Chapter 3
Branch Loop, and IO Port

Sections

3.1 Loop and Jump Instructions
3.2 IO PORT
3.3 Time Delay Generation and Calculation

Sections

3.1 **Loop and Jump Instructions**
3.2 IO PORT
3.3 Time Delay Generation and Calculation

Branch instruction and Looping

- Looping in PIC
- Loop inside loop
- Other conditional jumps
- All conditional branches are short jumps
- Calculating the short branch address
- Unconditional branch instruction

Loop in PIC

- Repeat a sequence of instructions or a certain number of times
- Two ways to do looping

  Using DECFSZ instruction

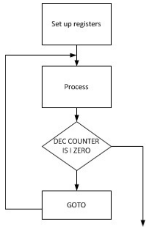
  Using BNZ\BZ instructions

DECFSZ instruction

•Decrement file register, skip the next instruction if the result is equal 0

•DECFSZ fileRef, d

•GOTO instruction follows DECFSZ



DECFSZ instruction

Example:

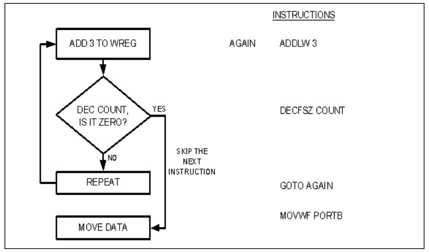
Write a program to

a)Clear WREG

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

```
COUNT    EQU      0x25
         MOVLW    d'10'
         MOVWF    COUNT
         MOVLW    0
AGAIN    ADDLW    3
         DECFSZ   COUNT, F
         GOTO     AGAIN
         MOVWF    PORTB
```


DECFSZ instruction



DECFSZ instruction

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

Solution:

Since COUNT holds is 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.

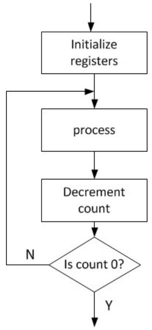
Thus, COUNT＝0H, FFH, FEH, ..., 2, 1, 0（total 256 times)

Using BNZ\BZ

Supported by PIC18 families Early families such as PIC16 and PIC12 doesn't support these instruction

These instructions check the status flag

```
Back  ..................
      ..................
      DECF fileReg, f
      BNZ  Back
```



Using BNZ\BZ

Example:

Write a program to

a)Clear WREG

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

```
COUNT    EQU     0x25
         MOVLW   d'10'
         MOVWF   COUNT
         MOVLW   0
AGAIN    ADDLW   3
         DECF    COUNT, F
         BNZ     AGAIN
         MOVWF   PORTB
```


DECFSZ instruction

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

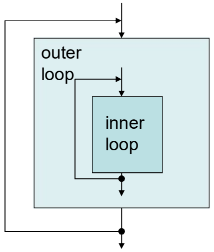
Solution:

Since COUNT holds is 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)

Nested Loop

- A single loop is repeated 256 times in maximum.
- If we want to repeat an action more times than 256, we use a loop inside a loop.
- This is called nested loop.
- For Example:
  - The inner loop is 256
  - The outer loop is 2
  - Total 256*2=512



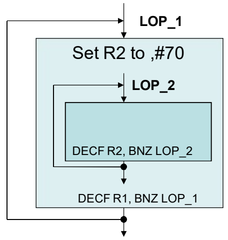
Loop inside a loop

Write a program to

a)Load the PORTB SFR register with the value 55H

b)Complement PORTB **700**times

```
R1          EQU   0x25
R2          EQU   0x26
COUNT_1     EQU     d'10'
COUNT_2     EQU     d'70'
            MOVLW   0x55
            MOVWF   PORTB
            MOVLW   COUNT_1
            MOVWF   R1
LOP_1:      MOVLW   COUNT_2
            MOVWF   R2
LOP_2:      COMF    PORTB, F
            DECF    R2, F
            BNZ     LOP_2
            DECF    R1, F
            BNZ     LOP_1
```



Other Conditional jumps

All of the 10 conditional jumps are 2-byte instructions

•They requires the target address

1 byte address (short branch address)

Relative address

•Recall: MOVF will affect the status Reg.

•In the BZ instruction, the Z flag is checked. If it is high, that is equal 1, it jumps to the target address.

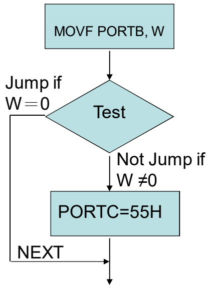
Flags Bits and Decision

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

BZ

- Jump if previous result is 0

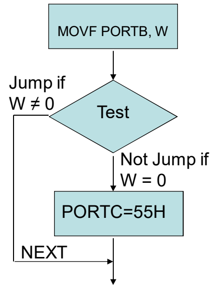
```
    MOVF PORTB, W
    BZ NEXT
    MOVLW 55H
    MOVWF PORTC
NEXT: ......
```



BNZ

- Jump if previous is not zero

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

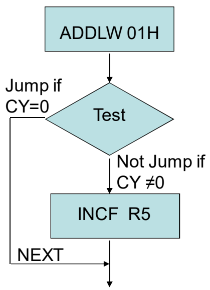


– This instruction examines the contents of the ACC and jumps if ACC is not 0.

BNC

- Jump if no carry（CY=0）

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



– CY is PSW.
– This instruction examines the CY flag, and if it is zero it will jump to the target address.

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
          BNZ  NEXT
          MOVLW    0x55
          MOVWF    MYLOC
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 0
             MOVLW 0x0      ;clear (W=0)
             MOVWF H_Byte,A ;clear H_Byte
             ADDLW 0x79     ;W=0+79H=79H
             BNC N_1        ;if CY=0,add next number directly
             INCF H_Byte    ;if CY=1, increment H_Byte
N_1:         ADDLW 0xF5     ;W=79+F5=6E and CY=1
             BNC N_2        ;if CY=0,add next number directly
             INCF H_Byte     ;if CY=1, increment H_Byte
N_2:         ADDLW 0xE2      ;W=6E+E2=50 and CY=1
             BNC OVER        ;jump if CY=0
             INCF  H_Byte     ;CY=1, increment H_Byte
OVER:        MOVWF L_Byte    ;Now H_Byte=2, L_Byte=50H
```


Relative Address Calculation in conditional brances Instructions

- 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

    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.

Relative Address Calculation in conditional brances Instructions

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 in ROM Loc the content is E301?

Ans:

E3 is opcode for BNC

When CPU is executing BNC N_1, the PC=00008.

The target address is 000A. The difference (000A-0008) is +2. So, the displacement in instruction is 01.

Relative Address Calculation in conditional brances Instructions

Example:

```
0000    0E00          MOVLW 0x0
0002    6E06          MOVWF H_Byte, A
0004    0F79          ADDLW 0x79
0006    E3XX          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    E301          BNC Over
0018    2A06          INCF 0x6, F, ACCESS
001A    6E05   Over:  MOVWF 0x5,
```

000E-0008=06, 6/2 = 3 => E303

Relative Address Calculation in conditional brances Instructions

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

Jump Instructions

The unconditional jump is a jump in which control is transferred unconditionally to the target location. The target address is directly coded in the instruction

•There are two unconditional jumps：

- GOTO（Long Jump）
- BRE（Short Jump）
- Examples in textbook

GOTO

- a 4-byte instruction

1110 1111	A_8-----A_1

1111	A_{20}-------------A_9

The 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.

  In condition branch, the instruction is

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

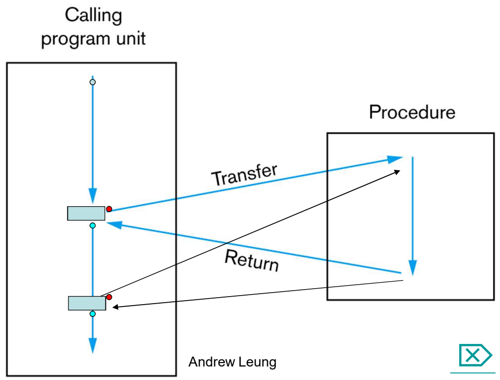
CALL and RCALL

- CALL: 4 bytes
  Call a subroutine in anywhere
  format is similar to GOTO
- RCALL: 2 bytes
  - Call a subroutine who address should be within Similar to BRA
  - Relative address

Andrew Leung 29

29

The Flow of Control Involving a Procedure



Andrew Leung 30

30

The Process of Calling a Subroutine

- After execution of the called subroutine, the CPU must know where to com back to.
- The process of calling a subroutine：
  - A subroutine is called by CALL instructions.
  - The CPU pushes the PC onto the stack (in PIC18 it is a hardware stack).
  - The CPU copies the target address to the PC.
  - The CPU fetches instructions from the new location.
  - When the instruction RETURN is fetched, the subroutine ends.
  - The CPU pops the return address from the stack.
  - The CPU copies the return address to the PC.
  - The CPU fetches instructions from the new location.

Andrew Leung 31

31

Sample

```
;MAIN program calling subroutines
         ORG   0
MAIN:
           :
           :
          CALL      SUBR_1
           :
          CALL      SUBR_2
           :
           :
HERE:    SJMP        HERE
;————end of MAIN
SUBR_1:  ....
         ....
         RETURN
;————end of subroutine 1
SUBR_2:  ....
         ....
         RETURM
END        ;end of the asm file
```

Andrew Leung 32

32

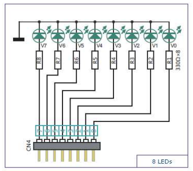
Sample

```
;MAIN program calling subroutines

MYREG EQU   0x8
PORTB EQU   0xF8
      ORG 0

BACK: MOVLW 0x55
      MOVWF PORTB
      CALL  DELAY
CALL  DELAY_500MS
      MOVLW 0xAA
      MOVWF PORTB
      CALL  DDELAY
      GOTO  BACK
DELAY:
      MOVLW 0xFF
      MOVWF MYREG
AGAIN:
      NOP
      NOP
      DECF MYREG, F
      BNZ   AGAIN
      RETURN
   END
```

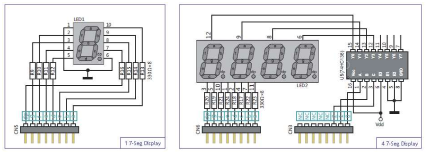

Sections

PIC18 Parallel Ports

- I/O ports are available in PIC18 to interact, monitor and control peripherals.



LED



7-Segment LED

PIC18 Parallel Ports

- The number of ports in the PIC18 family depends on the number of pins on the chip.
- PIC18F4520 has five ports: Port A-E
- Different ports have different number of pins:
  - Port A: 7 pins
  - Port B, C, D: 8 pins
  - Port E: 3 pins

PIC18 Parallel Ports

- The behaviour of each port is controlled by two special function registers (SFRs):
  - PORTx – indicates the voltage levels on the pins of the device
  - TRISx – data direction register
- e.g., For Port B, we have PORTB and TRISB

TRISx and PORTx SFRs

- Each of the Ports A-E can be used for input and output.
- TRISx SFR is used for designating the direction of a port
  - 0 for output (e.g., Controlling LED state)
  - 1 for input (e.g., Key scan)
- e.g., To output data to Port B:
  - Write 0s to TRISB SFR
  - Write data to PORTB SFR

TRISx and PORTx SFRs

- e.g., output the hex value 0x26 to Port C

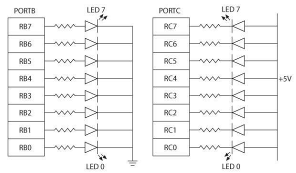
```
clrf TRISC
movlw 0x26
movwf PORTC
```

- e.g., Read the current value of Port D into WREG register

```
setf TRISD
movf PORTD, W
```


Interfacing with LED

- Two ways of connecting LEDs to I/O ports:
  - Common Cathode: LED cathodes are grounded and logic 1 from the I/O port turns on the LEDs.
  - Common Anode: LED anodes are connected to the power supply and logic 0 from the I/O port turns on the LEDs.

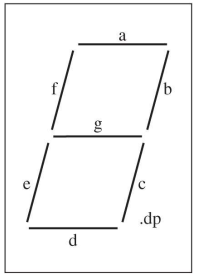


Common Cathode
Active high

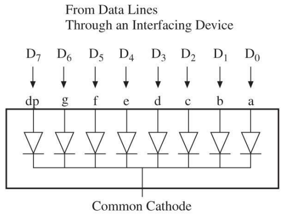
Common Anode
Active low

Interfacing with 7-Segment LED

- Often used to display BCD numbers (1 through 9) and a few alphabets
- A group of eight LEDs physically mounted in the shape of the number eight plus a decimal point
- Each LED is called a segment and labeled as ‘a’ through ‘g’.

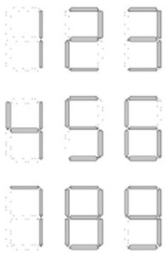


Interfacing with 7-Segment LED



- **In a common cathode seven-segment LED**
  - **All cathodes are connected together to ground and the anodes are connected to data lines**
- **Logic 1 turns on a segment.**
- **Example: To display digit 1, all segments except b and c should be off.**
- **Byte 00000110 = 06H will display digit 1.**

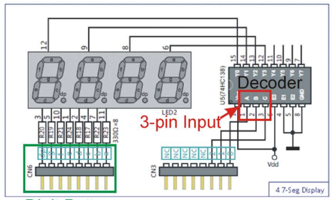
Interfacing with 7-Segment LED



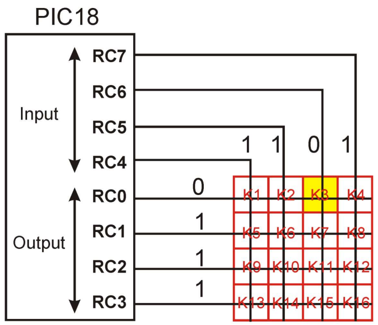
Digit Shown	Illuminated Segment (1 = illumination) a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Interfacing with 4-digit 7-Segment LED

- Decoder selects the position where digit pattern is displayed.
- Use time multiplexing if we need to display all four digits.



Interfacing with Keypad

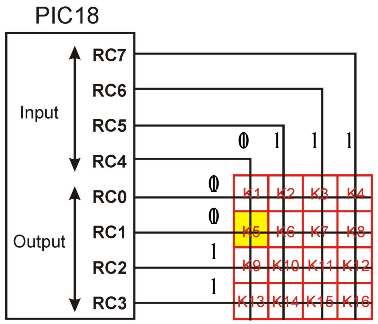


Andrew Leung

45

45

When K5 is pressed

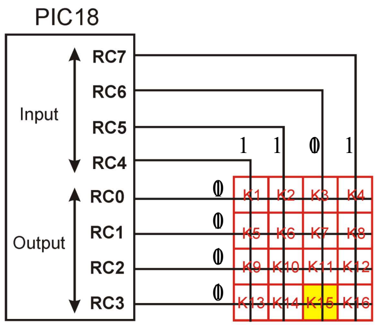


Andrew Leung

46

46

When K15 is pressed



Andrew Leung

47

47

PICkit3

- Device Programmer – Burn assembled code into the device FLASH.
- In-circuit debugger – allow real-time interactions with the board even when debugging.

Andrew Leung

48

48

I/O ports and bit-addressability

- Often need to access individual bit of the port instead of the entire 8 bits.
- PIC18 provides instructions that alter individual bits without altering the rest of the bits in the port.
- Most common bit-oriented instructions:

Instructions	Function
bsf fileReg, bit	Bit Set fileReg
bcf fileReg, bit	Bit Clear fileReg
btg fileReg, bit	Bit Toggle fileReg
btfsc fileReg, bit	Bit test fileReg, skip if clear
btfss fileReg, bit	Bit test fileReg, skip if set

49

bsf, bcf and btg

- bsf fileReg, bit
  - e.g., `bsf PortB, 5` sets Bit 5 of Port B to be 1
- bcf fileReg, bit
  - e.g., `bcf PortB, 5` sets Bit 5 of Port B to be 0.
- btg fileReg, bit
  - e.g., `btg PortB, 5` toggles Bit 5 of Port B (i.e., sets it to 1 if the current value is 0 & vice versa)

50

Checking the state of an I/O port

- btfsc (bit test file, skip if clear) and btfss (bit test file, skip if set) are used to make branching decision based on the status of a given bit.
- e.g., btfsc PORTD, 2 skips the next instruction if Bit 2 of Port D equals 0.
- e.g., btfss PORTD, 2 skips the next instruction if Bit 2 of Port D equals 1.

51

Example: btfss

- e.g., Write a program to (a) keep monitoring RB2 bit until it becomes high (b) When RB2 becomes high, write 45H to Port C

```
  bsf TRISB, 2; set RB2 as input
  clrf TRISC; set Port C as output
  movlw 0x45
Again: btfss PORTB, 2
  bra Again
  movwf PORTC
```


52

Example: btfss

- e.g., Write a program to check RB2.
  - If RB2 = 0, send the letter ‘N’ to Port D
  - If RB2 = 1, send the letter ‘Y’ to Port D

```
  bsf TRISB, 2; set RB2 as input
  clrf TRISD; set Port D as output
Again: btfss PORTB, 2
  bra Over
  movlw A’Y’
  movwf PORTD
  bra Again
Over:  movlw A’N’
  movwf PORTD
  bra Again
```


Example: btfsc

- e.g., Perform the same function using btfsc

```
  bsf TRISB, 2; set RB2 as input
  clrf TRISD; set Port D as output
Again: btfsc PORTB, 2
  bra Over
  movlw A‘N’
  movwf PORTD
  bra Again
Over:  movlw A‘Y’
  movwf PORTD
  bra Again
```


Sections

3.1 Loop and Jump Instructions
3.2 IO PORT
3.3 **Time Delay Generation and Calculation**

Time Delay

- We have written a delay subroutine in Ex3-8.
- How to calculate exact delays？
- How to generate various time delay？

Machine Cycle

- For the CPU to execute an instruction takes a certain number of clock cycles.
- In the PIC18 family, these clock cycles are referred to as *instruction cycles*.
- In PIC18, 4 clock cycles=1 machine cycle
- Different instructions take need different machine cycles

Andrew Leung 57

57

Example

The following shows 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=1MHz, 1MC=1 µs (microsecond)
(b) 16/4 MHz = 4 MHz
machine cycle (**MC**) = 1/4MHz=0.25 µs
(c) 20 MHz/4 = 5 MHz
MC = 1/5 MHz = 0.2 µs

Andrew Leung 58

58

Example

For a PIC18 system of 4MHz11.0592 MHz, 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) CALL (h) BNZ
Solution:
The machine cycle for a system of 11.0952 MHz is 1.085 µs.
Table A-1 shows machine cycles for each instructions.

Instruct	Machine cycles	Time to execute
(a) MOVLW	1	1x1 µs = µs
(b) DECF	1	1x1 µs = µs
(c) MOVWF	1	1x1 µs = µs
(d) ADDLW	1	1x1 µs = µs
(e) NOP	1	1x1 µs = µs
(b) GOTO	2	2x1 µs = µs
(c) CALL	2	2x1 µs = µs
(d) BNZ	1 or 2	1 or 2 µs

BNZ takes 2 MCs if it jump, and takes 1 MC when falling through. (The pipeline is clear if jump)

Andrew Leung 59

59

Example

Find the size of the delay in the following program, if the crystal frequency is 4 MHz.

```
                                        MC
MYREG   EQU 0x08
DELAY:  MOVLW 0xFF                      1
        MOVWF MYREG                     1
AGAIN:  NOP                             1
        NOP                             1
        DECF  MYREG, F                  1
        BNZ   AGAIN                     2
        RETURN                          1
```

[(255 x 5)+1+1+1] x 1 µs = 1278 µs.

Note that BNZ takes two MCs if it jump, and take only one when falling through the loop. The actual time is 1277 µs.

Andrew Leung 60

60

Example

Find the size of the delay in the following program, if the crystal frequency is 4 MHz.

```
                                            MC
R2      EQU 0x7
R3      EQU 0x8
DELAY
        MOVLW   D'200'                      1
        MOVWF   R2                          1
AGAIN   MOVLW   D'250'                      1
        MOVWF   R3                          1
HERE    NOP                                 1
        NOP                                 1
        DECF    R3, F                       1
        BNZ     HERE,                       2
        DECF    R2, F                       1
        BNZ     AGAIN                       2
        RETURN                              1
```

For HERE loop, the delay is 5x250=1250µs. The AGAIN loop repeats the HERE loop 200 times. Therefore, we have 200x1250=250 000 µs. (without overhead)

Overhead=5x200-200 + 1 + 1 + 1 – 1

The delay = 250 ms + 0.8 ms = 250.8 ms

Andrew Leung

61

61

Example

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

Solution:

10 MHZ => 1 MC = 400 ns. `1 second' needs 2,500,000 MCs

If we have a basic loop with 5 MC. That means, we may need to execute this basic loop 500,000 times. 500,000 can be decompose as 20x100x250

```
DELAY:          MOVLW   D'20'
                MOVWF   R4
BACK            MOVLW   D'100'
                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
```

Andrew Leung

62

62

You are able to (1/2)

- Code PIC18 Assembly language instructions using loops
- Code PIC18 Assembly language conditional jump instructions
- Explain conditions that determine each conditional jump instruction
- Code GOTO/BRA jump instructions for unconditional jumps

Andrew Leung

63

63

You are able to (2/2)

- Calculate target addresses for jump instructions
- Code PIC subroutines
- Code PIC IO ports
- Discuss crystal frequency versus machine cycle
- Code PIC programs to generate a time delay

Andrew Leung

64

64