DATA CONVERSION PROGRAMS

In microcontrollers, certain information is stored in the form of packed BCD. A packed BCD contains two decimal digits in a byte. Eg: 34, 76 etc. An unpacked BCD contains one decimal digit in a byte. Eg: 05, 07 etc. To display these information, we have to convert it into ASCII. ASCII codes of the digits 0 to 9 are 30 to 39 respectively. It means that to display the digit 7, we have to first convert it into its ASCII code 37 (ie. 30 is added). Similarly to get the numerical value of ASCII code, we have to subtract 30 from it.

Packed BCD to ASCII conversion:

To convert packed BCD to ASCII, follow the steps:

1. Convert it into unpacked BCD
2. Add 30 to each digit.

Example: Packed BCD 0x29 (00101001)
Unpacked BCD 0x02 and 0x09 (00000010 and 00001001)
Adding 0x30 0x32 and 0x39 (00110010 and 00111001)

ASCII to packed BCD conversion:

To convert ASCII to packed BCD, follow the steps:

1. Convert it to unpacked BCD
2. Combine the digits

Example: Let the digits are '4' and '7' which are represented as 0x34 and 0x37. We have to combine it to a single byte as 47.

Converting 0x34 to unpacked BCD gives 0x04
Converting 0x37 to unpacked BCD gives 0x07
Combining it to packed BCD gives 0x47

Programming Example:

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

Solution:

```
#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;
}
```

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

Solution:

```
#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;
}
```

Binary (Hex) to Decimal conversion:

Hexadecimal format is a convenient method to represent binary data. The binary data 00–FFH converted to decimal will give us 000 to 255. One method of converting it to decimal is dividing it by 10 and keeping the remainder.

Example: Covert 0xFD to decimal. Its decimal value is 253.

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

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

Solution:

```
#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 SERIALIZATION

Sometimes we need to send a byte of data one bit at a time through a single pin of a microcontroller. There are two ways to send data serially:

1. Using serial port
2. Data serialization.

Using serial port method, the programmer has very limited control over the sequence of data transfer. In data serialization, the data bits can be send under the program control. So the programmer can determine the timing and bit sequence of the data.

Programming Examples:

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.

Solution:

```
#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;
}
```

If we want to send the MSB first, then the following changes can be made in the for loop:

```
  for(x=0;x<8;x++)
    {
      if(regALSB & 0x80)
          PORTC |= (1<<serPin);
     else
          PORTC &= ~(1<<serPin);
      regALSB = regALSB << 1;
    }
```

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.

Solution:

```
//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;
}
```

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.

Solution:

```
#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;
}
```

MEMORY ALLOCATION

We can store predefined fixed data in any of the following three spaces in AVR microcontroller:

- SRAM
- EEPROM
- Flash (code ROM)

The size of these spaces depends on the microcontroller. To allocate space for data in EEPROM and Flash, different C compilers use different directives or built-in functions. Following is one of the method to allocate space in EEPROM and Flash.

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