MODULE 2: AVR C PROGRAMMING

AVR C data types

Following table shows commonly used data types in C. The sizes and range of values allowed for each data type may vary from one compiler to another.

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	–32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	–2,147,483,648 to +2,147,483,648
float	32-bit	±1.175e-38 to ±3.402e38
double	32-bit	±1.175e-38 to ±3.402e38

To generate smaller Hex files, suitable data type must be used. Since ATmega32 is an 8 bit microcontroller, *unsigned char* is the most preferred data type in many applications. Using this data type, we can represent from 0 to 255 (0x00 to 0xFF). For example, the following code send values 00–FF to Port B.

```
unsigned char z;
DDRB = 0xFF;
for(z = 0; z <= 255; z++)
  PORTB = z;
```

The following code send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, D to Port B.

```
unsigned char myList[]= "012345ABCD";
unsigned char z;
DDRB = 0xFF;
for(z=0; z<10; z++)
  PORTB = myList[z];
```

To represent 8 bit signed numbers (positive/negative), we use char data type. In this, the most significant bit is used to represent sign. Only seven bits are available for magnitude.
For example, the following code has an array of signed numbers. After executing this code, we get the hex values of these numbers such as 0xFC,0xFD, 0xFE , 0xFF, 0x00, 0x01, 0x02 etc.

```
char mynum[] = {-4,-3,-2,-1,0,+1,+2,+3,+4};
unsigned char z;
DDRB = 0xFF;
for(z=0; z<=8; z++)
  PORTB = mynum[z];
```

Generating time delays in C:

- There are three methods to generate a time delay in AVR C
    1. Using a simple for loop
    2. Using predefined C functions
    3. Using AVR timers

- When using loop to generate time delay, the following factors accuracy of the delay:
    1. The crystal frequency. The duration of the clock period for the instruction cycle is a function of this crystal frequency.
    2. The compiler used. If we compile a given C program with different compilers, each compiler produces different hex code.

- Predefined function, _delay_ms() can be used to generate time delay, which is defined in delay.h. The problem with predefined functions is the portability issue, because different compilers use different name for delay functions. For example, the following code sends 0x55 and 0xAA to portB continuously with one second delay.

```
while (1)
{
  PORTB=0X55;
  _delay_ms(1000);     //1000ms=1sec
  PORTB=0XAA;
  _delay_ms(1000);
}
```

AVR C – I/O programming and Logic operations.

One of the most important and powerful features of the C language is its ability to perform bit manipulation.

Bit-wise logic operators:

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

1. 0x35 & 0x0F = 0x05 /* ANDing */
2. 0x04 | 0x68 = 0x6C /* ORing */
3. 0x54 ^ 0x78 = 0x2C /* XORing */
4. ~0x55 = 0xAA /* Inverting 55H */

Compound Assignment Operators:

Operation	Abbreviated Expression	Equal C Expression
And assignment	a &= b	a = a & b
OR assignment	a |= b	a = a | b

Bit-wise Shift Operators:

Operation	Symbol	Format of Shift Operation
Shift right	>>	data >> number of bits to be shifted right
Shift left	<<	data << number of bits to be shifted left

1. 0b00010000 >> 3 = 0b00000010 /* shifting right 3 times */
2. 0b00010000 << 3 = 0b10000000 /* shifting left 3 times */
3. 1 << 3 = 0b00001000 /* shifting left 3 times */

Programming Examples: (For more examples, refer text book)

LEDs are connected to pins of Port B. Write an AVR C program that shows the count from 0 to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

Solution:

```
#include <avr/io.h>                 //standard AVR header
int main(void)
{
  DDRB = 0xFF;                      //Port B is output
  while (1)
  {
    PORTB = PORTB + 1;
  }
  return 0;
}
```

Write an AVR C program to get a byte of data from Port B, and then send it to Port C.

Solution:

```
#include <avr/io.h>                 //standard AVR header
int main(void)
{
  unsigned char temp;

  DDRB = 0x00;                      //Port B is input
  DDRC = 0xFF;                      //Port C is output

  while(1)
  {
    temp = PINB;
    PORTC = temp;
  }
  return 0;
}
```

Write an AVR C program to get a byte of data from Port C. If it is less than 100, send it to Port B; otherwise, send it to Port D.

Solution:

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
  DDRC = 0;                            //Port C is input
  DDRB = 0xFF;                         //Port B is output
  DDRD = 0xFF;                         //Port D is output
  unsigned char temp;
  while(1)
  {
    temp = PINC;                       //read from PINB
    if ( temp < 100 )
      PORTB = temp;
    else
      PORTD = temp;
  }
  return 0;
}
```

Write an AVR C program to monitor bit 5 of port C. If it is HIGH, send 55H to Port B; otherwise, send AAH to Port B.

Solution:

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

int main(void)
{
  DDRB = 0xFF;                 //PORTB is output
  DDRC = 0x00;                 //PORTC is input
  DDRD = 0xFF;                 //PORTB is output

  while(1)
  {
    if (PINC & 0b00100000)     //check bit 5 (6th bit) of PINC
      PORTB = 0x55;
    else
      PORTB = 0xAA;
  }

  return 0;
}
```

A door sensor is connected to bit 1 of Port B, and an LED is connected to bit 7 of Port C. Write an AVR C program to monitor the door sensor and, when it opens, turn on the LED.

Solution:

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

int main(void)
{
  DDRB = DDRB & 0b11111101;         //pin 1 of Port B is input
  DDRC = DDRC | 0b10000000;         //pin 7 of Port C is output

  while(1)
  {
    if (PINB & 0b00000010)          //check pin 1 (2nd pin) of PINB
      PORTC = PORTC | 0b10000000;   //set pin 7 (8th pin) of PORTC
    else
      PORTC = PORTC & 0b01111111;   //clear pin 7 (8th pin) of PORTC
  }
  return 0;
}
```

The data pins of an LCD are connected to Port B. The information is latched into the LCD whenever its Enable pin goes from HIGH to LOW. The enable pin is connected to pin 5 of Port C (6th pin). Write a C program to send “The Earth is but One Country” to this LCD.

Solution:

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

int main(void)
{
  unsigned char message[] = "The Earth is but One Country";
  unsigned char z;

  DDRB = 0xFF;                      //Port B is output
  DDRC = DDRC | 0b00100000;         //pin 5 of Port C is output

  for ( z = 0; z < 28; z++)
  {
    PORTB = message[z];
    PORTC = PORTC | 0b00100000;     //pin LCD_EN of Port C is 1
    PORTC = PORTC & 0b11011111;     //pin LCD_EN of Port C is 0
  }
  while (1);
  return 0;
}
```

Write an AVR C program to read pins 1 and 0 of Port B and issue an ASCII character to Port D according to the following table:

pin1	pin0	
0	0	send '0' to Port D (notice ASCII '0' is 0x30)
0	1	send '1' to Port D
1	0	send '2' to Port D
1	1	send '3' to Port D

Solution:

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

int main(void)
{
  unsigned char z;
  DDRB = 0;                    //make Port B an input
  DDRD = 0xFF;                 //make Port D an output
  while(1)                     //repeat forever
  {
    z = PINB;                  //read PORTB
    z = z & 0b00000011;        //mask the unused bits
    switch(z)                  //make decision
    {
      case(0):
      {
        PORTD = '0';           //issue ASCII 0
        break;
      }
      case(1):
      {
        PORTD = '1';           //issue ASCII 1
        break;
      }
      case(2):
      {
        PORTD = '2';           //issue ASCII 2
        break;
      }
      case(3):
      {
        PORTD = '3';           //issue ASCII 3
        break;
      }
    }
  }
  return 0;
}
```

Write an AVR C program to monitor bit 7 of Port B. If it is 1, make bit 4 of Port B input; otherwise, change pin 4 of Port B to output.

Solution:

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

int main(void)
{
  DDRB = DDRB & 0b01111111;                  //bit 7 of Port B is input

  while (1)
  {
    if(PINB & 10000000)
      DDRB = DDRB & 0b11101111;              //bit 4 of Port B is input
    else
      DDRB = DDRB | 0b00010000;              //bit 4 of Port B is output
  }

  return 0;
}
```

Write an AVR C program to get the status of bit 5 of Port B and send it to bit 7 of port C continuously.

Solution:

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

int main(void)
{
  DDRB = DDRB & 0b11011111;          //bit 5 of Port B is input
  DDRC = DDRC | 0b10000000;          //bit 7 of Port C is output

  while (1)
  {
    if(PINB & 0b00100000 )
      PORTC = PORTC | 0b10000000;    //set bit 7 of Port C to 1
    else
      PORTC = PORTC & 0b01111111;    //clear bit 7 of Port C to 0
  }
  return 0;
}
```

Write an AVR C program to toggle all the pins of Port B continuously.
(a) Use the inverting operator. (b) Use the EX-OR operator.

Solution:

(a)

```
#include <avr/io.h>                  //standard AVR header
int main(void)
{
      DDRB = 0xFF;                   //Port B is output
      PORTB = 0xAA;
      while (1)
            PORTB = ~ PORTB;         //toggle PORTB
      return 0;
}
```

(b)

```
#include <avr/io.h>                  //standard AVR header
int main(void)
{
      DDRB = 0xFF;                   //Port B is output
      PORTB = 0xAA;
      while (1)
            PORTB = PORTB ^ 0xFF;
      return 0;
}
```

Write an AVR C program to monitor bit 7 of Port B. If it is 1, make bit 4 of Port B input; else, change pin 4 of Port B to output.

Solution:

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

int main(void)
{
  DDRB = DDRB & ~(1<<7);                     //bit 7 of Port B is input

  while (1)
  {
    if(PINB & (1<<7))
      DDRB = DDRB & ~(1<<4);                 //bit 4 of Port B is input
    else
      DDRB = DDRB | (1<<4);                  //bit 4 of Port B is output
  }

  return 0;
}
```