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

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.

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

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

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;
}
```

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

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;
}
```

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

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

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)

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)

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;
}
```

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;
}
```

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)

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;
}
```

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.

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.

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;
}
```

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;
}
```

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;
}
```

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;
}
```

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
```

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;
}
```

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;
```

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
}
```

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
}
```

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
}
```

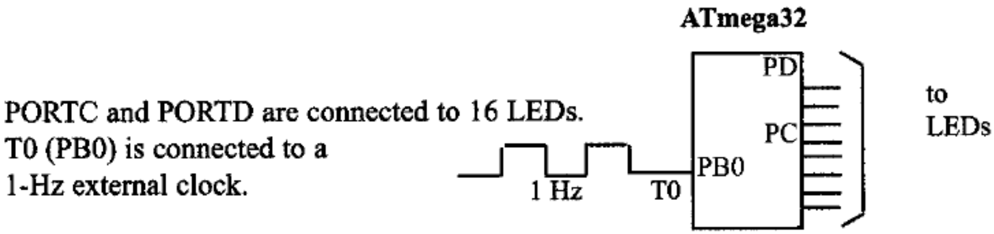
Counter Programming in C

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.

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.

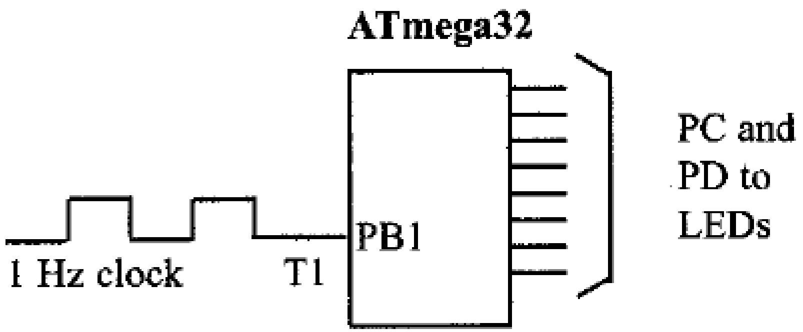
```
#include "avr/io.h"

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
      }
}
```



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
      }
}
```



پایان

موفق و پیروز باشید