Chapter 5

Instruction to Data Copy

2/21/14

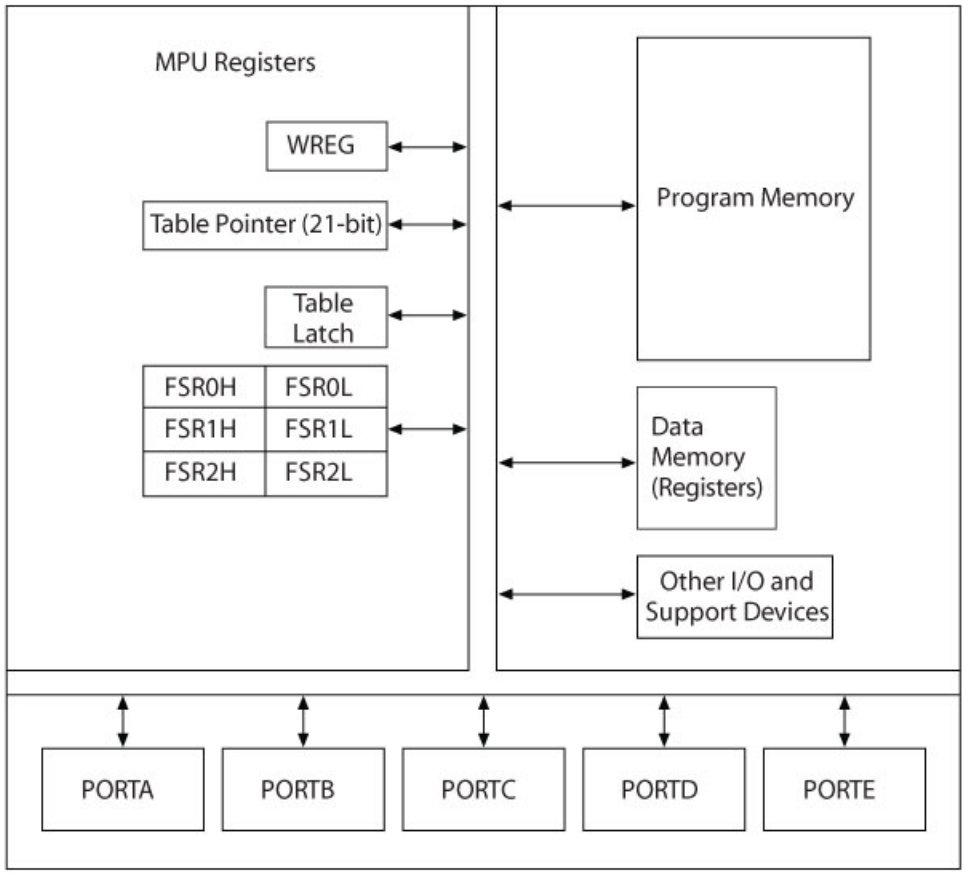
Data Copy (Move) and Set/Clear Operations

The data copy operations are classified as:

- Loading 8-bit data directly in WREG
- Copying data between WREG and data (file) register including I/O ports
- Copying data from one data (file) register to another data (file) register
- Clearing or setting all data bits in data (file) register
- Exchanging low-order four bits (nibble) with high-order four bits in data (file) register

Frequently Used Registers in ALU, Memory, and I/O Ports for Data Copy

- Literal to W
- W→ F
- F→ F
- F→ Program memory
- Swapping lower order with the higher-order



Addressing Modes

- A way of specifying of an operand
  - Direct addressing
    - The operand is a part of the instruction
  - Indirect addressing
    - An address is specified in a register and the MPU looks up the address in that register

MOV (Copy) Operations

Instructions

- MOVLW 8-bit
- MOVWF F, a
- MOVF F,d, a
- MOVFF Fs, Fd

Examples

- MOVLW 0xF2
  Load F2H in WREG
- MOVWF REG1, 0
  Copy WREG in REG1
- MOVF REG1, 1
  Copy REG1 in REG1 (check for flag)
- MOVF REG1, 0
  Copy REG1 into WREG
- MOVFF REG1,REG2
  Copy REG1 into REG2

d=0 or omitted then → WREG
a=0 or omitted then → data reg is from Access Bank

SET/CLR Instructions

Instructions

- CLRF F,a
- SETF F, a
- SWAP F,d,a

Examples

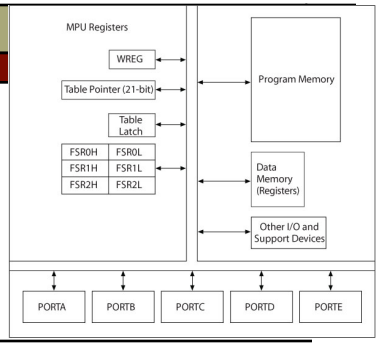
- CLRF REG1,0

  Clear REG1 located in access bank
- SETF REG1,0

  Set all bits in REG1
- SWAP REG1,1,0

  Exchange low and high nibbles in REG1, save in REG1

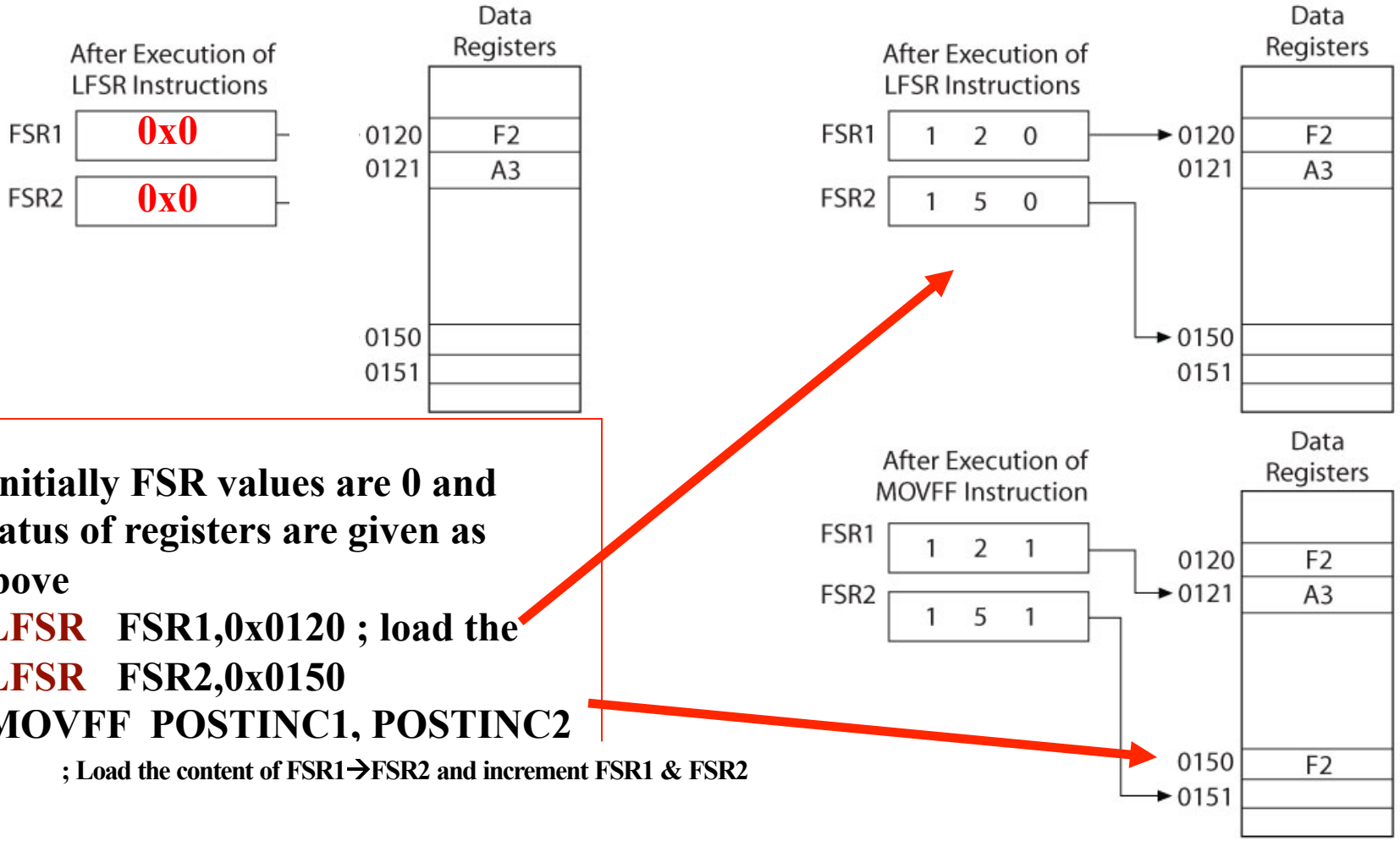
Using File Select Registers (FSRs) as Pointers to Data Registers – Indirect addressing



- Memory pointer is a register that holds the address of a data register
  - This is called indirect addressing
  - Easy to move/copy an entire block
- Three registers: FSR0, FSR1, and FSR2
  - Each FSR has a High and Low byte associated with an index
  - Used as memory pointers to data registers
- Each can be used in five different formats (operands) :
  - INDF0: Use FSR0 as pointer (index)
  - POSTINC0: Use FSR0 as pointer and increment FSR0
  - POSTDEC0: Use FSR0 as pointer and decrement SR0
  - PREINC0: Increment FSR0 first and use as pointer
  - PLUSW0: Add W to FSR0 and use as pointer

```
LFSR        FSR1,120        ; LOAD 12-BIT ADDRESS 120h INTO FSR1
```

Indirect Addressing - Example



- **Initially FSR values are 0 and status of registers are given as above**
- **LFSR FSR1,0x0120 ; load the**
- **LFSR FSR2,0x0150**
- **MOVFF POSTINC1, POSTINC2**

; Load the content of FSR1→FSR2 and increment FSR1 & FSR2

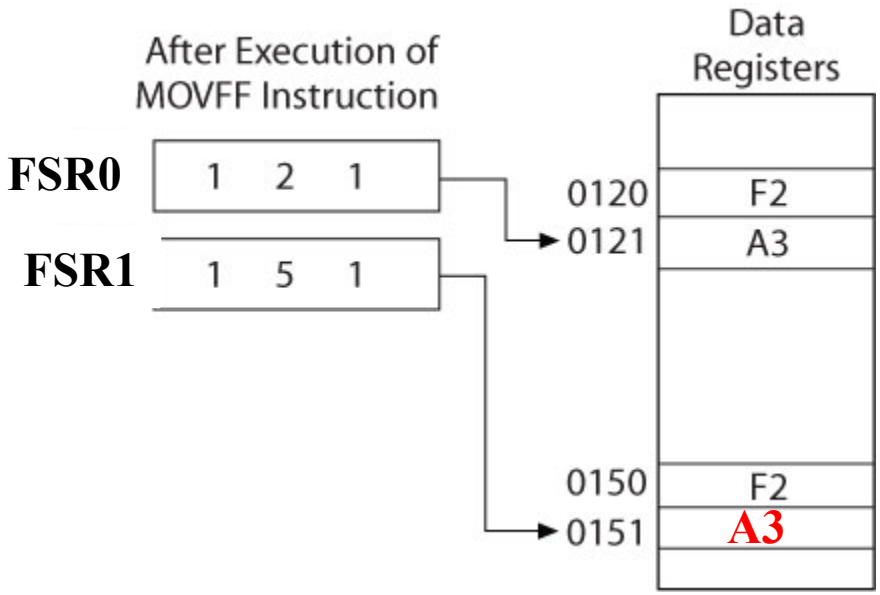
Indirect Addressing – Example (1)

Examples:

```
MOVFF     INDF0,INDF1    ; COPY BYTE FROM REGISTERS SHOWN BY
                         ;FSR0 TO FSR1- NO CHANGE IN FSR
ADDWF     POSTINC0,1     ; ADD BYTE FROM REGISTERS SHOWN BY
                         ;FSR0 AND W→REG ;
                         ;FSR0 IS INCREMENTED
```

- Hence, we will have A3 in register 0x151 after the MOVFF instruction

- Note that the pointer indexes are not changing!

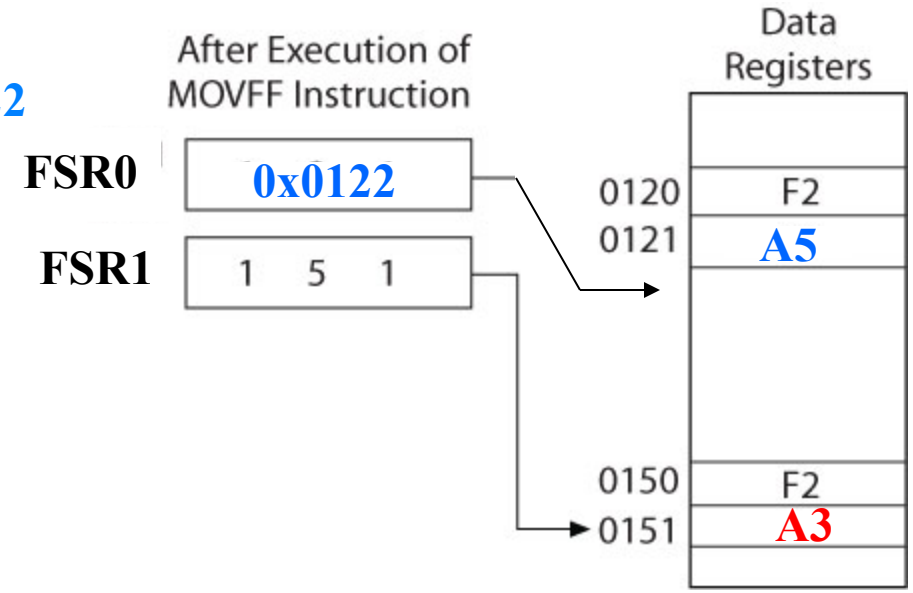


Indirect Addressing Example (2)

Examples:

```
MOVFF     INDF0,INDF1     ; COPY BYTE FROM REGISTERS SHOWN BY
                                ;FSR0 TO FSR1- NO CHANGE IN FSR
ADDWF     POSTINC0,1      ; ADD BYTE FROM REGISTERS SHOWN BY
                                ;FSR0 AND W→REG ;
                                ;FSR0 IS INCREMENTED
```

Assume W=2; after the ADD operation, A3+2=A5→Reg 0x0121, then FSR=0x0122



Initializing the RAM – Application of Indirect Addressing

```
Main:
     CLRF    FSR1H
     MOVLW   0x40
     MOVWF   FSR1L
NEXT
     SETF    POSTINC1
     BTFSS   FSR1L, 4
     GOTO    NEXT
     sleep
```

File Registers

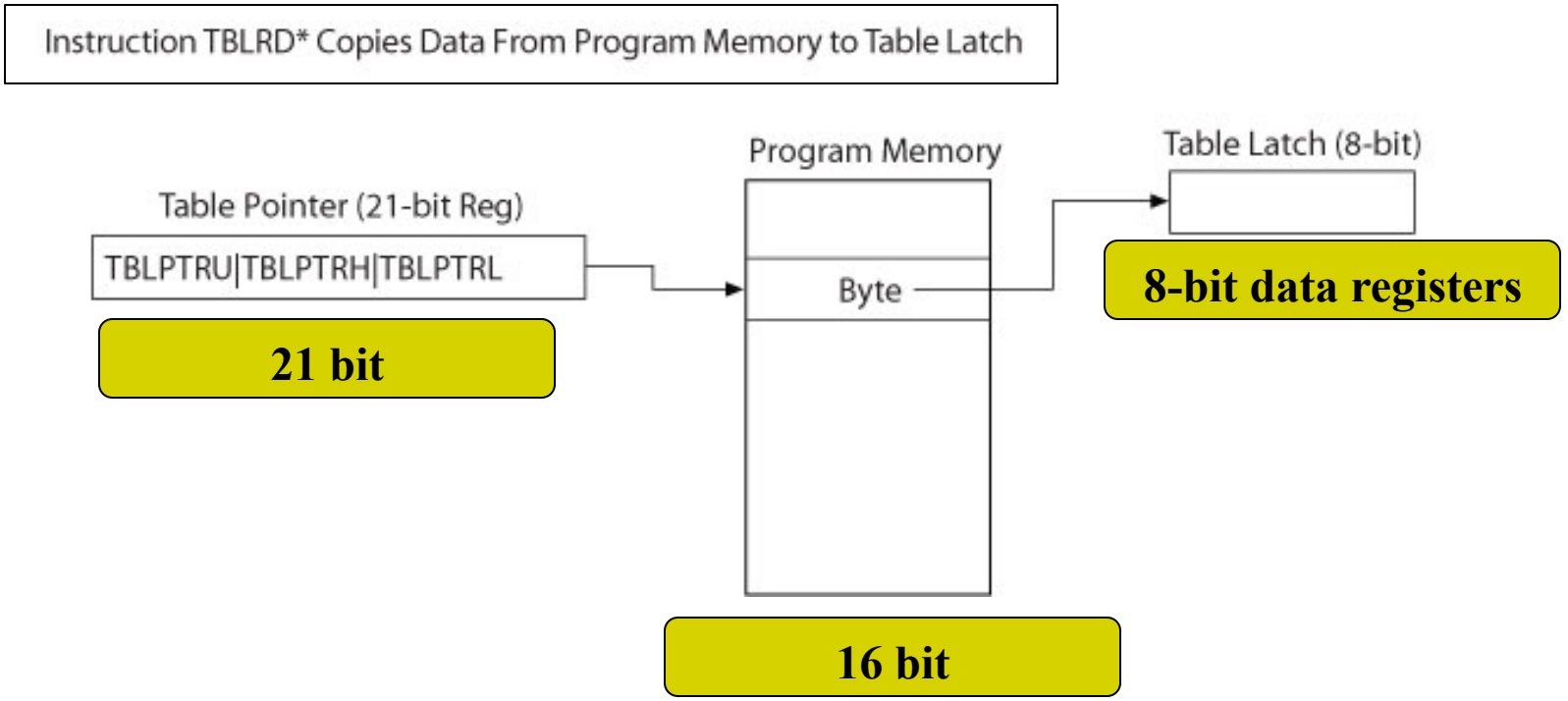
Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	44	00	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00

```
Main:
     CLRF    FSR1H
     CLRF    FSR1L
CLR_LP
     SETF    POSTINC1
     MOVLW   0x0F
     SUBWF   FSR1H, W ;(F-W)->W
     BNZ     CLR_LP

     sleep
```

What is this doing?

Using Table Pointers to Copy Data



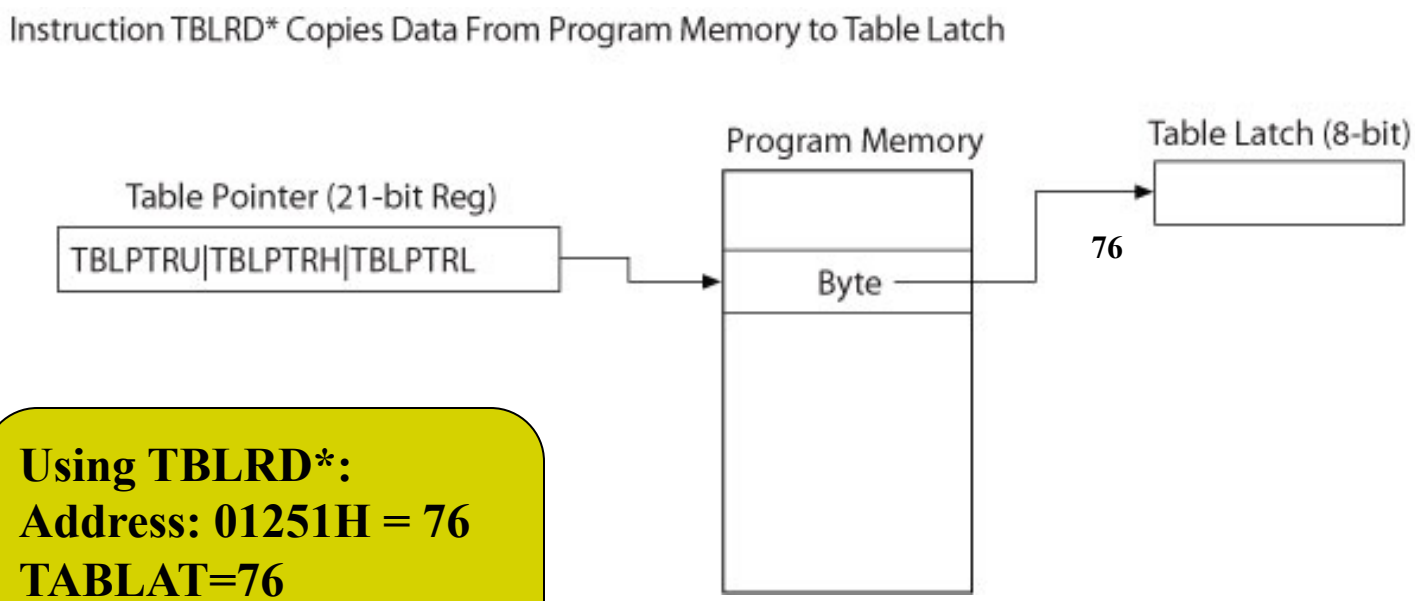
Using Table Pointers to Copy Data from Program Memory into Table Latch (REFER TO PAGE 82-83)

- TBLRD*
  - Copy from Program Memory into Table Latch Using Table Pointer
- TBLRD*+
  - Copy from Program Memory into Table Latch and Increment Table Pointer
- TBLRD*-
  - Copy from Program Memory into Table Latch and Decrement Table Pointer
- TBLRD+*
  - Increment Table Pointer first and then copy from Program Memory into Table Latch

Using Table Pointers to Copy Data from Table Latch into Program Memory

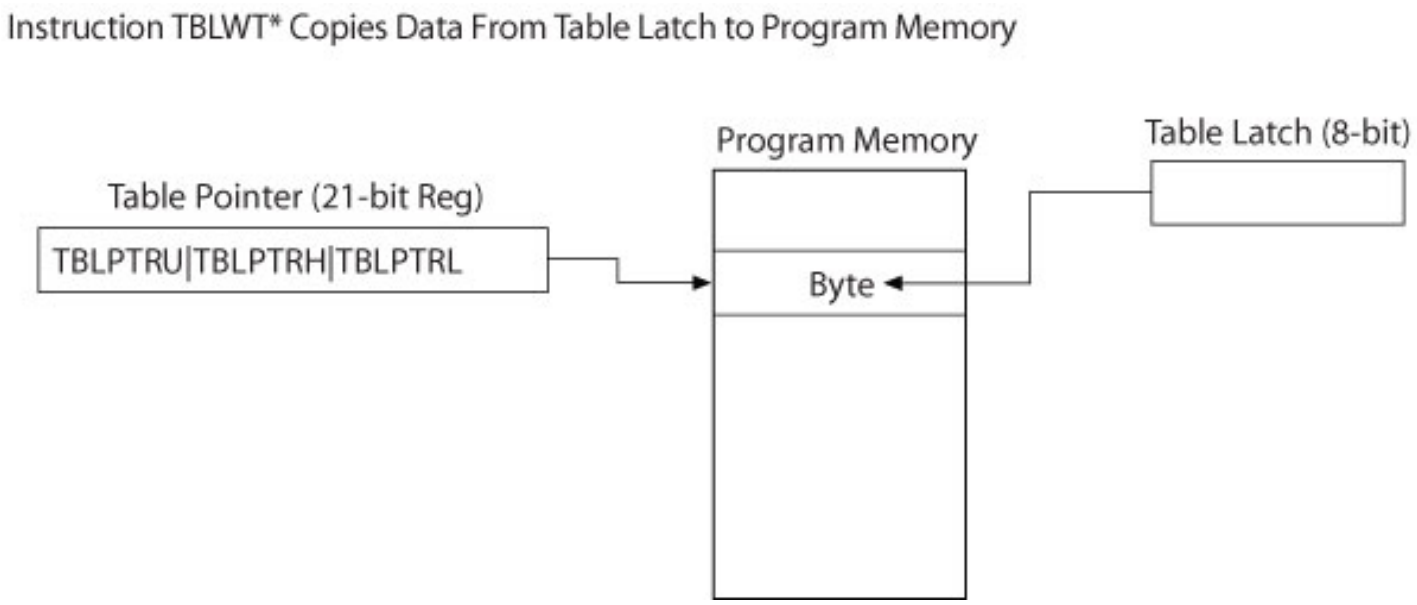
- TBLWT*
  - Copy from Table Latch into Program Memory Using Table Pointer
- TBL WT*+
  - Copy from Table Latch into Program Memory and Increment Table Pointer
- TBL WT*-
  - Copy from Table Latch into Program Memory and Decrement Table Pointer
- TBLWT+*
  - Increment Table Pointer first and then copy from Table Latch into Program Memory

Copying Data from Program Memory to Table Latch



Using TBLRD*:
Address: 01251H = 76
TABLAT=76
TBLPTR=01251H

Copying Data from Table Latch to Program Memory



Arithmetic Operations

- The PIC18F MPU performs the following arithmetic operations:
  - Add
  - Subtract
  - Multiply
  - Complement (1s and 2s)
  - Increment
  - Decrement

Addition

Instructions:

- ADDLW 8-bit
- ADDWF F, d, a
- ADDWFC F, d, a

1. Impacts all the flags
2. Result can be saved in W if d=0
3. ADDWFC is used for numbers > 8-bit

Examples

- ADDLW 0xA2
  Add A2H to WREG
- ADDWF REG1, 0
  Add WREG to REG1 and save the sum in W
- ADDWF REG1, 1
  Add WREG to REG1 and save the sum in REG1
- ADDWFC REG2, 0
  Add WREG, REG2 and Carry from previous operation and save the sum in WREG

ADDWF Example

WREG 0 1 0 0 1 1 1 1 (4FH)

+

REG20 0 1 0 0 1 0 0 0 (48H)

Carry 1 1

---

1 0 0 1 0 1 1 1 (97H) = 151d

N=1, OV=1, Z=0, DC=1, C=0

ADDWF Example

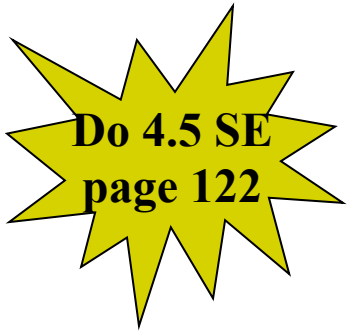
Assume we want to add 0x7292 and 0x1F91

REG10=91, REG11=1F

REG20=92, REG21=72

REG10+REG20=22 & C=1 **(291d)**

N=1, OV=1, Z=0, DC=1, C=1



ADDWFC=REG11+REG21+C

→0x72+ 0x1F+1= 0x92 (1 0 0 1 0 0 1 0)

N=?, OV=?, Z=?, DC=?, C=?

Subtraction

Instructions:

- SUBLW 8-bit

L-WREG

- SUBWF F, d, a

F-WREG

- SUBFWB F, d, a
- SUBWFB F, d, a

If the number is larger than 8 bit!

Examples

- SUBLW 0xA2

  Subtract WREG from A2H
- SUBWF REG1, 0

  Subtract WREG from REG1 and save the result in W
- SUBWF REG1, 1

  Subtract WREG from REG1 and save the result in REG1
- SUBFWB REG2, 0

  Subtract REG2, and Borrow from WREG and save the result in W
- SUBWFB REG2, 0

  Subtract WREG, and Borrow from REG2 and save the result in W

Example

WREG	0 1 1 1 1 1 1 1	(7FH - Subtrahend)
2's Com. of 7FH	1 0 0 0 0 0 0 1	(81H - 2's Complement of 7FH)
	+	
	0 0 1 0 1 0 0 0	(28H - Minuend)
WREG	1 0 1 0 1 0 0 1	(A9H)

Assume [W] = 7F and [REG5]=28

SUBWF REG5

Means:28-7F→W

(Adding values larger than 8 bits)

```
    2  9  F  2
+   3  5  8  7
Carry  1
-------------
    5  F  7  9
```

REG10=F2
REG11=29
REG12=87
REG13=35

Increment, Decrement, and Complement Operations

Instructions:

- INCF F, d, a
- DECF F, d, a
- COMF F, d, a
- NEGF F, a

Examples:

- INCF REG1,1

  Increment REG1 and save result in REG1
- DECF REG2, 0

  Decrement REG2 and save result in W
- COMF REG3,0

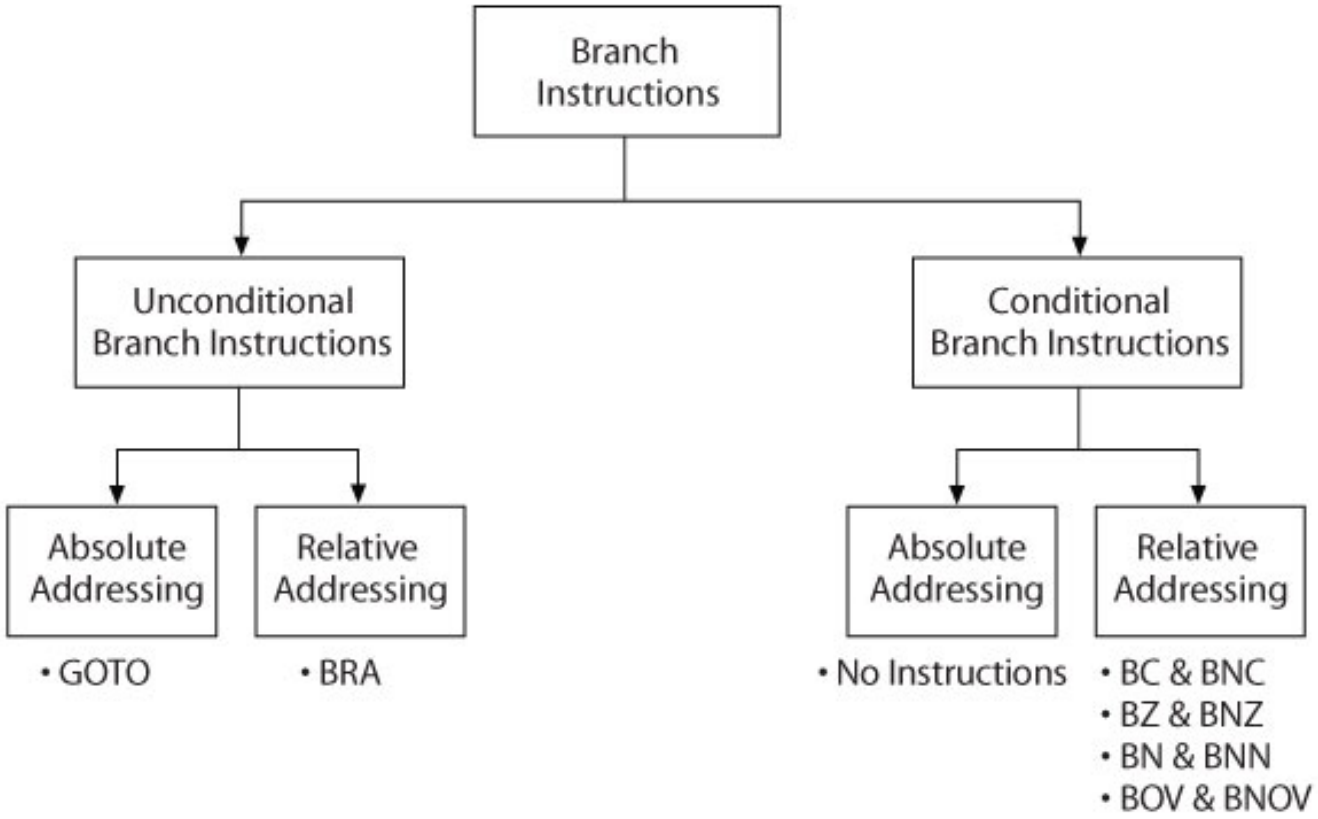
  Complement REG3 and save result in W
- NEGF REG1, 0

  Take 2s complement of REG1

Redirection of Program Execution (Branch and Skip Operations)

- The PIC18F MPU includes three groups of instructions that change the direction of execution:
  - Branch
  - Skip
  - Call (discussed in Chapter 7)

Branch Instructions in PIC18 Family



Unconditional Branch Instructions

Instructions:

- BRA n (label)
  - Unconditional relative branch within ± 512 words
- GOTO k (label)
  - Unconditional absolute branch anywhere in 2 MB memory locations

Conditional Relative Branch Instructions

- BC n: Branch if Carry
- BNC n: Branch if No Carry
- BZ n: Branch if Zero
- BNZ n: Branch if No Zero
- BN n: Branch if Negative
- BNN n: Branch if No Negative
- BOV n: Branch if Overflow
- BNOV n: Branch if No Overflow
- (n is label – represents 8-bit signed number in words)

n is the address the PC should be set to upon executing the command!

For Example:

```
BC    0x020 ; goto address 20
BZ    LOOP; goto loop
```

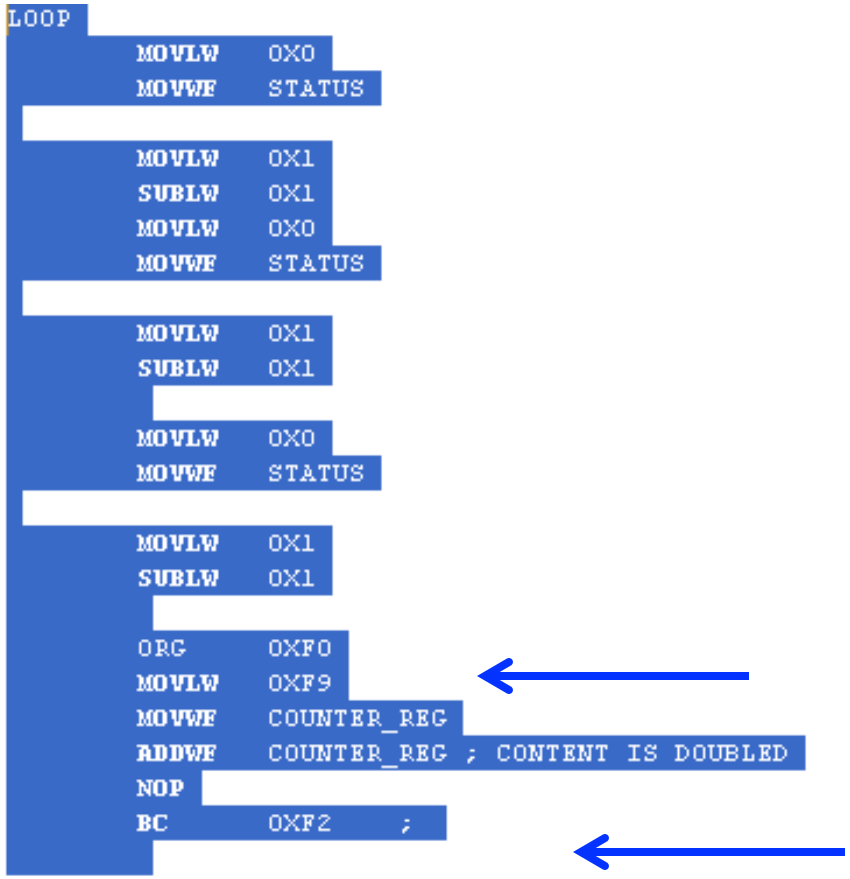
Examples of Relative Conditional Branch Instructions

Correction to the the TEXT:

BC 0x6

This means:
If carry flag is set then jump to 0x06

The n value is where the jump goes to (what the next PC value should be)



Examples of Relative Conditional Branch Instructions

```
LOOP
        MOVLW   0X0
        MOVWF   STATUS

        MOVLW   0X1
        SUBLW   0X1
        MOVLW   0X0
        MOVWF   STATUS

        MOVLW   0X1
        SUBLW   0X1

        MOVLW   0X0
        MOVWF   STATUS

        MOVLW   0X1
        SUBLW   0X1

        ORG     0XF0
        MOVLW   0XF9
        MOVWF   COUNTER_REG
        ADDWF   COUNTER_REG ; CONTENT IS DOUBLE
        NOP
        BC      0XF2    ;
```

```
17    0020    0E00    MOVLW 0
18    0022    6ED8    MOVWF 0xfd8, ACCESS
19    0024    0E01    MOVLW 0x1
20    0026    0801    SUBLW 0x1
21    0028    0E00    MOVLW 0
22    002A    6ED8    MOVWF 0xfd8, ACCESS
23    002C    0E01    MOVLW 0x1
24    002E    0801    SUBLW 0x1
25    0030    0E00    MOVLW 0
26    0032    6ED8    MOVWF 0xfd8, ACCESS
27    0034    0E01    MOVLW 0x1
28    0036    0801    SUBLW 0x1
29    0038    FFFF    NOP
```

```
120   00EE    FFFF    NOP
121   00F0    0EF9    MOVLW 0xf9
122   00F2    6E01    MOVWF 0x1, ACCESS
123   00F4    2601    ADDWF 0x1, F, ACCESS
124   00F6    0000    NOP
125   00F8    E2FC    BC 0xf2
126   00FA    0003    SLEEP
```

BC n refers to the next PC if the condition is met!

Arithmetic Instructions with Skip

- Compare File (Data) Register with W:
  - CPFSEQ F, a ;Skip next instruction if F = W
  - CPFSGT F, a ;Skip next instruction if F > W
  - CPFSLT F, a ;Skip next instruction if F > W
- Increment File (Data) Register:
  - INCFSZ F, d, a ;Skip next instruction if F = 0
  - INFSNZ F, d, a ; Skip next instruction if F ≠ 0
- Decrement File (Data) Register:
  - DECFSZ F, d, a ; Skip next instruction if F = 0
  - DCFSNZ F, d, a ; Skip next instruction if F ≠ 0

Creating some Delay:

Example : When W=0, REG11=1

```
COUNT1   EQU     D'5'
REG10    EQU     0x10
REG11    EQU     0x11

     ORG     0x20
     MOVLW   0x05
     MOVWF   REG10
     DECFSZ  REG10   Skip next instruction if F = 0
     BRA     0x24    Go to location 0x24
     SETF    REG11
     NOP
     NOP
     NOP
     NOP
     NOP
     NOP
     end
```

What is the delay?

Program Memory

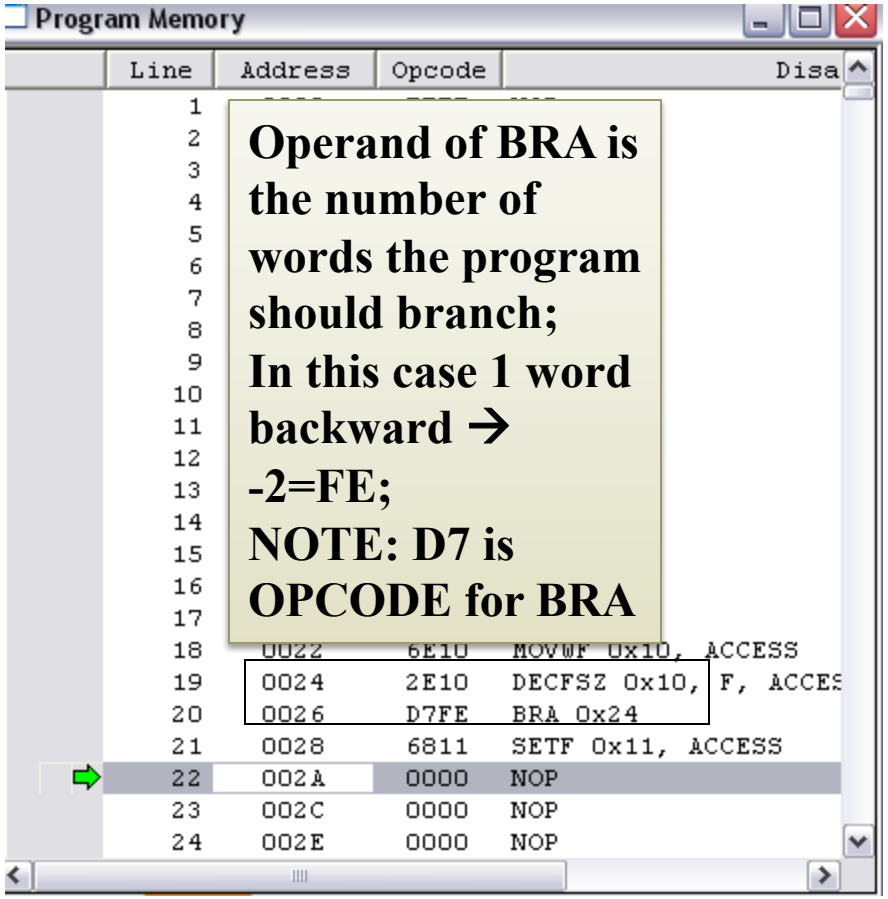
	Line	Address	Opcode	Disa
	1	0000	FFFF	NOP
	2	0002	FFFF	NOP
	3	0004	FFFF	NOP
	4	0006	FFFF	NOP
	5	0008	FFFF	NOP
	6	000A	FFFF	NOP
	7	000C	FFFF	NOP
	8	000E	FFFF	NOP
	9	0010	FFFF	NOP
	10	0012	FFFF	NOP
	11	0014	FFFF	NOP
	12	0016	FFFF	NOP
	13	0018	FFFF	NOP
	14	001A	FFFF	NOP
	15	001C	FFFF	NOP
	16	001E	FFFF	NOP
	17	0020	0E05	MOVLW 0x5
	18	0022	6E10	MOVWF 0x10, ACCESS
	19	0024	2E10	DECFSZ 0x10, F, ACCES
	20	0026	D7FE	BRA 0x24
	21	0028	6811	SETF 0x11, ACCESS
➡	22	002A	0000	NOP
	23	002C	0000	NOP
	24	002E	0000	NOP

Creating some Delay:

Example : When W=0, REG11=1

```
COUNT1   EQU     D'5'
REG10    EQU     0x10
REG11    EQU     0x11

     ORG     0x20
     MOVLW   0x05
     MOVWF   REG10
     DECFSZ  REG10   Skip next instruction if F = 0
     BRA     0x24    Go to location 0x24
     SETF    REG11
     NOP
     NOP
     NOP
     NOP
     NOP
     NOP
     NOP
     end
```



Branch (BRA) Examples

```
17    0020    0E05    MOVLW 0x5
18    0022    6E01    MOVWF 0x1, ACCESS
19    0024    2E01    DECFSZ 0x1, F, ACCESS
20    0026    D7FD    BRA 0x22
21    0028    0003    SLEEP
```

```
;-----  CASE 1; USING UNCONDITIONAL BRANCH
       MOVLW   0X05
LOOP   MOVWF   COUNTER
       DECFSZ  COUNTER
       BRA     LOOP
```

FD = -3; Going back 3 words
Starting with PC=0x28 going to PC=22
That is (28-26) / 2 = 3

```
17    0020    0E05    MOVLW 0x5
18    0022    6E01    MOVWF 0x1, ACCESS
19    0024    0000    NOP
20    0026    0000    NOP
21    0028    D7FB    BRA 0x20
22    002A    0003    SLEEP
```

```
;-----  CASE 2; USING UNCONDITIONAL BRANCH
LOOP   MOVLW   0X05
       MOVWF   COUNTER
       NOP
       NOP
       BRA     LOOP
```

PC+2 + 2*n
28+2 + 2(-5)=2A-A=20

FB = -5; Going back 5 words
Starting with PC=0x2A going to PC=20; before the branch is executed the PC=2A
That is (2A-20) / 2 = 5

Branch Example

```
LOOP
        MOVLW   0X0
        MOVWF   STATUS

        MOVLW   0X1
        SUBLW   0X1
        BZ      0X82    ;CURRENT PC+2=2A
                        ;THIS IS GOTO 0X82--> OPCODE: E0 2C; JUMPING TO 0X0850!
                        ;ERROR MESSAGE (THE PROGRAM DOES COMPILE)
                        ;CORE-W0016: Halted due to PC incrementing over the
                        ;Maximum PC address and wrapping back to Zero

        BZ      0X0E    ;CURRENT PC+2=2A -->OPCODE: E0 F2

        MOVLW   0X89
        MOVWF   COUNTER_REG
        ADDWF   COUNTER_REG ; CONTENT IS DOUBLED
        NOP
        BC      0X20    ; CAN BE "LOOP" OR "0X20"-->OPCODE: E2 F9
```

BAD JUMP!

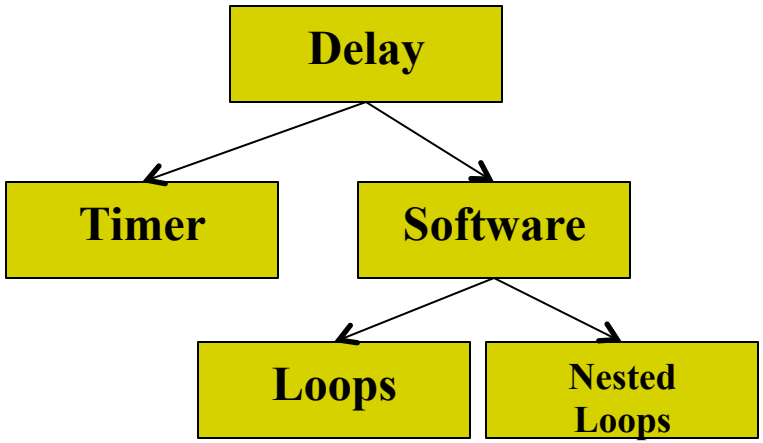
GOOD JUMP!

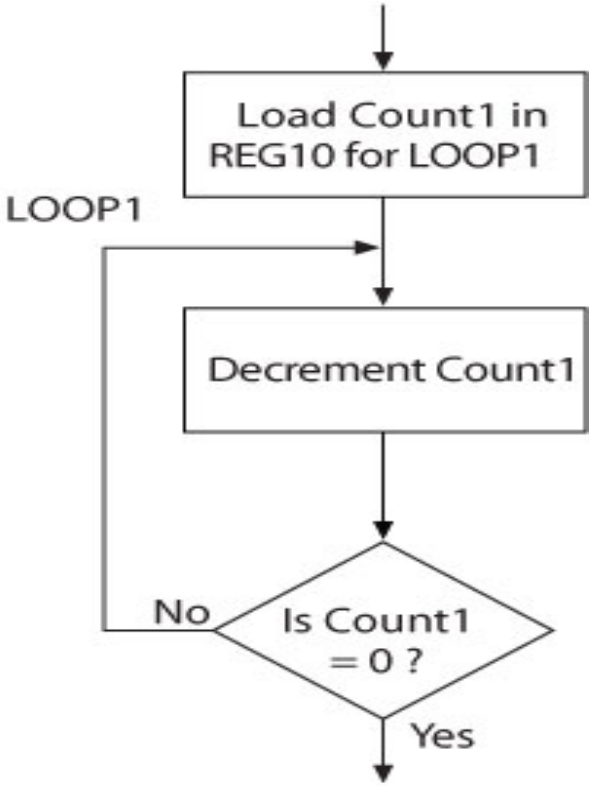
Note that is the program starts at 0x20
And we use BZ 0x2 → the jump will to PC=20

GOOD JUMP!

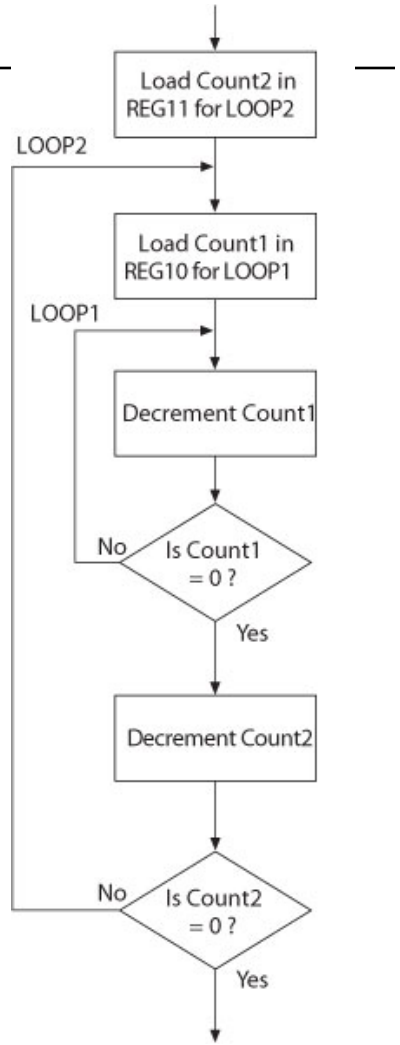
```
17    0020    0E00    MOVLW 0
18    0022    6ED8    MOVWF 0xfd8, ACCESS
19    0024    0E89    MOVLW 0x89
20    0026    6E01    MOVWF 0x1, ACCESS
21    0028    2601    ADDWF 0x1, F, ACCESS
22    002A    0000    NOP
23    002C    E2F9    BC 0x20
```

Flowchart for Delay Loop





Flowchart for Delay Using Nested Loop



Nested Loop Delay Analysis

```
---------------------------------------------
PIC ASSEMBLER LISTING
Line    Address Opcode  Instruction
---------------------------------------------
0001    000000          ;Line removed by MPASMWIN preprocessor:         Title "Ex5-12 Delay Using Nested Loop"
0002    000000          ;Line removed by MPASMWIN preprocessor:         List p=18F452,
0003    000000          ;Line removed by MPASMWIN preprocessor:         #include <p18F452.inc>              ;This is a header file for 18F452
0004    000000
0005    000000          COUNT1              EQU                         D'250'
0006    000000          REG10               EQU                         0x10
0007    000000          REG11               EQU                         0x11
0008    000000
0009    000000
0010    000000  6A11                        CLRF            REG11       ;Set up REG11= COUNT2 =0 for 256 execution
0011    000002  0EFA    LOOP2:              MOVLW           COUNT1      ;Load decimal count in W
0012    000004  6E10                        MOVWF           REG10       ;Set up REG10 as a counter
0013    000006  0610    LOOP1:              DECF            REG10,1     ;Decrement REG10 - 1W/1C/4CLK
0014    000008  E1FE                        BNZ                         LOOP1           ;Go back to LOOP1 if REG 10 =/ 0
0015    00000A  0611                        DECF            REG11,1     ;Decrement REG11
0016    00000C  E1FA                        BNZ                         LOOP2           ;Go back to load 250 in REG10 and start LOOP1
again
0017    00000E                              END
---------------------------------------------
Number of errors = 0
```

Delay Analysis

DECF→1 Cycle; 4 clock periods
BNZ→ 2 Cycles; 8 clock period (w/branch)
→Iclk=12 clock periods (instruction clock cycles)
→Tc=1/40MHz=25ns
→T(one loop)=(1/F) * Iclk = 12 . 25ns=300 ns
→Total Delay(N loop)=Count*T=75 us
→Total Execution time: TL=Iclk*Tc*Nloop

```
---------------------------------------------
PIC ASSEMBLER LISTING
Line    Address Opcode  Instruction
---------------------------------------------
0001    000000          ;Line removed by MPASMWIN preprocessor:     Title "Ex5-12 Delay Using Nested Loop"
0002    000000          ;Line removed by MPASMWIN preprocessor:     List p=18F452,
0003    000000          ;Line removed by MPASMWIN preprocessor:     #include <p18F452.inc>          ;This is a header file for 18F452
0004    000000
0005    000000          COUNT1          EQU                         D'250'
0006    000000          REG10           EQU                         0x10
0007    000000          REG11           EQU                         0x11
0008    000000
0009    000000
0010    000000  6A11                    CLRF        REG11           ;Set up REG11= COUNT2 =0 for 256 execution
0011    000002  0EFA    LOOP2:          MOVLW       COUNT1          ;Load decimal count in W
0012    000004  6E10                    MOVWF       REG10           ;Set up REG10 as a counter
0013    000006  0610    LOOP1:          DECF        REG10,1         ;Decrement REG10 - 1W/1C/4CLK
0014    000008  E1FE                    BNZ                         LOOP1           ;Go back to LOOP1 if REG 10 =/ 0
0015    00000A  0611                    DECF        REG11,1         ;Decrement REG11
0016    00000C  E1FA                    BNZ                         LOOP2           ;Go back to load 250 in REG10 and start LOOP1
again
0017    00000E                          END
---------------------------------------------
Number of errors = 0
```

Remember: 1W/1C/4Clk → 1 Word instruction with one clock cycle with 4 clk periods

Delay Analysis

→In reality, however:
Last loop branches, hence only 8clock cycle
→Total Actual Execution time:
TL=Iclk*Tc*(Nloop-1) + 8*Tc

```
---------------------------------------------
PIC ASSEMBLER LISTING
Line    Address Opcode  Instruction
---------------------------------------------
0001    000000          ;Line removed by MPASMWIN preprocessor:     Title "Ex5-12 Delay Using Nested Loop"
0002    000000          ;Line removed by MPASMWIN preprocessor:     List p=18F452,
0003    000000          ;Line removed by MPASMWIN preprocessor:     #include <p18F452.inc>          ;This is a header file for 18F452
0004    000000
0005    000000          COUNT1          EQU                         D'250'
0006    000000          REG10           EQU                         0x10
0007    000000          REG11           EQU                         0x11
0008    000000
0009    000000
0010    000000  6A11                    CLRF        REG11           ;Set up REG11= COUNT2 =0 for 256 execution
0011    000002  0EFA    LOOP2:          MOVLW       COUNT1          ;Load decimal count in W
0012    000004  6E10                    MOVWF       REG10           ;Set up REG10 as a counter
0013    000006  0610    LOOP1:          DECF        REG10,1         ;Decrement REG10 - 1W/1C/4CLK
0014    000008  E1FE                    BNZ                         LOOP1           ;Go back to LOOP1 if REG 10 =/ 0
0015    00000A  0611                    DECF        REG11,1         ;Decrement REG11
0016    00000C  E1FA                    BNZ                         LOOP2           ;Go back to load 250 in REG10 and start LOOP1
again
0017    00000E                          END
---------------------------------------------
Number of errors = 0
```

Remember: 1W/1C/4Clk → 1 Word instruction with one clock cycle with 4 clk periods

Nested Loop Delay Analysis

Second Loop: (1/F=25 nsec)
4 Instructions: 4+4+4+8 clock periods=20 Iclk
T(loop2)=20*1/F=500 ns
T(loop1+loop2)=75+0.5=75.5 us
Total Delay=256 * 75.5=19 .328 msec
(ignoring the last cycle in each case)

```
---------------------------------------------
PIC ASSEMBLER LISTING
Line    Address Opcode  Instruction
---------------------------------------------
0001    000000          ;Line removed by MPASMWIN preprocessor:     Title "Ex5-12 Delay Using Nested Loop"
0002    000000          ;Line removed by MPASMWIN preprocessor:     List p=18F452,
0003    000000          ;Line removed by MPASMWIN preprocessor:     #include <p18F452.inc>          ;This is a header file for 18F452
0004    000000
0005    000000          COUNT1          EQU                         D'250'
0006    000000          REG10           EQU                         0x10
0007    000000          REG11           EQU                         0x11
0008    000000
0009    000000
0010    000000  6A11                    CLRF        REG11           ;Set up REG11= COUNT2 =0 for 256 execution
0011    000002  0EFA    LOOP2:          MOVLW       COUNT1          ;Load decimal count in W
0012    000004  6E10                    MOVWF       REG10           ;Set up REG10 as a counter
001                                                                 ment REG10 - 1W/1C/4CLK
001                                                                 ;Go back to LOOP1 if REG 10 =/ 0
0015    00000A  0611                    DECF        REG11,1         ;Decrement REG11
0016    00000C  E1FA                    BNZ         LOOP2           ;Go back to load 250 in REG10 and start LOOP1
again
0017    00000E                          END
---------------------------------------------
Number of errors = 0
```

Delay of about 75 usec

Calculations for Generating Waveforms

Write a program to generate a square wave of 10KHz freq. by turning Bit0 of PORT C ON & OFF using a 40 MHz clk (25 nsec)

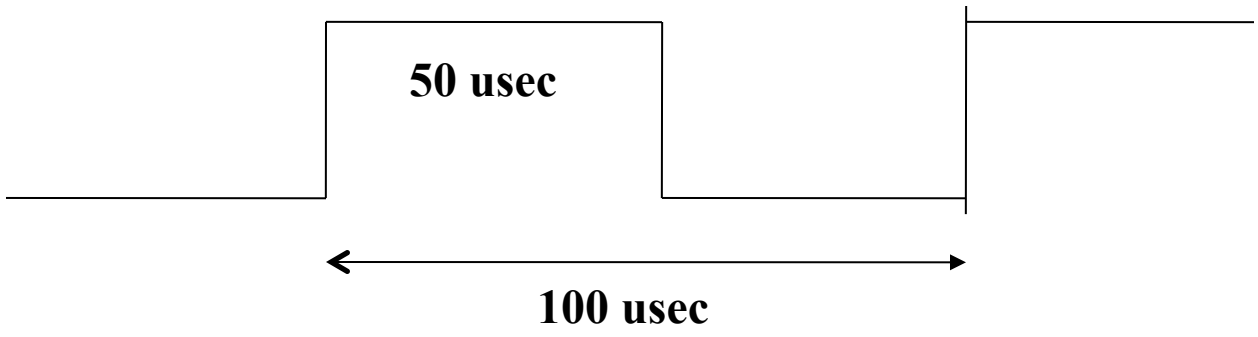
```
LOOP1:   DECF    REG10,1   ;Decrement REG10 - 1W/1C/4CLK
         BNZ     LOOP1     ;Go back to LOOP1 if REG 10 =/ 0
```

- To generate 50 µS delay, count:

$$50\ \mu S = 12 \times 25\ nS \times Count$$

$$\text{Therefore, Count} = \frac{50 \times 10^{-6}}{12 \times 25 \times 10^{-9}} \cong 166$$

Number of times the program should loop



Calculations for Generating Waveforms

Write a program to generate square wave of 10KHz freq. by turning Bit0 of PORT C using a 40 MHz clk (25 nsec)

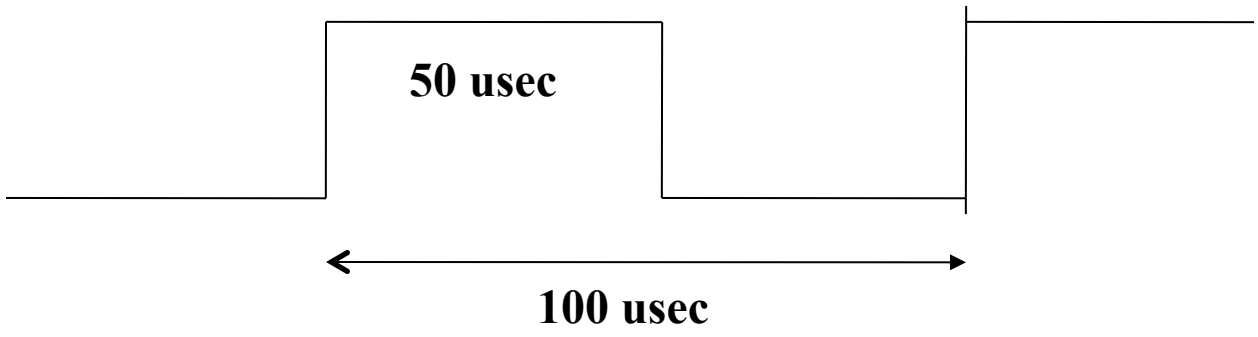
```
        BSF     PORTB,0
LOOP1:  DECF    REG10,1     ;Decrement REG10 - 1W/1C/4CLK
        BNZ     LOOP1       ;Go back to LOOP1 if REG 10 =/ 0
        BCF     PORTB,0
```

- To generate 50 µS delay, count:

$$50\ \mu S = 12 \times 25\ nS \times Count$$

$$\text{Therefore, } Count = \frac{50 \times 10^{-6}}{12 \times 25 \times 10^{-9}} \cong 166$$

Number of times the program should loop



Generating Waveforms

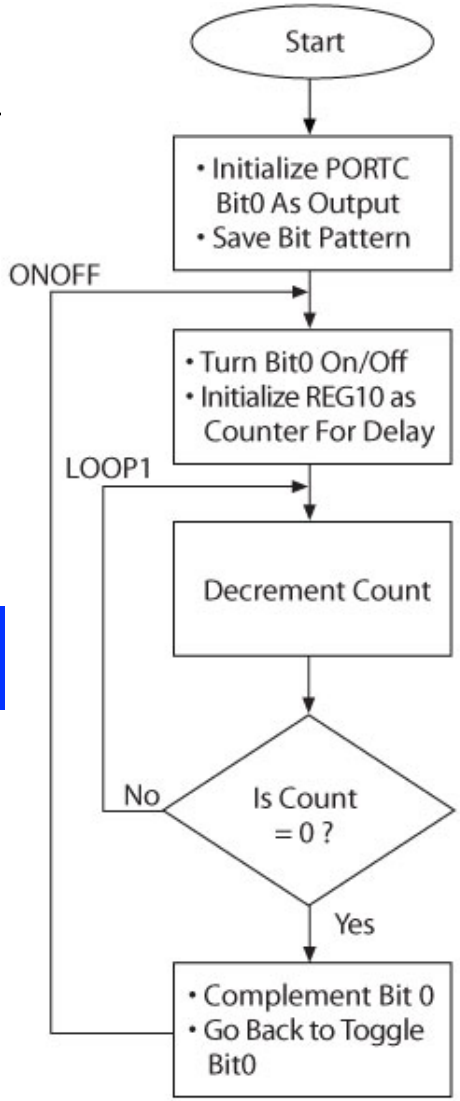
```
REG1    EQU             0x01            ;Address of Data Register1
REG10   EQU             0x10            ;Address of Data Register10

                ORG     0x20
START:          MOVLW   B'11111110'     ;Load number to set up  BIT0 as an output
                MOVWF   TRISC           ;Initialize BIT0 as an output pin
                MOVWF   REG1            ;Save Bit pattern in REG1

ONOFF:          MOVFF   REG1,PORTC      ;(8 Clk)-Turn on/off BIT0
                MOVLW   D'166'          ;(4 Clk) -Load decimal count in W
                MOVWF   REG10           ;(4 Clk) Set up REG10 as a counter

LOOP1:          DECF    REG10, 1        ;(4 Clk) - Decrement REG10
                BNZ     LOOP1           ;(8/4 Clk)-Go back to LOOP1 if REG 10 not equal  0

                COMF    REG1,1          ;(4 Clk)-Complement Bit Pattern
                BRA     ONOFF           ;(8 Clk)-Go back to change LED
                END
```



Delay Recalculations to Account for Outside Instructions – Single Loop

$$\text{On/Off Time} = 50\,\mu S = (\text{Loop Instruction Cycles } I_{CLK} \times \text{CLK} \times \text{Count } N_{10}) + (1 \times \text{Cycles Outside Loop} \times \text{CLK})$$

$$= (12 \times 25\ nS \times \text{Count}) + (1 \times 28 \times 25\ nS)$$

$$= (300\ nS \times \text{Count}) + 700\ nSx$$

$$\text{Count} = \frac{50\,\mu S - 0.7\,\mu S}{300\ nS} = \frac{49.3 \times 10^{-6}}{300 \times 10^{-9}} \approx 164$$

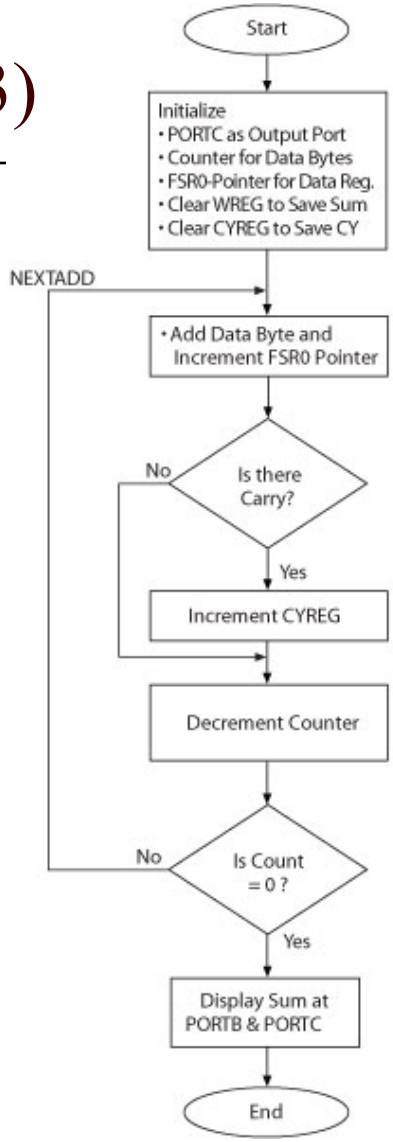
Adding a Block of Data (P.163)

- Add five data bytes stored in data registers (0x10-0x14). The sum should be displayed at PORTB and PORTC.

- Use FSR0 – Pointer
- Use a carry register: CYREG
- Assume values are given:

File Registers

Address	00	01	02	03	04	05	06	07	08	09	0A
0010	F6	67	7F	A9	72	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00
0050	00	00	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00



Adding a block of numbers

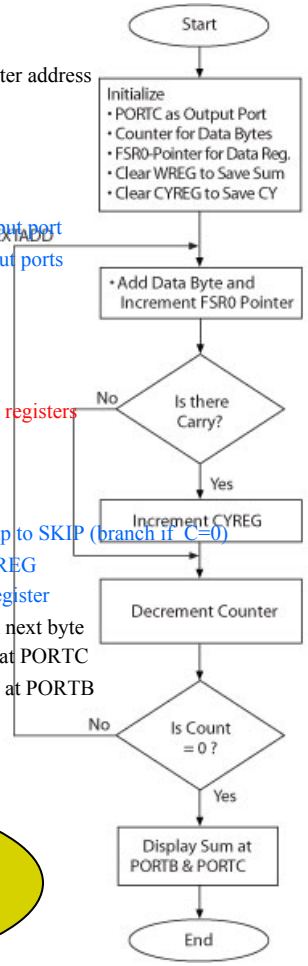
Initialize

Initialize Ports

Set the Ptr

Move the Ptr

```
0005  000000  BUFFER     EQU     0x10    ;Define the beginning data register address
0006  000000  COUNTER    EQU     0X01    ;Set up register 01 as a counter
0007  000000  CYREG      EQU 0X02
0008  000000
0009  000000              ORG     0x20    ;Begin assembly at 0020H
0010  000020  0E00  START:  MOVLW   0x00    ;Byte to initialize port as an output port
0011  000022  6E93         MOVWF   TRISB   Initialize Ports B &C as an output ports
0012  000024  6E94         MOVWF   TRISC
0013  000026  0E05         MOVLW   0X05    ;Count for five bytes
0014  000028  6E01         MOVWF   COUNTER ;Set up counter
      00002A  6A02         CLRF    CYREG   ;Clear carry register
      00002C  EE00         LFSR    FSR0,BUFFER ;Set up FSR0 as pointer for data registers
0016  00002E  F010
0017  000030  0E00         MOVLW   0x00    ;Clear W register to save sum
      000032  24EE  NEXTADD: ADDWF  POSTINC0,W ;Add byte and increment FSR0
      000034  E301         BNC     SKIP    ;Check for carry-if no carry jump to SKIP (branch if  C=0)
0020  000036  2A02         INCF    CYREG   ;If there is carry, increment CYREG
0021  000038  0601  SKIP:  DECF    COUNTER,1,0 ;Next count and save count in register
0022  00003A  E1FB         BNZ     NEXTADD ;If COUNT =/ 0, go back to add next byte
0023  00003C  6E82         MOVWF   PORTC   ;Display low-order byte of sum at PORTC
0024  00003E  C002         MOVFF   CYREG, PORTB ;Display high-order byte of sum at PORTB
0024  000040  FF81
0025  000042  0003         SLEEP
0026  000044              END
--------------------------------------------
Number of errors = 0
```

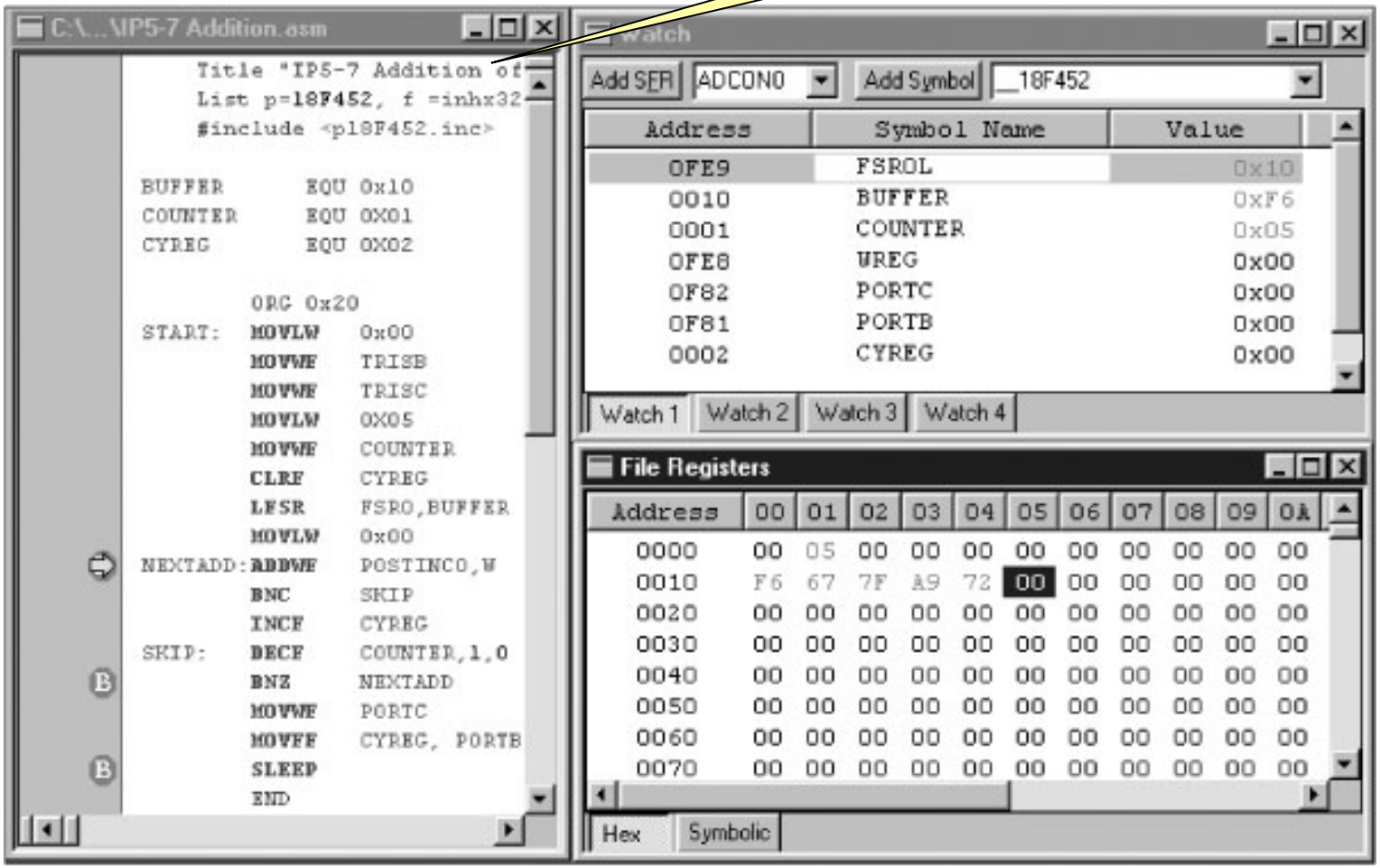


How do you modify to make the BUFFER size variable?

Lab:
Assume the total is less than 2FFFF and we can only display two byes
Using PORT B and C
Modify the program to calculate the sum of registers 10-19
If the sum is zero set bit 0 of PORTA
If the sum>50H set bit 1 of PORT A

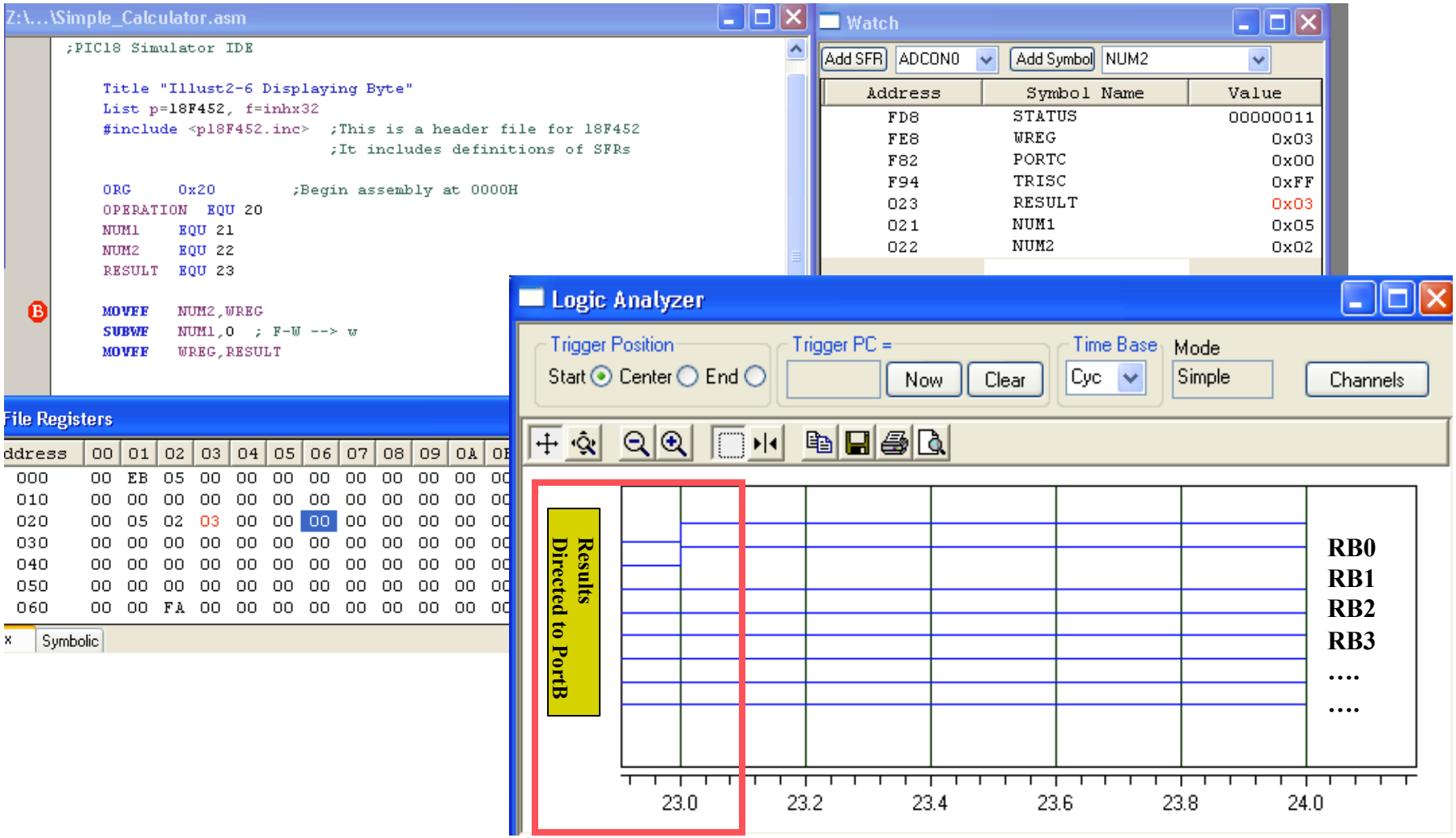
Displaying Breakpoints

Heading and Include files must be included:

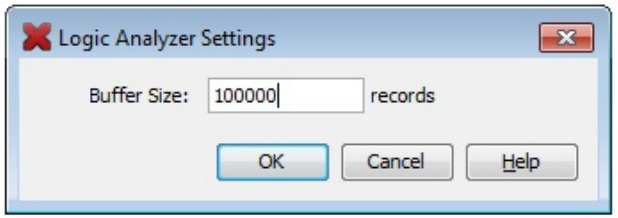


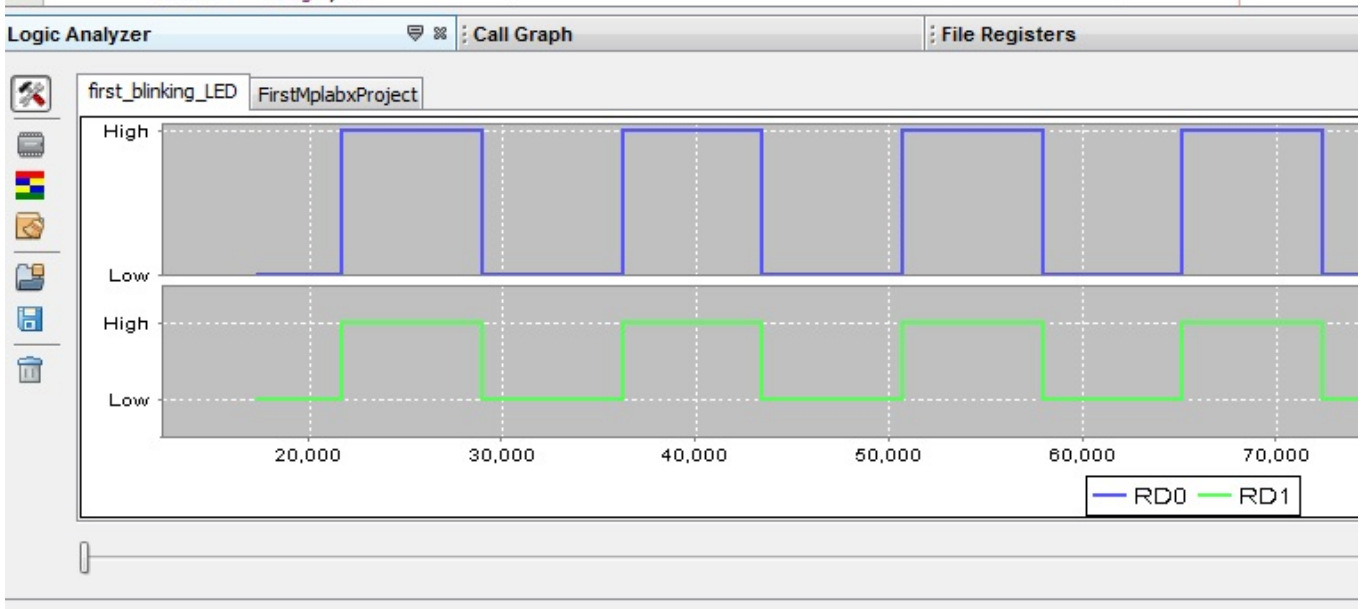
Using the Logic Analyzer

Problem: 5-2→3 going to REG3



```
59    ;//** Register Declaration:
60    reg1     equ 0x01
61    reg10    equ 0x10
62    reg11    equ 0x11
63
64    ;//** M A I N   C O D E  ***********/
65        org 0x20
66    start1
67        movlw   B'11111100'
68        movwf   TRISD
69        movwf   reg1
70
71    onoff
72        movff   reg1,PORTD   ; this  determines the clock speed
73        movlw   Inner_loop
74        movwf   reg10
75        movlw   Outer_loop
76        movwf   reg11
77
78    loop1
79        decf    reg10,1
80        bnz     loop1
81       decf     reg11,1
82        bnz     loop1
83
84        comf    reg1,1
```





Good Programing

```
;//** Constants
Inner_loop  equ D'10'
Outer_loop  equ D'1'

;//** Register Declaration:
reg1    equ 0x01
reg10   equ 0x10
reg11   equ 0x11

;//** M A I N   C O D E  ***********/
    org 0x20
start1
    movlw   B'11111100'
    movwf   TRISD
    movwf   reg1

onoff
    movff   reg1,PORTD   ; this  determines the clock speed
    movlw   Inner_loop
    movwf   reg10
    movlw   Outer_loop
    movwf   reg11
```

Example on page 164 – Hide it