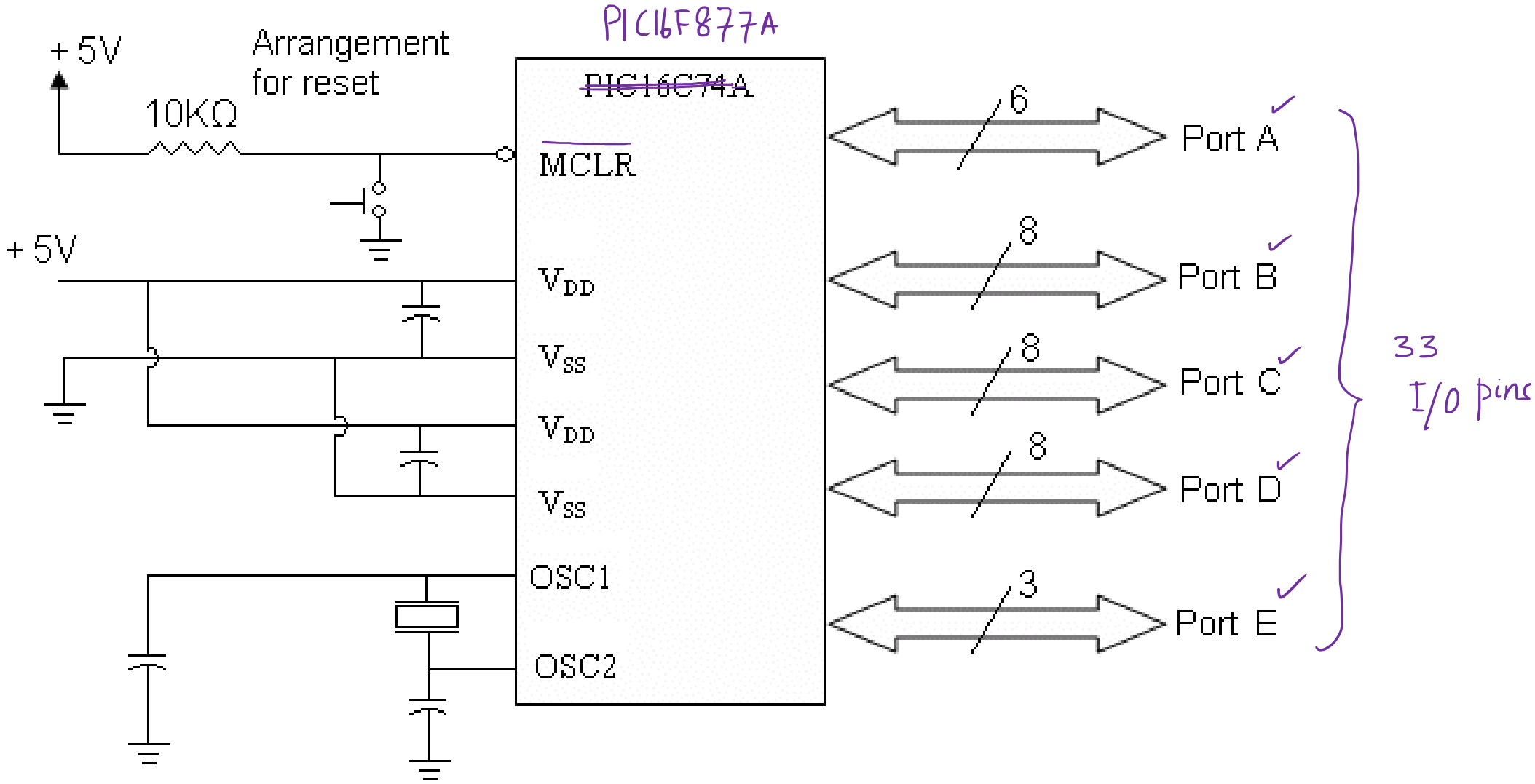
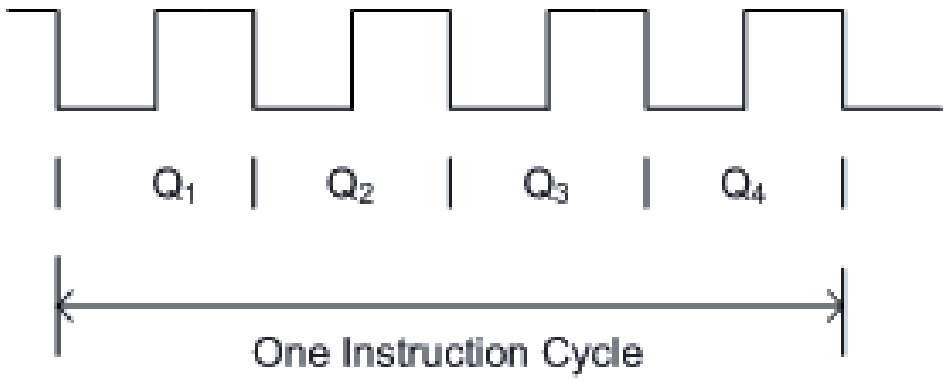
Pin Configuration of PIC16F877A



Functions of Various Port Pins

Port	Altenative uses of I/O pins	No.of I/O pins
Port A	A/D Converter inputs	6
Port B	External interrupt inputs	8
Port C	Serial port, Timer I/O	8
Port D	Parallel slave port	8
Port E	A/D Converter inputs	3
	Total I/O pins	33
	Total pins	40

An Instruction Cycle



4 clock cycles

Basic Architecture of PIC16F877A

8-bit Microcontroller

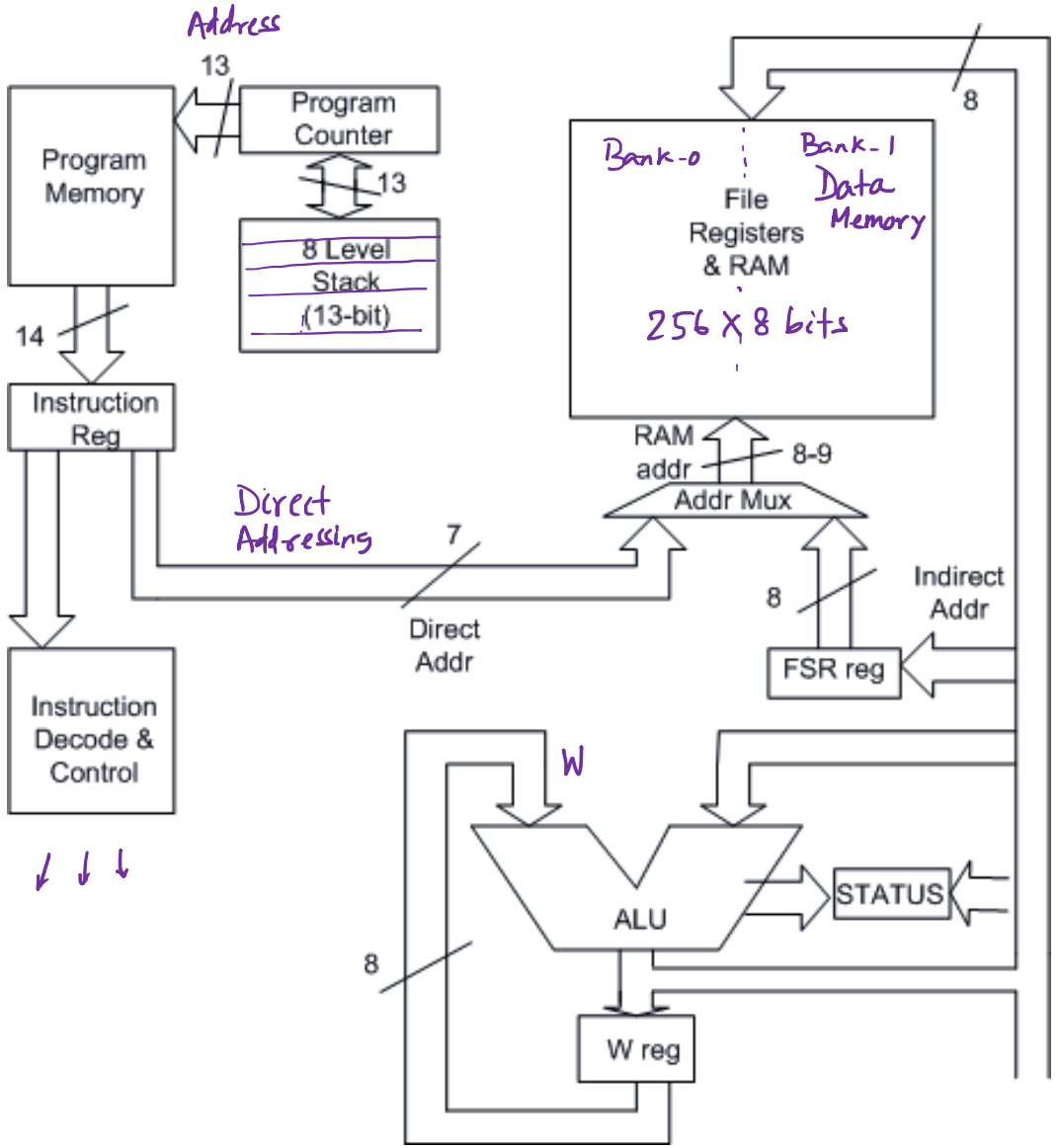
Instructions are stored in the Program Memory → Program Memory

13
2 X 14 bits

Instruction

Opcode — Operation code

Operand — Data / Address



Except for W register, all registers are mapped to the Data Memory.

FSR = File Selection Register (Indirect Data Addressing)
Memory Pointer

W Reg = Working Register

CPU Registers!

STATUS Register

The STATUS register is a 8-bit register that stores the status of the processor. This also stores carry, zero and digit carry bits.

STATUS – address 03H, 83H

Bank-0 Bank-1 Bank-0 & Bank-1

RP1	RP0	Bank Selected
0	0	Bank-0
0	1	Bank-1
1	X	Bank-2

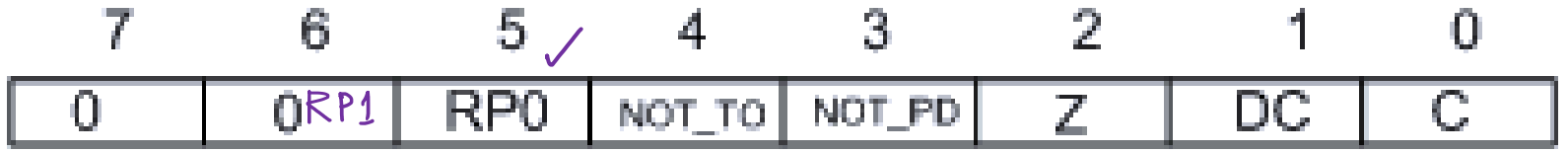


Figure: STATUS register

C = Carry bit

DC = Digit carry (same as auxiliary carry)

Z = Zero bit

NOT_TO and NOT_PD – Used in conjunction with PIC's sleep mode

RP0– Register bank select bit used in conjunction with direct addressing mode.

FSR Register

Bank-0 Bank-1

(File Selection Register, address = 04H, 84H)

FSR is an 8-bit register used as data memory address pointer. This is used in indirect addressing mode.

FSR stores/has the address of the data memory (operand) in indirect addressing mode.

INDF Register

Bank-0 Bank-1

ADDWF INDF Indirect addressing
W ← W + (FSR)
← (FSR)

(INDirect through FSR, address = 00H, 80H)

INDF is not a physical register. Accessing INDF access is the location pointed to by FSR in indirect addressing mode.

PC ← Program Counter (13-bit Reg)

5-bit 8-bit PCL

Bank-0 Bank-1

PCL Register

(Program Counter Low Byte, address = 02H, 82H)

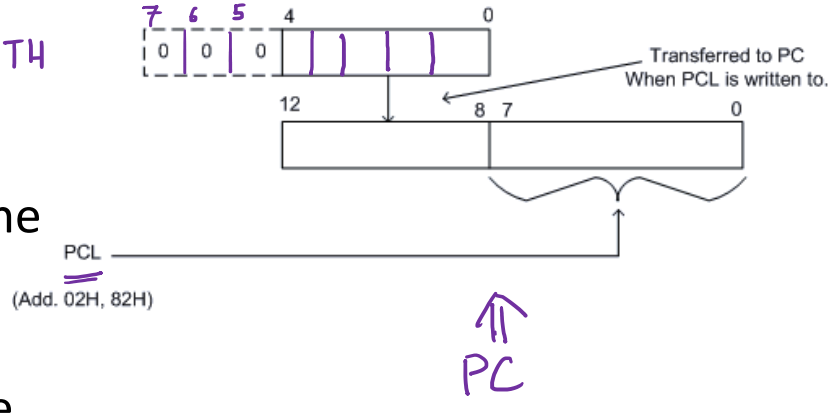
PCL is actually the lower 8-bits of the 13-bit program counter. This is a both readable and writable register.

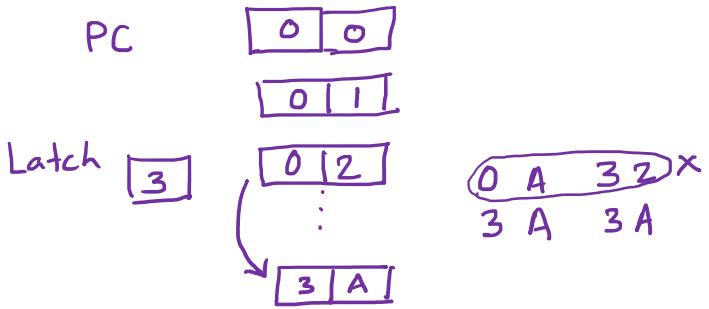
PCLATH Register

Bank-0 Bank-1

(Program Counter Latch, address = 0AH, 8AH)

PCLATH is a 8-bit register which can be used to decide the upper 5bits of the program counter. PCLATH is not the upper 5bits of the program counter. PCLATH can be read from or written to without affecting the program counter. The upper 3bits of PCLATH remain zero and they serve no purpose. When PCL is written to, the lower 5bits of PCLATH are automatically loaded to the upper 5bits of the program counter, as shown in the figure.





Register File Map

Data Memory Map

Bank-0 | Bank-1

	Bank-0	Bank-1	
00	INDF ✓	INDF ✓	80
01	TMR0	OPTION	81
02	PCL ✓	PCL ✓	82
03	STATUS ✓	STATUS	83
04	FSR ✓	FSR ✓	84
05	PORTA	TRISA	85
06	PORTB	TRISB	86
07	PORTC	TRISC	87
08	PORTD	TRISD	88
09	PORTE	TRISE	89
0A	PCLATH ✓	PCLATH ✓	8A
0B	INTCON ✓	INTCON ✓	8B
0C	PIR1	PIE1	8C
0D	PIR2	PIE2	8D
0E	TMR1L	PCON	8E
0F	TMR1H	.	8F
10	T1CON	.	90
11	TRM2	.	91
12	T2CON	PR2	92
13	SSPBUF	SSPADD	93
14	SSPCON	SSPSTAT	94
15	CCPR1L	.	95
16	CCPR1H	.	96
17	CCP1CON	.	97
18	RCSTA	TXSTA	98
19	TXREG	SPBRG	99
1A	RCREG	.	9A
1B	CCPR2L	.	9B
1C	CCPR2H	.	9C
1D	CCP2CON	.	9D
1E	ADRES	.	9E
1F	ADCON0	ADCON1	9F
20	General Purpose RAM	General Purpose RAM	A0
7F			BFH
			FFH
	Bank - 0	Bank - 1	

Special Purpose Registers (SPRs)

(32 bytes)

RP0 = 0

RP0 = 1

(96 bytes) General Purpose RAM

Important Registers are available / common in both the banks.

Instruction Set:

The instruction set for ~~PIC16C74A~~ PIC 16 F 877A consists of only 35 instructions. Some of these instructions are byte oriented instructions and some are bit oriented instructions.

The **byte oriented instructions** that require two parameters (For example, movf f, F(W)) expect the f to be replaced by the name of a special purpose register (e.g., PORTA) or the name of a RAM variable (e.g., NUM1), which serves as the source of the operand. ‘f' stands for file register. The F(W) parameter is the destination of the result of the operation. It should be replaced by:

F, if the destination is to be the source register.

W, if the destination is to be the working register (i.e., Accumulator or W register).

movwf PORTA
↑
Data memory/file

W → PORTA
byte

clrf PORTA
↑
Data memory

PORTA ← 00H
byte

The **bit oriented instructions** also expect parameters (e.g., btfsc f, b). Here 'f' is to be replaced by the name of a special purpose register or the name of a RAM variable. The 'b' parameter is to be replaced by a bit number ranging from 0 to 7.

Processing of a bit

bcf PORTA, 0 ⇒ 0th bit of PORTA is cleared.

bcf PORTA, 5 ⇒ 5th bit of PORTA is cleared.

bsf PORTA, 5 ⇒ 5th bit of PORTA is set.

For example:

```
Z equ 2
btfsc STATUS, Z
```

Z has been equated to 2. Here, the instruction will test the Z bit of the STATUS register and will skip the next instruction if Z bit is clear.

The **literal instructions** require an operand having a known value (e.g., 0AH) or a label that represents a known value.

Literal value of the operand is available in the instruction.

For example:

```
NUM equ 0AH ;        Assigns 0AH to the label NUM ( a constant )
movlw NUM ;          will move 0AH to the W register.
```

movlw 05H (05H: literal) W ← 05H

Mnemonics	Description	Instruction Cycles
bcf f, b	Clear bit b of register f	1
bsf f, b	Set bit b of register f	1
clrw	Clear working register W	1
clrf f	Clear f	1
movlw k	Move literal 'k' to W	1
movwf f	Move W to f	1
movf f, F(W)	Move f to F or W	1
swapf f, F(W)	Swap nibbles of f, putting result in F or W	1
andlw k	And literal value into W	1
andwf f, F(W)	And W with F and put the result in W or F	1
andwf f, F(W)	And W with F and put the result in W or F	1
iorlw k	inclusive-OR literal value into W	1
iorwf f, F(W)	inclusive-OR W with f and put the result in F or W	1
xorlw k	Exclusive-OR literal value into W	1
xorwf f, F(W)	Exclusive-OR W with f and put the result in F or W	1
addlw k	Add the literal value to W and store the result in W	1
addwf f, F(W)	Add W to f and store the result in F or W	1
sublw k	Subtract the literal value from W and store the result in W	1
subwf f, F(W)	Subtract f from W and store the result in F or W	1
rlf f, F(W)	Copy f into F or W; rotate F or W left through the carry bit	1
rrf f, F(W)	Copy f into F or W; rotate F or W right through the carry bit	1

Conditional jumps

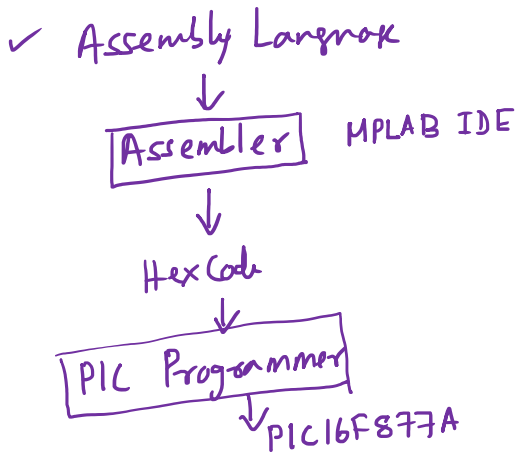
btfsc f, b	Test 'b' bit of the register f and skip the next instruction if bit is clear	1 / 2
btfss f, b	Test 'b' bit of the register f and skip the next instruction if bit is set	1 / 2
decfsz f, F(W)	Decrement f and copy the result to F or W; skip the next instruction if the result is zero	1 / 2
incfcz f, F(W)	Increment f and copy the result to F or W; skip the next instruction if the result is zero	1 / 2
goto label	Go to the instruction with the label "label"	2
call label	Go to the subroutine "label", push the Program Counter in the stack	2
retrun	Return from the subroutine, POP the Program Counter from the stack	2
retlw k	Retrun from the subroutine, POP the Program Counter from the stack; put k in W	2
retie	Return from Interrupt Service Routine and re-enable interrupt	2
clrwdt	Clear Watch Dog Timer	1
sleep	Go into sleep/ stand by mode	1
nop	No operation	1

Experiment No 6 (Experiments with PIC16F877A Microcontroller)

Objectives:

(a) Familiarization with MPLAB IDE and PIC Microcontroller Programming

(b) To light up an LED and realize an 8-bit binary UP counter



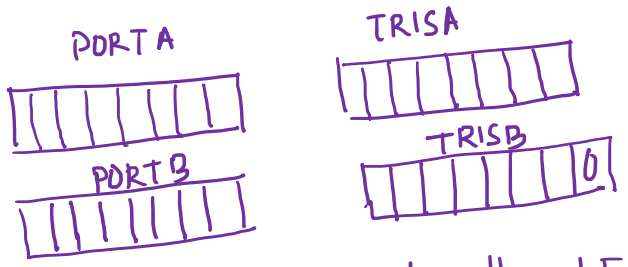
PART A) Familiarisation with MPLAB IDE PIC Microcontroller Programming

Go through the tutorial of MPLAB IDE and PIC Programmer. Get familiarized with various features of MPLAB.

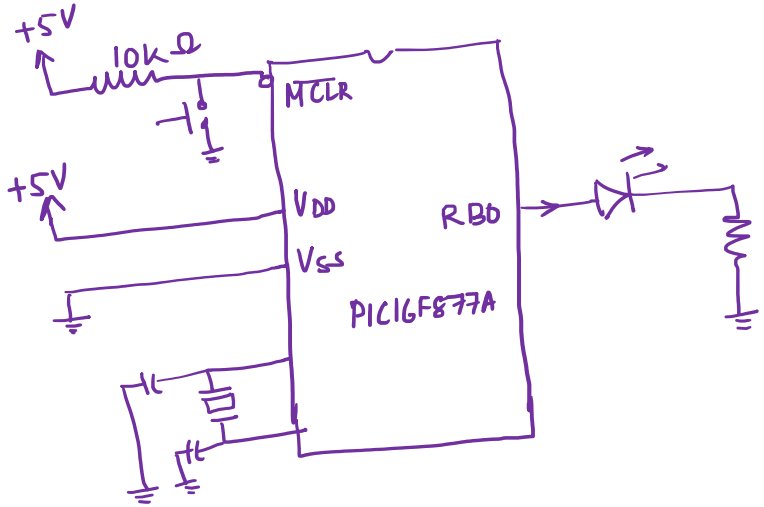
We use the MPLAB IDE Software for designing, editing & compiling codes. Look out for the MPLAB Tutorial for details, on how to begin with a new project & end up with a .hex file of the required code (after simulator testing.)

Read the PICPgm Tutorial to know, how to feed the code (.hex file) into the microcontroller.
Feed the Test File into the MicroController to understand the process.

Bank-0 Bank-1



Output → 0
Input → 1

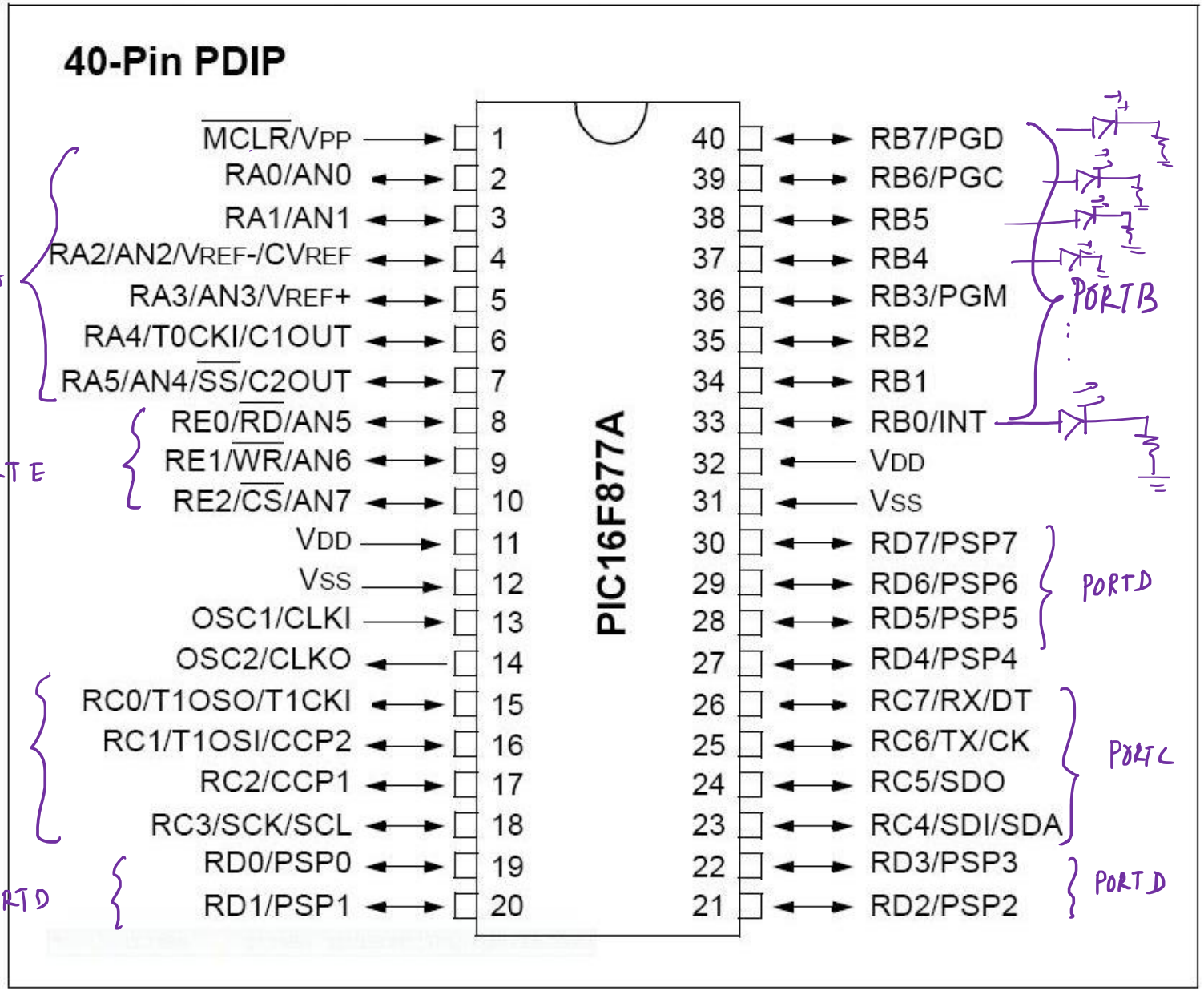


Program to turn 'ON' the LED Connected to PORTB Pin-0

```
        bcf   STATUS, RP1
        bsf   STATUS, RP0      ;  Switching to Bank-1
        bcf   TRISB, 0
        bcf   STATUS, RP0      ;  Switching to Bank-0
LOOP:   bsf   PORTB, 0         ;  PORTB Pin-0 is made→1
        Call  Delay
        bcf   PORTB, 0         ;  PORTB Pin-0 is made→0
        goto  LOOP
```

} Blinks the LED Connected to PORTB Pin-0

Pin Diagram of PIC16F877A



PART B) To Light up an LED

Problem Statement:

Connect an LED to the RB0 of PORTB (Pin No. 33 of the IC), and write a code that will light up the given LED.

Hints for writing the program:

Write codes sequentially for each of the following

a) Select the Register Bank that contains the TRISB Register
b) Clear (equate to 0) the 0^{th} Bit of TRISB Register (to configure RB0 as an Output Pin)
c) Select the Register Bank that contains the PORTB Register
d) Set (equate to 1) the 0^{th} Bit of PORTB Register, so that RB0 outputs a High voltage and lights up the LED.

Compile the code, test it using the MPLAB Simulator, and feed the .hex file into the IC.

PART C) To realize a software 8-bit DOWN counter using PIC Microcontroller

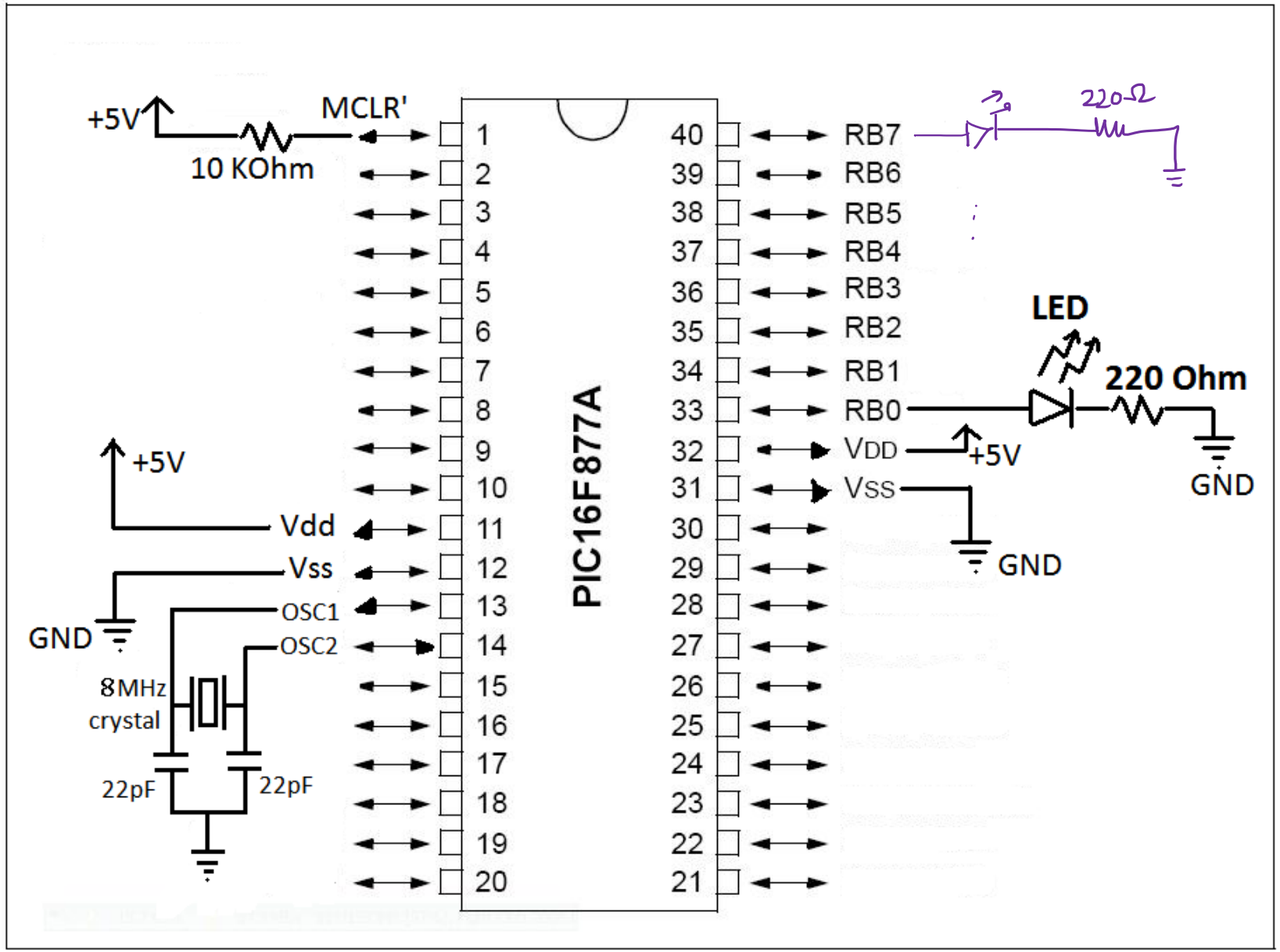
Problem Statement

To realize an 8-bit UP counter through PORTB

Procedure:

Connect 8 LEDs to the port pins of PORTB through current limiting resistors (220 Ohm each). Write a delay routine to generate approximately 0.5 s delay. Define an 8-bit variable ‘Counter’. Initialize ‘Counter’ to FFH. Use the instruction ‘decf Counter, W’ to decrement the counter. Output the content of W to PORTB. Introduce a delay of 0.5 s and repeat in an infinite loop.

Circuit Diagram of the Microcontroller Circuit for LED Lighting



Blinking of LED with a Delay

```
    MAIN_PROG     CODE
Count  Res   1
    start

            bsf     STATUS,  RP0
            bcf     STATUS,  RP1
            bcf     TRISB,   0

            bcf     STATUS,  RP0

    Loop

            bsf     PORTB,   0          ;  LED is made 'ON'

            movlw   0xFF        W ← FF H
            movwf   count       W → Count
```

```
Delay1
        nop
        decfsz      count
        goto        Delay1

        bcf         PORTB,    0         ; LED is made 'OFF'

        movlw       0xFF                W ← FFH
        movwf       count               W → Count
Delay2
        nop
        decfsz      count
        goto        Delay2

        goto Loop

    END                     ; directive 'end of program'
```