

## 3.2 branching, looping, time delay

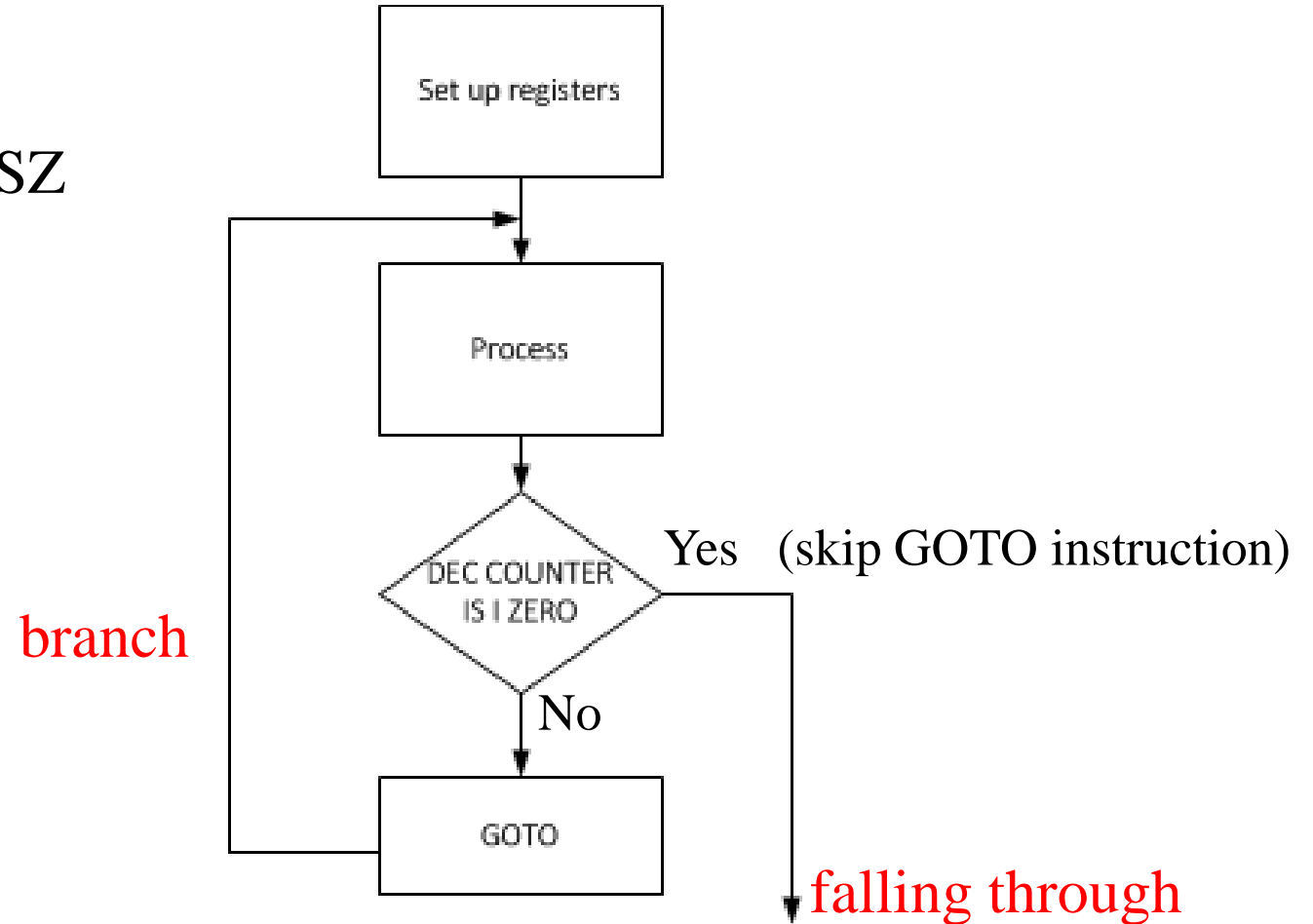
- conditional branch and looping
- unconditional branch
- time delay loop

### 3.2.1 conditional branch and looping

- branch – transfer program control to a different location
- loop – repeat a sequence of instructions a certain number of times
- 2 ways to do looping
  - using **DECFSZ** instruction
  - using conditional **branch** instructions

# DECFSZ instruction

- decrement file register, skip the next instruction if the result is equal to zero  
**DECFSZ** fileReg, d
- GOTO instruction follows DECFSZ



## Example:

Write a program to

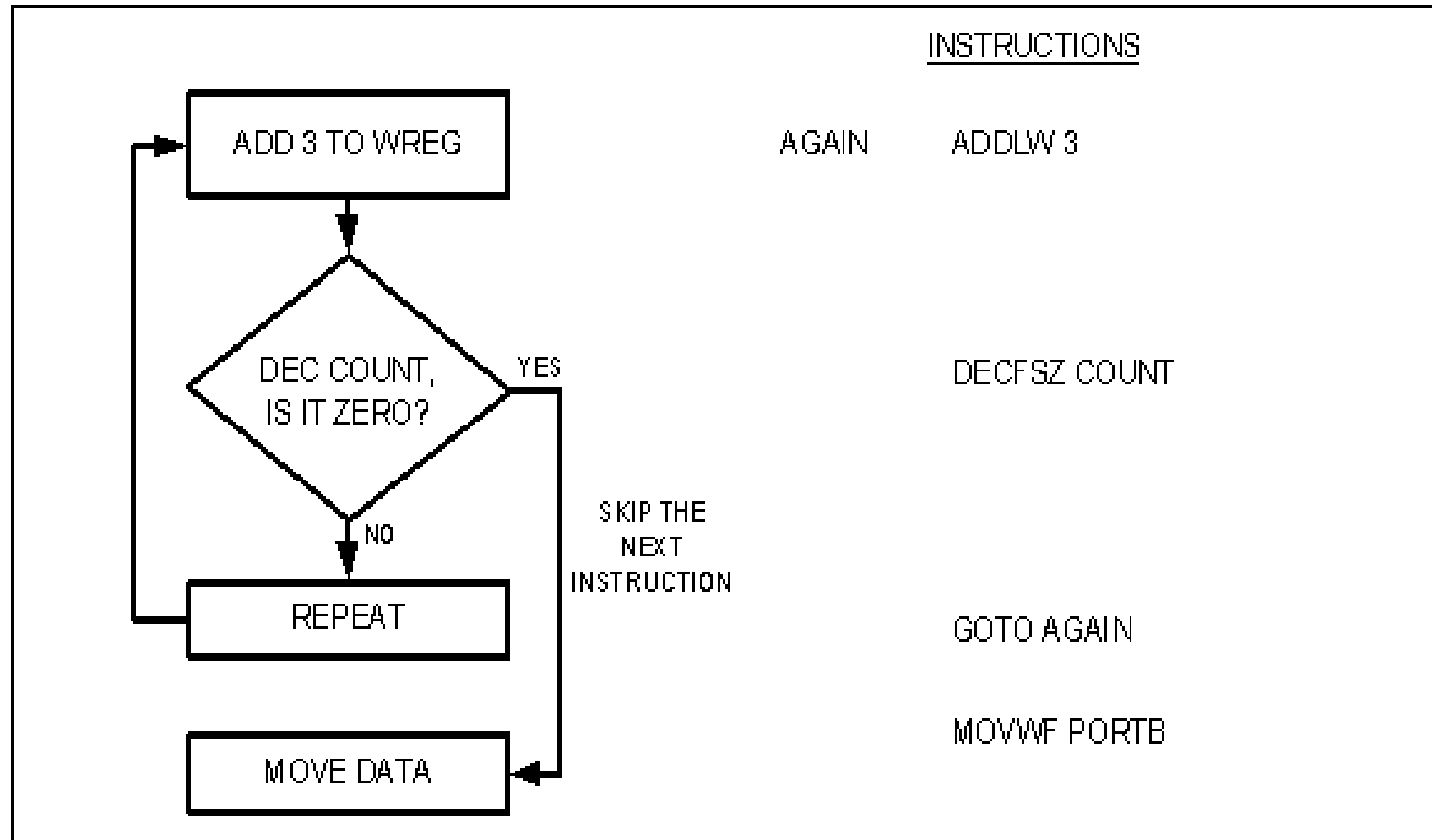
a) Clear WREG

b) Add 3 to WREG ten times and place the summation result in PORT B

COUNT	EQU	0x25	; use loc 25H for counter
	MOVLW	d'10'	; WREG = 10 for counter
	MOVWF	COUNT	; load counter
	MOVLW	0	; clear WREG
AGAIN	ADDLW	3	; add 3 to WREG
	DECFSZ	COUNT, F	; decrement counter, skip if zero
	GOTO	AGAIN	; repeat until counter = 0
	MOVWF	PORTB	; send sum to PORT B



DECFSZ COUNT, W is wrong!



What is the maximum number of times that the loop in the previous example can be repeated?

**Solution:**

Since COUNT holds an 8-bit register, it can hold a maximum of FFH, therefore the loop can be repeated a maximum of 256 times by setting COUNT = 0.

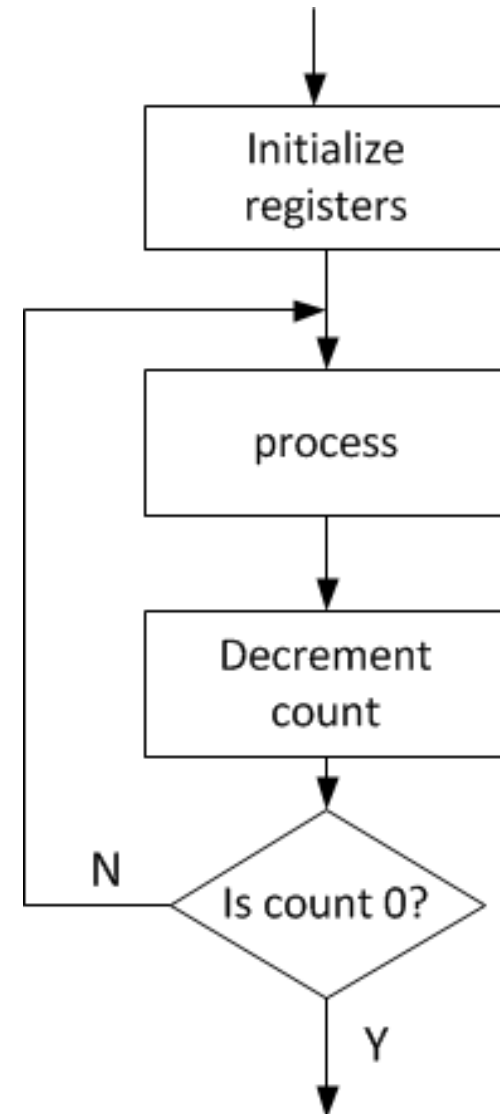
Thus, COUNT = 0H, FFH, FEH, ..., 2, 1, 0 (total 256 times)

# BNZ

- **b**branch if **n**ot **z**ero  
    **BNZ** label
- supported by PIC18 families
- the instruction checks the Z flag

Back

.....  
.....  
**DECF** fileReg, f  
**BNZ** Back



## Example:

Write a program to

a) Clear WREG

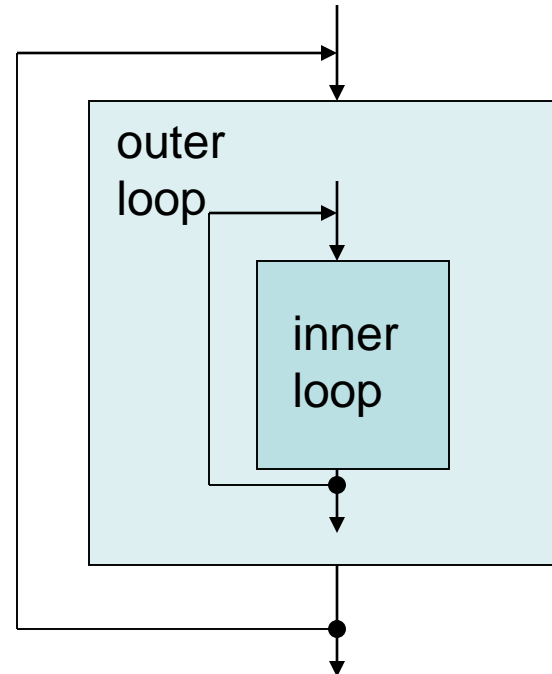
b) Add 3 to WREG ten times and place the result in PORT B

```
COUNT    EQU    0x25                ; use loc 25H for counter
        MOVLW    d'10'              ; WREG = 10 for counter
        MOVWF    COUNT              ; load counter
        MOVLW    0                   ; clear WREG
AGAIN    ADDLW    3                   ; WREG=WREG+3
        DECF     COUNT, F           ; decrement counter
        BNZ      AGAIN              ; repeat until counter = 0
        MOVWF    PORTB              ; send sum to PORT B
```



## Nested loop

- there is limit in the number of times a loop can be repeated
- if we want to repeat an action more times, use a loop inside a loop (nested loop)
- Example:
  - inner loop 200 times
  - outer loop 200 times
  - total  $200 \times 200 = 40,000$  times

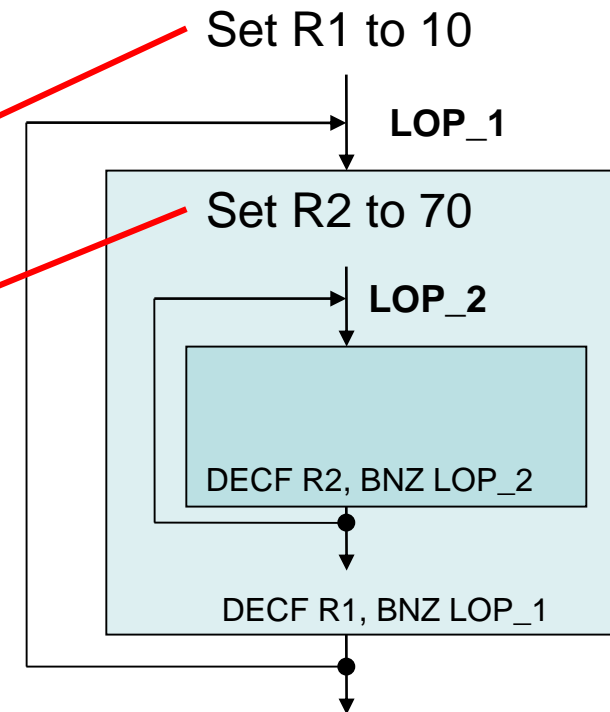


Write a program to

a) Load PORTB with the value 55H

b) Complement PORTB **700** times

```
R1      EQU    0x25      ; loc for counter 1
R2      EQU    0x26      ; loc for counter 2
COUNT_1 EQU    d'10'
COUNT_2 EQU    d'70'
        MOVLW   0x55
        MOVWF   PORTB    PORTB = 55
        MOVLW   COUNT_1
        MOVWF   R1
LOP_1:  MOVLW   COUNT_2
        MOVWF   R2
        COMF    PORTB, F  ; complement PORT B
        DECF    R2, F
        BNZ     LOP_2     ; repeat 70 times
        DECF    R1, F
        BNZ     LOP_1     ; repeat 10 times
```



## Other conditional jumps

- conditional branch/jump instructions are 2-byte
- they require the target address
  - 1-byte address (short branch address)
  - relative address with reference to PC
- longest distance?
- if the condition is true, it jumps to the target address

## Flag bits and decision

BC	k	Branch relative if Carry
BNC	k	Branch relative if Not Carry
BN	k	Branch relative if Negative
BNN	k	Branch relative if Not Negative
BOV	k	Branch relative if Overflow
BNOV	k	Branch relative if Not Overflow
BZ	k	Branch relative if Zero
BNZ	k	Branch relative if Not Zero

k is label (target address)

## BZ

- Jump if previous result is 0 ( $Z = 1$ )
- MOVF will affect the status register

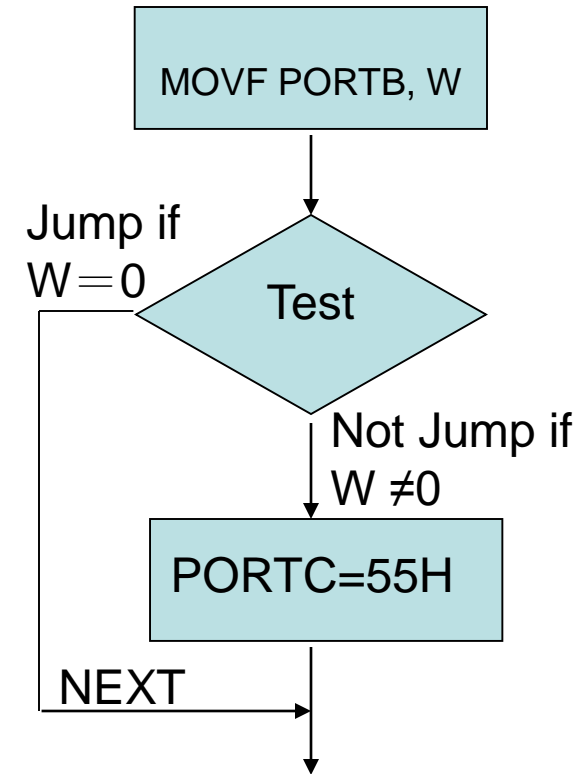
**MOVF PORTB, W**

**BZ NEXT**

**MOVLW 55H**

**MOVWF PORTC**

**NEXT: .....**

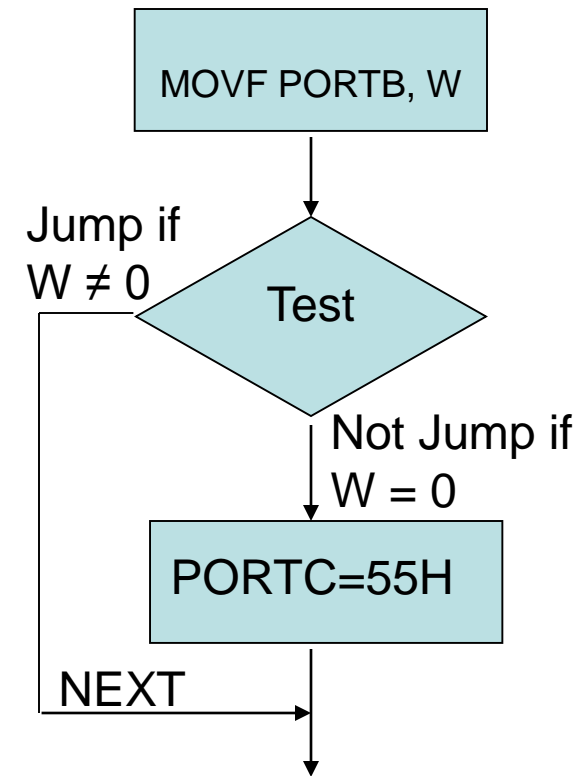


## inverse the condition

- Jump if previous is not zero ( $Z = 0$ )

```
MOVF PORTB, W  
BNZ NEXT  
MOVLW 55H  
MOVWF PORTC
```

```
NEXT: ...
```

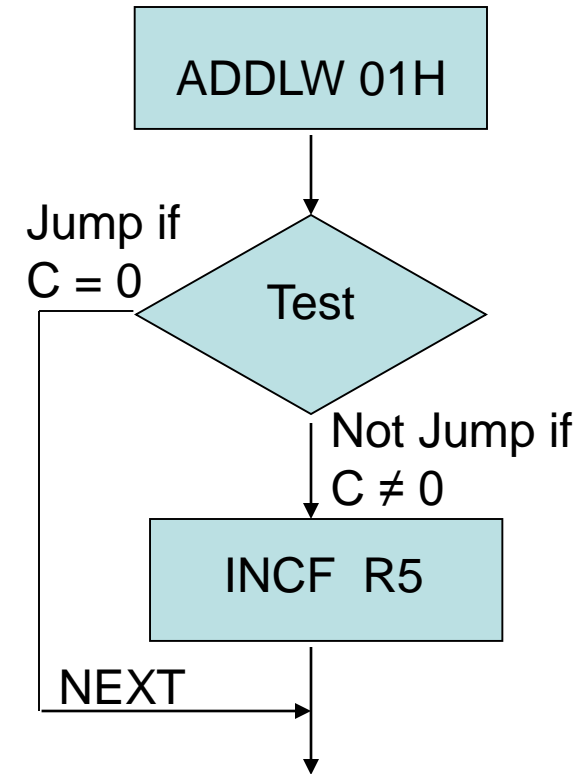


# BNC

- Jump if no carry ( $C = 0$ )

```
MOVLW    FFH
ADDLW    01H
BNC      NEXT
INCF     R5
```

```
NEXT:    ...
```



## Example

Write a program to determine if the loc. 0x30 contains the value 0.  
If so, put 55H in it.

Solution:

```
MYLOC EQU 0x30
        MOVF MYLOC, F ; copy MYLOC to itself ?
        BNZ NEXT      ; branch if MYLOC is not 0
        MOVLW 0x55
        MOVWF MYLOC    ; put 55H to loc. 0x30
NEXT:    .....
```




# Example

Find the sum of the values 79H, F5H, and E2H.

Put the sum in file register loc 5H (low byte) and 6H (high byte).

## Solution:

L_Byte	EQU 0x5	
H_Byte	EQU 0x6	
	ORG 0H	
	MOVLW 0x0	;clear (W=0)
	MOVWF H_Byte,A	;clear H_Byte
	ADDLW 0x79	;W=0+79=79
	BNC N_1	;if C=0,add next number
	INCF H_Byte,F	;if C=1, increment H_Byte
N_1:	ADDLW 0xF5	;W=79+F5=6E and C=1
	BNC N_2	;if C=0,add next number
	INCF H_Byte,F	;if C=1, increment H_Byte
N_2:	ADDLW 0xE2	;W=6E+E2=50 and C=1
	BNC OVER	;jump if C=0
	INCF H_Byte,F	;C=1, increment H_Byte
OVER:	MOVWF L_Byte	;Now H_Byte=2, L_Byte=50H
	END	



## Calculate relative address

- The conditional jump is a jump in which control is transferred to the target location if it meets the required condition.
  - BZ, BNC...
  - The target address cannot be farther than -256 / +254 from the current program counter

How to know the target address is out of range or not?

Ans: Assembler will tell you

How to solve the problem if our conditional jump needs to go to a target address which is farther than -256 / +254 from the program counter.

In condition branch, the instruction is

8 bit Opcode	8 bit signed number (for displacement)
--------------	--

Target address=PC + (2 x 8 bit signed number)

The relative displacement is a common used technique in many processor.

Example:

000000 0E00	00004 MOVLW 0x0
000002 6E06	00005 MOVWF H_Byte
000004 0F79	00006 ADDLW 0x79
000006 E301	00007 BNC N_1
000008 2A06	00008 INCF H_Byte,F
00000A 0FF5	00009 N_1 ADDLW 0xF5
00000C E301	00010 BNC N_2
00000E 2A06	00011 INCF H_Byte,F
000010 0FE2	00012 N_2 ADDLW 0xE2
000012 E301	00013 BNC OVER
000014 2A06	00014 INCF H_Byte,F
000016 6E05	00015 OVER MOVWF L_Byte

Why the machine code of BNC N\_1 is E301?

Ans:

E3 is opcode for BNC

When CPU is executing BNC N\_1, the PC = 000008.

The target address is 00000A. The difference (00000A - 000008) is +2. So, the displacement in instruction is 01.

### Example:

0000	0E00		MOVLW 0x0
0002	6E06		MOVWF H_Byte, A
0004	0F79		ADDLW 0x79
<b>0006</b>	<b>E3XX</b>		<b>BNC N_1</b>
0008	0000		NOP
000A	0000		NOP
000C	2A06		INCF 0x6, F, ACCESS
000E	0FF5	N_1:	ADDLW 0xf5
0010	E301		BNC N_2
0012	2A06		INCF 0x6, F, ACCESS
0014	0FE2	N_2:	ADDLW 0xe2
0016	E301		BNC Over
0018	2A06		INCF 0x6, F, ACCESS
001A	6E05	Over:	MOVWF 0x5,

~~000E-0008=06~~,  $6/2 = 3 \Rightarrow$  E303

Example:

0000	0E00	MOVLW 0x0
0002	6E06	MOVWF H_Byte, A
0004	0F79	ADDLW 0x79
0006	E303	BNC N_1
0008	0000	NOP
000A	0000	NOP
000C	2A06	INCF 0x6, F, ACCESS
000E	0FF5	N_1:ADDLW 0xf5
0010	E301	BNC N_2
0012	2A06	INCF 0x6, F, ACCESS
0014	0FE2	N_2:ADDLW 0xe2
0016	E3XX	BNC N_1
0018	2A06	INCF 0x6, F, ACCESS
001A	6E05	Over: MOVWF 0x5,

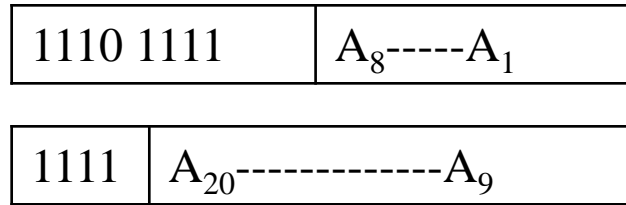
000E-0018=-10,  $-10/2 = -5$ , -5 is signed format is FB=>E3FB

### 3.2.2 unconditional branch/jump

- a jump in which control is transferred unconditionally to the target location
- the target address is directly coded in the instruction
- there are 2 unconditional jumps :
  - **GOTO** (Long Jump)
  - **BRA** (Short Jump)

# GOTO

- a 4-byte instruction



- address is  $A_{20}$ ----- $A_1$ 0
- jump to anywhere in the program memory

# BRA

- a 2-byte instruction
  - The first 5 bits are the opcode
  - The next 11 bits form a signed number displacement, which (need to times 2 first ) is added to the PC to get the target address

5 bit Opcode	11 bit signed number (for displacement)
--------------	---

Target address = PC + (2 x 11 bit signed number)

**Limited range : -2048 to 2046 bytes of the relative address of the current PC value**



### 3.2.3 time delay loop

- You have written a program with time delay loops in tutorial 3.
- How to calculate the actual delay time?
- How to generate a specific time delay?

## Machine Cycle

- to execute an instruction, CPU takes a certain number of clock cycles
- in the PIC18 family, these clock cycles are referred to as *instruction cycle or machine cycle*.
- in PIC18, 4 clock cycles = 1 machine cycle
- different instructions need different number of machine cycle(s)

## Example

The following shows the crystal frequency for three different PIC18 based systems. Find the period of the machine cycle in each case.

(a) 4 MHz    (b) 16 MHz    (C) 20 MHz

### **Solution:**

(a)  $4/4 = 1\text{MHz}$

1 machine cycle =  $1\mu\text{s}$  (microsecond)

(b)  $16/4 = 4\text{ MHz}$

1 machine cycle =  $1/4\text{MHz} = 0.25\ \mu\text{s}$

(c)  $20\text{ MHz}/4 = 5\text{ MHz}$

1 machine cycle =  $1/5\text{ MHz} = 0.2\ \mu\text{s}$

## Example

For a PIC18 system of 4MHz, find how long it takes to execute each of the following instructions.

- (a) MOVLW      (b) DECF      (c) MOVWF      (d) ADDLW  
(e) NOP      (f) GOTO      (g) BNZ

### Solution:

The machine cycle for a system of 4 MHz is 1  $\mu$ s.

The following table shows machine cycles for each instruction.

Instruction	Machine cycles	Time to execute
(a) MOVLW	1	$1 \times 1 \mu\text{s} = 1\mu\text{s}$
(b) DECF	1	$1 \times 1 \mu\text{s} = 1\mu\text{s}$
(c) MOVWF	1	$1 \times 1 \mu\text{s} = 1\mu\text{s}$
(d) ADDLW	1	$1 \times 1 \mu\text{s} = 1\mu\text{s}$
(e ) NOP	1	$1 \times 1 \mu\text{s} = 1\mu\text{s}$
(f) GOTO	2	$2 \times 1 \mu\text{s} = 2\mu\text{s}$
(g) BNZ	1 or 2	1 or 2 $\mu\text{s}$

BNZ takes 2 machine cycles if it jumps, and takes 1 machine cycle when falling through.

## Example

Find the actual delay time in the following program, if the crystal frequency is 4 MHz.

	machine cycle
<b>MYREG EQU 0x08</b>	
<b>DELAY: MOVLW 0xFF</b>	<b>1</b>
<b>MOVWF MYREG</b>	<b>1</b>
<b>AGAIN: NOP</b>	<b>1</b>
<b>NOP</b>	<b>1</b>
<b>DECF MYREG, F</b>	<b>1</b>
<b>BNZ AGAIN</b>	<b>1 or 2</b>
<b>RETURN</b>	<b>1</b>

The actual time is  $1277 \mu\text{s}$ .  
 $1+1+255 \times 5-1+1$

# Example

Find the actual delay time in the following program, if the crystal frequency is 4 MHz.

<b>R2</b>	<b>EQU 0x7</b>	
<b>R3</b>	<b>EQU 0x8</b>	machine cycle
<b>DELAY:</b>		
	<b>MOVLW D'200'</b>	<b>1</b>
	<b>MOVWF R2</b>	<b>1</b>
<b>AGAIN:</b>	<b>MOVLW D'250'</b>	<b>1</b>
	<b>MOVWF R3</b>	<b>1</b>
<b>HERE:</b>	<b>NOP</b>	<b>1</b>
	<b>NOP</b>	<b>1</b>
	<b>DECF R3, F</b>	<b>1</b>
	<b>BNZ HERE</b>	<b>2</b>
	<b>DECF R2, F</b>	<b>1</b>
	<b>BNZ AGAIN</b>	<b>2</b>
	<b>RETURN</b>	<b>2</b>

For HERE loop, the delay is  $5 \times 250 = 1250 \mu\text{s}$ . The AGAIN loop repeats the HERE loop 200 times. Therefore, we have  $200 \times 1250 \mu\text{s} = 250,000 \mu\text{s}$ . (without overhead)

Overhead =  $5 \times 200 - 200 + 1 + 1 + 2 - 1$

The actual time is 250.8 ms.

# Example

Write a PIC18 program to generate 1 second delay. The crystal frequency is 10 MHz.

## Solution:

10 MHZ => 1 machine cycle = 400 ns. '1 second' needs 2,500,000 machine cycles

If we have a basic loop with 5 machine cycles. That means, we may need to execute this basic loop 500,000 times. 500,000 can be decompose as 8x250x250

```
DELAY:      MOVLW    D'8'
             MOVWF    R4
BACK:        MOVLW    D'250'
             MOVWF    R3
AGAIN:       MOVLW    D'250'
             MOVWF    R2
HERE:        NOP
             NOP
             DECF     R2, F
             BNZ      HERE
             DECF     R3, F
             BNZ      AGAIN
             DECF     R4, F
             BNZ      BACK
             RETURN
END
```

## Summary

- ◆ conditional branch
- ◆ unconditional branch
- ◆ time delay loop