MULTIPLEXING 7 SEGMENT DISPLAY USING PIC MICROCONTROLLER

Let us try to understand about multiplexing of seven segment display using PIC microcontroller to implement decimal counter which will increment 0000 to 9999.

Seven Segment Displays are arrays of seven “Light Emitting Diode” (LED) segments with additional decimal point (dp) which is also a LED. Each these segments are marked as a,b,c,d,e,f,g and dp.

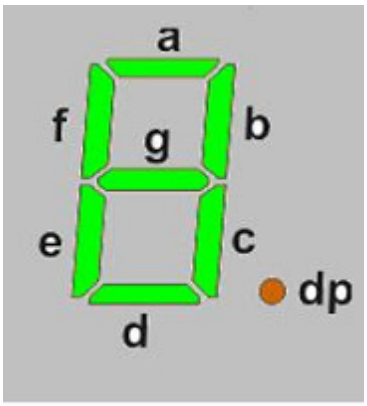

Fig 1

When we use these segments for practical applications current and voltage tolerance values should be known to us. Over voltage or current may damage LED segment. It is always better to use current limiting resistors in series with each LED segment to avoid such damages. The segment is light up only when both a LED segment and its associated common lead (either anode or cathode) are selected.

Multiplexing is necessary to interface two or more seven segment displays to a microcontroller. Software program can control these multiplexed seven segments to ON/OFF in a cyclical fashion. This also helps to reduce power in battery operated systems.

For each microcontroller pin, there is a maximum current limitation it can source or sink. For this reason low current LEDs are using if several displays are connected to the microcontroller. For PIC microcontroller direct drive of the LED is possible because of high sink source capability.

Two types of seven segment displays are in use.

Common Anode Type Seven Segment Display

b) Common Anode (Power): Anodes of all LED segments are connected to common pin , by applying low voltage (i.e. in general Ground voltage of 0V) to the segment input, ON each segment.

Common Cathode Type Seven Segment Display

a) Common Cathode (i.e. Ground): Cathodes of all LED segments are connected to common pin and to turn ON the segments you need to apply +ve voltage of either 5V or any other value of voltage that is supported by display.

How seven segments display numbers?

The numbers 0, 2, 3, 4,5,6,7, 8, 9 to displayed. For 7 to be display in segment means segments a,b,c should be enabled right ?????? All other should be off. Because it is Common ANODE and input side Cathode.

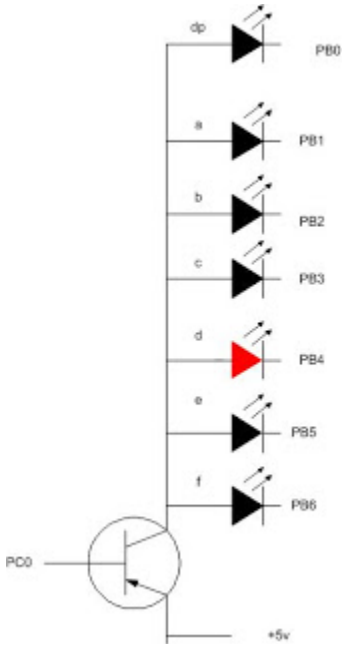

Fig 2

To enable led you need logic 0 at (PB0-PB7) and Logic 0 at PC0, PNP transistor

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
g	f	e	d	c	b	a	DP

1 1 1 1 0 0 0 1

Table 1

Common cathode, logic 0, enables the led

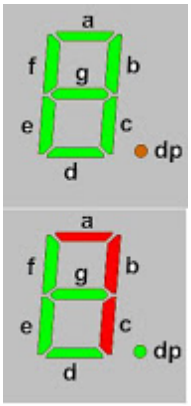

Fig 3

0000 - 9999 Counter with Common Anode 7 Segment Display Multiplexed with PNP Transistor.

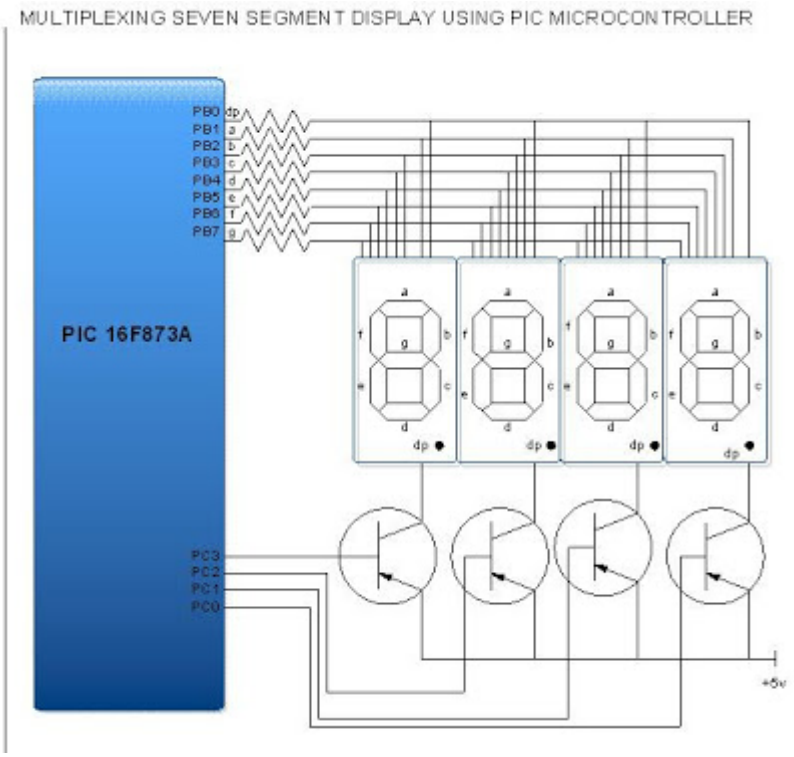

Fig 4

Here in our circuit we used common anode type with all plusses shared common pin. The cathodes (segments a - g) of each display are all connected together and then connected to a PIC I/O line (PORTB) through a series resistor. Each common anode is connected to transistor and it is controllable by a PIC I/O line (PORTC).

Explanations:

PORTB of microcontroller connected to segments ‘a’ to ‘g’ and digits are connected to PORTC. In order to ON each segment, PORTB pin must be set to logical '0’ and to Off, its pin must be set to logical ‘1’ because we have chosen common anode type seven segment display, its input end is cathode. Transistor base is connected to PORTC. If transistor is in conducting mode, segment will ON state otherwise OFF.
In order to display numbers, byte representing number must be sent to the PORTB and transistors must be activated by enabling corresponding PORTC pin by applying low level at the base of transistor. At any particular time, only one of the LED segments is turned on. Immediately after the segment is lit, the segment is turned off and the next segment in the implementation is turned on. Thus each digit is On-Off by controlling the anode for that display. If we do each one fast enough, we will see the entire display flicker free.

Programming:

Timer 0 used to refresh the display for every 1ms delay. Time0 Clocked from the system clock divided by 256 is used to generate delay. The timer is preloaded with the number xxx; it will generate interrupt for every 1ms. The digits to be displayed store it in array variable. For every 1 ms interrupt will be generate and in the Interrupt subroutine, array variables load to the PORTB and corresponding value to PORTC to select each digit. Counter will be incremented for every 1ms.

In order to display digits, corresponding values must be calculated. To display digit 7, 1111 0001 in binary (0xf1 in hexadecimal). Here logic ‘0’ enables segment, by setting PORTC bits ‘0’, which set PNP transistor in conducting state.

Number	Segments	Hex
0	1001 0000	0x81
1	1111 0011	0xf3
2	0100 1001	0x49
3	0110 0001	0x61
4	0011 0011	0x33
5	0010 0101	0x25
6	0000 0101	0x05

7	1111 0001	0xf1
8	0000 0001	0x01
9	0010 0001	0x21

Table 2

C Program code for Multiplexing seven segment display using PIC microcontroller

```
#include
const unsigned char DispArry[]={0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,0x01,0x21};
 unsigned long DigitCount,DispValue;
 unsigned char delay,DispIncIndx,CountBuff[6];
 bit flag;

void ClrMem(unsigned char *,unsigned char *);
void SetDisp(void);

void interrupt isr(void)
{
        if(TMR0IF && TMR0IE)
        {
                TMR0IF=0;
                TMR0=100;
                if(!(DispIncIndx))
                {
                        PORTC=0X7F;
                }
                else
                {
                        PORTB=DispArry[CountBuff[DispIncIndx]];

                        switch(DispIncIndx)
                        {
                                case 1:
                                        PORTC=0X7E;
                                        break;
                                 case 2:
                                        PORTC=0X7d;
                                        break;
                                case 3:
                                        PORTC=0X7b;
                                        break;
                                case 4:
                                        PORTC=0X77;
                                        break;
                                default:
                                        PORTC=0x7f;
                                        break;
                         }
                }
                DispIncIndx++;
                if((DispIncIndx)>7)
                {
```

```
                        DispIncIndx=0;
                        flag=0;
                }

        }
}

void main(void)
{
        unsigned int j;
        ClrMem(CountBuff,CountBuff+5);

        TRISB=0X00;
        TRISC=0X00;
        TMR0=100;               // 4ms
        GIE=1;
        PEIE=1;
        TMR0IE=1;
        TMR0IF=0;
        PORTB=0XFF;
        PORTC=0X7F;
        OPTION=0X84;
        DispIncIndx=0;
        DispIncIndx=0;
        DigitCount=0;
        flag=0;

        for(;;)
        {

                        asm("nop");
                        if(!flag)
                        {
                                flag=1;
                                DigitCount++;
                                if(DigitCount>9999)
                                        DigitCount=0;

                           SetDisp();

                        }

        }
}

void ClrMem(unsigned char *start,unsigned char *end)
{
 while(*start<*end)
 {
                *start=0x00;
                start++;
 }
}

void SetDisp(void)
```

```
{
unsigned long value,k=10;
unsigned char indx=0;
        value=DigitCount;
        while(indx<6)
        {
                CountBuff[indx]=value%10;
                value/=10;
                indx++;
        }
}
```

Source : http://asic-soc.blogspot.in/2010/05/multiplexing-7-segment-display-using.html#more