

Basic Microprocessor's Operations: MCU for Dummies'

1. Objective of Experiment

This experiment aims to help understand how a microprocessor works, by implementing simple applications using the PIC16F84, a single chip microprocessor in the PIC family of controllers, produced by Microchip Technology.

2. Example Applications

2.1. LED Control Program

Using the PIC16F84 microcontroller, a simple control program to enable or disable the LEDs (light-emitting diodes) is written and the corresponding circuit is constructed. Throughout this exercise with regard to software and hardware building, the development environment of the PICs, the way to use the assembler and the ROM writer, and the basic circuit construction of the PIC16F84 processor can be understood.

2.2. Digital Dice

A device for showing numerical information on the 7-segment LED display is developed using the PIC16F84. Here, a switch is used as the device's input component and a group of seven LEDs as the output component. Although the ultimate device to develop is a digital dice, a device simply displaying a given number on the seven LEDs when the switch is 'closed' is implemented first, so that one can quickly learn how to control the 7-segment LED display. After this, a digital dice that displays in random the number from 1 to 6 on the 7-segment LEDs is built.

3. Experiment Equipments and Parts

3.1. LED Control Program

PC, Breadboard, Power Supply, ROM Writer, PIC16F84, Crystal Oscillator (4MHz), Switch, LEDs, Resistors (330 Ω , 10K Ω)

3.2. Digital Dice

PC, Breadboard, Power Supply, ROM Writer, PIC16F84, Crystal Oscillator (4MHz), Switch, 7-segment LED, Resistors (330 Ω , 2.2K Ω , 10K Ω)

4. Theoretical Background

4.1. LED Control Program

4.1.1 Pull-up Resistor

In this experiment, a pull-up resistor is used in the switch input circuit. Three types of the switch input circuit are presented from Figure 4.1.1 to Figure 4.1.3, for the purpose of showing why a pull-up resistor is necessary. First of all, suppose an input circuit as shown in Figure 4.1.1. This is an incomplete circuit because the state ('high' or 'low') of the device is not clearly determined when the switch is 'open', while its state is confirmed 'low' when the switch is 'closed'. To solve this problem, the circuit in Figure 4.1.2 can be considered. Now, the state is clearly 'high' when the switch is 'open', but shorts arise when the switch is 'closed' as VCC (power) is directly connected to GND (ground). Eventually, Figure 4.1.3 presents the third type of switch input circuit that uses a pull-up resistor in order to prevent shorts and to ensure that inputs to logic systems settle at expected logic levels if devices are disconnected. In this circuit, the state is 'low' when the switch is 'closed' (ON), and 'high' with the switch 'open' (OFF).

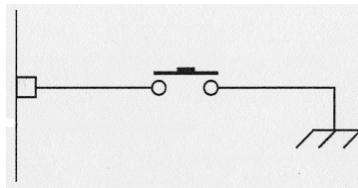


Figure 4.1.1 Switch Input Circuit (Type 1)

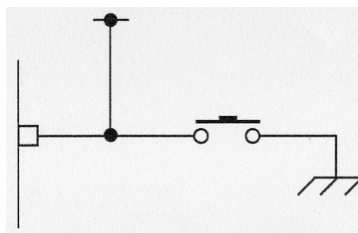


Figure 4.1.2 Switch Input Circuit (Type 2)

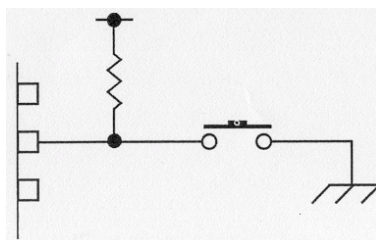


Figure 4.1.3 Switch Input Circuit (Type 3)

4.1.2 Crystal Oscillator

A crystal oscillator, often abbreviated OSC and also known as x-tal, is an electronic circuit that creates an electrical signal with a very precise frequency. This frequency is commonly used to keep track of time, to provide a stable clock signal for digital integrated circuits, and to stabilize frequencies for radio transmitters and receivers. VCC (power) and GND (ground) pins provide power for the oscillator circuitry, generating automatically clocks. A circuit symbol of the crystal oscillator is shown in Figure 4.1.4.

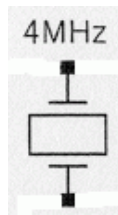


Figure 4.1.4 Schematic Symbol of Crystal Oscillator

4.1.3 LED Output Circuit

LEDs emit light when an electric current passes through them. Figure 4.1.6 illustrates an output circuit of the LED. As LED's anode (labeled a or +) is connected toward the port, electricity flow through the resistor and the light is on when the port releases 'high' current – i.e., a positive voltage is applied to the anode. The resistor between the LED and the port is to limit the current to a safe value. Without this, the LED could be destroyed because too much current will pass through and burn it out. To recap, the port releases 'high' current to get an LED to light up; and the LED does not glow when the port sends 'low' electrical flow.

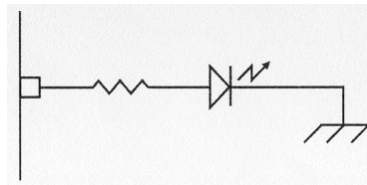


Figure 4.1.6 LED Output Circuit

4.1.4 PIC16F84's Basic Wiring

The pins OSC1, OSC2, VDD, VSS, and MCLR should be wired to make a PIC16F84 operate. Figure 4.1.7 illustrates a typical "basic" wiring of the PIC16F84 processor. The OSC1 and OSC2 pins are used to connect to the oscillation part of the circuit, i.e., a crystal in this case. VSS and VDD are the power supply pins - VDD is the positive supply, and VSS is the negative supply, or 0V. Ground is taken to the VSS pin. When the device has power and is running a program, one can easily reset the device by shorting MCLR to VSS (ground).

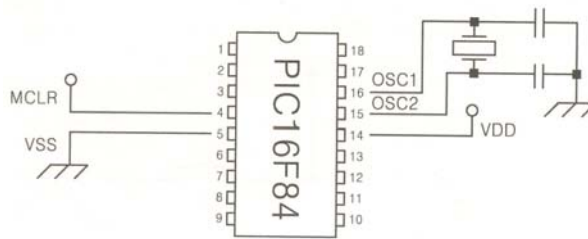


Figure 4.1.7 PIC16F84's "Basic" Wiring Diagram

4.1.5 MRPIC-IDE: Integrated Development Environment

MRPIC-IDE, an Integrated Development Environment for programming and emulating Microchip PIC microcontroller, is used to program the PIC. This development system includes a Text Editor, Assembler, Debugger, in-circuit emulator, etc. Figure 4.1.8 shows the initial screen displayed after starting MRPIC-IDE. To write a source program, a project should be created. To create a project, select “New Project” from <File> menu on MRPIC-IDE Main Window. Consequently, Project Window will pop up (see Figure 4.1.9). Project includes information such as device, type of compiler, and frequency of oscillation. In order to generate Project file, fill the components with related value in Project Window and then click “OK”. The content of Project file can be edited later by “Edit Project”, as shown in Figure 4.1.10.

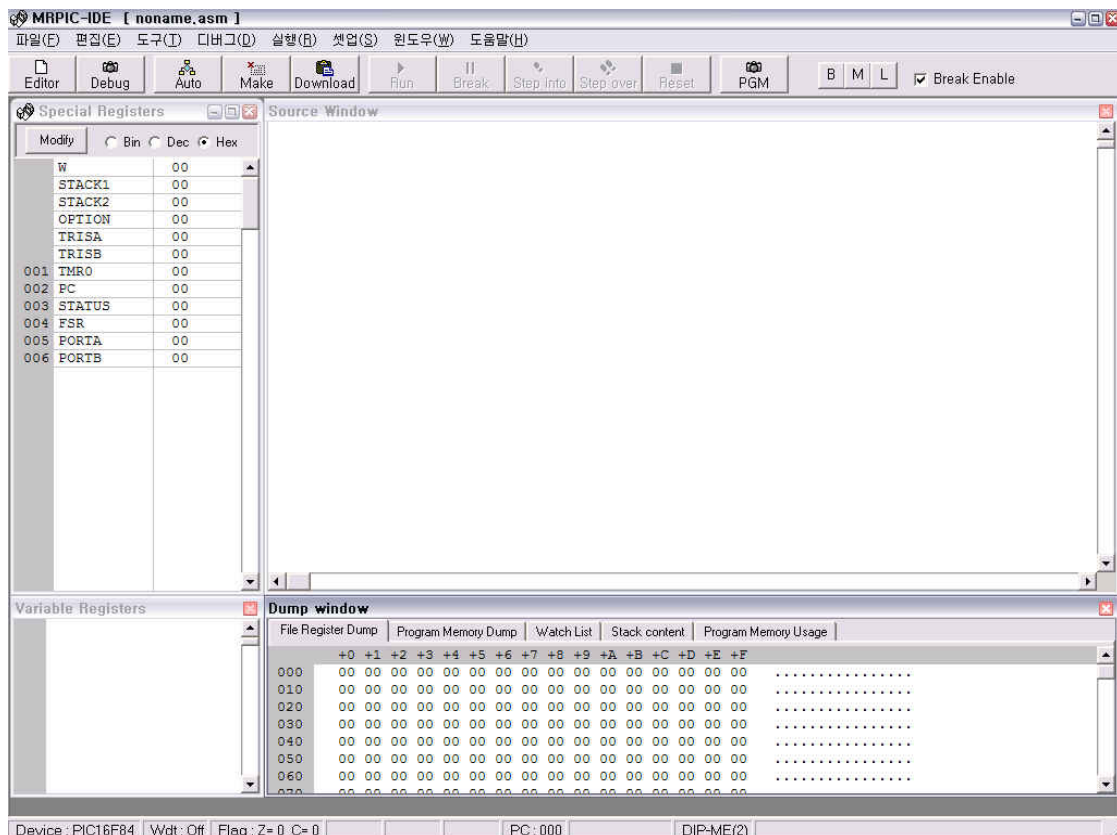


Figure 4.1.8 MRPIC-IDE's Initial Screen

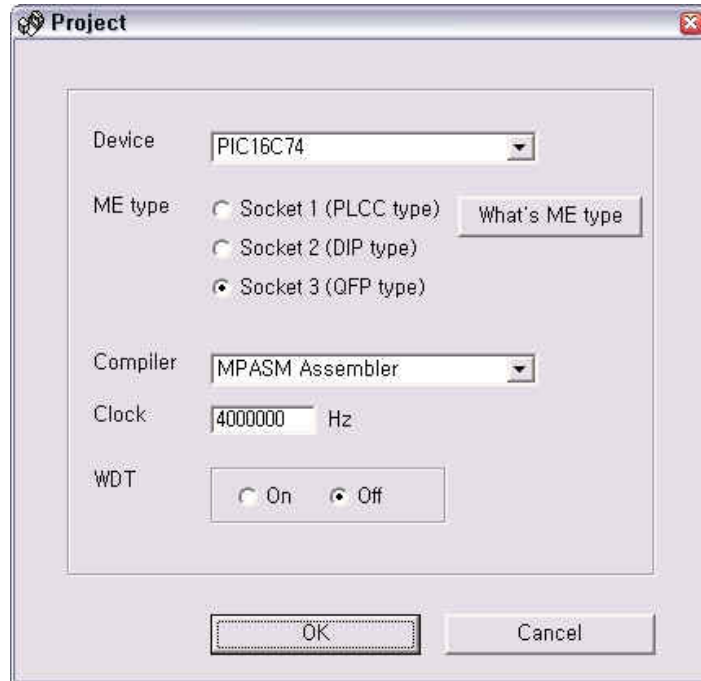


Figure 4.1.9 Making Project

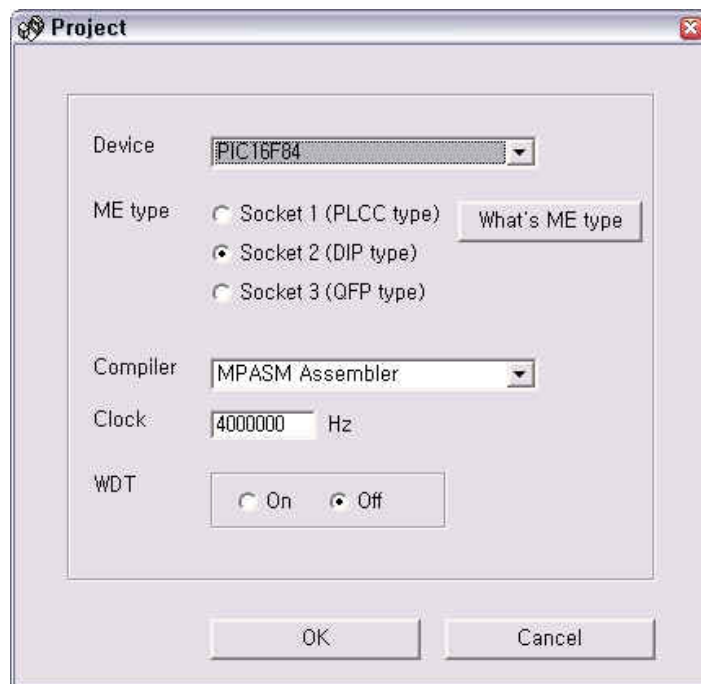


Figure 4.1.10 Edit Project

As appearing in Figure 4.1.11, an empty editor turns up when Project file is saved by clicking “OK” in Project Window. A source program having “asm” as its file extension can be written and saved in this editor. Or, if the source program is written using other text editors such as Microsoft’s Note, this text file

can also be open and edited using “Open” in <File> menu, by changing its extension to “asm”.

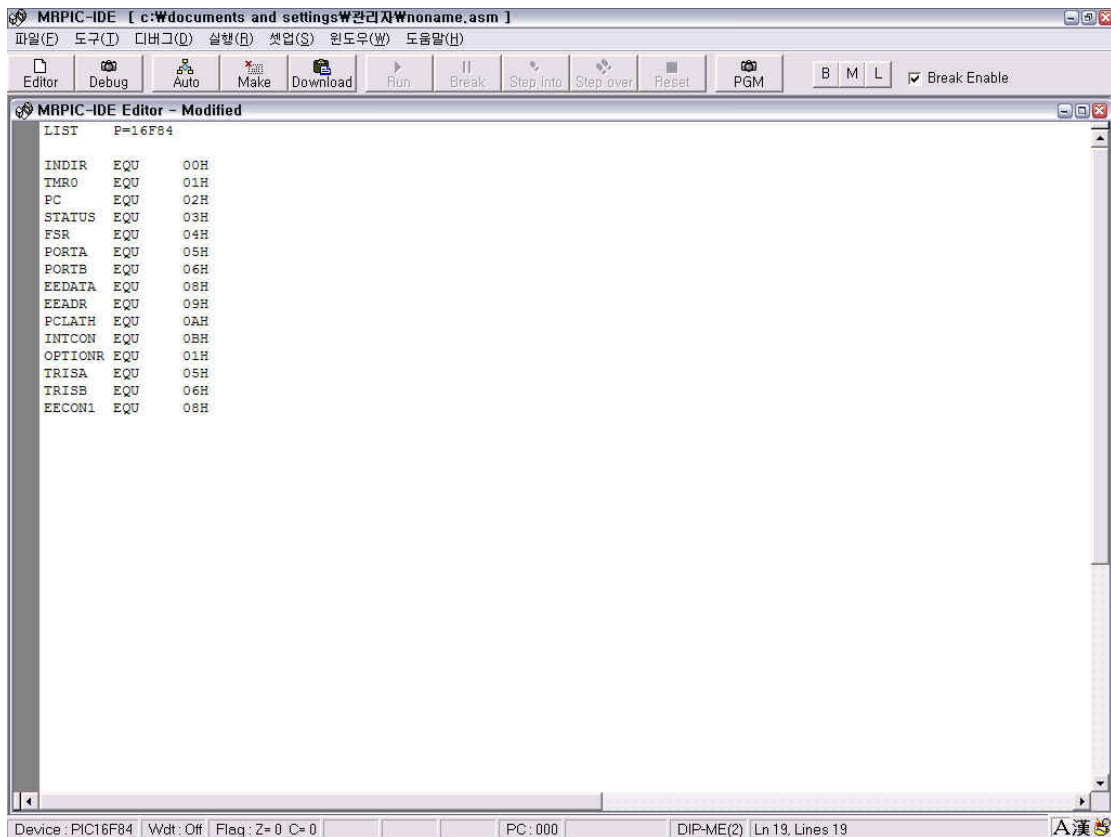


Figure 4.1.11 Editor

After generating source code in Editor and saving it, the next stage is Assemble that translates a source file into an executable object file. MR.PIC-IDE has a built-in PIC-dedicated Cross Assembler, MPASM. To start Assemble, click “MAKE” button in the menu bar. If Assemble is completed without error, a green status bar will briefly show up on the screen. If the red status bar appears, there is some error in the process of Assemble and debugging is needed.

Once the program has been assembled, the next work to carry out is ROM writing. Carefully matching the corresponding pins, place the PIC16F84 into the ROM writer socket, and click <PGM> button in MRPIC-IDE program. The program for ROM writing is then executed and the PIC16F84 will be programmed, as shown in Figure 4.1.12. Make sure that the chip’s name is matched.

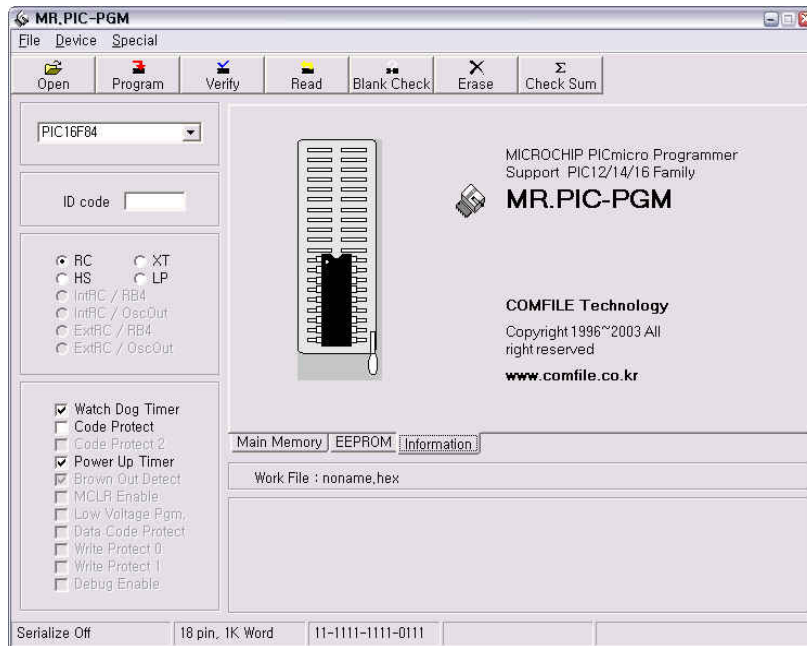


Figure 4.1.12 MR.PIC-PGM

If <PGM> is clicked after compile being successfully completed, memory information regarding the performed compilation is automatically loaded, as presented in Figure 4.1.13. This information is also displayed by selecting the “Main Memory” tab located in the middle of MRPIC-IDE Program Window.

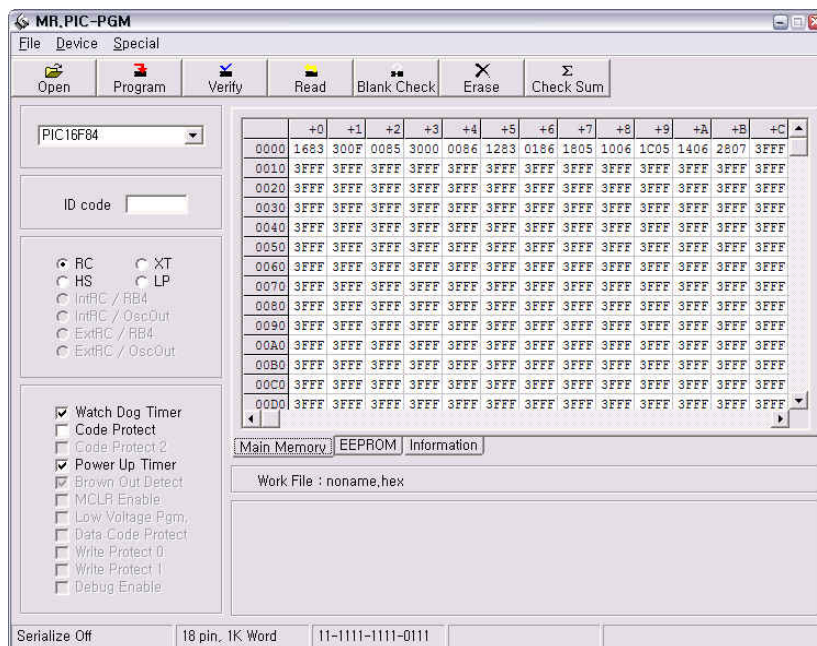


Figure 4.1.13 Main Memory Information

In this stage, options are selected. The selectable options are listed in the left area of the window. The chosen options work like an one-time programmable fuse, and so they fix chip's characteristics in the Writing process. Thus, it is very important to be careful in choosing options. In this experiment, options are chosen according to the information in the window appearing in Figure 4.1.14. Here, the selected option XT means adopting the crystal oscillator for clocking, and a watchdog timer is not used (unselected).

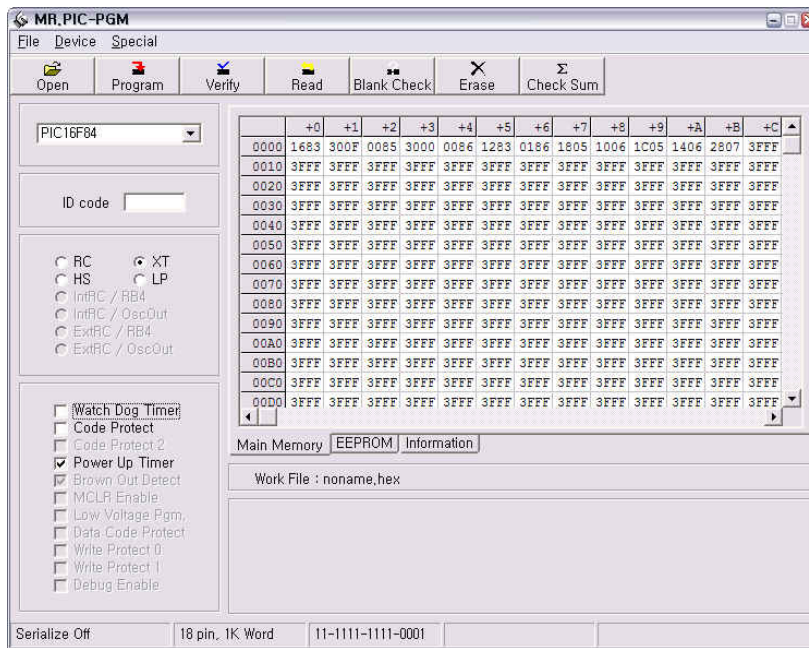


Figure 4.1.14 Option Selection

Finally, the last thing to do is Writing. In order to execute Writing, click “Program” button in MRPIC-IDE Main Window’s tool bar. Make sure that the light of the ROM writer’s LEDs is coming on during the writing.

4.2. Digital Dice

4.2.1 7-segment LED's Power Supply

There are two types of 7-segment LED displays, the common anode type and the common cathode one. In the common anode type, VCC (diode's anode pin) is shared. This shared pin is set to 'high', whereas the other pins from 'a' to 'g' retain active 'low' – i.e., ON for 0, OFF for 1. Conversely, Ground (diode's cathode pin) is shared in the common cathode array. Thus, the shared GND pin is set to 'low', while the other pins from 'a' to 'g' are active 'high'. Figure 4.2.1 is an example of the common anode 7-segment LED display. Here, a NPN transistor is used for the 'power control input' that is a device to control the

power for the 7-segment LEDs. For Over-Current Protection, a resistor of 330 ohm is connected to the collector. Besides, a resistor of 2.2K ohm is connected to the base for high input impedance. Electrical supply is controlled by sending varying levels of current from the base. That is, if the 'power control input' device is 'high', the current flows through. Otherwise, the current interrupted. In this way, the amount of current flowing through the gate from the collector may be regulated and unnecessary energy release in the 7-segment LEDs can be prevented.

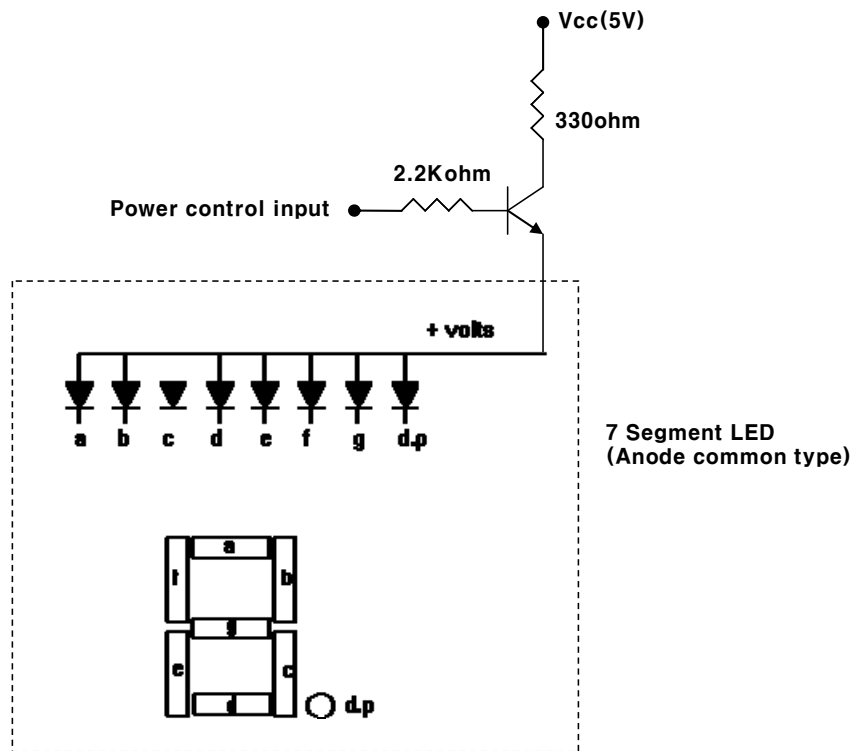


Figure 4.2.1 Seven Segment LED's Power Supply

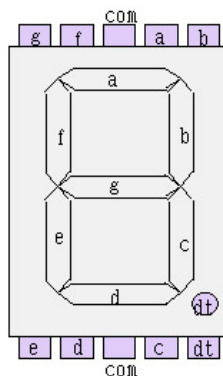


Figure 4.2.2 Seven Segment LED's Pins Arrangement

4.2.2 7-segment LED's Pins Identification

The 7-segment LED display consists of seven LEDs (precisely, eight LEDs including the dot). Each of the LEDs is given a name, the character between 'a' and 'g'. However, it might not be easy to identify which LED pin is which because the names of the pins are not explicitly labeled. A way to easily identify the pins is employing digital multimeters (often abbreviated DMM). In a Digital Multimeter, the signal under test is converted to a voltage. When a DMM is placed within resistor measurement range, its black lead releases about +3V voltage, which is enough power to light up a LED. Thus, the black lead is considered as positive (+) voltage, and the red lead as negative (-). Fix the black lead to one of the 7-segment LED pins, and try connecting the red lead to the other pins. In this way, one can figure out whether it is a common anode type or a common cathode type, in the first place. If it is a common anode array, fix the black lead to the identified common anode pin, and identify the other pins ('a' to 'g') by trying the rest of the pins one by one with the red lead. Figure 4.2.2 shows a general pins arrangement of the 7-segment LED display. As the arrangement can differ according to the devices, it is important to verify which pin is which using the aforementioned identification method.

4.2.3 PIC16F84 OPTION Register

In order to implement a digital dice using the 7-segment LEDs, it is important to understand PIC16F84's OPTION register. The OPTION register, as the name suggests, offers a series of options – various control bits in the OPTION register are used to configure the TMR0/WDT prescaler, the external INT interrupt, TMR0, and the weak pull-ups on PORTB. In other words, a certain function in the PIC16F84 can be ON or OFF depending on the state of the bits in the OPTION register. It is a readable and writable register freely changeable during programming, and so it differs from fuse setting that is fixed during Writing.

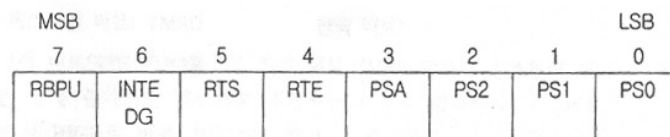


Figure 4.2.3 OPTION Register

4.2.4 OPTION Register's Bits Information

- **TMR0**

PIC16F84 has an 8-bit timer (called TMR0), which can count up to 256 (from 0 to 255). As a matter of fact, TMR0 can be used as a timer or a counter. In implementing a digital dice with the 7-segment LEDs, TMR0 is exploited to generate a random number between 1 and 6. That is, while continuing to increase TMR0 value in a regular interval, a digital dice device captures the time when the switch is 'closed', divides the captured time by 6, and then displays the remainder of the division on the 7-segment LED display.

- **RTS (Bit 5; TMR0 Clock Source Select bit)**

Bit 5 RTS decides if TMR0 register is used as a timer or as a counter of pulses. That is, this pin enables a free-run timer to increment its value either from an internal oscillator (when the bit is 0), or via external impulses (when the bit is 1). The first is a timer and the latter is a counter.

1 = external impulses

0 = internal clock

- **RTE (Bit 4; TMR0 Source Edge Select bit)**

As described above, beside the internal oscillator clock, timer status can also be increased by the external clock on RA4/TOCKI pin. If this option of external clock was selected (i.e., RTS bit is 1), it would be possible to define the edge of a signal (rising or falling), on which timer would increase its value. If RTE bit is 0, it is a rising edge – i.e., TMR0 value is increased when a signal is changed from ‘low’ to ‘high’. On the other hand, RTE bit with its value 1 indicates a falling edge, TMR0 being increased when a signal is changed from ‘high’ to ‘low’. Note again that RTE bit does not exert influence if TMR0 was enabled with an internal oscillator clock (i.e., RTS bit is 0).

1 = falling edge

0 = rising edge

- **PS2, PS1, PS0 (Bit 2, 1, 0; Prescaler Rate Select bit)**

The low three bits, PS0, PS1, and PS2, set the prescaler ratio - the time period between incrementing TMR or WDT. If the prescaler ratio is 1:2, TMR0 (or WDT) value is increased by one when there are two input signals. If the prescaler ratio is 1:4, four input signals are required to increase TMR0 (or WDT) by one. Figure 4.2.4 and 4.2.5 illustrate a simplified scheme of relation between a timer and a prescaler.

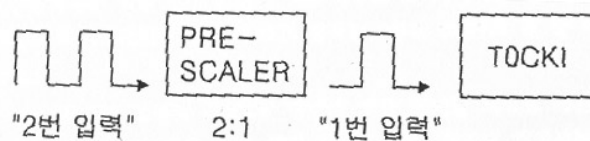


Figure 4.2.4 Prescaler Ratio 1:2 (Timer : Prescaler)

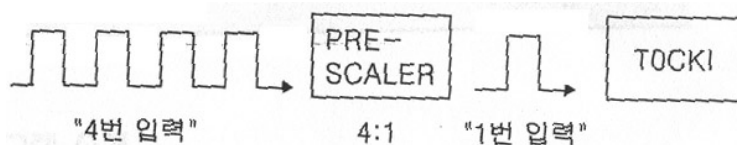


Figure 4.2.5 Prescaler Ratio 1:4 (Timer : Prescaler)

4.2.5 Summary

When the OPTION register is initialized with 0, its state would be:

- TMR0's source is an internal clock and thus it is used as a Timer.
- TMR0's edge select bit is 0 (rising edge), although it has no meaning when TMR0 is a Timer.
- Default prescaler ratio is 1:2.

That is, TMR0 is used as a timer, and the external impulse device becomes obsolete in this case (it might be better to connect it to VSS or VDD). Physically, TMR0 timer is a register whose value is continually increasing to the maximum value of 8-bits, 255 (0FFH), and then it starts all over again from zero. The internal timer TMR0 is constantly incrementing as long as the chip is working. Applications of this internal timer are diverse. It is widely used in digital clocks and electronic meters to measure signal intervals. Besides, it can be used as a random number generator, which is employed in digital dice implementation.

5. Experiment Method and Procedures

5.1. LED Control Program

Exercise 1 (LED Control Program): Using a breadboard, the circuit for the LED control program is configured. The overall circuit diagram is illustrated in Figure 5.1.1. An Assembly code that operates the PIC16F84 to enable or disable the LEDs is programmed on the PC. The code is shown in Table ???. Note that this code is also presented in Preliminary Study 5.

```
LIST    P=16F84

;      FILE DEFINITION

INDIR   EQU    00H           ; PAGE 0
TMR0    EQU    01H
PC       EQU    02H
STATUS  EQU    03H
FSR     EQU    04H
PORTA   EQU    05H
PORTB   EQU    06H
EEDATA  EQU    08H
EEADR   EQU    09H
PCLATH  EQU    0AH
INTCON  EQU    0BH

OPTIONR EQU    01H           ; PAGE 1
TRISA   EQU    05H
TRISB   EQU    06H
EECON1  EQU    08H

;      BIT DEFINITION

CF       EQU    .0           ; STATUS
DC       EQU    .1
ZF       EQU    .2
PD       EQU    .3
TO       EQU    .4
RP0      EQU    .5

;      MAIN ROUTINE

ORG     0

BSF     STATUS,RP0
MOVLW  B'00001111'
MOVWF  TRISA
MOVLW  B'00000000'
MOVWF  TRISB
BCF     STATUS,RP0
CLRF   PORTB

LOOP    BTFSC  PORTA,0
        BCF   PORTB,0
        BTFSS PORTA,0
```

```

BSF    PORTB, 0

GOTO   LOOP

END

```

Table1. Assembly Code for LED Control Program

The Assembly code is written to the PIC16F84 using the ROM writer linked to the PC. The PIC16F84 processor is then placed into the breadboard. Make sure that the LEDs glow when the switch is ‘closed’. If not, examine that the fault is in the program code or in the circuit. After debugging, check again if the LEDs are turned on.

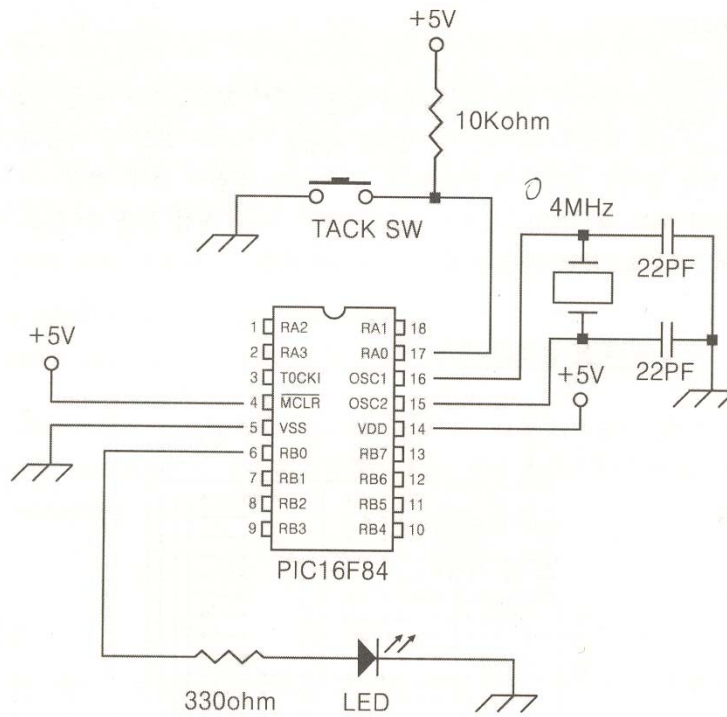


Figure 5.1.1 LED Control Program Circuit Diagram


```

;INITIAL SETTING

ORG    0
BSF    STATUS,RP0    ;BANK1
MOVLW  B'11111111'
MOVWF  TRISA         ;A is input port
MOVLW  B'00000000'
MOVWF  TRISB         ;B is output port
BCF    STATUS,RP0    ;BANK0

;MAIN LOOP

;=====Initial number
MAIN   MOVLW  B'00111111'
;=====Set to '0'
MOVWF  PORTB
BTFSC  PORTA,2       ;Check switch.
GOTO   MAIN         ;Pushed '1' WAIT until push
CALL   DELAY        ;Not pushed '0'
GOTO   SHOWTME

;=====Display number when push the switch
SHOWTME MOVLW  B'00000110' ;Setting displayed number to '1'
MOVWF  PORTB        ; Displayed number
CALL   DELAY
BTFSS  PORTA,2       ;Check the switch if it is still pushed
GOTO   SHOWTME      ;Pushed'0', then stay SHOWTIME!
GOTO   MAIN         ;Not pushed '1'

DELAY  MOVLW  .250
MOVWF  DELAY_COUNT1
DELAY_1 CLRF  DELAY_COUNT2
DELAY_2 DECFSZ DELAY_COUNT2
        GOTO  DELAY_2
        DECFSZ DELAY_COUNT1
        GOTO  DELAY_1
        RETURN

END

```

Table2. Assembly Code for Displaying Given Number

Using the development tool MRPIC-IDE and the ROM writer, compile the programmed code and write it to the PIC16F84. The circuit should be constructed as presented in Figure 5.2.1.

Caution: Remember to connect the LED the correct way round and never connect the LED directly to a battery or power supply. Otherwise, the chip can be burnt out.

Additional Information: In the circuit depicted in Figure 5.2.1, the common cathode 7-segment LED array is employed. That is, the common pin is connected to ground. The 'power control input' described in Figure 4.2.1 is not used here. Those who are interested try to work on the circuit presented in Figure.

4.2.1 as well, in order to learn the way to control power supply for the 7-segment LEDs.

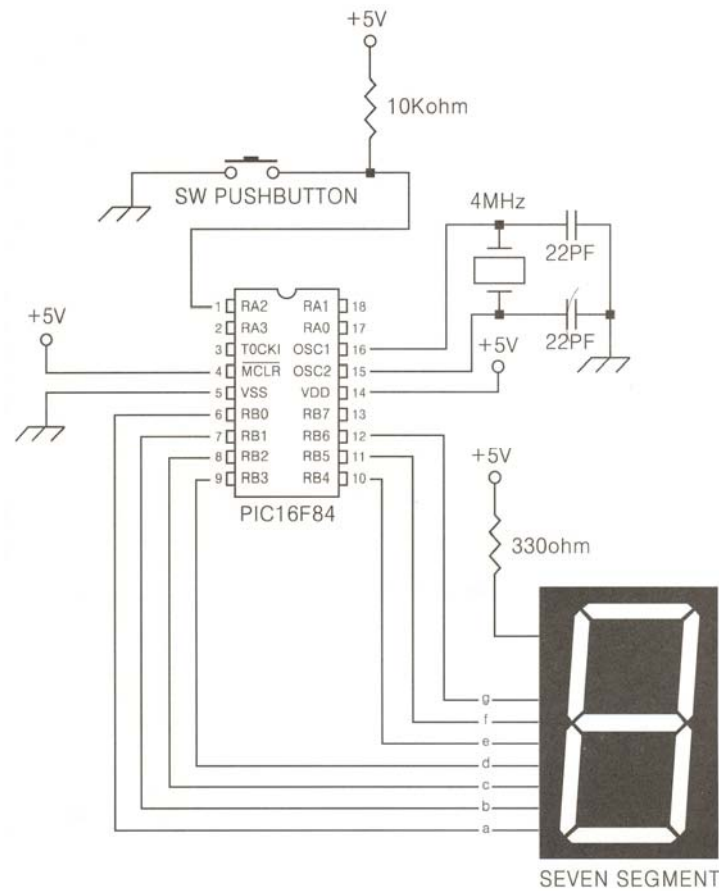


Figure 5.2.1: Circuit Diagram for Exercise 2 (Displaying Number) and Exercise 3 (Digital Dice)

Push down the switch and verify the expected number is coming up. If it does not work as expected, find out the flaw is in the program or in the circuit. After modifying the program code or the circuit, according to the cause of the flaw, verify again the device's operation. If it works fine, try to change the number displayed on the LEDs with other numbers and repeat the process.


```

LOOP
    GOTO LOOP
    MOVF TMR0,W
    GOTO LOOP

SEGMENT_TABLE
    ANDLW 07H
    ADDWF PC
    RETLW B'00000110'
    RETLW
    RETLW
    RETLW
    RETLW
    RETLW
    RETLW
    RETLW
    RETLW
    END

```

;If PORT A 2 bit is '1', idling,
; or '0', execute below commend
;Obtain TMR0 value to Wreg
;Jump to SEGMENT_TABLE
;Transmit Wreg to PORTB
;Obtain 3 low bit of Wreg and save
;Add Wreg to PC(program counter)
;Exactly, jump to below as Wreg
; and excute.
;For example, Wreg is '2', jump
; two line and execute next code.
;As a result, jump to one of below
; 8 code
; W dice
; 0 1
; 1 2
; 2 3
; 3 4
; 4 5
; 5 6
; 6 your character1
; 7 your character2
;Using 'RETLW' commend 8 times,
; this code save suitable value
; for display to Wreg, and return.
; previous position.

Table3. Code for Digital Dice (Incomplete)

<Reference>	[RETLW]	Loads literal data into Wreg and returns.
Instruction Format:	[label] RETLW k	
Operand:	k: literal data (00~256)	
Operation:	k → Wreg, Stack → PC	
Description:	When returning from a subroutine, loads 8-bits literal data into the Wreg register, and move back to the calling program by loading the return address stored in Stack into PC register.	

Using MRPIC-IDE and ROM writer, compile the program code and write it to the PIC16F84. After putting the PIC16F84 chip into the breadboard providing the circuit for the digital dice, verify that it works as intended. If not, debugging should be followed.

6. Advanced Exercise

Rewrite the main routine code in Exercise 1 (LED Control Program) using other instructions, and verify that it works as intended.