به نام خدا

آشنایی با زبان AVR C

تبدیل انواع داده

Dr. Aref Karimiafshar
A.karimiafshar@iut.ac.ir

BCD Code

- BCD (Binary coded decimal) number system
  - Binary representation of 0 to 9
  - Because in everyday life we use the digits 0 to 9

- BCD
  - Unpacked BCD
  - Packed BCD

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

اعداد BCD:
ما با عدد 0 تا 9 کار میکنیم و بیشتر دوست داریم با این ها کار بکنیم و میایم یک نمایش باینری از اعداد 0 تا 9 مطرح میکنیم
دو نوع BCD داریم:
– Unpacked BCD
– Packed BCD

BCD میاد همین اعداد 0 تا 9 ما به صورت روزمره ازشون استفاده میکنی رو به صورت باینری نمایش میده تا بتونه در کامپیوتر ذخیره بکنه

BCD Code

- Unpacked BCD

In unpacked BCD, the lower 4 bits of the number represent the BCD number, and the rest of the bits are 0. For example, "0000 1001" and "0000 0101" are unpacked BCD for 9 and 5, respectively. Unpacked BCD requires 1 byte of memory, or an 8-bit register, to contain it.

- Packed BCD

In packed BCD, a single byte has two BCD numbers in it: one in the lower 4 bits, and one in the upper 4 bits. For example, "0101 1001" is packed BCD for 59H. Only 1 byte of memory is needed to store the packed BCD operands. Thus, one reason to use packed BCD is that it is twice as efficient in storing data.

در حالت Unpacked BCD ما از 4 بیت برای نمایش اعداد 0 تا 9 استفاده میکنیم و باتوجه به اینکه AVR رجیسترهای 8 بیتی داره ما نیاز داریم بقیه بیت ها رو هم پر بکنیم و برای این کار ما نیاز 4 بیت کم ارزش رو با نمایش باینری این اعداد 0 تا 9 کار میکنیم و برای 4 بیت پر ارزش با صفر پر میکنیم پس در این حالت ما یک نمایش 8 بیتی داریم

Packed BCD

از این فضای رجیستری 8 بیتی ما بهتر استفاده میکنه ینی در هر 8 بیت دو عدد BCD رو نمایش میده

مثلا برای 59 :

عدد 5 میشه و عدد 9 که کنار هم می ذاریمش و میشه حالت Packed BCD

در حالت کلی حالت Packed BCD به صورت موثرتری نسبت به UNPacked BCD داره عمل میکنه

ASCII Numbers

- On ASCII keyboards, when the key "0" is activated
  - "0110000" (30H) is provided to the computer

ASCII and BCD Codes for Digits 0–9

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

با کدهای اسکی هم به عنوان فرمت دیگری استفاده می کنیم
یکسری کیبوردها داریم و وقتی یک کلیدی روی ان ها زده میشه یک کد اسکی برای سیستم کامپیوتری ما فراهم میکنه

Packed BCD to ASCII Conversion

- In many systems, we have RTC (real time clock)
  - The RTC provides
    - The time of day (hour, minute, second)
    - The date (year, month, day)

  Continuously and regardless of whether the power is on or off
  - This data is provided in packed BCD
    - To be displayed on a device such as LCD
      - It must be in ASCII format
- To convert packed BCD to ASCII
  - First, convert it to unpacked BCD
  - Then, unpacked BCD is tagged with 011 0000 (30H)

Packed BCD	Unpacked BCD	ASCII
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 &	0011 0010 &
	0000 1001	0011 1001

چجوری این ها با استفاده از RTC بهم دیگه تبدیل کنیم و کجاها می تونیم از اون ها استفاده بکنیم؟
یکی از مواردی که نیاز پیدا خواهیم کرد که این تبدیل هارو انجام بدیم در نمایش تاریخ و ساعت سیستم است
نکته: فرمت این داده ای که RTC در اختیار ما قرار میده به صورت نمایش یک Packed BCD است از طرفی هم برای اینکه ما بیایم اونو روی یکسری ابزارها مثل LCD نمایش بدیم نیازمند این هستیم ک اینها رو به فرمت اسکی تبدیل بکنیم پس میایم و اون نمایش BCD که RTC در اختیار ما قرار میده به اسکی تبدیل میکنیم که بتونیم به راحتی روی LCD نمایش بدیم
برای اینکه بیایم این تبدیل فرمت رو انجام بدیم باید یکسری کارهایی انجام بدیم که دوتا گام داره:
گام اول: اون رو تبدیل می کنیم به unpacked BCD و وقتی که به unpacked BCD تبدیل شد می تونیم با برچسب زدن مقدار 30 به اون مقدارهایی که به صورت BCD به دست اوردیم اون هارو به فرمت اسکی تبدیل بکنیم

مثلا اینجا عدد 29 هگز رو در فرمت packed BCD داریم و وقتی بخوایم اینو تبدیل بکنیم به فرمت اسکی اول میایم هر کدوم از این اعداد رو به فرمت unpacked BCD در میاریم ینی 2 جدا و 9 هم جدا و بعد کافیه تگ 30 هگز رو به این ها اضافه بکنیم تا به اون عدد اسکی که میخوایم برسیم

Packed BCD to ASCII Conversion

Write an AVR C program to convert packed BCD 0x29 to ASCII and display the bytes on PORTB and PORTC.

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
   unsigned char x, y;
   unsigned char mybyte = 0x29;

   DDRB = DDRC = 0xFF;                 //make Ports B and C output
   x = mybyte & 0x0F;                  //mask upper 4 bits
   PORTB = x | 0x30;                   //make it ASCII
   y = mybyte & 0xF0;                  //mask lower 4 bits
   y = y >> 4;                         //shift it to lower 4 bits
   PORTC = y | 0x30;                   //make it ASCII

   return 0;
}
```

مثال: میخوایم یک برنامه ای بنویسیم ک یک عدد packed BCD مثلا 29 هگز رو به صورت اسکی تبدیل بکنه و اونو نمایش بدیم روی پورت B , C

فرض می کنیم عدد 29 هگز رو در یک خانه ای از حافظمون ذخیره کردیم
قسمت بالاتر اونو می ریزیم ماسک می کنیم و این باعث میشه که بتونیم 4 بیت کم ارزش اونو به دست بیاریم این کمک میکنه که عدد 9 رو به دست بیاریم = x
و برای به دست اوردن 4 بیت بالاتر این دفعه مقدار عدد اصلی رو با F0 هگز and می کنیم تا قسمت بالاتر اون به دست بیاد و این رو داخل y می ریزیم و چون این مقدار در قسمت بالاتر اون قرار داره میایم 4 بیت اونو شیفت میدیم به سمت راست و داخل y می ریزیم
در نهایت x رو به صورت اسکی روی پورت B و y به صورت اسکی روی پورت C نمایش میدیم

ASCII to Packed BCD Conversion

- To convert ASCII to packed BCD
  - First, convert it to unpacked BCD
  - Then, combine it to make packed BCD

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 which is 47H

برای تبدل کد اسکی به Packed BCD:
روندها:
ابتدا اون هارو به صورت unpacked BCD تبدیل میکنیم و بعد اون هارو تبدیل میکنیم به اون نوع Packed BCD که مد نظر ما هستش
مثال:
4 و 7 دو تا کلیدی بوده که زده شده که کد اسکیشون هم نوشته و بعد اون هارو به فرمت unpacked BCD درمیاریم و بعد اونا رو به حالت Packed BCD درمیاریم ینی اون چیزی که مد نظر ما هستش

ASCII to Packed BCD Conversion

Write an AVR C program to convert ASCII digits of '4' and '7' to packed BCD and display them on PORTB.

```
#include <avr/io.h>                 //standard AVR header

int main(void)
{
   unsigned char bcdbyte;
   unsigned char w = '4';
   unsigned char z = '7';
   DDRB = 0xFF;                     //make Port B an output
   w = w & 0x0F;                    //mask 3
   w = w << 4;                      //shift left to make upper BCD digit
   z = z & 0x0F;                    //mask 3
   bcdbyte = w | z;                 //combine to make packed BCD
   PORTB = bcdbyte;

   return 0;
}
```

در این برنامه دوتا کد اسکی رو به یک Packed BCD تبدیل میکنه:

یک متغییر تعریف میکنیم که بیاد این متغییر BCD مارو در خودش ذخیره بکنه به عنوان bcdbyte
دو تا عدد رو توی حافظمون داریم به صورت کد اسکی و قراره اینارو روی پورت B نمایش بدیم
در ابتدا قسمت پایین تر اون عدد مورد نظر را با 0f هگز and میکنیم و بعد اینو شیفت میدیم به سمت چپ که در 4 بیت بالاتر قرار بگیره
و بعد عدد دوم رو قسمت پایین ترش رو برداریم و بعد این دوتا قسمت رو با or کردن کنار هم دیگه قرار بدیم و این چیزی که در bcdbyte قرار می گیره 4 بیت بالاترش قسمت عدد مرتبط با w هستش ینی 4 و قسمت پایین ترش عدد z ما ینی 7 هستش و شد الان یک Packed BCD

Checksum byte in ROM

- To ensure the integrity of data
  - Systems must perform the checksum calculation
- We perform checksum calculation
  - When transmit data from one device to another
  - When save or restore data to a storage device
- The checksum will detect any corruption of data
- Checksum process uses what is called
  - *Checksum byte*
    - The extra byte that is tagged to the end of a series of bytes of data

ذخیره سازی اعداد در حافظه یا زمانی که قراره داده رو از یه جایی به یه جای دیگه ای منتقل بکنیم پس برای اینکه از صحت داده هامون در یک انتقال مطمئن بشیم نیاز داریم یکسری چک هایی رو انجام بدیم که یکی از این چک ها میتونه Checksum باشه
بنابراین در دو نوع پرکاربرد که میاین Checksum مطرح میکنن:
یکی زمانی که داده قراره از یک ابزار به ابزار دیگه ای منتقل بشه و اینجا میاین یک Checksum طراحی می کنن و بعد اون چک میکنن که ببینن این انتقال درست انجام شده یا نه
و مورد دیگه ای که مطرح مال زمانیه که داده ها رو در یک ابزاری ذخیره میکنیم و قصد داریم اونو بازیابی بکنیم یا ... ممکنه یکسری اتفاق های ناخواسته بیوفته و داده هایی رو از دست بدیم اینجا هم معموله که میاین Checksum رو تعریف می کنن برای اینکه بتونن اگر داده های دچار اشکالی شده اونو تشخیص بدن

فرایند ایجاد و طراحی Checksum به این صورته که میاین یک بایتی رو در انتهای یک سری از بایت ها که مدنظرشون هست قرار میدن تحت عنوان Checksum byte
نحوه کار به این صورته که چندین بایت از داده رو که ما قصد داریم در رابطه با اون ها این چک رو انجام بدیم میایم و اون هارو با هم دیگه جمع میکنیم و درون یک بایت جدیدی تحت عنوان Checksum byte قرار میدیم

Checksum byte in ROM

- To calculate checksum byte of a series of bytes of data
  - Add the bytes together and drop the carries
  - Take the 2's complement of the total sum. This is the checksum byte, which becomes the last byte of the series
- To perform the checksum operation
  - Add all the bytes, including the checksum byte
  - The result must be zero
    - If it is not zero, one or more bytes of data have been changed

کل فرایند:
ما توی حافظه یا جایی که قراره انتقال صورت بدیم یکسری بایت پشت سر هم داریم و اونها رو بدون اینکه کری خروجی از اون ها رو در نظر بگیرید با هم دیگه جمع می کنیم و بعد مکمل اون جمع کلی رو میگیریم و این میشه Checksum byte که در انتهای اون سری از بایت هایی که قراره Checksum رو انجام بدیم و صحت اونارو چک بکنیم قرار گرفته
برای اینکه عملیات Checksum رو انجام بدیم کافیه که همه اون بایت هایی رو که به صورت اورجینال داشتیم به علاوه اون بایت Checksum با هم دیگه جمع بکنیم و اگر مجموع همه این ها صفر شد ینی داده های ما به صورت کامل همون داده های اصلی ما هستش و هیچ اشکالی در اونها پیش نیومده ولی اگر جمع این ها صفر نشد نشون دهنده اینه که یکی یا چند تا از داده های ما تغییر پیدا کرده و باعث شده این جمع صفر نشه

Checksum byte in ROM

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H.
(a) Find the checksum byte, (b) perform the checksum operation to ensure data integrity, and (c) if the second byte, 62H, has been changed to 22H, show how checksum detects the error.

(a) Find the checksum byte.

```
    25H
+   62H
+   3FH
+   52H
  1 18H  (dropping carry of 1 and taking 2's complement, we get E8H)
```

(b) Perform the checksum operation to ensure data integrity.

```
    25H
+   62H
+   3FH
+   52H
+   E8H
  2 00H  (dropping the carries we get 00, which means data is not corrupted)
```

فرض میکنیم 4 بایت داده به صورت هگز است:
25 هگز و 62 هگز و 3f هگز و 52 هگز
در گام اول میخوایم یک Checksum برای این بایت ها که توی حافظه پشت سر هم قرار دارند پیدا بکنیم برای این کار میایم و اون هارو بدون کری خروجی در نظر میگیریم و با هم جمع میکنیم و بعد مکمل دو اون عدد رو حساب میکنیم
توی قسمت b : میخوایم Checksum رو انجام بدیم که مطمئن بشیم داده های ما سالم هستند وتغییری در اون ها پیش نیومده مثلا داده رو منتقل کردیم به ابزار دیگری و این ها ذخیره شده و می خوایم چک بکنیم داده ها سالم اند یا نه برای این کار کافیه عددهای مرحله قبل رو با Checksum جمع بکنیم اگر جواب این جمع شد صفرینی داده ها تغییر نکردند
حین انتقال مقدار 62 هگز تغییر کرده و 22 هگز تشخیص داده شده : c
صفحه بعدی..

Checksum byte in ROM

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H.
(a) Find the checksum byte, (b) perform the checksum operation to ensure data integrity, and (c) if the second byte, 62H, has been changed to 22H, show how checksum detects the error.

(c) If the second byte, 62H, has been changed to 22H, show how checksum detects the error.

```
    25H
+   22H
+   3FH
+   52H
+   E8H
  1 C0H  (dropping the carry, we get C0H, which means data is corrupted)
```

این بار جمع این اعداد صفر نمیشه و صفر نشدن به معنای این است که یکی یا چندتا از داده های ما دچار مشکل شده و مقدار صحیحی رو در خودش نداره

Checksum byte

Write an AVR C program to calculate the checksum byte for the data: 25H, 62H, 3FH, and 52H.

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
      unsigned char mydata[ ]  = { 0x25,0x62,0x3F,0x52} ;
      unsigned char sum = 0;
      unsigned char x;
      unsigned char chksumbyte;

      DDRA = 0xFF;                     //make Port A output
      DDRB = 0xFF;                     //make Port B output
      DDRC = 0xFF;                     //make Port C output

      for(x=0; x<4; x++)
      {
            PORTA = mydata[ x] ;       //issue each byte to PORTA
            sum = sum + mydata[ x] ;   //add them together
            PORTB = sum;               //issue the sum to PORTB
      }
      chksumbyte = ~sum + 1;           //make 2's complement (invert +1)
      PORTC = chksumbyte;              //show the checksum byte
      return 0;
}
```

اعداد 25 هگز و 62 هگز و 3f و 52 هگز رو در خانه های پشت سر هم حافظه داریم و می خوایم Checksum byte اون هارو به دست بیاریم
از اخر خط 3: میایم مکمل دو sum رو حساب میکنیم و اونو به عنوان Checksum byte داریم

Checksum byte

Write a C program to perform the checksum operation. If the data is good, send ASCII character 'G' to PORTD. Otherwise, send 'B' to PORTD.

```
#include <avr/io.h>                          //standard AVR header
int main(void)
{
     unsigned char mydata[ ]  = { 0x25,0x62,0x3F,0x52,0xE8} ;
     unsigned char chksum = 0;
     unsigned char x;
     DDRD = 0xFF;                            //make Port D an output
     for(x=0;x<5;x++)
       chksum = chksum + mydata[ x] ;        //add them together
     if(chksum == 0)
       PORTD = 'G';
     else
       PORTD = 'B';
     return 0;
}
```

توی این برنامه می خوایم برنامه ای بنویسیم که Checksum byte رو چک بکنه ینی داده های ما مقادیر صحیحی دارند هنوز یا نه و اگر مقادیر صحیح هستش مقدار G رو روی پورت D می نویسیم و اگر صحیح نبودن روی پورت D مقدار B قرار میدیم

Binary (hex) to decimal and ASCII

The printf function is part of the standard I/O library in C and can do many things including converting data from binary (hex) to decimal, or vice versa. But printf takes a lot of memory space and increases your hex file substantially. For this reason, in systems based on the AVR microcontroller, it is better to know how to write our own conversion function instead of using printf.

One of the most widely used conversions is binary to decimal conversion. In devices such as ADCs (Analog-to-Digital Converters), the data is provided to the microcontroller in binary. In some RTCs, the time and dates are also provided in binary. In order to display binary data, we need to convert it to decimal and then to ASCII. Because the hexadecimal format is a convenient way of representing binary data, we refer to the binary data as hex. The binary data 00–FFH converted to decimal will give us 000 to 255. One way to do that is to divide it by 10 and keep the remainder.

Hex	Quotient	Remainder
FD/0A	19	3 (low digit) LSD
19/0A	2	5 (middle digit)
		2 (high digit) (MSD)

یک نوع دیگه فرمتی که برای داده داریم و به صورت وسیع استفاده میشه دریافت اطلاعات ورودی به صورت باینری هستش و بعد ما نیاز پیدا خواهیم کرد اینارو به صورت دسیمال یا اسکی نمایش بدیم
ما در زبان سی کتابخونه standard I/O رو داریم که داخل اون یک تابع داریم تحت عنوان printf
printf خیلی از انواع داده ای که مدنظر ما هستش از جمله تبدیل باینری به دسیمال و برعکس رو انجام میده اما نکته ای که در مورد printf وجود داره اینه که printf فضای حافظه زیادی رو اشغال میکنه و ما دوست داریم فایل هگز تا جاییی که می تونه سایز کمی داشته باشه پس با توجه به اینکه printf نیاز به فضای زیادی داره در سیستم هایی که ما در رابطه با حافظه اون ها نگرانی داریم پس بهتره به جای اینکه از printf بریم بیایم اون تبدیلاتی که مدنظرمون است رو خودمون اعمال بکنیم که حجم حافظه ما و فایل هگز ما خیلی زیاد نشه
یکی از این تبدیلات Binary (hex)to decimal and ASCII است
وقتی یک عددی هگز است می تونه بین 00 تا FF هگز باشه برای یک بایت و وقتی اونو تبدیل میکنیم به دسیمال می تونه بین 000 تا 255 باشه یکی از روش هایی که این تبدیل رو میتونیم با استفاده از اون انجام بدیم با استفاده از تقسیم های متوالی بر 10 هستش و با استفاده از باقی مانده ای که ته تقسیم باقی می مونه می تونیم این تبدیل رو انجام بدیم
مثلا یک عدد هگز داریم و می خوایم اونو تبدیل بکنیم به دسیمال پس میایم مرتب اون عدد هگز رو بر 10 تقسیم می کنیم

Binary (hex) to decimal and ASCII

Write an AVR C program to convert 11111101 (FD hex) to decimal and display the digits on PORTB, PORTC, and PORTD.

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
     unsigned char x, binbyte, d1, d2, d3;
     DDRB = DDRC = DDRD =0xFF;         //Ports B, C, and D output
     binbyte = 0xFD;                   //binary (hex) byte
     x = binbyte / 10;                 //divide by 10
     d1 = binbyte % 10;                //find remainder (LSD)
     d2 = x % 10;                      //middle digit
     d3 = x / 10;                      //most-significant digit (MSD)
     PORTB = d1;
     PORTC = d2;
     PORTD = d3;

     return 0;
}
```

مثال:
یک برنامه ای بنویسیم که یک عدد هگز FD رو به یک عدد دسیمال تبدیل بکنه و اونو نمایش بدیم روی یک سری پورت

Data type conversion functions in C

Many compilers have some predefined functions to convert data types.
To use these functions, the stdlib.h file should be included.
Notice that these functions may vary in different compilers.

Data Type Conversion Functions in C

Function signature	Description of functions
int atoi(char *str)	Converts the string str to integer.
long atol(char *str)	Converts the string str to long.
void itoa(int n, char *str)	Converts the integer n to characters in string str.
void ltoa(int n, char *str)	Converts the long n to characters in string str.
float atof(char *str)	Converts the characters from string str to float.

تبدیلات دیگری هم در سی وجود داره:
که توی جدول نوشته

Data serialization in C

Serializing data is a way of sending a byte of data one bit at a time through a single pin of a microcontroller. There are two ways to transfer a byte of data serially:

1. Using the serial port. In using the serial port, the programmer has very limited control over the sequence of data transfer.

2. The second method of serializing data is to transfer data one bit a time and control the sequence of data and spaces between them. In many new generations of devices such as LCD, ADC, and EEPROM, the serial versions are becoming popular because they take up less space on a printed circuit board. Although we can use standards such as I^2C, SPI, and CAN, not all devices support such standards. For this reason we need to be familiar with data serialization using the C language.

توزیع داده یا انتقال داده به صورت سریال:
به صورت کلی خود بحث سریال یک روشی از ارسال یک بایت به صورت بیت به بیت است ینی ممکنه محدودیت پهنای باند داشته باشیم و این امکان برای ما وجود نداره که همزمان تمام بایت ها رو بتونیم انتقال بدیم و بیت به بیت از طریق یک پین توی میکروکنترلر بخوایم این کارو انجام بدیم
به صورت کلی برای این انتقال که یک انتقال کم هزینه هم هست دو روش داریم:
1- استفاده از پورت های سریال و پروتکل های اماده ای که وجود داره منتها توی این حالت اون برنامه نویس دستش بستس و تحت کنترل اون پروتکل های اماده ای است که وجود داره
2- خودمون روی اون داده ای که قراره ارسال بشه کنترل داشته باشیم و بخوایم این کارو به صورت دستی انجام بدیم که این هم معمولا ابزارهایی که قراره با اون کار بکنیم به نیاز پیدا می کنیم که خودمون بسته به کاربرد بخوایم از اون استفاده بکنیم ؟؟

Data serialization in C

Write an AVR C program to send out the value 44H serially one bit at a time via PORTC, pin 3. The LSB should go out first.

```
#include <avr/io.h>
#define serPin 3

int main(void)
{
     unsigned char conbyte = 0x44;
     unsigned char regALSB;
     unsigned char x;
     regALSB = conbyte;
     DDRC |= (1<<serPin);

    for(x=0;x<8;x++)
       {
         if(regALSB & 0x01)
             PORTC |= (1<<serPin);
        else
             PORTC &= ~(1<<serPin);
         regALSB = regALSB >> 1;
       }
      return 0;
}
```

می خوایم یک برنامه ای بنویسیم که مقدار 44 هگز رو به صورت سریالی ینی یک بیت در همزمان از طریق پین شماره 3 پورت C ارسال بکنه و پروتکل ما هم اینه که اول کم ارزش ترین بیت رو ارسال بکنه

این داده رو در یک بایت ذخیره میکنیم ینی توی conbyte ذخیره اش کردیم حالا conbyte رو توی یک regALSB می ریزیم برای اینکه اصل اونو دستکاری نکنیم این کارو کردیم
توی for بیت میایم اون عدد رو با یک andاش میکنیم که بتونیم کم ارزش ترین بیتش رو بررسی بکنیم حالا اگر این بیت یکه ما میایم توی پورت C اونجایی که قراره این داده رو ارسال بکنیم ینی شماره سه میایم و یک رو ارسال میکنیم و اگر اون بیت یک نبود صفر ارسال میشه روی پورت و بعد یک شیفت به سمت راست میدیم تا این کار دوباره تکرار بشه و تمام بیت های اون regALSB ارسال بشه

Data serialization in C

Write an AVR C program to send out the value 44H serially one bit at a time via PORTC, pin 3. The MSB should go out first.

```
#include <avr/io.h>
#define serPin 3
int main(void)
{
     unsigned char conbyte = 0x44;
     unsigned char regALSB;
     unsigned char x;
     regALSB = conbyte;
     DDRC |= (1<<serPin);
     for(x=0;x<8;x++)
       {
         if(regALSB & 0x80)
             PORTC |= (1<<serPin);
        else
             PORTC &= ~(1<<serPin);
         regALSB = regALSB << 1;
       }
      return 0;
}
```

مثل مثال قبله ولی توی این پروتکل ما به این صورت است که اولین بیتی که ارسال میشه پرارزش ترین بیت ما باشه

تهش اون رجیستر رو به سمت چپ شیفت میدیم تا تمام این 8 بیت رو بتونیم ارسال بکنیم

Data serialization in C

Write an AVR C program to bring in a byte of data serially one bit at a time via PORTC, pin 3. The LSB should come in first.

```
//Bringing in data via PC3 (SHIFTING RIGHT)
#include <avr/io.h>                  //standard AVR header
#define serPin 3
int main(void)
{
     unsigned char x;
     unsigned char REGA=0;
     DDRC &= ~(1<<serPin);           //serPin as input
     for(x=0; x<8; x++)              //repeat for each bit of REGA
       {
        REGA = REGA >> 1;            //shift REGA to right one bit
        REGA |= (PINC &(1<<serPin)) << (7-serPin); //copy bit serPin
       }                             //of PORTC to MSB of REGA.
     return 0;
}
```

اگر بخوایم اون طرف همین داده رو دریافت بکنیم ینی اگر قراره باشه یک بایت از داده به صورت سریال بیت به بیت در هر لحظه روی پورت C شماره 3 بخونه اینجا یک نمونه کد داریم
و پروتکلی که اینجا برای دریافت اطلاعات داریم به این صورت هستش که از بیت کم ارزش می خوایم شروع بکنیم

Data serialization in C

Write an AVR C program to bring in a byte of data serially one bit at a time via PORTC, pin 3. The MSB should come in first.

```
#include <avr/io.h>                    //standard AVR header
#define serPin 3

int main(void)
{
     unsigned char x;
     unsigned char REGA=0;
     DDRC &= ~(1<<serPin);      //serPin as input
     for(x=0; x<8; x++)         //repeat for each bit of REGA
       {
        REGA = REGA << 1;       //shift REGA to left one bit
        REGA |= (PINC &(1<<serPin))>> serPin; //copy bit serPin of
       }                        //PORT C to LSB of REGA.
     return 0;
}
```

قراره اولین بیتی که وارد سیستم میشه MSB ما باشه

Flash, RAM and EEPROM memory allocation in C

- Different C compiler
  - May have their built-in functions or directives to access each type of memory
- In codeVision
  - To define a constant variable in Flash
    - Put FLASH directive before it
  - To define a variable in EEPROM
    - Put EEPROM directive in front of it

```
flash unsigned char mynum[ ]  = "Hello";    //use Flash code space
eeprom unsigned char = 7                    //use EEPROM space
```

دستیابی به انواع حافظه ها در avr سی:
دسترسی به حافظه ها خصوصا فلش و EEPROM توی کامپایلرهای متفاوتی که اطراف خودمون داریم و استفاده میکنیم خیلی می تونه متفاوت باشه و ما فانکشن ها و دایرکتیوهای متفاوتی برای دسترسی به حافظه ها داشته باشیم

EEPROM access in C

Write an AVR C program to store 'G' into location 0x005F of EEPROM.

```
#include <avr/io.h>                 //standard AVR header
int main(void)
{
   while(EECR & (1<<EEWE));         //wait for last write to finish
   EEAR = 0x5f;                     //write 0x5F to address register
   EEDR = 'G';                      //write 'G' to data register
   EECR |= (1<<EEMWE);              //write one to EEMWE
   EECR |= (1<<EEWE);               //start EEPROM write
     return 0;
}
```

ساختار کلی برای نوشتن توی EEPROM:
می خوایم کد اسکی G رو در یک محلی از حافظه EEPROM بنویسیم
اولین گام: اول چک می کنیم ببینیم نوشتنی در حال اجرا است یا نه اگر بود صبر میکنیم که تموم بشه
بعد ادرس رو در رجیستر ادرس که EEAR هست قرار میدیم
و داده رو در رجیستر داده ینی EEDR قرار می دیم
و کامنت هارو که قراره عملیات نوشتن رو انجام بدن برای اون رجیستر کنترلی ینی EECR ارسال میکنیم
بیت مرتبط با EEMWE رو یک میکنیم و بعد شروع می کنیم به نوشتن که این هم با فعال سازی بیت EEWE انجام میشه و بعد که این بیت رو فعال کردیم توی چند سیکل اون خونه مورد نظر از حافظه اون داده ای رو که در EEDR قرار دادیم می نویسه و در اون قرار میگیره

EEPROM access in C

Write an AVR C program to read the content of location 0x005F of EEPROM into PORTB.

```
#include <avr/io.h>                 //standard AVR header
int main(void)
{
   DDRB = 0xFF;                     //make PORTB an output
   while(EECR & (1<<EEWE));         //wait for last write to finish
   EEAR = 0x5f;                     //write 0x5F to address register
   EECR |= (1<<EERE);               //start EEPROM read by writing EERE
   PORTB = EEDR;                    //move data from data register to PORTB
}
```

Programming Timers in C

As we saw , the general-purpose registers of the AVR are under the control of the C compiler and are not accessed directly by C statements. All of the SFRs (Special Function Registers), however, are accessible directly using C statements. As an example of accessing the SFRs directly, we saw how to access ports PORTB–PORTD .

In C we can access timer registers such as TCNT0, OCR0, and TCCR0 directly using their names.

```
PORTB = 0x01;
DDRC = 0xFF;
DDRD = 0xFF;

TCCR1A = 0x00;
TCCR1B = 0x06;

TCNT1H = 0x00;
TCNT1L = 0x00;
```

استفاده از تایمر در زبان سی:

Example

Write a C program to toggle all the bits of PORTB continuously with some delay. Use Timer0, Normal mode, and no prescaler options to generate the delay.

```
#include "avr/io.h"
void T0Delay ( );
int main ( )
{
      DDRB = 0xFF;       //PORTB output port

      while (1)
      {
            PORTB = 0x55;      //repeat forever
            T0Delay ( );       //delay size unknown
            PORTB = 0xAA;      //repeat forever
            T0Delay ( );
      }
}

void T0Delay ( )
{
      TCNT0 = 0x20;              //load TCNT0
      TCCR0 = 0x01;              //Timer0, Normal mode, no prescaler
      while ((TIFR&0x1)==0);     //wait for TF0 to roll over
      TCCR0 = 0;
      TIFR = 0x1;                //clear TF0
}
```

روی پورت B می خوایم پایه هاشو یکی در میان ست و ریست بکنیم برای این کار از تایمر شماره صفر توی مد نرمال بدون هیچ اسکیلی می خوایم استفاده بکنیم:

Example

Write a C program to toggle only the PORTB.4 bit continuously every 70 µs. Use Timer0, Normal mode, and 1:8 prescaler to create the delay. Assume XTAL = 8 MHz.

XTAL = 8MHz ➔ $T_{machine\ cycle}$ = 1/8 MHz

Prescaler = 1:8 ➔ T_{clock} = 8 × 1/8 MHz = 1 µs

70 µs/1 µs = 70 clocks ➔ 1 + 0xFF – 70 = 0x100 – 0x46 = 0xBA = **186**

```
#include "avr/io.h"

void T0Delay ( );

int main ( )
{
      DDRB = 0xFF;       //PORTB output port

      while (1)
      {
            T0Delay ( );            //Timer0, Normal mode
            PORTB = PORTB ^ 0x10;   //toggle PORTB.4
      }
}

void T0Delay ( )
{
      TCNT0 = 186;       //load TCNT0
      TCCR0 = 0x02;      //Timer0, Normal mode, 1:8 prescaler
      while ((TIFR&(1<<TOV0))==0);  //wait for TOV0 to roll over

      TCCR0 = 0;         //turn off Timer0
      TIFR = 0x1;        //clear TOV0
}
```

می خوایم روی پورت B پایه شماره 4 با تاخیر 70 میکروثانیه این هارو ست و ریست بکنیم
از تایمر صفر استفاده می کنیم توی مد نرمال با اسکیل 1/8
و فرض میکنیم فرکانس کاری ما 8 مگاهرتز است

Example

Write a C program to toggle only the PORTB.4 bit continuously every 2 ms. Use Timer1, Normal mode, and no prescaler to create the delay. Assume XTAL = 8 MHz.

XTAL = 8 MHz ➔ $T_{machine\ cycle}$ = 1/8 MHz = 0.125 µs

Prescaler = 1:1 ➔ T_{clock} = 0.125 µs

2 ms/0.125 µs = 16,000 clocks = 0x3E80 clocks 1 + 0xFFFF – 0x3E80 = 0xC180

```
#include "avr/io.h"

void T1Delay ( );

int main ( )
{
      DDRB = 0xFF;            //PORTB output port

      while (1)
      {
            PORTB = PORTB ^ (1<<PB4); //toggle PB4
            T1Delay ( );       //delay size unknown
      }
}


void T1Delay ( )
{
      TCNT1H = 0xC1;    //TEMP = 0xC1
      TCNT1L = 0x80;

      TCCR1A = 0x00;    //Normal mode
      TCCR1B = 0x01;    //Normal mode, no prescaler

      while ((TIFR&(0x1<<TOV1))==0);      //wait for TOV1 to roll over

      TCCR1B = 0;
      TIFR = 0x1<<TOV1;       //clear TOV1
}
```

توی این مثال از تایمر یک استفاده می کنیم
و تاخیر ما اینجا 2 میلی ثانیه هستش

Counter Programming in C

Timers can be used as counters if we provide pulses from outside the chip instead of using the frequency of the crystal oscillator as the clock source. By feeding pulses to the T0 (PB0) and T1 (PB1) pins, we use Timer0 and Timer1 as Counter 0 and Counter 1, respectively. Study the next Examples to see how Timers 0 and 1 are programmed as counters using C language.

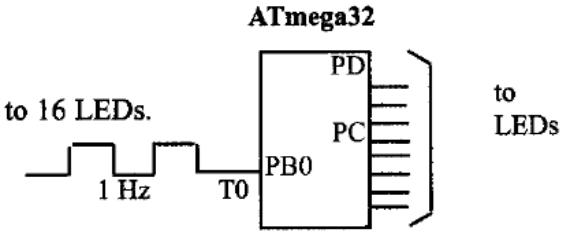
چجوری توی زبان سی می تونیم از تایمرها به عنوان کانتر استفاده بکنیم؟
مثل همون زبان اسمبلی هست: استفاده از یک پالس خارجی
برای تایمر صفر از T0
برای تایمر یک از T1

Example

Assuming that a 1 Hz clock pulse is fed into pin T0, use the TOV0 flag to extend Timer0 to a 16-bit counter and display the counter on PORTC and PORTD.

```
#include "avr/io.h"

int main ( )
{
      PORTB = 0x01;            //activate pull-up of PB0
      DDRC = 0xFF;             //PORTC as output
      DDRD = 0xFF;             //PORTD as output

      TCCR0 = 0x06;            //output clock source
      TCNT0 = 0x00;

      while (1)
      {
            do
            {
                  PORTC = TCNT0;
            }while((TIFR&(0x1<<TOV0))==0);//wait for TOV0 to roll over

            TIFR = 0x1<<TOV0;        //clear TOV0
            PORTD ++;                //increment PORTD
      }
}
```

PORTC and PORTD are connected to 16 LEDs.
T0 (PB0) is connected to a
1-Hz external clock.



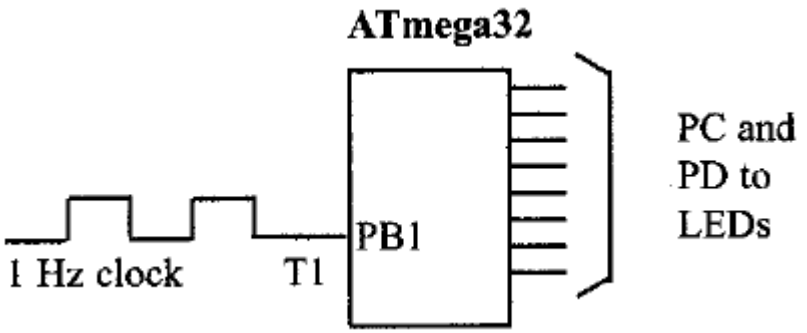
می خوایم تعداد پالس هایی که از یک منبع خارجی به پایه T0 متصل هست رو بشماریم
می خوایم از فلگ TOV0 استفاده بکنیم و تایمر را به نحوی توسعه بدیم که بتونه تا 16 بیت بشماره و نتیجه را روی پورت C , D نمایش بده

به لبه پایین رونده حساس هست اینجا

Example

Assume that a 1-Hz external clock is being fed into pin T1 (PB1). Write a C program for Counter1 in rising edge mode to count the pulses and display the TCNT1H and TCNT1L registers on PORTD and PORTC, respectively.

```
#include "avr/io.h"

int main ( )
{
      PORTB = 0x01;          //activate pull-up of PB0
      DDRC = 0xFF;           //PORTC as output
      DDRD = 0xFF;           //PORTD as output

      TCCR1A = 0x00;         //output clock source
      TCCR1B = 0x07;         //output clock source

      TCNT1H = 0x00;         //set count to 0
      TCNT1L = 0x00;         //set count to 0

      while (1)              //repeat forever
      {
            do
            {
                  PORTC = TCNT1L;
                  PORTD = TCNT1H;       //place value on pins
            }while((TIFR&(0x1<<TOV1))==0);//wait for TOV1

            TIFR = 0x1<<TOV1;           //clear TOV1
      }
}
```



تایمر یک:
یک پالسی با کلاک 1 هرتز به پایه T1 متصل هستش و میخوایم برنامه ای بنویسیم که تعداد این پالس ها رو شمارش بکنه و روی پورت C , D بنویسه و لبه بالارونده این پالس مد نظرمون هست اینجا

پایان

موفق و پیروز باشید