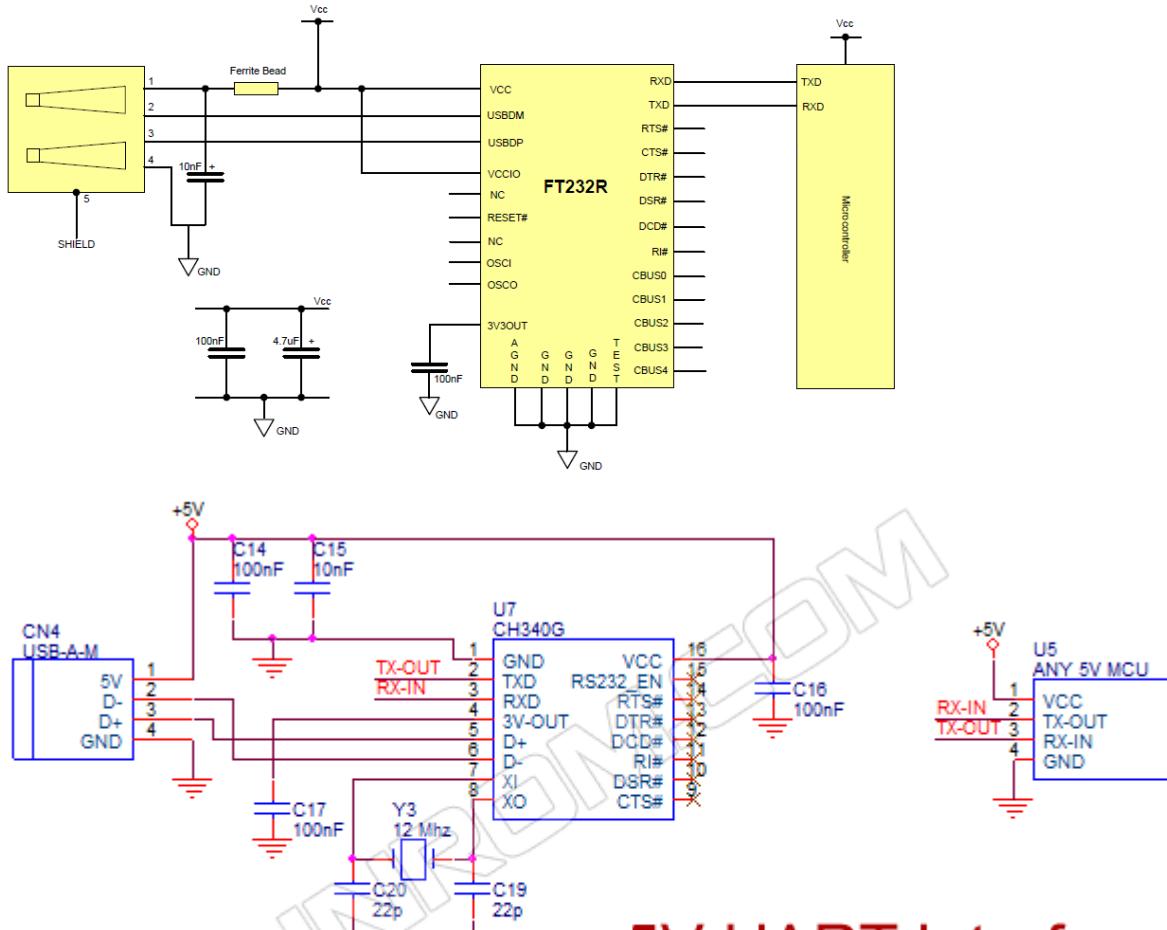
Data rate (bps)	Distance (m)
2400	60
4800	30
9600	15
19200	7.6
38400	3.7
56000	2.6

برای درایور RS232 می توانید از چیپ MAX232 و برای درایور RS485 از چیپ MAX487 یا 75179 یا 75176 استفاده کنید.



مبدل USB به UART

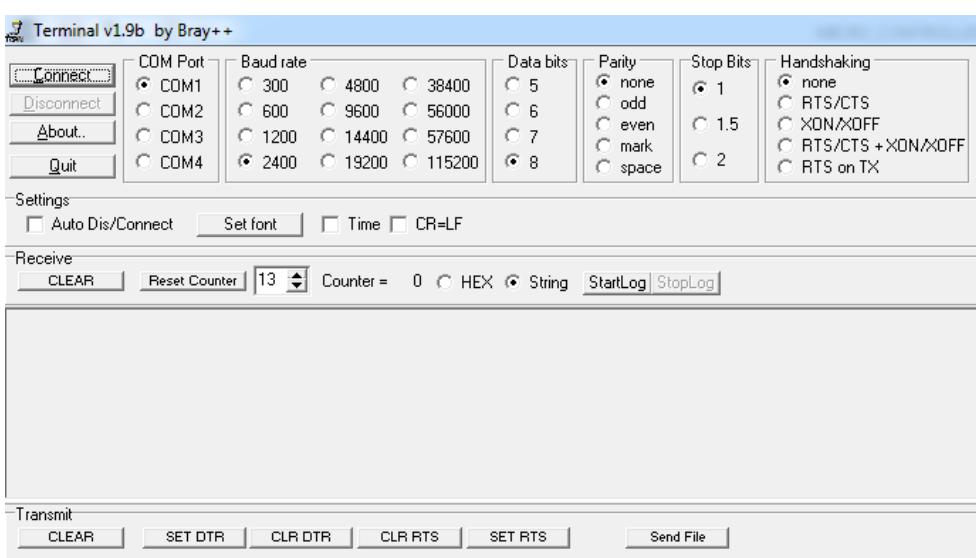
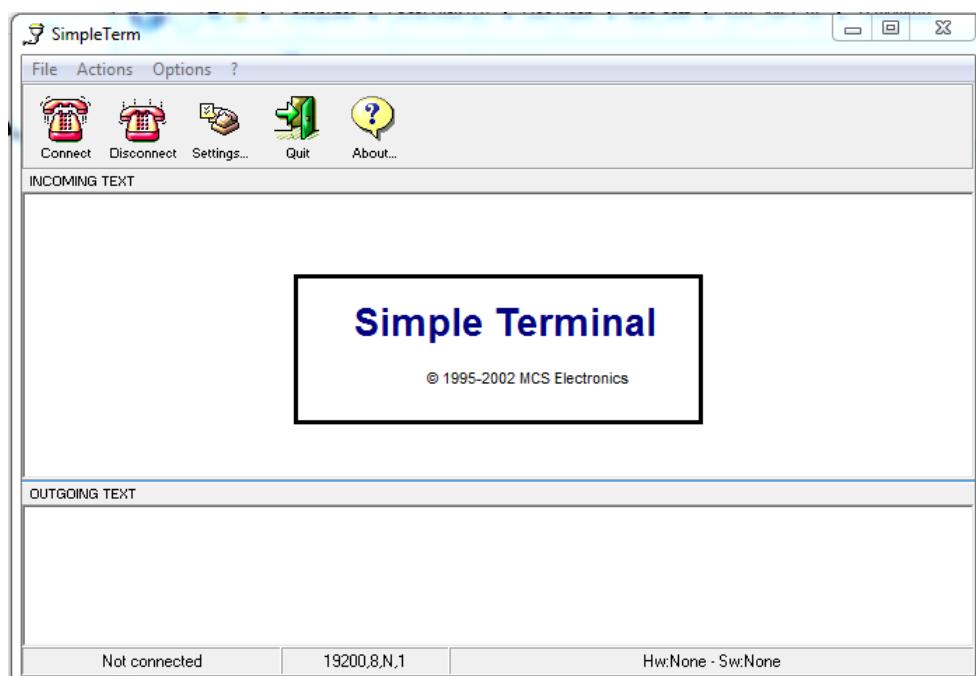
برای این منظور می توانید از چیپ های FT232 , PL2303 , CH340G استفاده کنید.



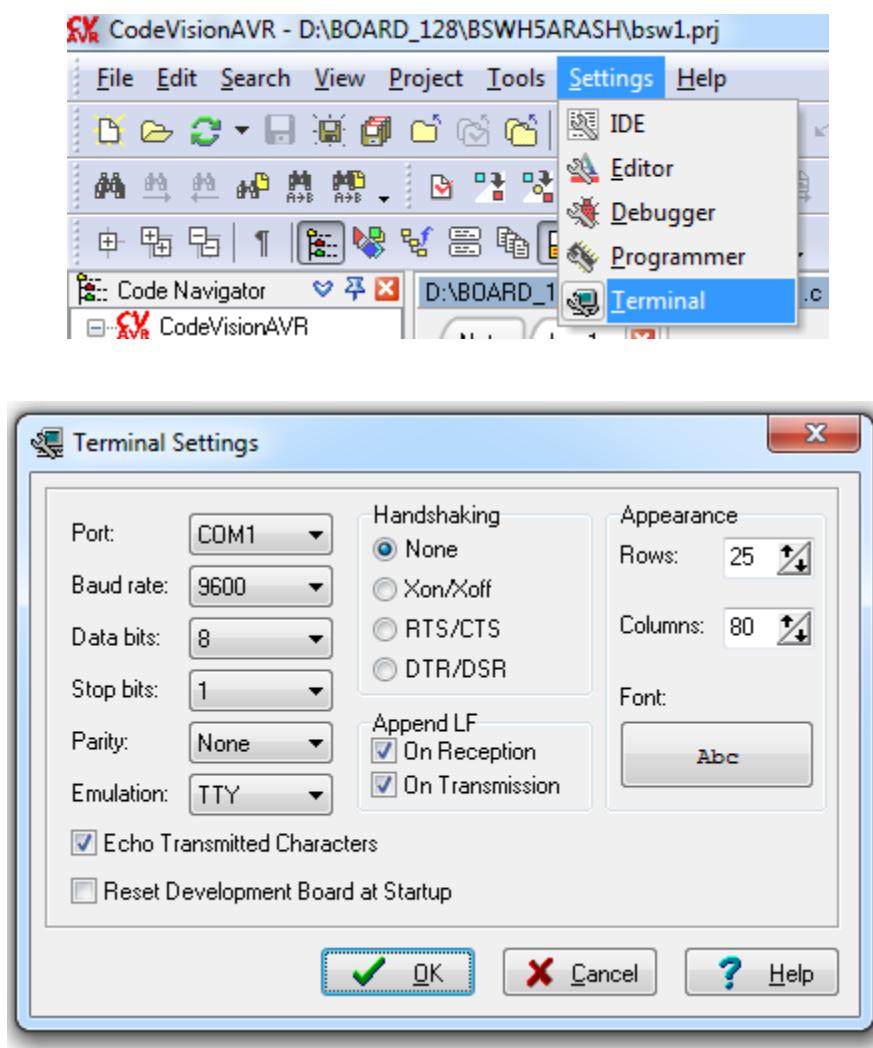
نرم افزارهای ترمینال Terminal:

اگر بخواهیم میکرو را با استفاده از **UART** به کامپیوتر وصل کنیم لازم است که در کامپیوتر نرم افزاری داشته باشیم تا ارتباط ما را با پورت سریال برقرار کند به این نرم افزارها **Terminal** گفته می شود . در فایل های مربوط به درس دونرم افزار **simpleterm** و **Terminal** در اختیار شما قرار گرفته ، در نرم افزار **CodeVision** نیز یک ترمینال وجود دارد.

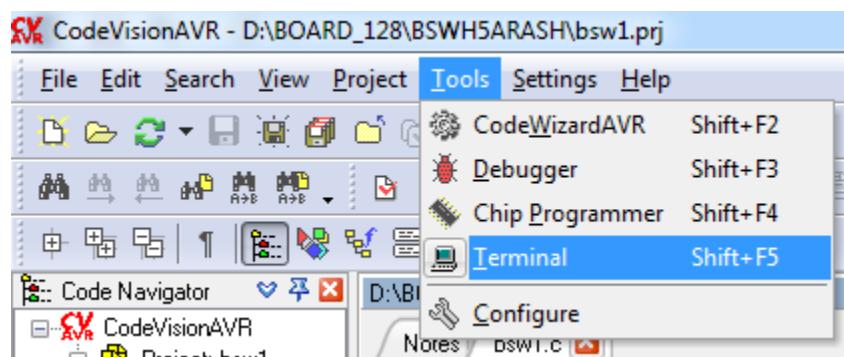
- توجه داشته باشید که لازم است قبل از استفاده از ترمینال باید آن را تنظیم کنید.



اگر می خواهید از ترمینال موجود در **CodeVision** استفاده کنید ابتدا از مسیر زیر تنظیم کنید.

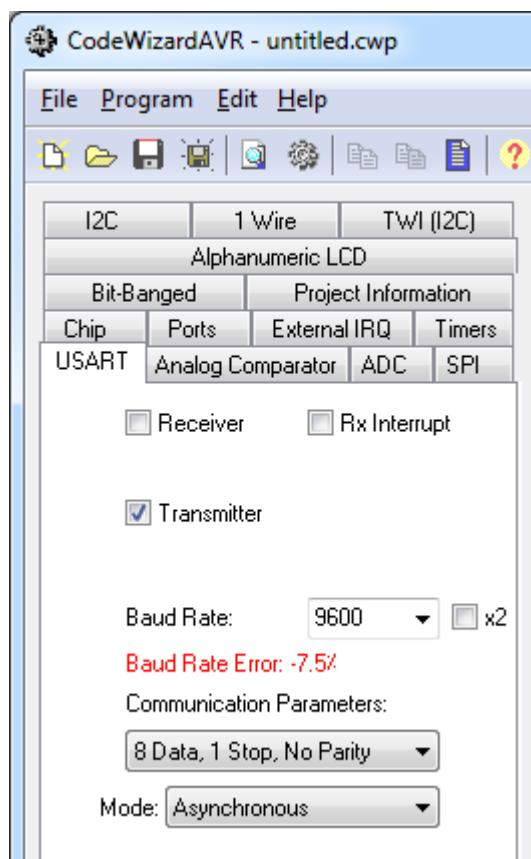


برای استفاده از Terminal نیز از مسیر زیر اقدام کنید.



مثال: برد آزمایش را توسط کابل **USB** به کامپیوتر متصل کنید برنامه ای بنویسید که هر دو ثانیه یک بار کلمه **SALAM** را به کامپیوتر ارسال نماید.

در ویزارد لازم است برگه **USART** را باز و بسته به نیاز **Transmitter** یا **Receiver** یا هر دو را فعال کنید. برای مثال ما **Receiver** کافی است.



توجه داشته باشید که **Baud Rate** را باید طوری تنظیم

کنید که خطای مشخص شده در زیر آن مقدار قابل قبولی باشد. در این تصویر فرکانس میکرو **1meg** بوده در نتیجه نرخ ارسال **9600** خطای بالایی دارد که برای رفع آن باید فرکانس میکرو را زیاد یا نرخ ارسال را کم کنیم.

در بخش **Communication Parameters** نیز وضعیت قاب دیتا را مشخص می کنیم.

در قسمت **Asynchronous Mode** نیز **Asynchronous** را انتخاب می کنیم.

پس از ذخیره پروژه مشاهده می شود که هدر فایل **stdio.h** به برنامه اضافه شده. توابع کار با پورت سریال در این کتابخانه قرار دارد.

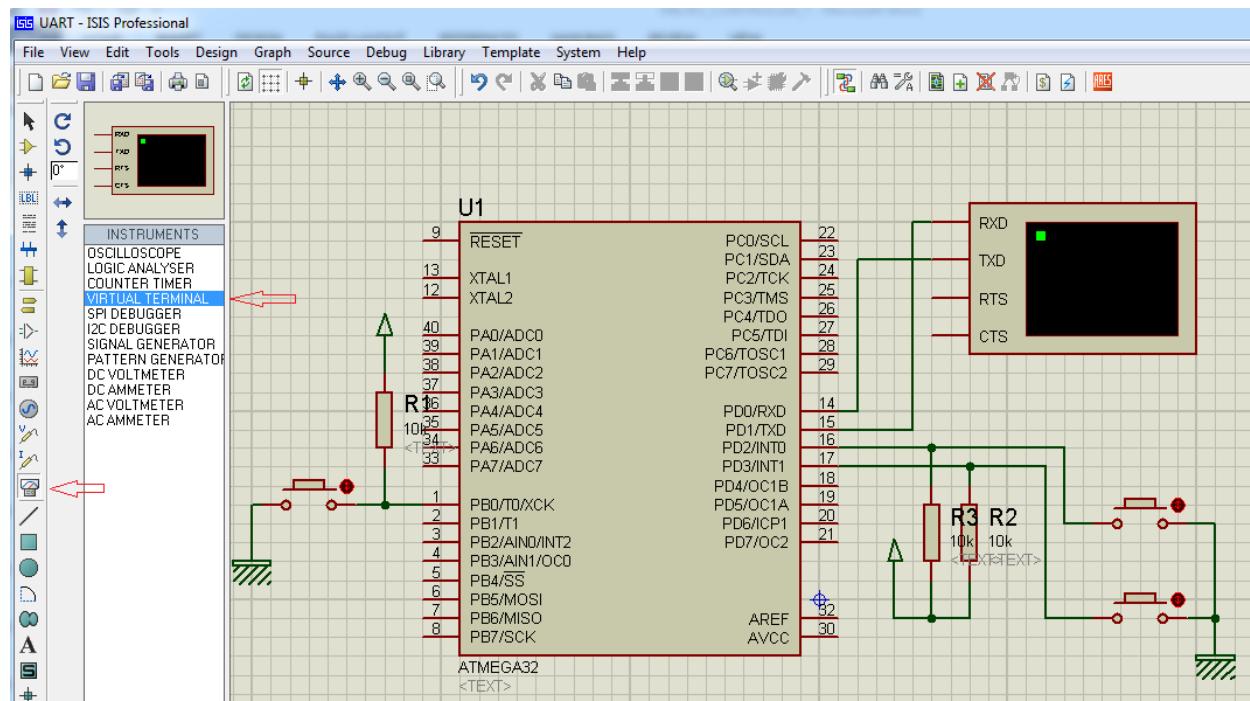
Putchar('');	{	ارسال دیتا
Putsf("");		
Puts(متغیررشته ای);		
=getchar();		دریافت دیتا

برنامه این مثال به شرح زیر است.

```
while(1)
{
    putsf("SALAM");
    putchar(13);
    putchar(10);
    delay_ms(2000);
}
```

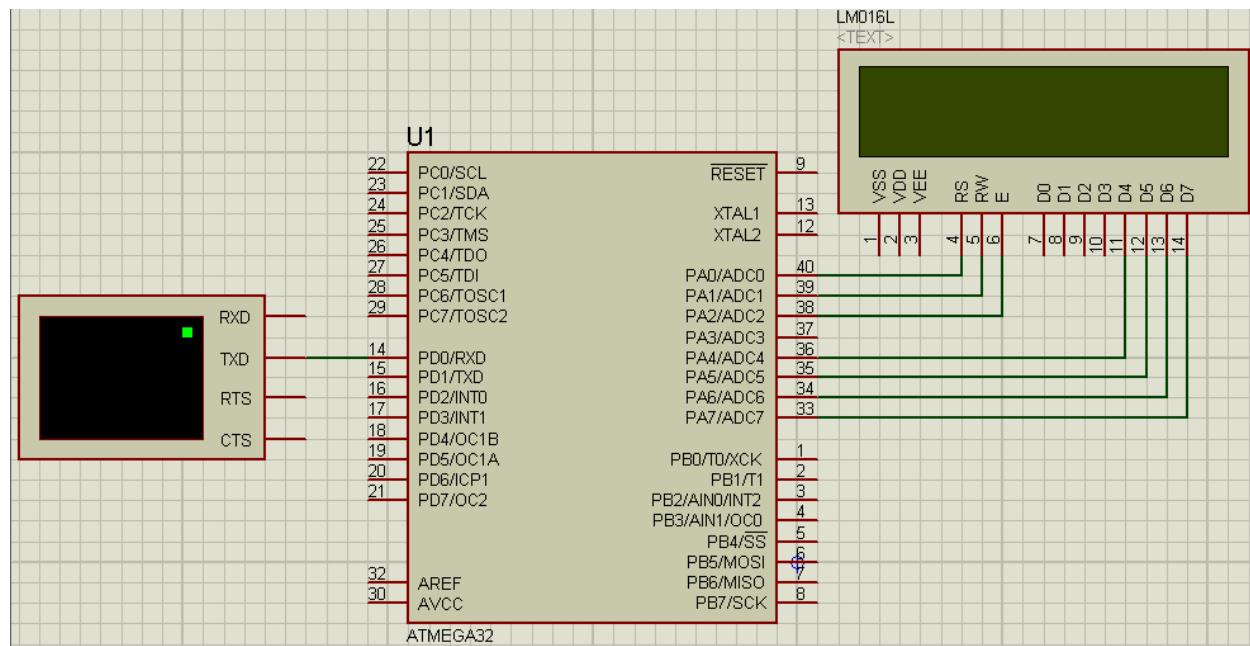
کاراکترهای 13 و 10 کد اسکی Line Feed و Enter می باشند این دو کد باعث می شوند بعد از نمایش کلمه SALAM به خط بعدی برود و زیر هم چاپ شوند.

نکته: در پروتئوس می توانید از VIRTUAL TERMINAL که در بخش VIRTUAL TERMINAL قرار دارد بجای کامپیوتر استفاده کنید.



تمرین ۱: با توجه به شکل بالا که منطبق با برد آموزشی می باشد برنامه ای بنویسید که با هر بار زدن کلیدهای **PB.0** و **PB.2** پیغام های متفاوتی به کامپیوتر ارسال شود.

تمرین ۲: با توجه به شکل زیر برنامه ای بنویسید که هر کاراکتری روی کیبورد کامپیوتر و در محیط **Terminal** تایپ شد بروی **LCD** دیده شود، اگر کلید **Enter** زده شد به خط دوم **LCD** برود و کاراکترهای دریافتی را آنجا تایپ کند و اگر کلید **ESC** زده شد **LCD** پاک شود.



تایمر و کانتر: Timer/Counter

در این میکرو دو تایمر / کانتر هشت بیتی و یک تایمر / کانتر شانزده بیتی پیش بینی شده است.

این تایمر / کانترها می توانند در سه حالت کاری **Normal, CTC, PWM** قرار گیرند.

: در این حالت یک شمارنده معمولی داریم که با هر پالس ورودی عدد داخل تایمر / کانتر یک واحد افزایش می یابد.

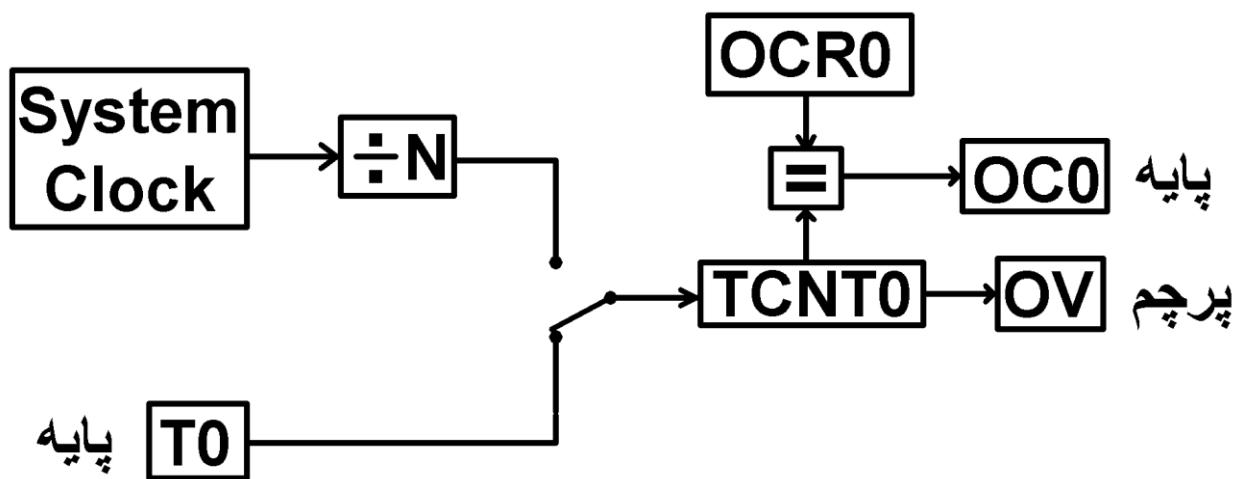
(Clear Timer on Compare Match): CTC

در این مد کاری می توان با تغییر در مقدار رجیستر **OCRn** بر روی پایه **OCn** یک موج مربعی با فرکانس های مختلف دریافت کرد.

(Pulse Width Modulator): PWM

در این مد کاری می توان با تغییر در مقدار رجیستر **OCRn** بر روی پایه **OCn** یک موج مربعی با پهنهای پالس های مختلف دریافت کرد.

مدار تایمر / کانتر را می توان به شکل زیر نمایش داد.



* توجه داشته باشید شکل بالا برای تایمر صفر رسم شده و قابل تعمیم به تایمر ۱ و ۲ نیز می باشد.

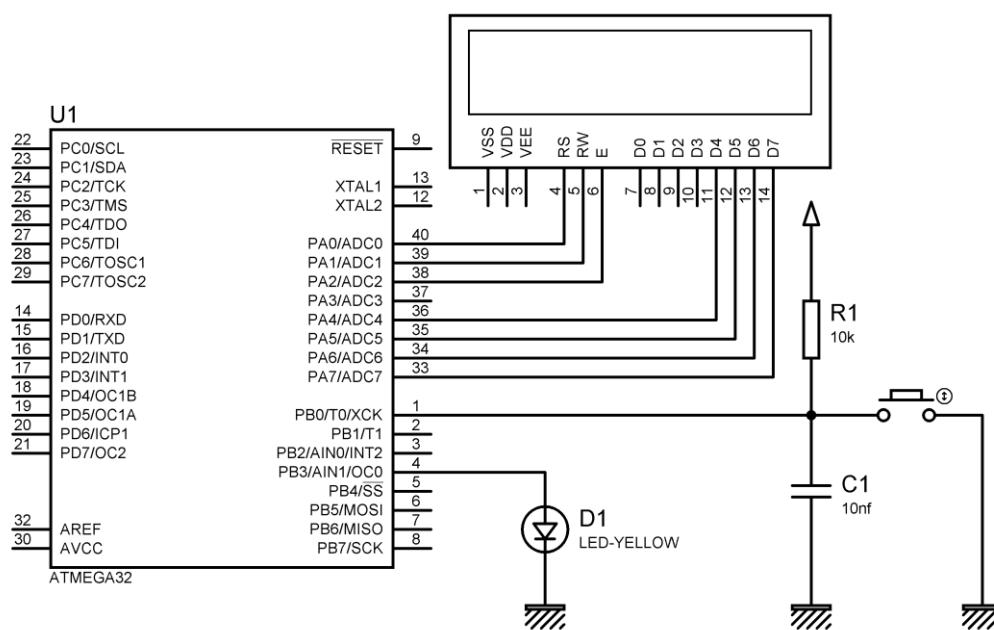
همانطور که در شکل مشاهده می شود پالس ورودی به تایمر ، می تواند از پایه **T0** تامین شود که در این صورت یک شمارنده(**counter**)، و یا تقسیمی از کلاک سیستم باشد که به علت منظم بودن یک تایمر خواهیم داشت.

N ÷ : فرکانس سیستم می تواند قبل از ورود به تایمر بر اعداد (1, 8, 64, 256, or 1024) تقسیم شود.

: رجیستر اصلی ، واحد تایمر/کانتر است و با هر پالس ورودی یک واحد افزایش می یابد. با توجه به اینکه تایمر صفر هشت بیتی است ، پس از ۲۵۵ پالس عدد داخل این رجیستر **0xff** خواهد شد و پالس ۲۵۶ ام باعث صفر شدن آن خواهد شد هر گاه این اتفاق رخ دهد پرچم سرریز **Over flow** خواهد شد. می توان با فعال کردن وقفه سرریز برنامه مدنظر را اجرا کرد.

: کاربر می تواند عددی را داخل این رجیستر قرار دهد ، محتوای این رجیستر همواره با عدد داخل رجیستر **TCNT0** مقایسه می شود هر گاه با هم مساوی شوند در پالس بعدی پایه **OC0** متاثر خواهد شد. این تاثیر می تواند ، صفر شدن ، یک شدن یا **NOT** شدن این پایه باشد.

مثال: یک کلید به پایه **T0** متصل کنید و تعداد دفعاتی که کلید زده می شود را توسط تایمر/کانتر صفر **OC0** متصل است نمایش دهید. در ضمن یک **LED** به پایه **LCD** متصل کنید و تایمر را طوری تنظیم کنید تا هر بار عدد داخل آن ۵ شد **LED** تغییر وضعیت دهد.

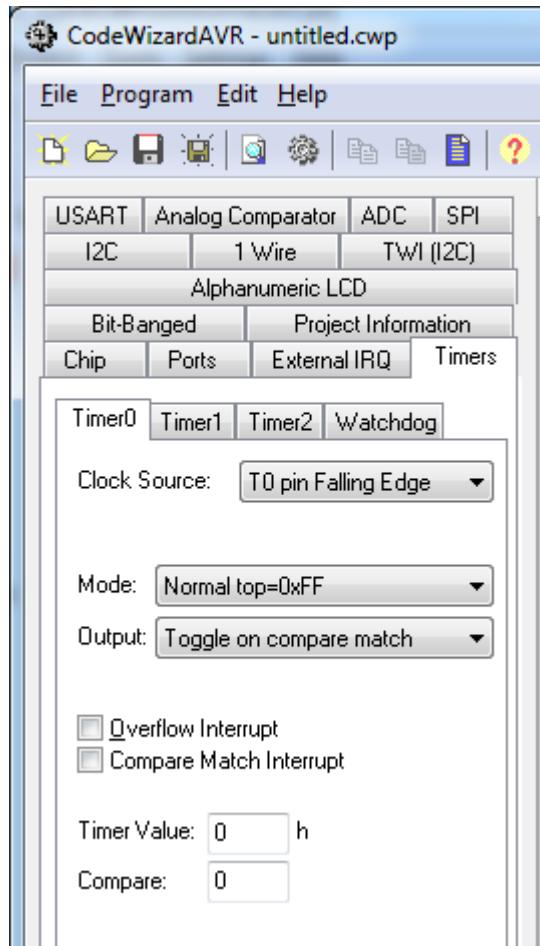


در ویزارد برگه **Timers** را باز و سپس برگه **Timer0** را انتخاب کنید. در قسمت **Clock Source** باز وسیس برگه **Timers** را انتخاب کنید. در قسمت **Mode** باز وسیس برگه **Normal** را انتخاب کنید. ما لبہ پایین رونده را انتخاب می‌کنیم. در قسمت **Output** باز وسیس برگه **Normal** را انتخاب کنید و در بخش **Output** چون می‌خواهیم پایه کنیم. در هر بار تساوی **OCR0** و **TCNT0** تغییر وضعیت دهد در حالت **Toggle** قرار می‌دهیم.

در دو قسمت **Interrupt** ها اگر تیک بزنید وقفه مربوط به سرریز و تساوی فعال خواهد شد.

در بخش **Timer Value** می‌توانیم مقدار اولیه به **TCNT0** و در بخش **Compare** می‌توانیم مقدار اولیه به **OCR0** بدهیم. که البته در برنامه نیز به راحتی می‌توان آن را انجام داد.

```
OCR0=5;
While(1)
{
    itoa(TCNT0,s);
    lcd_gotoxy(0,0);
    lcd_puts(s);
    lcd_putsf("  ");
}
```



مثال : اگر در مثال بالا هنگام تنظیم ویزارد در بخش **Clock Source** گزینه **Clock Value** یکی از فرکانس ها را و سایر تنظیمات همان حالت قبلی را انتخاب کنیم، و در قسمت **Clock Value** یکی از فرکانس ها را و سایر تنظیمات همان حالت قبلی بماند بر روی پایه **OC0** چه خواهیم داشت؟

پاسخ: فرض کنید در قسمت **Clock Value** فرکانس **125kHz** را انتخاب کرده باشیم در نتیجه زمان تناوب آن $T=1/125000=8\mu s$ خواهد شد. با توجه به اینکه با هر 256 پالس، پایه **OC0** یک بار تغییر وضعیت می دهد پس با گذشت هر $256*8=2048 \mu s$ این پایه از صفر به یک یا برعکس تغییر وضعیت خواهد داد در نتیجه یک پالس مربعی که زمان تناوب آن $2*2048=4096 \mu s$ است خواهیم داشت. البته فرکانس موج تولید شده را می توان از رابطه زیر بدست آورد.

$$f = \frac{f_{sys}}{n * 512}$$

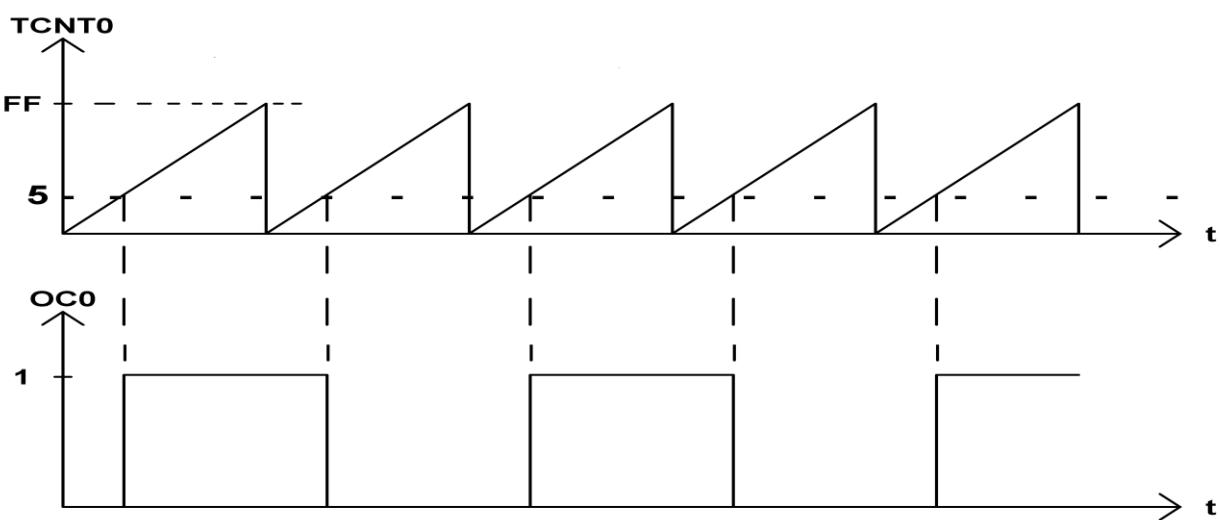
که در آن:

F_{sys} : فرکانس کار میکرو (که در مثال ما **1mhz** است)

N : ضریب تقسیم فرکانس (که در مثال ما **8** است)

$$f = \frac{1000000}{8*512} = 244.14$$

شكل زیر نیز می تواند در تفهیم چگونگی تولید موج مربعی مفید باشد.

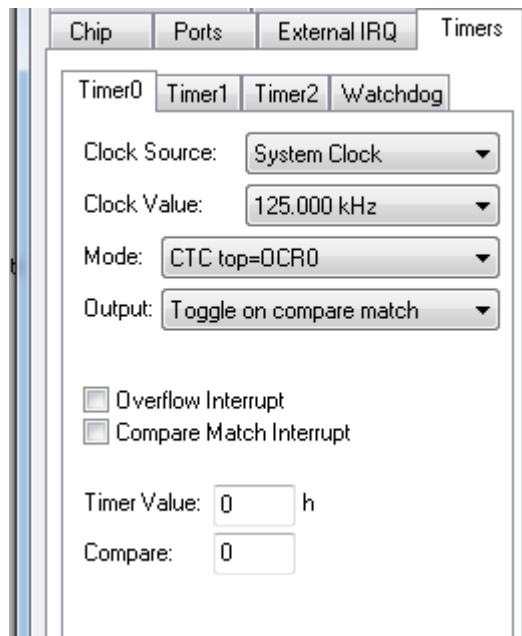


(Clear Timer on Compare Match): CTC

در این مد با تغییر در مقدار رجیستر **OCR0** طبق رابطه زیر می‌توان فرکانس‌های مختلفی را روی پایه **OC0** دریافت کرد.

$$f_{ctc} = \frac{f_{sys}}{2n(OCR0 + 1)}$$

تنظیمات ویزارد برای تایمر در حالت **CTC** به شکل زیر است.



مثال: برنامه زیر را روی برد آزمایش اجرا کنید و نتیجه

را که بصورت صوت روی **Buzzer** یا بلندگو شنیده خواهد شد،

تجربه کنید.

while(1)

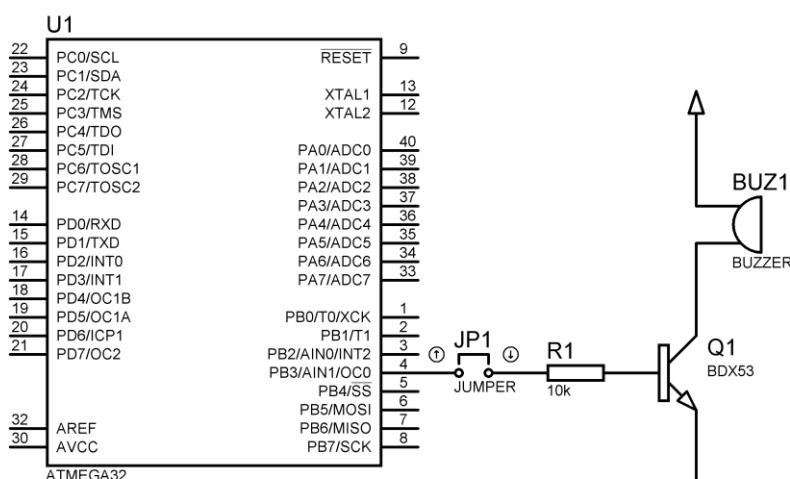
{

OCR0++;

delay_ms(10);

}

مدار روی برد به شکل زیر است.



لازم است روی بود ، بیس ترانزیستور را

با یک سیم به پایه 4 میکرو متصل کنید.

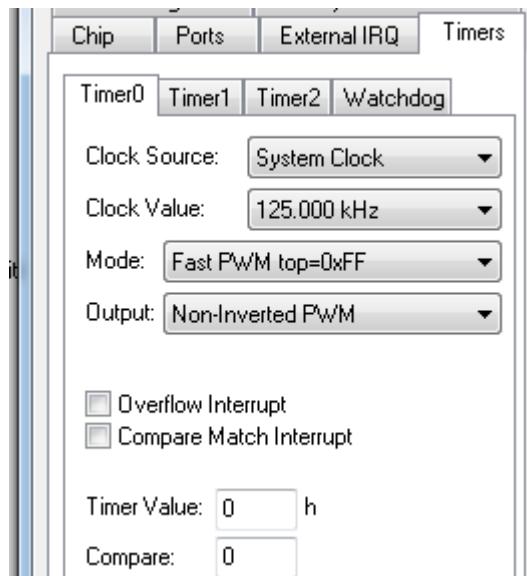
تمرین: برنامه‌ای بنویسید که با دو کلید متصل به PD.2 و PD.3 بتوان فرکانس خروجی را کم و زیاد کرد.

(Pulse Width Modulator):PWM

در این مدد کاری می‌توان با تغییر در مقدار رجیستر $OCRn$ بر روی پایه OCn یک موج مربعی با پهنهای پالس‌های مختلف دریافت کرد. فرکانس تولید در این مدد از رابطه زیر بدست می‌آید.

$$f_{pwm} = \frac{f_{sys}}{n * 256}$$

تنظیمات ویژارد برای تایмер در حالت **PWM** به شکل زیر است.



در قسمت **Output** می‌توان دو حالت **Non-Inverted** یا

یکدیگر هستند را انتخاب کرد.

مثال: برنامه زیر را به میکرو وارد کنید و خروجی **OC0** را روی

اسیلوسکوپ مشاهده کنید.

while(1)

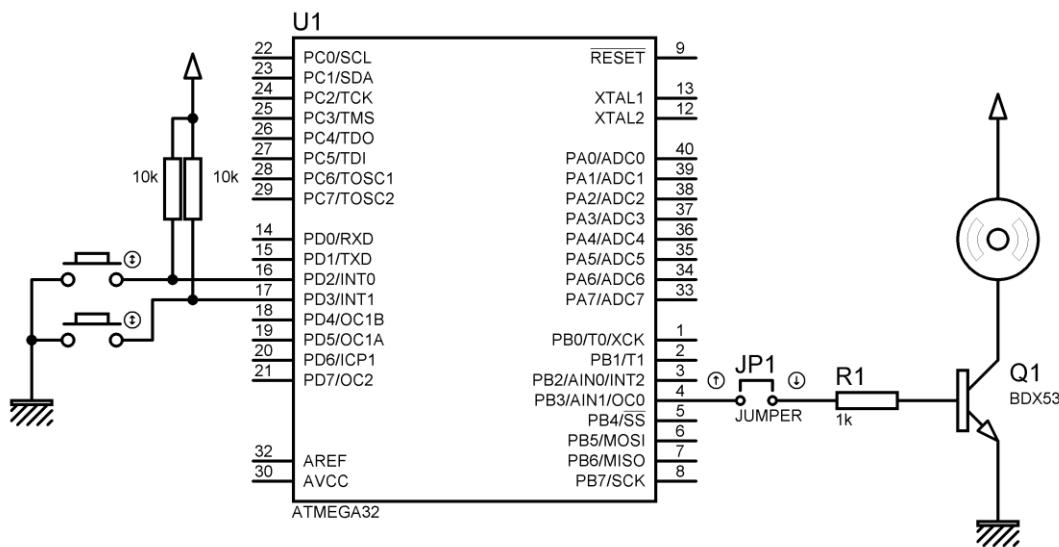
{

OCR0++;

delay_ms(10);

}

مثال: با توجه به مدار زیر برنامه ای بنویسید که توسط دو کلید عرض پالس و در نتیجه سرعت موتور کنترل شود.



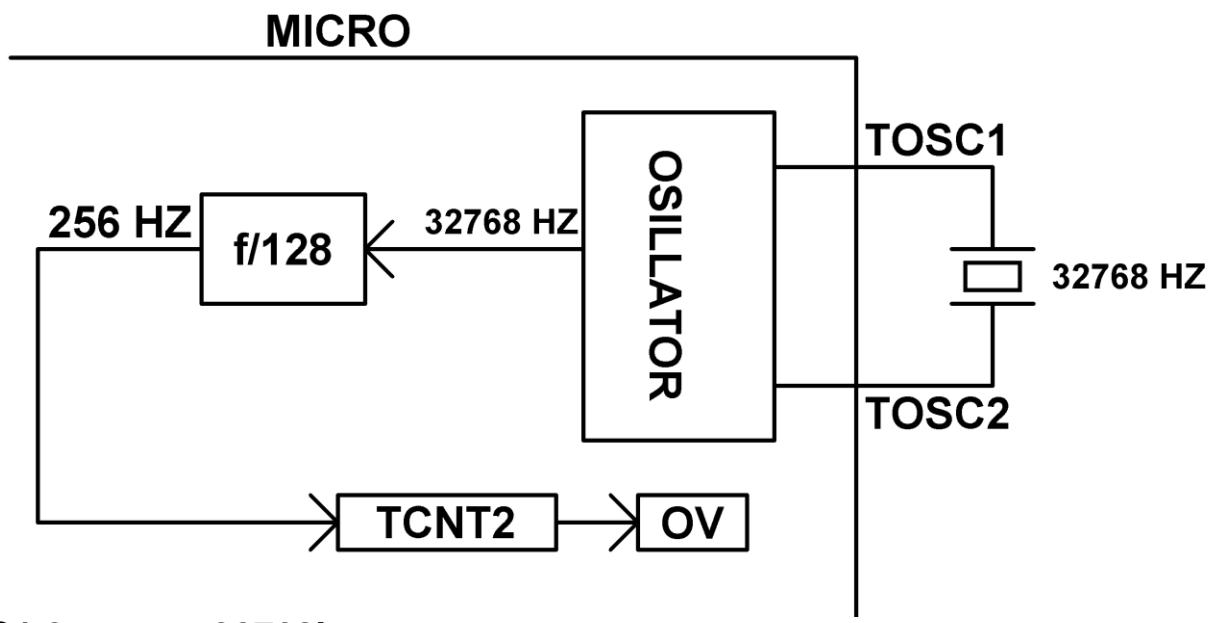
```

while(1)
{
    if(PIND.2==0)
    {
        p++;
        if (p>250) p=250;
        OCR0=p;
        delay_ms(20);
    }
    if(PIND.2==0)
    {
        p--;
        if(p<5) p=5;
        OCR0=p;
        delay_ms(20);
    }
}

```

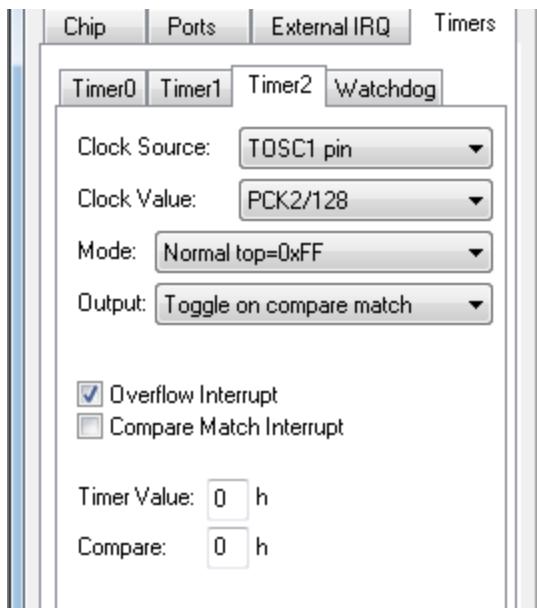
Real Time Counter : RTC

هر گاه نیاز به زمان سنجی واقعی داشته باشیم لازم است که بتوانیم پالس یک ثانیه را تولید کنیم و با استفاده از آن دقیقه و ساعت را در تایمر ۲ سازوکار شکل زیر فراهم شده تا بتوان پالس یک ثانیه را تولید کرد.



برای تولید پالس یک ثانیه لازم است یک کریستال با فرکانس 32768hz به دو پایه $TOSC1,2$ متصل کنیم ، با این کار اسیلاتور داخل میکرو راه اندازی شده و یک موج مربعی با فرکانس 32768hz تولید می کند با تنظیماتی که در ویزارد برای تایمر ۲ انجام می دهیم این فرکانس را بر 128 تقسیم می کنیم ، در نتیجه یک فرکانس 256hz خواهیم داشت که آن را به تایمر ۲ می دهیم از آنجایی که تایمر ۲ هشت بیتی است با هر 256 پالسی که به آن وارد می شود این تایمر پر شده و سرریز اتفاق می افتد یعنی پرچم OV یک می شود حال اگر اینترپت سرریز تایمر ۲ را فعال کرده باشیم هر ثانیه یک بار روتین سرویس آن اجرا می شود و کافی است در برنامه سرویس ثانیه را یک واحد افزایش دهیم.

برای تنظیم ویزارد به شکل زیر عمل می کنیم.



*بخش **output** را در حالت **Toggle** قرار داده ایم ،

می توان یک **LED** را روی پایه **OC2** قرار داد تا در هر ثانیه

یک بار خاموش یا روشن شود.

پس از تولید کد برنامه ، زیر برنامه ای برای وقفه سرریز تایمر ۲ ایجاد خواهد شد که داخل آن ثانیه را افزایش می دهیم.

//Timer2 overflow interrupt service routine

```
interrupt [TIM2_OVF] void timer2_ovf_isr(void)
```

```
{
```

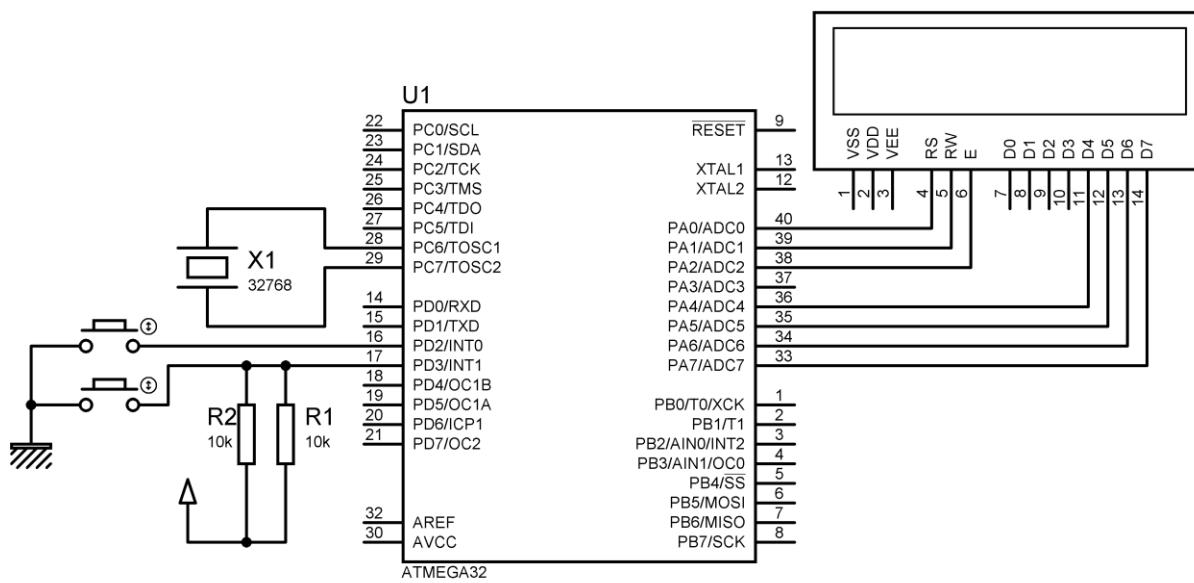
//Place your code here

S++; // در این خط ثانیه اضافه می شود.

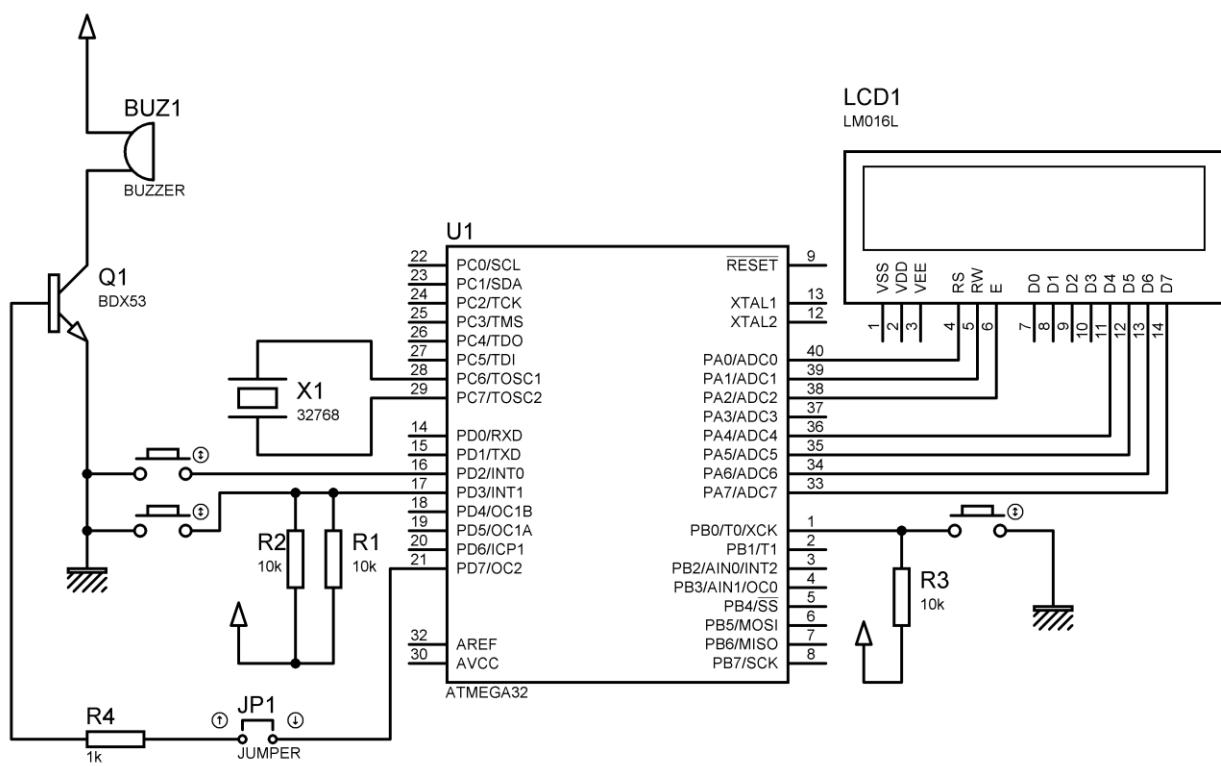
```
}
```

تمرین: یک کریستال به پایه های **TOSC1,2** و **PORTA** به **LCD** و یک **RTC** با استفاده از **PD.2** و **PD.3** متصل کنید و با **LCD** را بتوانیم ساعت و دقیقه را تنظیم کنیم . همچنین با **RTC** ساعت و دقیقه را تنظیم کنیم .

(توجه داشته باشید سخت افزار پیشنهادی روی برد آزمایش فراهم می باشد)



تمرین: روی برد آزمایش کلید سوم به **PB.0** متصل است و **Buzzer** را می توانید به **PB.0** متصل کنید . برنامه ای بنویسید که با کلید روی **PB.0** مشخص کنیم می خواهیم کدام آیتم را تغییر دهیم و با دو کلید **PD.2** و **PD.3** آن را کم و زیاد کنیم . روی خط اول **LCD** ساعت و روی خط دوم زمان زنگ ساعت **ALARM** را داشته باشیم. هر گاه ساعت با آلارم برابر شد با صدای مناسب **Buzzer** به مدت یک دقیقه کار کند. (توجه داشته باشید سخت افزار پیشنهادی روی برد آزمایش فراهم می باشد)



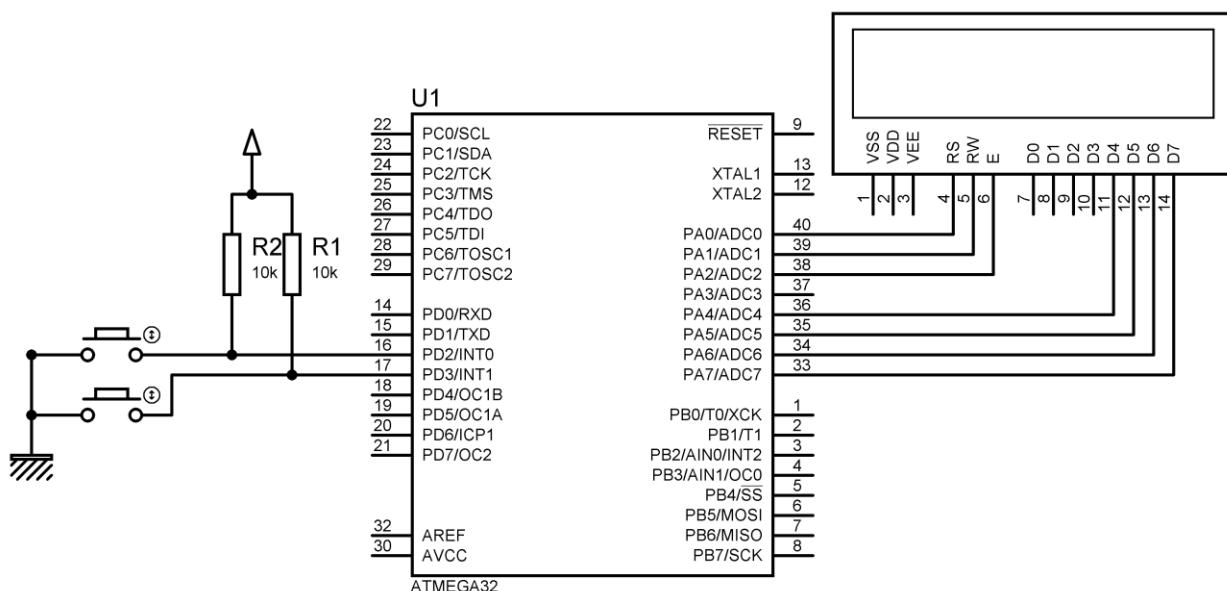
وقفه : Interrupt

همان طور که قبلا اشاره شد برای سرویس دهی به دستگاه های جانبی دو روش سرکشی **Polling** و **وقفه** را می توان استفاده کرد.

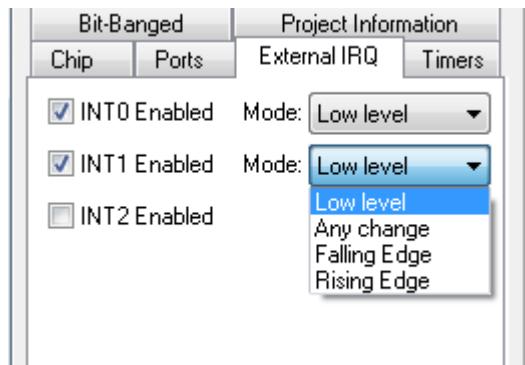
در میکرو مورد بحث ما سه وقفه خارجی **INT0, INT1, INT2** وجود دارد. با یک مثال روش تنظیم راه اندازی و استفاده از آن ها را فرامی گیریم.

مثال: یک **LCD** به **PORTA** و دو عدد کلید به وقفه های صفر و یک متصل کنید، برنامه ای بنویسید که وسط خط اول **lcd** یک شمارنده **0** تا **9** داشته باشیم (برنامه اصلی). حال اگر وقفه صفر حادث شد روی خط دوم **lcd** کلمه **INT0** و اگر وقفه یک حادث شد روی خط دوم **lcd** کلمه **INT1** به مدت دو ثانیه ظاهر و سپس پاک شود(روتین سرویس) و شمارش که به علت اجرای وقفه متوقف شده بود ادامه یابد.

(توجه داشته باشید سخت افزار پیشنهادی روی برد آزمایش فراهم می باشد)



برای تنظیم وقفه ها در External IRQ برجه wizard فعال می کنیم. سپس در قسمت Mode باید مشخص کنیم که پاسخ به وقفه ، در چه وضعیتی از پالس وقفه داده و روتین سرویس آن وقفه اجرا شود.



*اگر گزینه Any change را انتخاب کنید ، برنامه مربوط به آن وقفه دو بار اجرا می شود ، روی لبه پایین رونده و بالا رونده .

برای این مثال INT0 را روی لبه پایین رونده و INT1 را روی لبه بالا رونده قرار دهید.

بعد از تولید کد ، مشاهده می کنید که برای هر یک از وقفه ها زیر برنامه ای آماده شده که می توانید روتین سرویس آن وقفه را آنجا بنویسید.

```

1 #include <mega32.h>
2
3 // External Interrupt 0 service routine
4 interrupt [EXT_INT0] void ext_int0_isr(void)
5 {
6     // Place your code here  INT0
7 }
8
9
10 // External Interrupt 1 service routine
11 interrupt [EXT_INT1] void ext_int1_isr(void)
12 {
13     // Place your code here  INT1
14 }
15
16 // Declare your global variables here
17
18
19 void main(void)
20 {

```

برنامه اصلی و وقفه ها بصورت زیر خواهد بود.

```

//External Interrupt 0 service routine

interrupt [EXT_INT0] void ext_int0_isr(void)
{
    lcd_gotoxy(0,1);
    lcd_putsf("INT0;(";
    delay_ms(2000);

    lcd_clear;()

}

//External Interrupt 1 service routine

interrupt [EXT_INT1] void ext_int1_isr(void)
{
    lcd_gotoxy(0,1);
    lcd_putsf("INT1;(";
    delay_ms(2000);

    lcd_clear;()

}

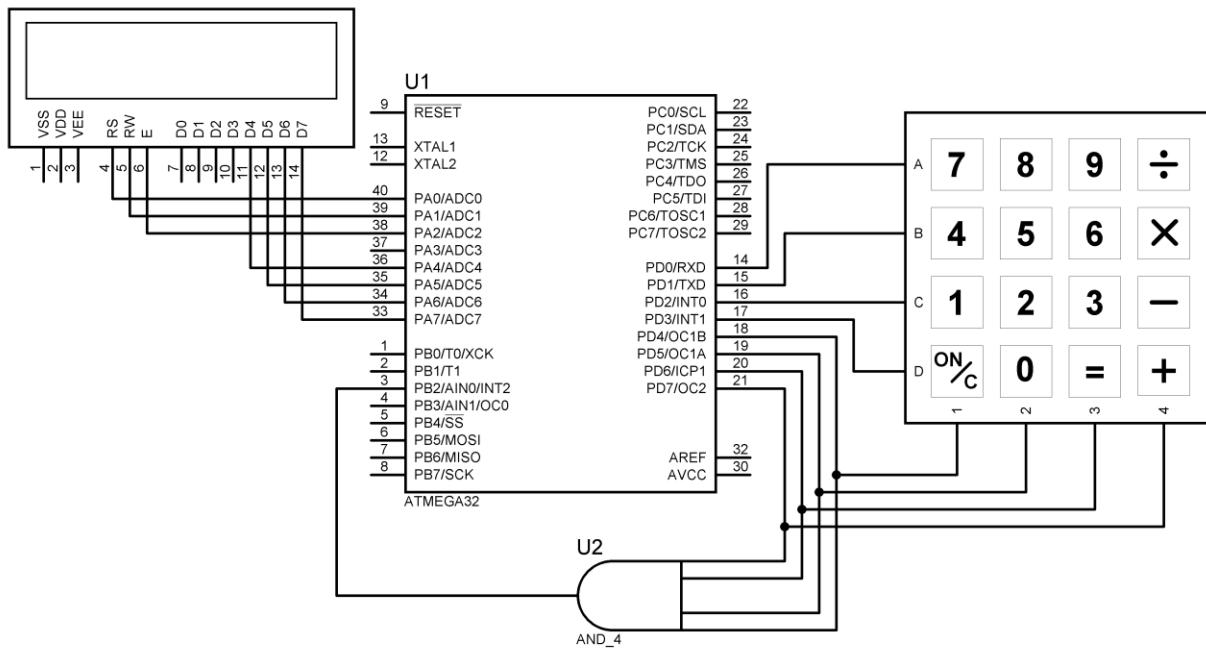
//Global enable interrupts

#asm("sei") فعال ساز وقفه سراسری

While(1)
{
    for(i=0;i<10;i++)
    {
        itoa(i,s);
        lcd_gotoxy(7,0);
        lcd_puts(s);
        delay_ms(500);}}

```

تمرین: یک LCD و یک صفحه کلید ۴*۴ را مانند شکل زیر به میکرو متصل کنید برنامه ای بنویسید که وسط خط اول lcd یک شمارنده ۰ تا ۹ داشته باشیم (برنامه اصلی). حال اگر کاربر هر کلیدی را زد عدد یا علامت آن کلید وسط خط دوم lcd نمایش داده شود (روتین سرویس).

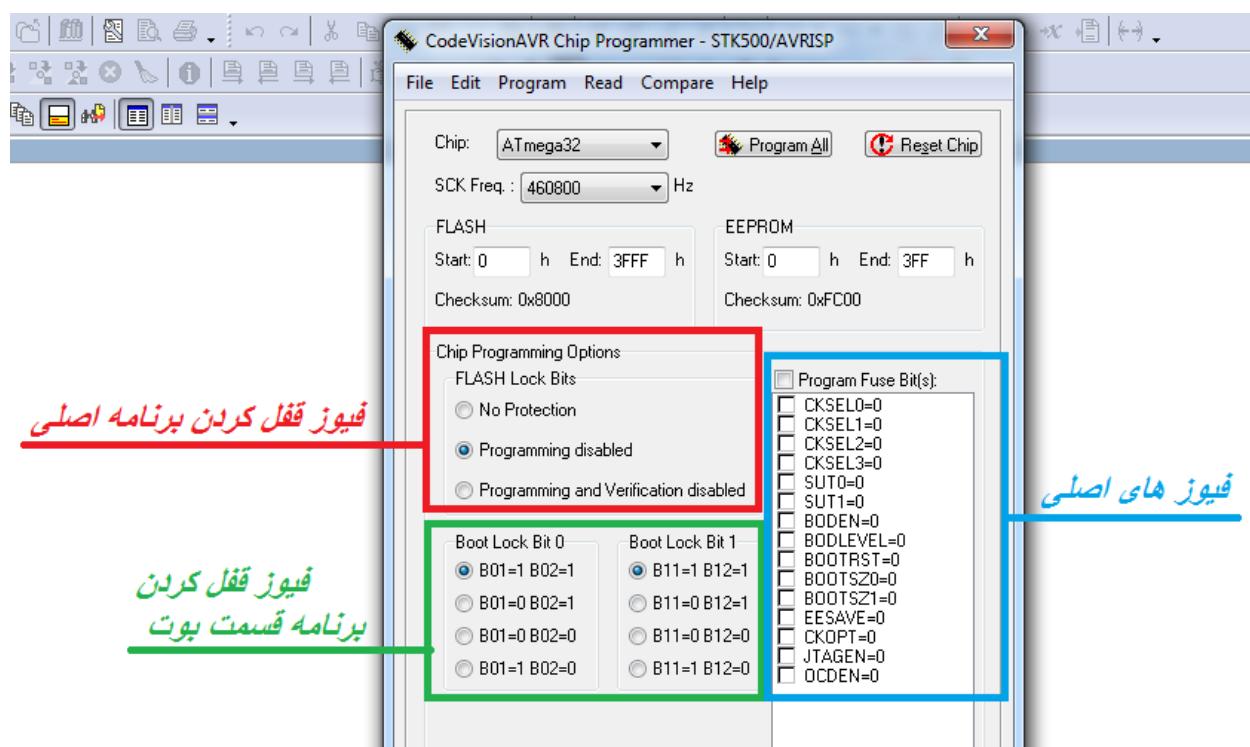


: Fuse bit

تعدادی بیت در حافظه **Flash** که می توان آن ها را توسط پرگرامر خواند، ویرایش کرد و دوباره برنامه ریزی نمود.

هر دسته یا هر یک از فیوز بیت ها می توانند بخشی از میکرو را در وضعیت مشخصی قرار دهند.

فیوز های ATMEGA32 عبارتند از:



کاربرد بعضی از فیوز بیت های اصلی به شرح زیر است:

CKSEL0_1_2_3: توسط این چهار فیوز می توان مشخص کرد که اولاً منبع کلاک سیستم از کجا تامین شود و ثانیاً فرکانس آن چقدر باشد.

BODEN: برای فعال شدن Brown-Out Detection باید این فیوز را فعال کنید.

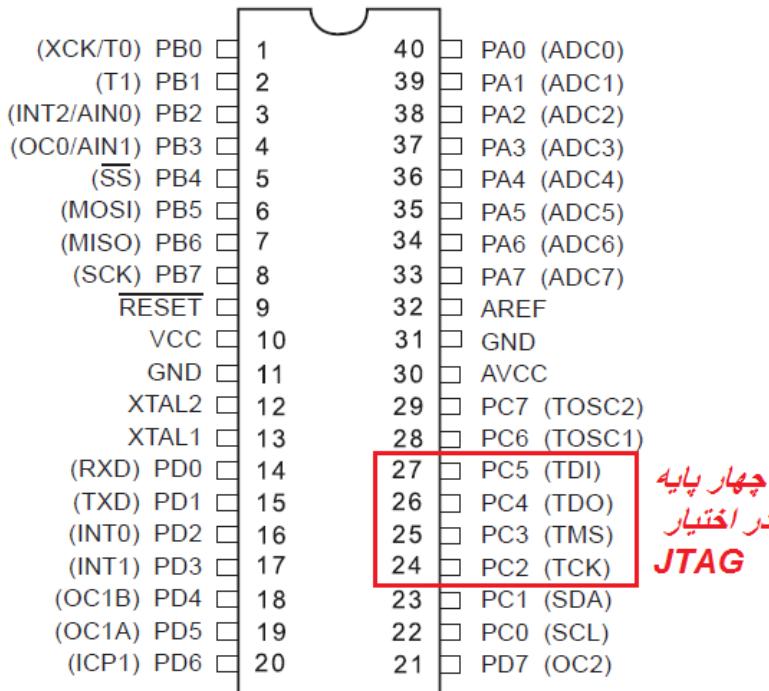
BODLEVEL: مشخص می کند Brown-Out Detection روی چه ولتاژی عمل کند.

0 = 4 volt

1 = 2.7 volt

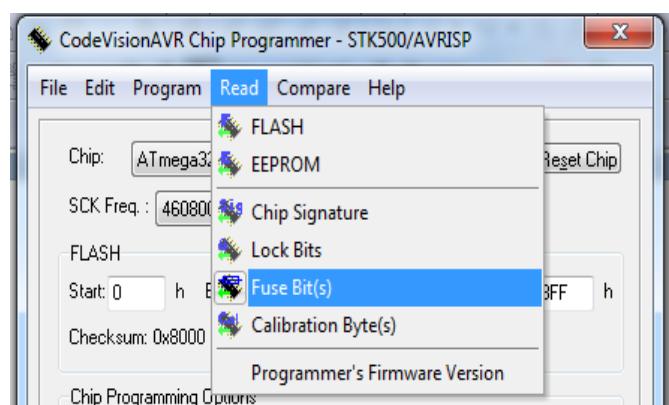
اگر این فیوز بیت فعال باشد هنگام پاک کردن حافظه **Flash** پاک می شود ولی حافظه دیتای ماندگار **EEPROM** پاک نمی شود.

برای فعال شدن **jtag** باید این فیوز فعال باشد. توجه داشته باشید در میکرو نو که می خرید این فیوز فعال است در نتیجه چهار پین از **PORTC** در اختیار **jtag** است و نمی توانید از تمام پایه های پورت **C** استفاده کنید.



(On-chip debug enabled) : OCDEN

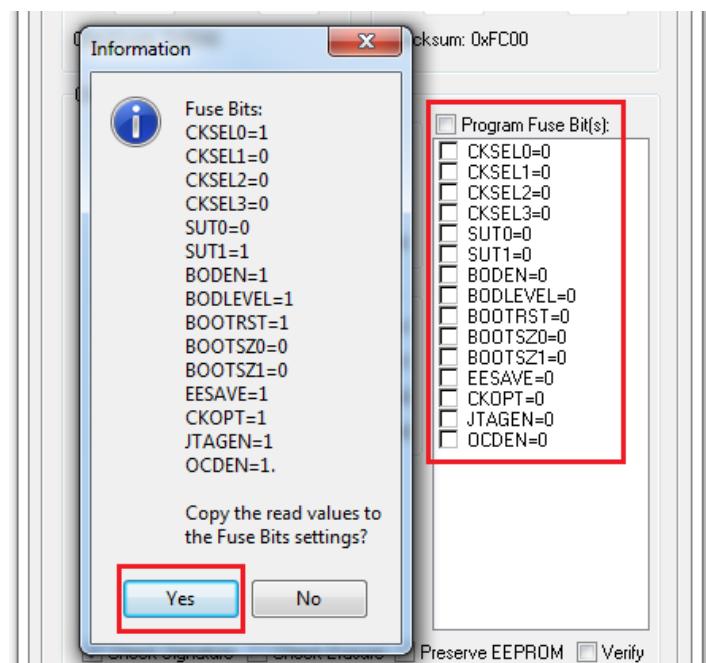
اگر این فیوز ، همراه با فیوز **JTAGEN** فعال باشد آنگاه می توانید با استفاده از رابط **jtag** برنامه داخل میکرو را خط به خط اجرا و در صورت نیاز رفع اشکال نمایید.



خواندن فیوز بیت ها:

برای این منظور در پنجره **Chip Programmer**

مسیر زیر را طی می کنیم.



بعد از این مرحله فیوزها خوانده شده ، و در یک

پنجره نمایش داده می شود، اگر گزینه yes را

انتخاب کنید وضعیت موجود فیوزها ، به قسمت

Program Fuse Bit(s) منتقل خواهد شد

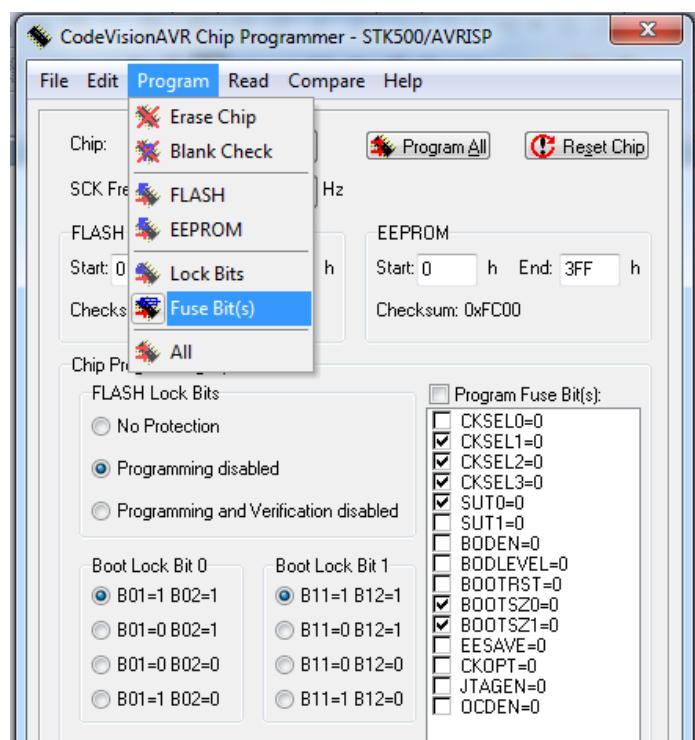
و امکان ویرایش آن ها وجود دارد.

برنامه ریزی فیوز بیت ها:

برای این منظور در پنجره

Chip Programmer

مسیر زیر را طی می کنیم.



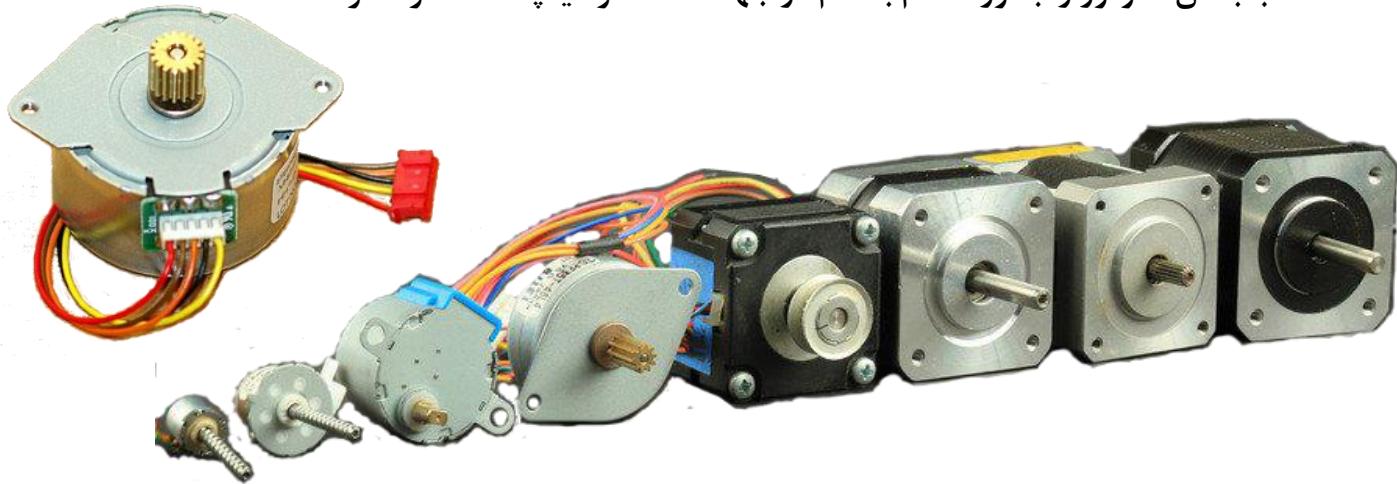
توجه داشته باشید که ابتدا با مراجعه به **Fuse Bit اطلاع کاملی از وضعیت Data sheet

پیدا کنید و سپس مبادرت به تغییر آن ها نمایید در غیر این صورت ممکن است میکرو در حالتی قرار

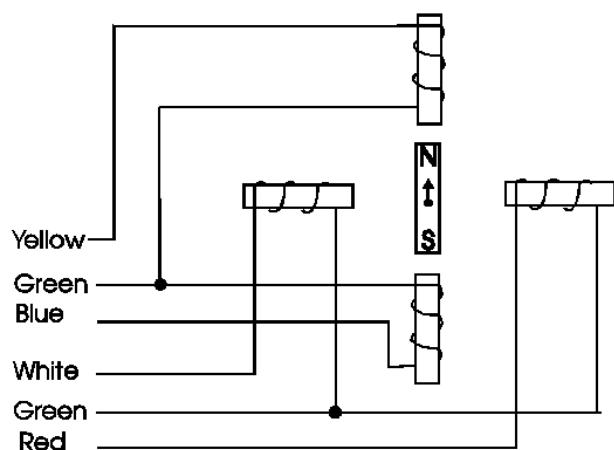
گیرد که دیگر در مدار شما کار نکند و نتوانید با پرگرامرهای معمولی آن را از این حالت خارج کنید.

موتور یله‌ای : STEP MOTOR

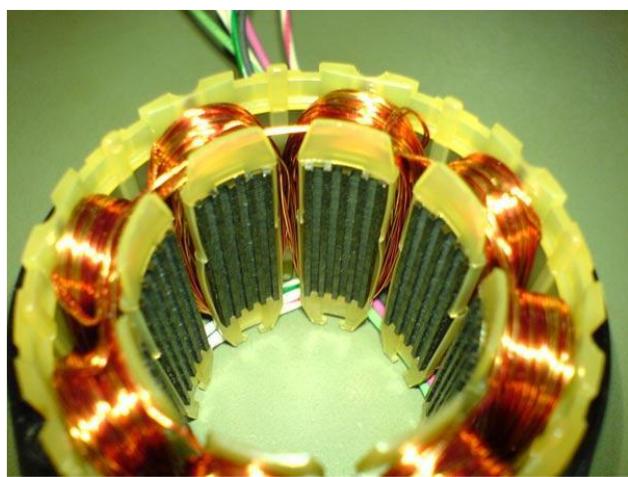
این موتور جزء موتورهای DC بدون جاروبک (Brushless) می‌باشد ، که می‌توان با اعمال دیتای مناسب به آن ، موتور را بصورت گام به گام در جهت ساعتگرد یا پاد ساعتگرد حرکت داد.



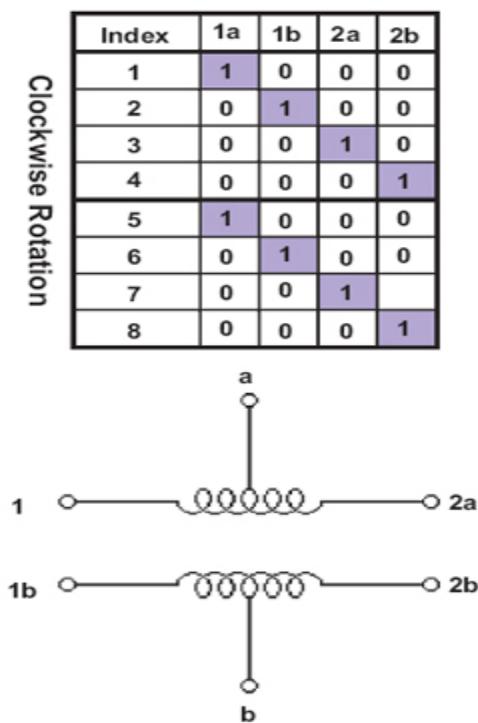
سیم پیچی این موتور را می‌توان بصورت شکل زیر نمایش داد.



برای حرکت دادن موتور می‌توان یک سیم پیچ یا
دو سیم پیچ را فعال کنیم به این ترتیب سیم پیچ ها
مغناطیس شده و روتور را که یک آهنربای دائمی است
یک گام حرکت می‌دهد.



با اعمال دیتاهای زیر می توان موتور را ساعتگرد یا پاد ساعتگرد به حرکت درآورد. دیتاهای برای حرکت توسط یک سیم پیچ، دو سیم پیچ و ترکیبی از هر دو نوشته شده است. در حالت ترکیبی می توان موتور را بصورت نیم پله حرکت داد.



Clockwise Rotation ↓

Index	1a	1b	2a	2b
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1
5	1	0	0	1
6	1	1	0	0
7	0	1	1	0
8	0	0	1	1

Alternate Full Step Sequence (Provides more torque)

Index	1a	1b	2a	2b
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1
9	1	0	0	0
10	1	1	0	0
11	0	1	0	0
12	0	1	1	0
13	0	0	1	0
14	0	0	1	1
15	0	0	0	1
16	1	0	0	1

Half Step Sequence

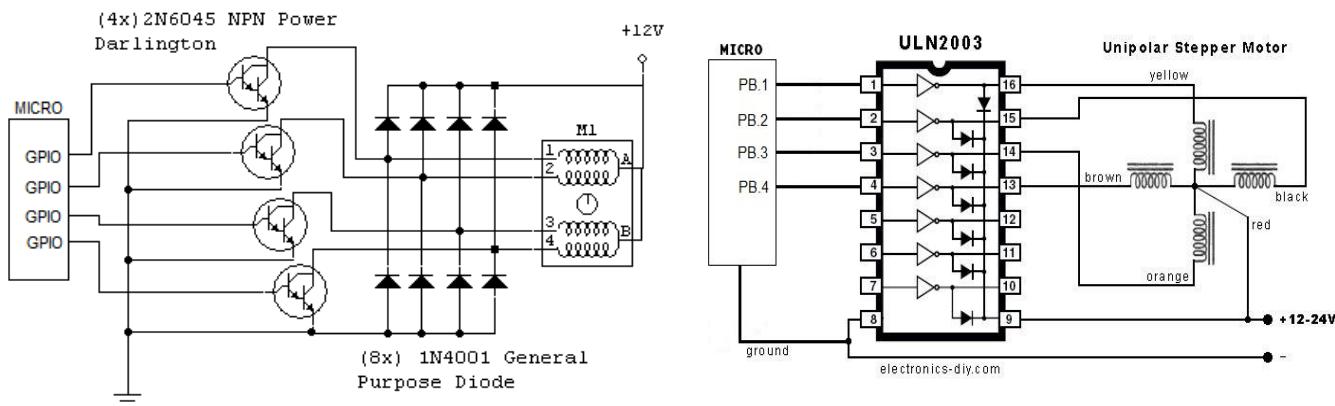
زاویه حرکت یا گام :

با تغییر از یک دیتا به دیتای دیگر موتور یک گام حرکت می کند برای چرخش 360° درجه ای موتور به تعداد مشخصی گام نیاز است اگر تعداد گام های لازم برای یک دور کامل n باشد در این صورت زاویه

$$\text{angle} = \frac{360}{n}$$

درایور : Driver

با توجه به این که جریان لازم برای راه اندازی و حرکت دادن موتورها زیاد است نمی توان موتور را مستقیم به میکرو متصل کرد. لازم است یک درایور بین موتور و میکرو قرار گیرد. برای این درایور می توان از المان های مجزا یا مدارهای مجتمع استفاده کرد.



مثال: بر روی برد آزمایش از درایور **ULN2003** استفاده شده است و به پایه های **PB1,2,3,4** متصل شده. برنامه ای بنویسید که موتور هر نیم ثانیه یک گام حرکت کند. سپس تعداد گام های لازم برای یک دور گردش کامل را شمارش و با استفاده از رابطه بالا زاویه هر گام را محاسبه کنید.

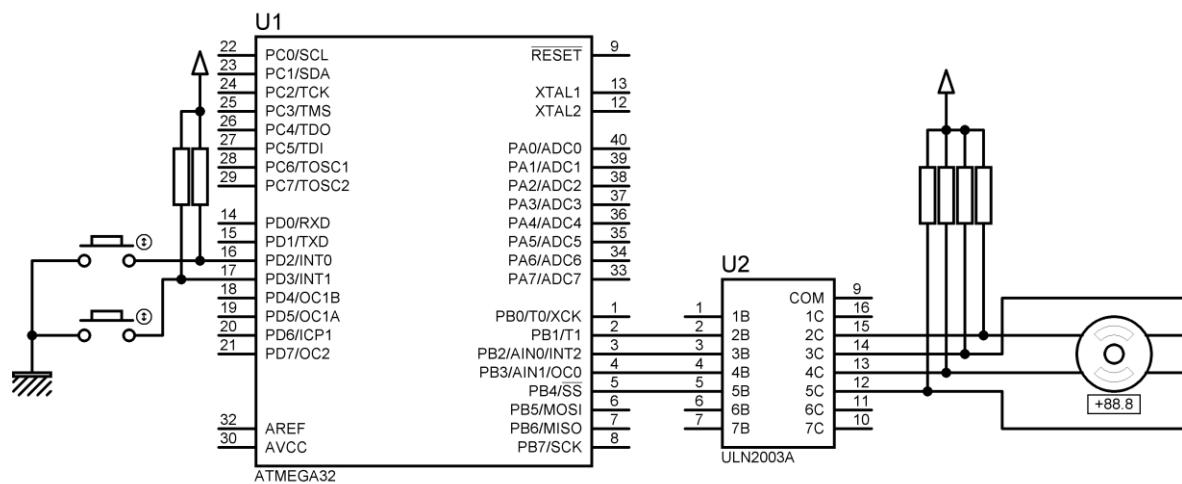
پاسخ: دیتای لازم با توجه به سخت افزار در مبنای ۲ و ۱۶ به صورت زیر است.

```
unsigned char d[ ]={0x02,0x04,0x08,0x10},i=0;
DDRB=0b00011110; // DDRB=0x1e;
while(1)
{
    for(i=0;i<4;i++)
    {
        PORTB=d[ i ];
        Delay_ms(500);
    }
}
```

0b00000010	0x02
0b00000100	0x04
0b00001000	0x08
0b00010000	0x10

تمرین: بر روی برد آزمایش دو کلید به پورت های PD.2 و PD.3 متصل است برنامه ای بنویسید که با نگه داشتن یکی از کلیدها موتور ساعتگرد و با کلید دیگر پاد ساعتگرد بچرخد.

- در برنامه پروتئوس می توانید موتور پله ای را از کتابخانه **Electromechanical** و با نام **MOTOR_BISTEPPER** بردارید. مشخصات موتور، ولتاژ و زاویه را می توانید تغییر دهید.



:Function تابع

ساختمان **C** بر پایه توابع است. کاربر می‌تواند هر بخش از برنامه را بصورت یک تابع نوشت و هرگاه لازم بود آن را فراخوانی و اجرا نماید. نوشتن برنامه بصورت توابع باعث می‌شود:

- ۱- خوانایی برنامه بالا رود.
 - ۲- رفع اشکال از برنامه ساده تر انجام می شود.
 - ۳- اصلاح و ارتقاء برنامه ساده تر صورت گیرد.
 - ۴- از توابع نوشته شده می توان در برنامه های دیگر استفاده کرد.

بسته به این که تابع ورودی یا خروجی داشته باشد چهار حالت زیر را خواهیم داشت:

- | | |
|--------------------------|-------------------------|
| void f1(void) | بدون ورودی - بدون خروجی |
| void f2(unsigned char x) | با ورودی - بدون خروجی |
| char f3(void) | با خروجی - بدون ورودی |
| int f4(float y) | با خروجی - با ورودی |

نکته: اگر تابعی دارای چند ورودی از یک نوع باشد پاید تک تک آن ها معرفی شوند مانند مثال زیر:

void f1 (int x,y,z) نادرست **void f1(int x,int y,int z)** درست

نکته: هر تابع فقط می‌تواند یک خروجی داشته باشد. و برای خروج دیتا از دستور `return` استفاده می‌شود.

Return نام متغیر ; ما **Return** مقدار ثابت ;

محل قرار گیری تابع می تواند یکی از اشکال زیر باشد:

(ب)

(الف)

معرفی و تعریف تابع <pre>{ }</pre> Void main (void) <pre>{ فراخوانی تابع }</pre>	; معرفی و تعریف تابع Void main (void) <pre>{ فراخوانی تابع }</pre> معرفی و تعریف تابع <pre>{ }</pre>
--	--

اگر توابع مستقل بوده ، و از داخل یک تابع ، تابع دیگری فراخوانی نشود شکل الف یا ب تفاوتی نخواهد داشت در غیر این صورت باید روش الف را استفاده کنید.

مثال: برنامه ای بنویسید که دو عدد چهار بیتی را از **PORTB** و **PORTA** وارد کند سپس حاصل جمع آن ها را روی **PORTC** و حاصل تفریق آن ها را روی **PORTD** نمایش دهد. این برنامه را یک بار بدون استفاده از تابع و یک بار با استفاده از تابع بنویسید.

```

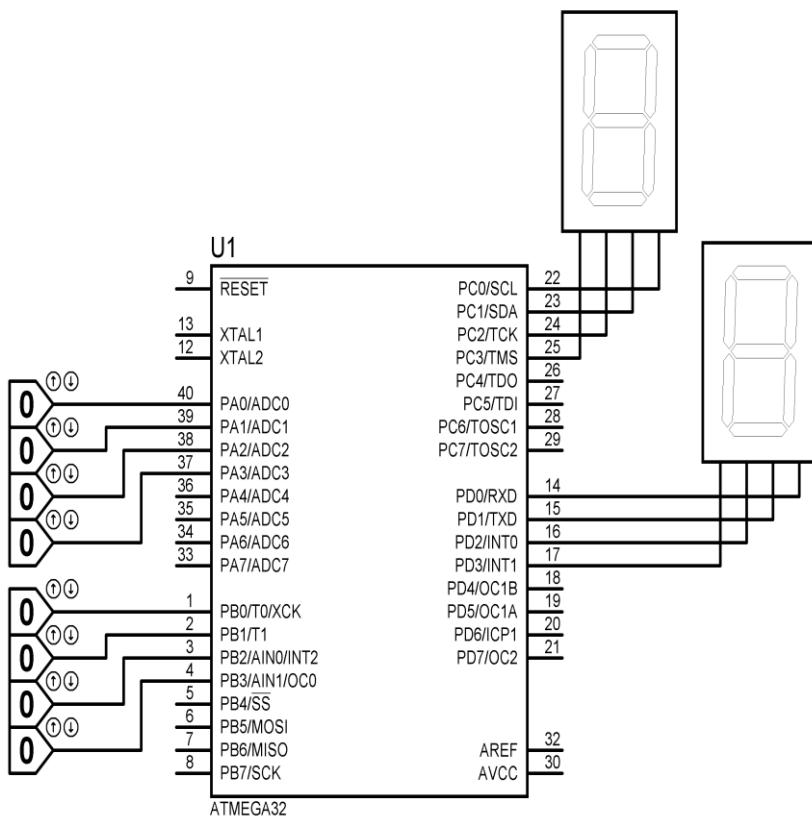
#include <mega32.h>

unsigned char a,b,c,d;

void main (void)

{
    DDRA=0x00;
    DDRB=0x00;
    DDRC=0xff;
    DDRD=0xff;
    While(1)
    {
        a=PIN A & 0x0f;
        b=PIN B & 0x0f;
        c=a+b;
        d=a-b;
        PORTC=c;
        PORTD=d;
    }
}

```



```
#include <mega32.h>

void init_port (void) ;

void input (void) ;

unsigned char sum (unsigned char x, unsigned char y) ;

unsigned char sub (unsigned char x, unsigned char y) ;

void output (unsigned char x, unsigned char y) ;

unsigned char a,b,c,d;

void main (void)

{

    Init_port();

    While(1)

    {

        input();

        c=sum(a,b);

        d=sub(a,b);

        output(c,d);

    }

}

void init_port (void)

{

    DDRA=0x00;

    DDRB=0x00;

    DDRC=0xff;

    DDRD=0xff;

}
```

```
void input (void)
{
    a=PINB & 0x0f;
    b=PINB & 0x0f;
}

unsigned char sum (unsigned char x, unsigned char y)
{
    unsigned char z;
    z=x+y;
    return z;
}

unsigned char sub (unsigned char x, unsigned char y)
{
    unsigned char z;
    z=x-y;
    return z;
}

void output (unsigned char x, unsigned char y)
{
    PORTC=x;
    PORTD=y
}
```

اشاره گر : POINTER

هرگاه لازم باشد به جای متغیر از آدرس آن متغیر استفاده کنیم می توان توسط اشاره گر با آدرس آن حافظه کار کرد. کاربرد اشاره گر در تخصیص حافظه پویا ، دسترسی آسان تر و کارکردن راحت تر با آرایه ها و ارسال متغیرها به توابع با ارجاع و..... می باشد.

در برنامه هایی که برای میکرو می نویسیم بیشترین کاربرد زمانی است که بخواهیم با فراخوانی یک تابع چند مقدار را بدست آوریم ، از آنجایی که هر تابع نمی تواند بیش از یک خروجی داشته باشد لازم است آرگومان های تابع از طریق فراخوانی با ارجاع به تابع ارسال شود. برای مثال در خواندن زمان از آی سی DS1307 باید از تابع **rtc_get_time** استفاده شود ، پس باید سه مقدار، ساعت_دقیقه_ثانیه از تابع خارج شود که امکان پذیر نیست و باید از فراخوانی با ارجاع استفاده شود. پس به جای خود متغیر از آدرس آن ها استفاده می کنیم.

rtc_grt_time(&h,&m,&s);

علامت & یعنی آدرس متغیر

تعریف متغیر از نوع اشاره گر:

برای این منظور مانند متغیر های معمولی عمل می شود با این تفاوت که قبل از نام متغیر اشاره گر یک علامت ستاره * قرار می گیرد.

Unsigned char a,b,*p,*q;

در خط بالا **a,b** دو متغیر معمولی و **p,q** از نوع اشاره گر هستند.

نکته: در هنگام نوشتن برنامه هر جا آدرس یک متغیر مدنظر بود از علامت & و هرجا محتوا یا مقدار متغیر مدنظر بود از علامت * استفاده می شود.

با توجه به تعریف زیر فرض می کنیم متغیر **a** در خانه 60 حافظه ، متغیر **b** در خانه 62 حافظه و متغیر

unsigned char a=12 , b=3 , c , *p , *q , *r; در خانه 64 حافظه ذخیره شده باشند.

با توجه به تعریف متغیری که در بالا انجام شد می توان شکل زیر را تصور کرد.

متغیر	آدرس	محتوی
	59	
a	60	12
	61	
b	62	3
	63	
c	64	
	65	
	66	

حال برنامه زیر را در نظر بگیرید:

- 1) $p = \&a;$
- 2) $q = \&b;$
- 3) $r = \&c;$
- 4) $*r = *p + *q;$
- 5) $*p = *q;$
- 6) $\text{PORTC} = a + b;$

در سه خط اول ، آدرس حافظه هایی که متغیر های a,b,c در آن جا ذخیره شده اند درون اشاره گر های p,q,r قرار می گیرند یعنی:

$$p=60 \quad q=62 \quad r=64$$

در خط چهارم محتوای محل هایی که p و q به آن ها اشاره می کنند با هم جمع شده در حافظه ای که اشاره گر r به آن اشاره می کند یعنی خانه 64 حافظه قرار می گیرد.

در خط پنجم محتوای متغیر مورد اشاره q یعنی مقدار متغیر b در متغیر مورد اشاره p یعنی متغیر a قرار می گیرد به عبارت دیگر مقدار a و b با هم برابر می شوند.

در نتیجه خط ششم باعث خواهد شد روی PORTC عدد 6 را داشته باشیم.

متغیر از نوع ساختار structure

هرگاه تعدادی متغیر غیر هم نوع داشته باشیم و بخواهیم آن ها در قالب یک متغیر تعریف کنیم لازم است که از متغیرهای نوع ساختار استفاده شود.

برای مثال یک فرم استخدام را در نظر می گیریم .

ردیف	نام	نام خانوادگی	سن	معدل
۱	علی	بارانی	۲۳	۱۵/۷۵
۲	آرش	غیور	۲۴	۱۶/۲۵
۳	پروین	میرزایی	۲۲	۱۷/۳۳
۴	سايه	حیدری	۲۶	۱۴/۵

هر سطر یا Record در این فرم دارای پنج field می باشد ، ردیف و سن اعداد صحیح ، نام و نام خانوادگی از نوع رشته و معدل از نوع اعشاری می باشند.

حال می خواهیم متغیری تعریف کنیم که هر پنج قسمت را در خود داشته باشد.

برای این منظور از متغیر نوع struct و به شکل زیر استفاده می کنیم.

struct { نام ساختار

عناصر ساختار

; نام متغیرها}

struct form{

int id;

char name[20];

char family[20];

int age;

float average;

}f;

دسترسی به عناصر ساختار:

برای دسترسی به هر یک از عناصر ساختار باید به شکل زیر عمل کنیم:

مقدار=نام_فیلد.نام متغیر

در مثال بالا نام متغیر **f** می باشد ، می خواهیم عناصر سطر اول را وارد کنیم.

```
f.id=1;
f.name=ali;
f.family=barani;
f.age=23;
f. average=15.75;
```

در برنامه کدویزن هنگامی که می خواهیم کنترل بیتی روی بیت های هر یک از پورت ها داشته باشیم

از این نوع متغیر استفاده می شود.

DDRA.5=1;

If (PIND.2==0) PORTD.7=1;

