MECE336 Microprocessors I
Subtraction and Lookup Tables

Dr. Kurtuluş Erinç Akdoğan

kurtuluserinc@cankaya.edu.tr

Course Webpage: http://MECE336.cankaya.edu.tr

ÇANKAYA ÜNİVERSİTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

Status Register (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C
bit 7							bit 0

bit 7-6 **Unimplemented:** Maintain as '0'

bit 5 **RP0**: Register Bank Select bits (used for direct addressing)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)

bit 4 **$\overline{\text{TO}}$**: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 **$\overline{\text{PD}}$**: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{\text{borrow}}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{\text{borrow}}$, the polarity is reversed)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

Subtraction: Background Twos-Complement

- Binary operation that can be used for subtraction
- Computation for a given binary number B
  - Take the bitwise complement of B (called ones-complement)
  - Add 1 to the result
- Examples: suppose we want to find how -28
- First we write out 28 in binary form.

  00011100
- Then we invert the digits. 0 becomes 1, 1 becomes 0.

  11100011
- Then we add 1.

  11100100
- That is how one would write -28 in 8 bit binary.

Subtraction: Background

Subtraction of Two Binary Numbers: B1 - B2

- Compute the twos-complement of B2
  - Add B1 and the twos-complement of B2
  - Result is B1 - B2
  - If the result is negative, there is "borrow‟ indicated with C flag is zero
- Examples

```
 (+8) 0000 1000                    0000 1000
-(+5) 0000 0101 -> Negate -> +1111 1011
-----                          -----------
 (+3)                          1 0000 0011 : discard carry-out
```

```
 (+3)    0000 0011
+(-8)    1111 1000
-----------------
 (-5)    1111 1011
```

Subtraction: Instructions SUBWF

- Substract Working Register from File Register
- subwf f,d: Subtract the W register from the content of memory location f. Result is written in
  - Working register W if d = 0
  - File register f if d = 1
- The C/borrow flag (bit 0) in the Status register is
  - 0 if there is borrow
  - 1 if there is no borrow

Example

- Write a program to subtract h’52’ - h’53’. Show the result at PORTB.

```
;======8_bit subtraction====
    LIST        P=16F84A
    INCLUDE   ‘’P16F84A.INC’’
                CLRF      PORTB
                BSF       STATUS, 5     ; in BANK1
                CLRF      TRISB         ;PORTB is output
                BCF        STATUS, 5    ; in BANK0
                MOVLW  h’52’            ; W=h’52’
                MOVWF   PORTB           ;PORTB=52
                MOVLW   h’53’           ; W=h’53’
                SUBWF    PORTB,F          ;PORTB=PORTB (h’52’ )-W(h’53’),result
negative
                COMPF     PORTB
                INCF        PORTB         ;2’s complement os result,
 LOOP
            GOTO     LOOP
            END
```

Subtraction: Instructions SUBLW

- Substract Working Register from Literal
- sublw k: Subtract the W register from a literal k. Result is written into W.
- The C/borrow flag (bit 0) in the Status register is
  - 0 if there is borrow
  - 1 if there is no borrow

Examples

- Example

  movlw 0

  sublw 0
- Means load W with 0x00. Subtract that from 0x00.
- Subtraction is by complementing the W register and adding 1 (2's complement), and adding to the literal.
- 0-0= 0xFF + 1 + 0x00 = 0x00 (C set)

- Example
- In general, the C bit (really a borrow rather than carry for subtraction) is set when the result is positive (including zero), as is normal in 2's complement subtraction.

  movlw 0x00

  sublw 0x33
- 0x33-0x00= 0xFF + 1 + 0x33 = 0x33 (C set)

Subtraction of Two 16-bit Numbers

- If the numbers greater than 1 byte (8 bit), we can subtract these numbers using 16-bit subtraction. When subtracting two 16-bit data operands, we need to be concerned with the propagation of a carry from the lower byte to the higher byte.
- For example look at the subtraction of h’3CE7’-h’3B8D’

```
   3C E7            3C 17
 - 3B 8D          - 3B 8D
 -------          -------
   01 5A            01 5A
```

- When the first byte is subtracted, there is a carry (E7-8D=59, C=1, positive). Subtract high byte directly.
- After the low byte subtraction; If C=0 subtract 1 from high byte of first number. And then subtract higher bytes. And control the carry again. If C=1, show output directly. If C=0, take 2’s complement of output and show the result.

Example

- Write a program to subtract two 16-bit numbers. The numbers are h'3CE7' and h' 3B8D'. Show low byte of the result at PORTB. When bit_1 of PORTA (RA1) is pressed, show high byte of the result at PORTB.

Solution: 2 byte (16-bit) numbers;

- Draw **FLOW CHART DIAGRAM**
- A=3CE7, B= 3B8D
- Low byte of A (AL)=E7, High byte of A (AH)=3C
- Low byte of B (BL)=8D, High byte of B (BH)=3B

```
;=======16-bit SUBTRACTION =======
        LIST        P=16F84A
        INCLUDE     "P16F84A.INC"
                    CLRF     PORTB
                    BSF      STATUS, 5     ; in BANK1
                    CLRF     TRISB          ;PORTB is output
                    MOVLW  h'FF'
                    MOVWF  TRISA           ; PORTA is input
                    BCF      STATUS, 5     ; in BANK0
          AL        EQU      h'0C'         ; Address of AL
          AH        EQU      h'0D'         ; Address of AH
          BL        EQU      h'0E'         ; Address of BL
          BH        EQU      h'0F'         ; Address of BH
BEGIN
                    MOVLW  h'E7'           ; W=h'E7'
                    MOVWF  AL              ;AL=h'A3'
                    MOVLW  h'3C'           ; W=h'3C'
                    MOVWF  AH              ;AH=h'3C'
                    MOVLW  h'8D'           ; W=h'8D'
                    MOVWF  BL              ;BL=h'8D'
                    MOVLW  h'3B'           ; W=h'3B'
                    MOVWF  BH              ;BH=h'3B'
```

```
;====cont. Prog====
SUB
        MOVF    BL,W            ;W=BL
        SUBWF   AL,F            ;AL=AL-W(BL)
        BTFSS   STATUS, 0;C=0 ?
        DECF    AH,F            ;if C=0, AH=AH-1
        MOVF    BH,W            ;W=BH
        SUBWF   AH,F            ;AH=AH-W(BH)
SHOW_LOW_BYTE
        MOVF    AL,W            ;W=AL
        MOVWF   PORTB           ;show low byte at PORTB
TEST_RA1
        BTFSC   PORTA,1         ;RA1 is pressed?
        GOTO    TEST_RA1        ;if NO
SHOW_HIGH_BYTE
        MOVF    AH,W            ; if YES , W=AH
        MOVWF   PORTB           ;show high byte at PORTB
LOOP
        GOTO    LOOP
        END
```

Look-up Tables

- The instruction **movlw** allows us to introduce within the program a byte of constant data such that:

  movlw D01000

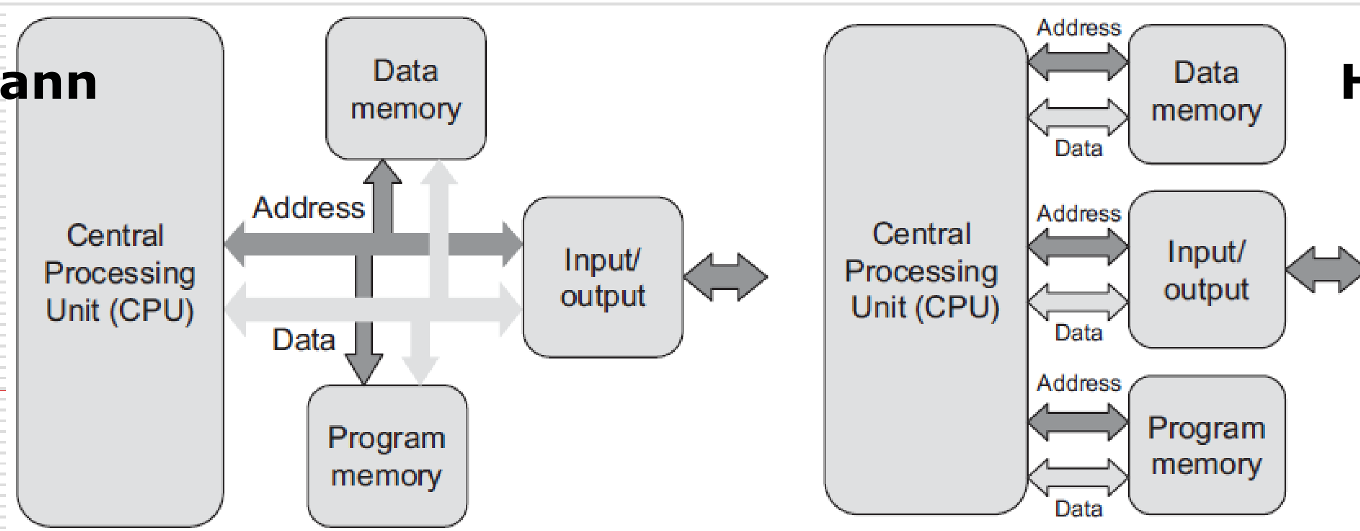
  movwf delcntr2

- This is fine for introducing single bytes of data into a program, or just a few.
- But suppose we want to place in the program a whole list of numbers, maybe
  - to generate a waveform or
  - to produce output patterns on a display.
- Suppose also that we want to be able to record where we are in the list with some sort of marker.
- The **movlw** instruction is then not really up to the job,
- We need to apply a way of setting up and accessing a block of data. This is called a **'lookup table'**.

Introducing the Look-up Table

- A **look-up table** is a **block of data** that is held in the **program memory** and which can be accessed by the program and used within it.
- In a **Von Neumann** structure with its single address and data buses, it is rather easy to set up and use look-up tables, as all memory locations are of equal size and all can be accessed with equal ease.
- In a **Harvard** structure, it is more difficult, as data must be moved from one distinct memory map to another.
- The situation is made worse by the difference in memory location size that usually exists between data and program memories.
- Therefore in a **Harvard structure, like the PIC's**, a special technique is used to create look-up tables. This introduces several important new ideas.

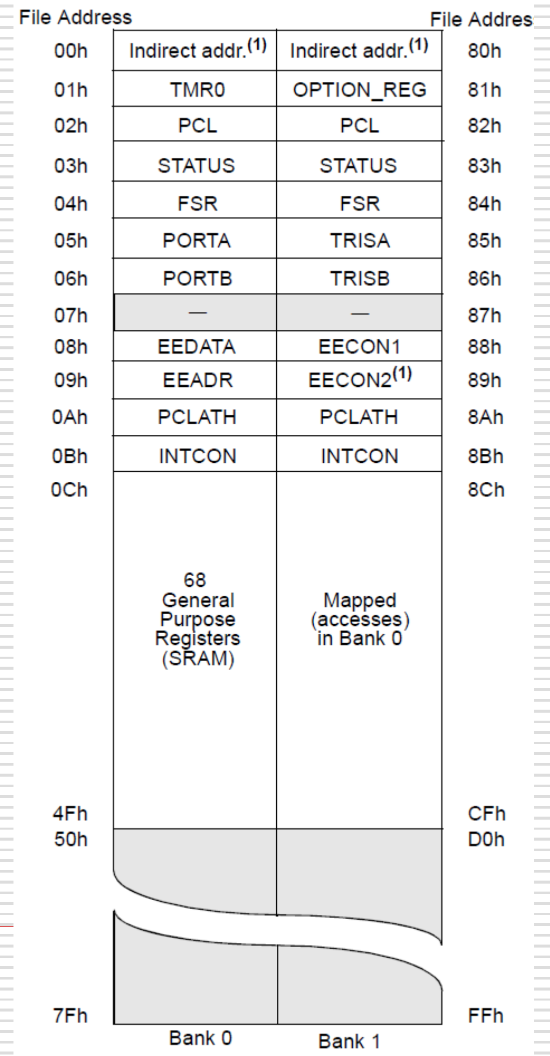
Von Neumann **Harvard**

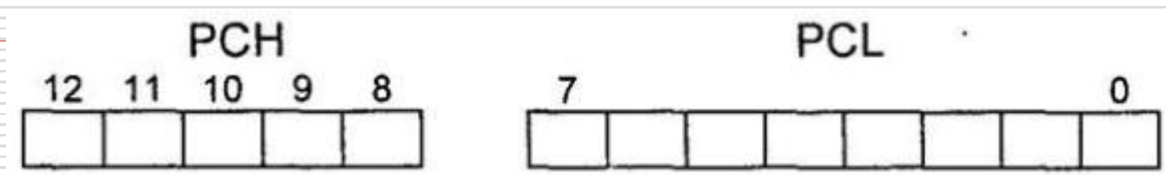


PROGRAM COUNTER

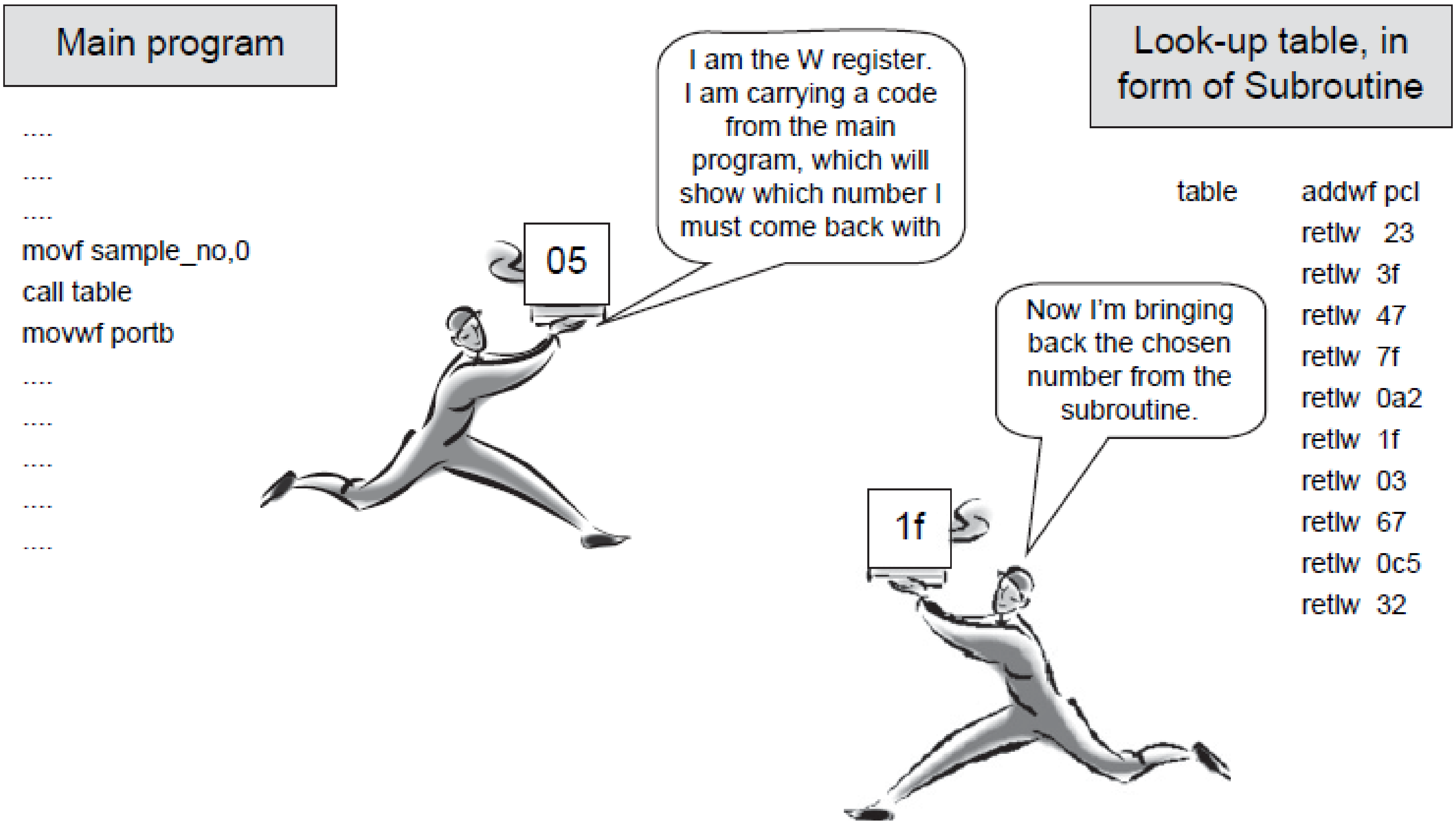
- The program counter (PC) specifies the address of the instruction to fetch for execution.
- The PC is 13 bits which potentially can address up to 2^{13}=8K=8192 instructions, although the PIC16F84 has only 1024 =1K=2^{10} (10 bits) instruction capacity.
- The Program counter normally increments up from instruction 1 at location h'000', but can skip or jump if commanded by a relevant instruction.
- Program counter (PC) is 13-bit; low 8-bit is PCL and high 5-bit is PCH. 10-bits are used for PIC16F84.
- The low byte is called the PCL register. This register is readable and writable.
- The high byte is called the PCH register. This register contains the PC<12:8> bits and is not directly readable or writable.
- PCLATH is used to write data to PCH

REGISTER FILE MAP -PIC16F84A

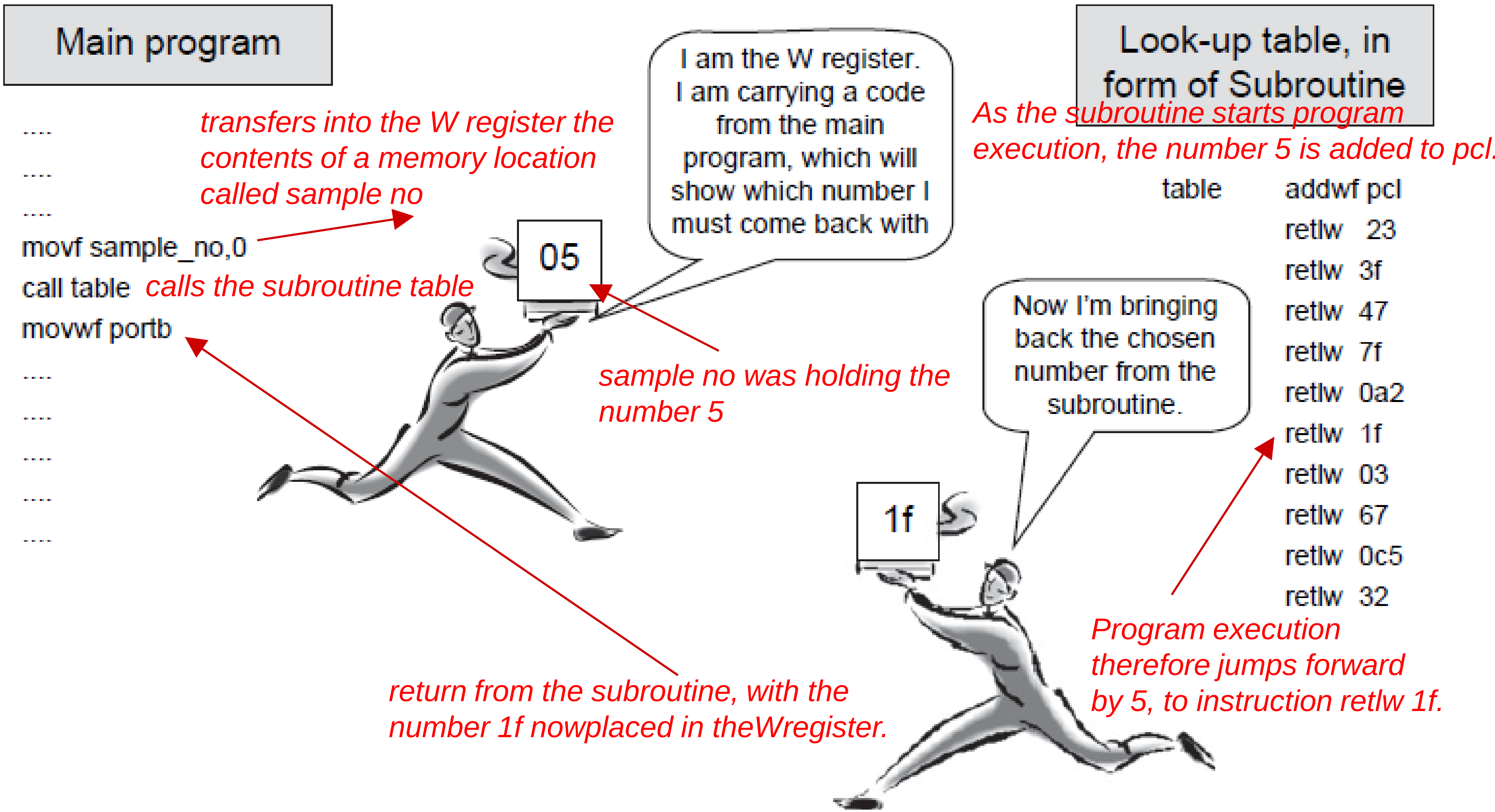




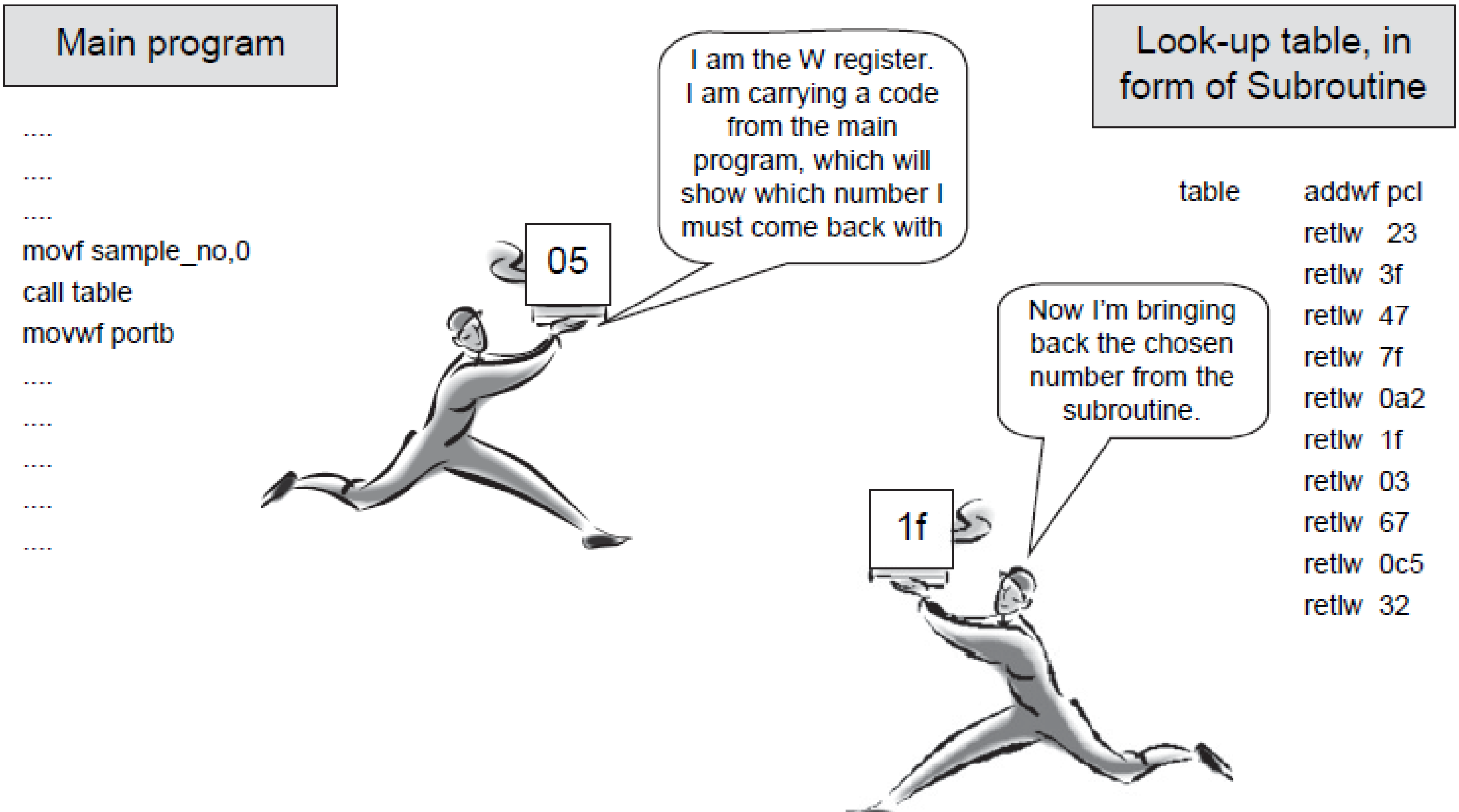
- The **look-up table** is formed as a **subroutine**.
- Every byte of data in the table is accompanied by a special instruction, **retlw**.
- This instruction is another 'return from subroutine' but with a difference – it requires an 8-bit literal operand.
- As it implements the subroutine return, it picks up its operand and puts it into the W register.
- The table is essentially a list of **retlw** instructions, each with its byte of data.



- What we need now is a technique which allows just one of those retlw instructions to be selected from the list.
- The first instruction in the subroutine,. **addwf pcl** adds the contents of the **W register** to **PCL**.
- **PCL** is the lower byte of the program counter.
- Once a number has been added to the program counter, program execution jumps forward by whatever that number was.
- If the number added is zero, then the next instruction is executed.
- In this example the CPU executes the **retlw** instruction it lands on, and then goes back to **main program**.
- Only two instructions are executed, the **addwf pcl** and the chosen **retlw**.



- In summary, the **W register is like a messenger** being sent to the subroutine. It goes to the subroutine carrying a code (which acts as a pointer) showing which line in the table is wanted.
- It comes back carrying the number stored in that line.
- There is one possible problem with this approach – by manipulating only the lower byte of the program counter we can only operate within the **first 256 words of program memory**, or within any page following



Example Program With Look-up Table

Program takes 8-bit values from a table and transfers them to the ping-pong LEDs with a delay between each data transfer. The overall effect is a display of randomly flashing LEDs

REGISTER FILE MAP -PIC16F84A

File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	—	—	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch–4Fh	68 General Purpose Registers (SRAM)	Mapped (accesses) in Bank 0	8Ch–CFh
50h–7Fh			D0h–FFh

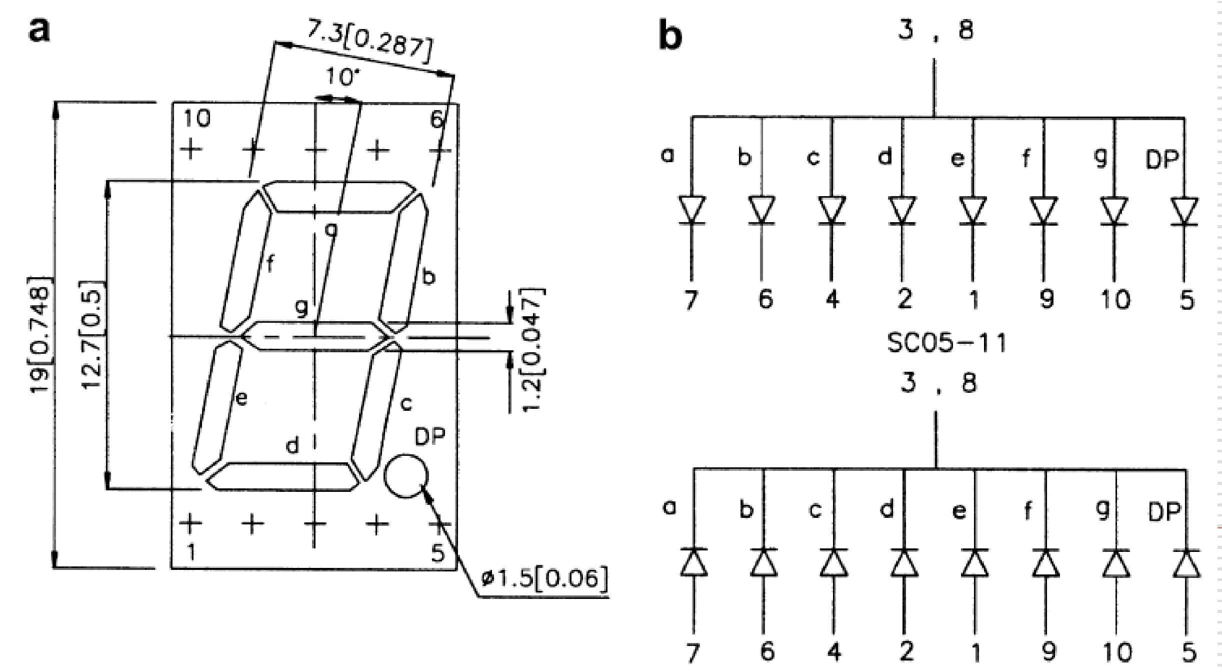
```
;******************************************************************
;Flashing LEDs 3.
;This program continuously outputs a series of LED patterns,
;using simulation or ping-pong hardware.
;TJW 5.3.05.                   Tested in simulation 11.3.05.
;******************************************************************
;Clock is 800kHz
;Configuration Word: WDT off, power-up timer on,
;                            code protect off, RC oscillator
;
;specify SFRs
pcl      equ   02
status   equ   03
porta    equ   05
trisa    equ   05
portb    equ   06
trisb    equ   06
;
pointer equ 10
delcntr1 equ 11
delcntr2 equ 12
;
        org    00
;Initialise
start   bsf    status,5        ;select memory bank 1
        movlw  B'00011000'
        movwf  trisa           ;port A according to above pattern
        movlw  00
        movwf  trisb           ;all port B bits output
        bcf    status,5        ;select bank 0
;
```

```
;The "main" program starts here
        movlw 00              ;clear all bits in port A
        movwf porta
        movwf  pointer        ;also clear pointer
loop    movf   pointer,0      ;move pointer to W register
        call  table
        movwf  portb          ;move W register, updated from table SR, to port B
        call  delay
        incf  pointer,1
        btfsc pointer,3       ;test if pointer has incremented to 8
        clrf   pointer        ;if it has, clear pointer to start over
        goto  loop
;
;************************************************************
;Subroutines
;************************************************************
;Introduces delay of 500ms approx, for 800kHz clock
...
 (delay subroutine omitted)
...

;Holds Lookup Table
table   addwf pcl
        retlw 23
        retlw 3f
        retlw 47
        retlw 7f
        retlw 0a2
        retlw 1f
        retlw 03
        retlw 67
;
        end
```

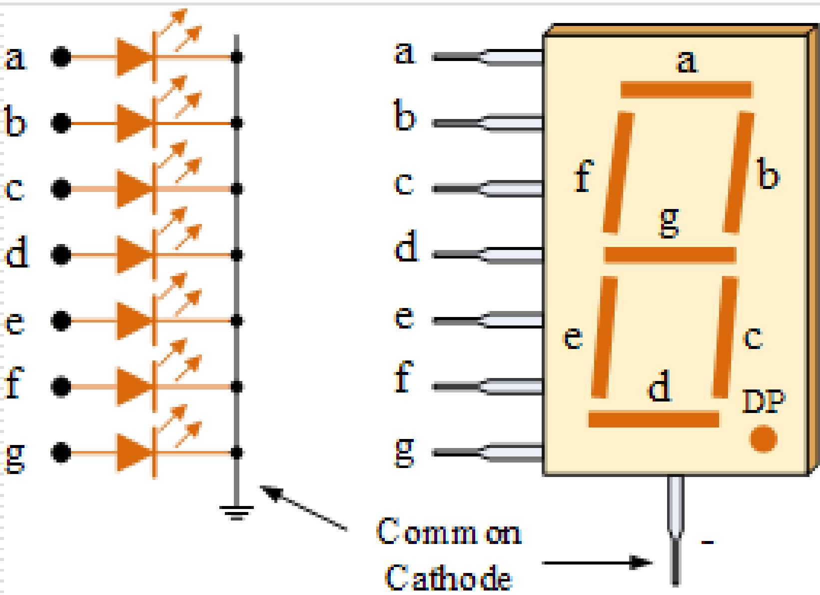
LED Arrays: Seven-segment Displays

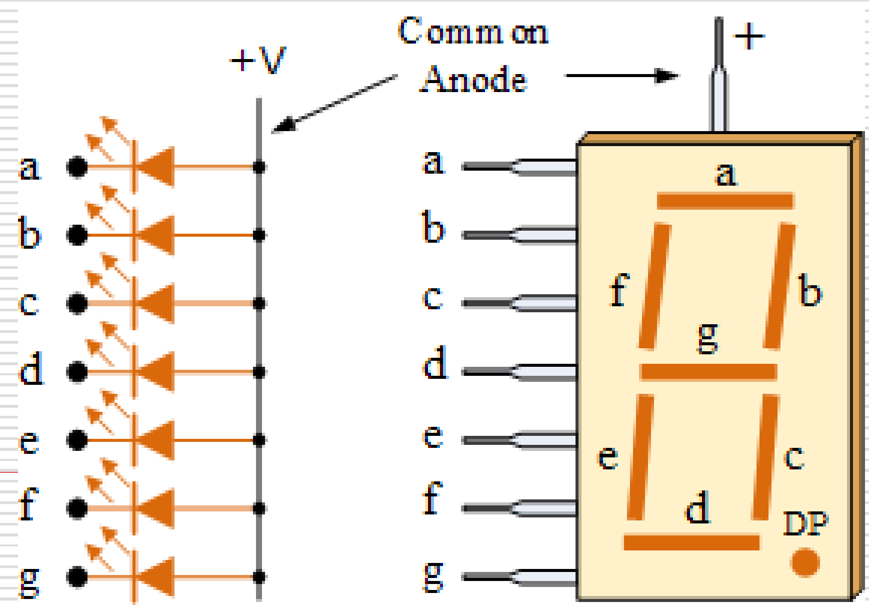
- By lighting different combinations of the seven segments, all numerical digits can be displayed, as well as a surprising number of alphabetic characters.
- A decimal point is usually included, as shown.
- The problem arises that if each segment is illuminated by an LED, then 14 connections are required, and that is just for one digit.
- The common anode/common cathode connection requires less connections



7-Segment Display: Types
Common Cathode (CC), Common Anode (CA)

- In the common cathode display, all the cathode connections of the LED segments are joined together to logic “0” or ground.
- The individual segments are illuminated by application of a “HIGH”, or logic “1” signal via a current limiting resistor to forward bias the individual Anode terminals (a-g).
- In the common anode display, all the anode connections of the LED segments are joined together to logic “1”.
- The individual segments are illuminated by applying a ground, logic “0” or “LOW” signal via a suitable current limiting resistor to the Cathode of the particular segment (a-g).



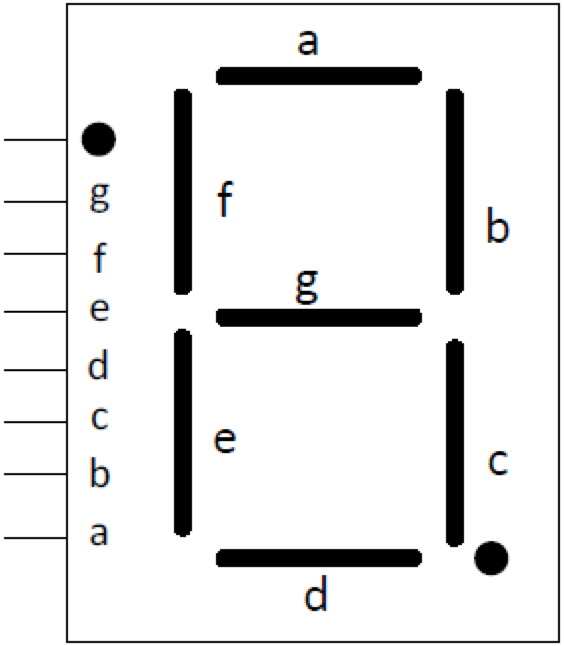


7-Segment Display: Truth Table (Common Cathode)

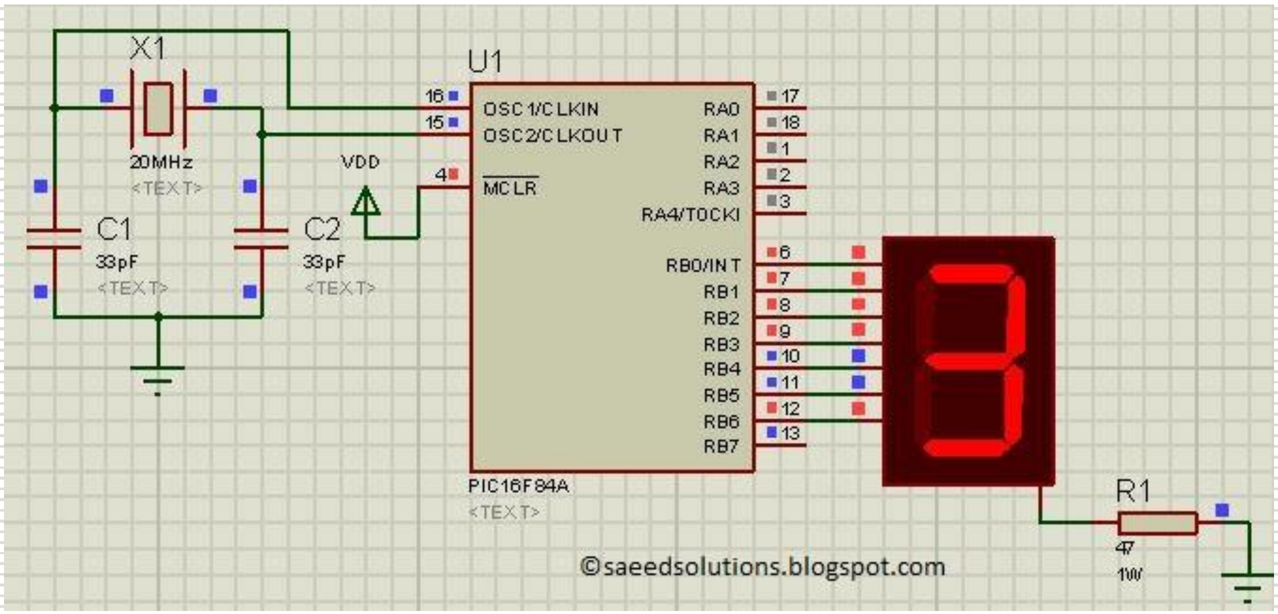
Digits

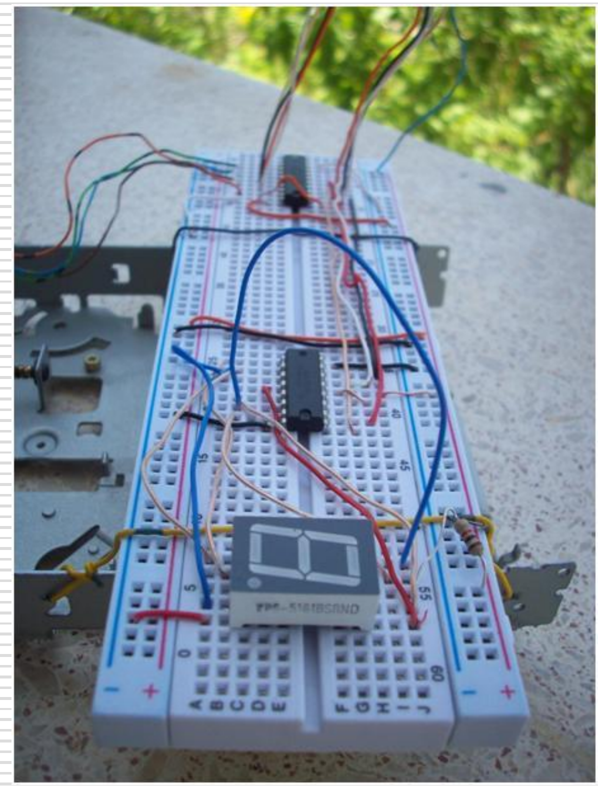
	•	g	f	e	d	c	b	a	Hex
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	1	1	1	1	0x6F
Digit with decimal point: add 0x80 to Hex									
0.	1	0	1	1	1	1	1	1	0xBF

Display



7-Segment Display: Connection Connection to PORTB of PIC





7-Segment Display: Lookup Table

Lookup Table for Digits

```
table
        addlw   pcl
        retlw   0x3F
        retlw   0x06
        retlw   0x5B
        retlw   0x4F
        retlw   0x66
        retlw   0x6D
        retlw   0x7D
        retlw   0x07
        retlw   0x7F
        retlw   0x6F
```

→ Extend to table including decimal point if desired

Numbers

Digit	Hex	Digit	Hex
0	0x3F	0.	0xBF
1	0x06	1.	0x86
2	0x5B	2.	0xDB
3	0x4F	3.	0xCF
4	0x66	4.	0xE6
5	0x6D	5.	0xED
6	0x7D	6.	0xFD
7	0x07	7.	0x87
8	0x7F	8.	0xFF
9	0x6F	9.	0xEF

Example

- Write a program to show '5' in 7-segment display.

```
;=======show '5' in7-segment display=======
            LIST        P=16F84A
            INCLUDE     ''P16F84A.INC''
            CLRF        PORTB
            BSF         STATUS, 5       ; in BANK1
            CLRF        TRISB           ;PORTB is output
            BCF         STATUS, 5       ; in BANK0
BEGIN
            MOVLW   h'05'                   ; W=h'05' (test number)
            CALL    LOOKUP_TABLE
            MOVWF   PORTB                   ;PORTB=6D
LOOP
            GOTO    LOOP
LOOKUP_TABLE
            ADDWF   PCL,F                   ;PCL=W(h'05')
            RETLW   h'3F'
            RETLW   h'06'
            RETLW   h'5B'
            RETLW   h'4F'
            RETLW   h'66'
            RETLW   h'6D'                   ;W=h'6D'
            RETLW   h'7D'
            ......
            END
```

Example

- Write a program to show '5' in 7-segment display.

```
BEGIN

MOVLW h'05' ; W=h'05' (test number)
CALL LOOKUP_TABLE
MOVWF PORTB ;PORTB=6D

LOOP
GOTO LOOP


LOOKUP_TABLE
    ADDWF PCL,F ;PCL=W(h'05')
    retlw 0x3F
    retlw 0x06
    retlw 0x5B
    retlw 0x4F
    retlw 0x66
    retlw 0x6D
    retlw 0x7D
    retlw 0x07
    retlw 0x7F
    retlw 0x6F
END
```

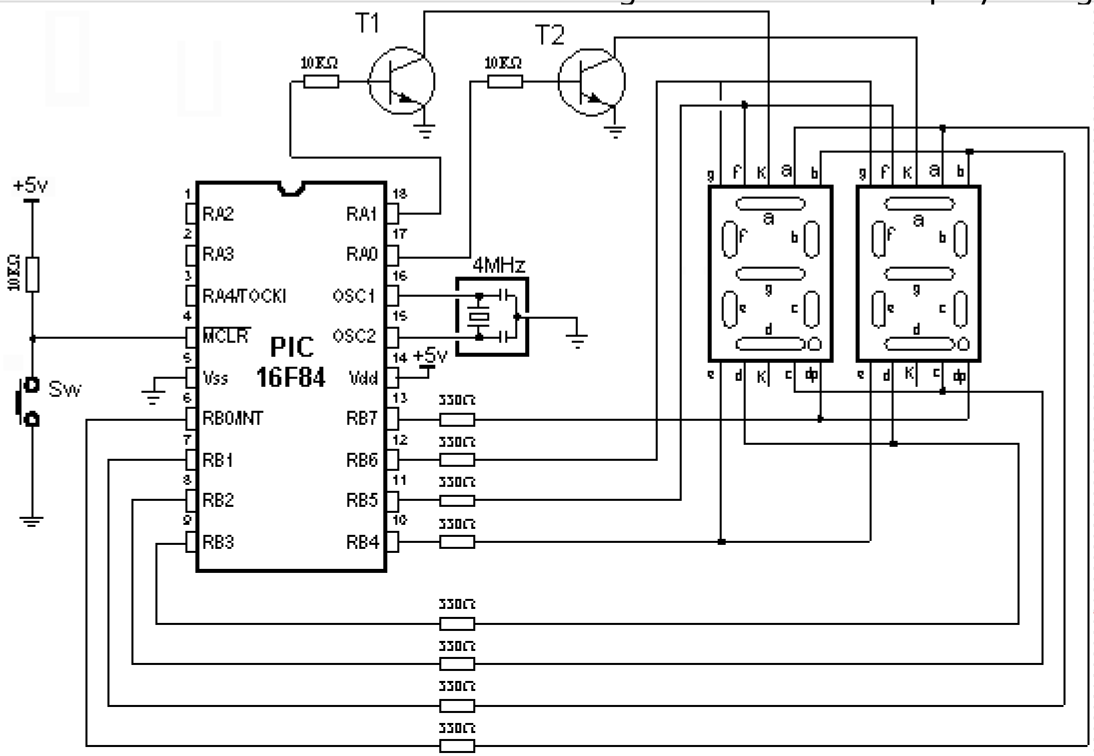
Example

Problem 17:

- A program partially written and with the delay subroutine can be found on the course webpage. Complete the program which is for a 7-segment display with common cathode to carry out following functions together. But first draw the **flow chart diagram**.
  - Display the number 9 on a 7-segment display at PORTB
  - Wait for 1 second
  - Subtract 7 from the displayed number and display the new number if the button at RA2 is pressed. If the result is negative, add 10 to the result and display the result of this computation.
  - go back to step 2.

7-Segment Display: Two Displays in Parallel

- To produce a 4, 5 or 6 digit display, all the 7-segment displays are connected in parallel.
- The common line (the common-cathode line) is taken out separately and this line is taken low for a short period of time to turn on the display.
- Each display is turned on at a rate above 100 times per second, and it will appear that all the displays are turned on at the same time.
- As each display is turned on, the appropriate information must be delivered to it so that it will give the correct reading.
- Up to 6 displays can be accessed like this without the brightness of each display being affected.



EXAMPLE: Two Digit Display

```
list p=16f84a;
    include "p16f84a.inc"
    __config _CP_OFF&_WDT_OFF&_XT_OSC;
N   equu            0x0A;  N = 10 -- delay = 10*100ms
    org 0;
main; Warning: the delay subroutine uses 0x0C, 0x0D -> we
cannot use these registers for the main program
    bsf             STATUS,5;
    clrf    TRISB; PORTB is output
    clrf    TRISA; RA2 is input
    bcf             STATUS,5;
    clrf    PORTB;
    comf    PORTB,1; all pins are 1 -> all segments are off
    clrf    PORTA;
loop
    bsf             PORTA,0; display 1 is selected
    bcf             PORTA,1;
    movlw   .8;
    call    common_anode; pin value for digit 8 in W
    movwf   PORTB; write value to PORTB
    call    delay_5ms; wait for 5 msec
    bcf             PORTA,0;
    bsf             PORTA,1; display 2 is selected
    movlw   .9;
    call    common_anode; pin value for digit 9 in W
    movwf   PORTB; write value to PORTB
    call    delay_5ms;
    goto    loop;  repeat the process

common_anode
    addwf   PCL,1;
    retlw 0x3F
    retlw 0x06
    retlw 0x5B
    retlw 0x4F
    retlw 0x66
    retlw 0x6D
    retlw 0x7D
    retlw 0x07
    retlw 0x7F
    retlw 0x6F

delay_5ms; delay subroutine for 100ms delay
    movlw   .250;
    movwf   0x0C; counter for outer loop
loop1;          outer loop with N1=250 iterations: N1 = 250, k1 =
0
    movlw   .5;
    movwf   0x0D; counter for inner loop;
loop2; inner loop with N2 = 98 iterations: N2 = 5, k2 = 0
    decfsz  0x0D,1; decrement counter2 (inner loop)
    goto    loop2;
    nop;
    decfsz  0x0C,1; decrement couter1 (outer loop)
    goto    loop1;
    return; return to delay_Nms subroutine

    end;
```