PIC Microcontroller and Embedded Systems
Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq
The Islamic Uni. Of Gaza
Chapter 3: Branch, Call and Time Delay Loop

- Branch instruction and looping
- Call instruction and stack
- PIC18 Time Delay and instruction pipeline
Objective

- Code PIC Assembly language instructions to create loops and conditional branch instructions
- Code Goto instructions for unconditional jump
Branch instructions 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
Looping 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
Example 3-1

Write a program to

- Clear WREG
- Add 3 to WREG ten times and place the result in SFR PORTB

Solution

```asm
COUNT EQU 0x25
MOVLW d'10'
MOVWF COUNT
MOVLW 0
AGAIN
ADDLW 3
DECFSZ COUNT,F
GOTO AGAIN
MOVWF PORTB
```
Figure 3-1. Flowchart for the DECFSZ Instruction
Using **BNZ\BZ** instructions

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

- These instructions check the status flag

```
DECF fileReg, f
BNZ Back
```
Example 3-2

Write a program to

- Clear WREG
- Add 3 to WREG ten times and place the result in SFR PORTB

Solution

COUNT EQU 0x25
MOVLW d'10'
MOVWF PORTB
MOVWF COUNT
MOVFWF 0
AGAIN
ADDLW 3
DECFW COUNT,F
BNZ AGAIN
MOVWF PORTB
Figure 3-2. Flowchart for Example 3-2
Example 3-3

- What is the maximum number of times that the loop can be repeated?
- All locations in the FileReg are 8-bit
- The max. loop size is 255 time
Loop inside a loop

- Write a program to
  a) Load the PORTB SFR register with the value 55H
  b) Complement PORTB 700 times

Solution

R1 EQU 0x25
R2 EQU 0x26
COUNT_1 EQU d'10'
COUNT_2 EQU d'70'
Solution

**MOVLW 0x55**
**MOVWF PORTB**
**MOVLW COUNT_1**
**MOVWF R1**
**LOP_1**
**MOVLW COUNT_2**
**MOVWF R2**
**LOP_2**
**COMPF PORTB, F**
**DECF R2, F**
**BNZ LOP_2**
**DECF R1, F**
**BNZ LOP_1**

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>25H</td>
<td>(R1) 10</td>
</tr>
<tr>
<td>26H</td>
<td>(R2) 70</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>F81H</td>
<td>(PORTB) 55</td>
</tr>
</tbody>
</table>
**Figure 3-3. Flowchart**

```
LOAD WREG
LOAD PORTB
LOAD WREG
LOAD COUNTER 1
LOAD WREG
LOAD COUNTER 2
TOGGLE PORTB
DECREMENT COUNTER 2

INSTRUCTIONS
MOVLW 0x55
MOVWF PORTB
MOVLW COUNT_1
MOVWF R1
LOP_1
MOVLW COUNT_2
MOVWF R1
LOP_2
COMPF PORTB, F
DECF R2, F
```
Figure 3-3. (continued)
Other conditional jumps

- All of the 10 conditional jumps are 2-byte instructions
- They require 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.
# Flag Bits and Decision Making

<table>
<thead>
<tr>
<th>Flag Bits</th>
<th>k</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>k</td>
<td>Branch relative if Carry</td>
</tr>
<tr>
<td>BNC</td>
<td>k</td>
<td>Branch relative if Not Carry</td>
</tr>
<tr>
<td>BN</td>
<td>k</td>
<td>Branch relative if Negative</td>
</tr>
<tr>
<td>BNN</td>
<td>k</td>
<td>Branch relative if Not Negative</td>
</tr>
<tr>
<td>BOV</td>
<td>k</td>
<td>Branch relative if Overflow</td>
</tr>
<tr>
<td>BNOV</td>
<td>k</td>
<td>Branch relative if Not Overflow</td>
</tr>
<tr>
<td>BZ</td>
<td>k</td>
<td>Branch relative if Zero</td>
</tr>
<tr>
<td>BNZ</td>
<td>k</td>
<td>Branch relative if Not Zero</td>
</tr>
</tbody>
</table>
Example 3-5

- 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 3-6

- Find the sum of the values 79H, F5H, and E2H. Put the sum in fileReg loc. 5H and 6H.

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Address</th>
<th>Data</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>5H</td>
<td>79</td>
<td>5H</td>
<td>6E</td>
<td>5H</td>
<td>50</td>
</tr>
<tr>
<td>(Low-</td>
<td>Byte) 0</td>
<td>(Low-</td>
<td>Byte) 0</td>
<td>(Low-</td>
<td>Byte) 0</td>
</tr>
<tr>
<td>6H</td>
<td>0</td>
<td>6H</td>
<td>1</td>
<td>6H</td>
<td>2</td>
</tr>
<tr>
<td>(High-</td>
<td>Byte) 0</td>
<td>(High-</td>
<td>Byte) 1</td>
<td>(High-</td>
<td>Byte) 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

79 + F5 = 16E
6E + E2 = 150
50
Solution

L_Byte EQU 0x5
H_Byte EQU 0x6

ORG 0h

MOVLW 0x0
MOVWF H(Byte)

ADDLW 0x79

BNC N_1
INCF H(Byte),F

ADDLW 0xF5
BNC N_2
INCF H(Byte),F

INCF H(Byte),F

N_2 ADDLW 0xE2
BNC OVER
INCF H(Byte),F
OVER MOVWF L(Byte)

END
### Example 3-7

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>MOVLW 0x0000</td>
<td>MOVLW 0x0</td>
</tr>
<tr>
<td>000002</td>
<td>MOVWF H_Byte</td>
<td>MOVWF H_Byte</td>
</tr>
<tr>
<td>000004</td>
<td>ADDLW 0x79</td>
<td>ADDLW 0x79</td>
</tr>
<tr>
<td>000006</td>
<td>E301 BNC N_1</td>
<td>BNC N_1</td>
</tr>
<tr>
<td>000008</td>
<td>2A06 INCF H_Byte,F</td>
<td>INCF H_Byte,F</td>
</tr>
<tr>
<td>00000A</td>
<td>OFF5 N_1 ADDLW 0xF5</td>
<td>N_1 ADDLW 0xF5</td>
</tr>
<tr>
<td>00000C</td>
<td>E301 BNC N_2</td>
<td>BNC N_2</td>
</tr>
<tr>
<td>00000E</td>
<td>2A06 INCF H_Byte,F</td>
<td>INCF H_Byte,F</td>
</tr>
<tr>
<td>000010</td>
<td>0FE2 N_2 ADDLW 0xEE2</td>
<td>N_2 ADDLW 0xEE2</td>
</tr>
<tr>
<td>000012</td>
<td>E301 BNC OVER</td>
<td>BNC OVER</td>
</tr>
<tr>
<td>000014</td>
<td>2A06 INCF H_Byte,F</td>
<td>INCF H_Byte,F</td>
</tr>
<tr>
<td>000016</td>
<td>6E05 OVER MOVWF L_Byte</td>
<td>OVER MOVWF L_Byte</td>
</tr>
</tbody>
</table>
### Example 3-7

#### Program Memory

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Opcode</th>
<th>Label</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>0E00</td>
<td></td>
<td>MOVLW 0</td>
</tr>
<tr>
<td>2</td>
<td>0002</td>
<td>6E06</td>
<td></td>
<td>MOVWF H.Byte, ACCESS</td>
</tr>
<tr>
<td>3</td>
<td>0004</td>
<td>0F79</td>
<td></td>
<td>ADDLW 0x79</td>
</tr>
<tr>
<td>4</td>
<td>0006</td>
<td>E301</td>
<td></td>
<td>BNC N_1</td>
</tr>
<tr>
<td>5</td>
<td>0008</td>
<td>2A06</td>
<td></td>
<td>INCF H.Byte, F, ACCESS</td>
</tr>
<tr>
<td>6</td>
<td>000A</td>
<td>OFF5</td>
<td>N_1</td>
<td>ADDLW 0xf5</td>
</tr>
<tr>
<td>7</td>
<td>000C</td>
<td>E301</td>
<td></td>
<td>BNC N_2</td>
</tr>
<tr>
<td>8</td>
<td>000E</td>
<td>2A06</td>
<td></td>
<td>INCF H.Byte, F, ACCESS</td>
</tr>
<tr>
<td>9</td>
<td>0010</td>
<td>OFE2</td>
<td>N_2</td>
<td>ADDLW 0xe2</td>
</tr>
<tr>
<td>10</td>
<td>0012</td>
<td>E301</td>
<td></td>
<td>BNC OVER</td>
</tr>
<tr>
<td>11</td>
<td>0014</td>
<td>2A06</td>
<td></td>
<td>INCF H.Byte, F, ACCESS</td>
</tr>
<tr>
<td>12</td>
<td>0016</td>
<td>6E05</td>
<td>OVER</td>
<td>MOVWF L.Byte, ACCESS</td>
</tr>
<tr>
<td>13</td>
<td>0018</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>14</td>
<td>001A</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>15</td>
<td>001C</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>16</td>
<td>001E</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>17</td>
<td>0020</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>18</td>
<td>0022</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>19</td>
<td>0024</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
</tbody>
</table>
### Example 3-8

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Opcode</th>
<th>Label</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>0EOA</td>
<td></td>
<td>MOVLW 0xa</td>
</tr>
<tr>
<td>2</td>
<td>0002</td>
<td>6E25</td>
<td></td>
<td>MOVWF COUNT, ACCESS</td>
</tr>
<tr>
<td>3</td>
<td>0004</td>
<td>0E00</td>
<td></td>
<td>MOVLW 0</td>
</tr>
<tr>
<td>4</td>
<td>0006</td>
<td>0F03</td>
<td>AGAIN</td>
<td>ADDLW 0x3</td>
</tr>
<tr>
<td>5</td>
<td>0008</td>
<td>0625</td>
<td></td>
<td>DECF COUNT, F, ACCESS</td>
</tr>
<tr>
<td>6</td>
<td>000A</td>
<td>E1FD</td>
<td></td>
<td>BNZ AGAIN</td>
</tr>
<tr>
<td>7</td>
<td>000C</td>
<td>6E81</td>
<td></td>
<td>MOVWF PORTB, ACCESS</td>
</tr>
<tr>
<td>8</td>
<td>000E</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>9</td>
<td>0010</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>10</td>
<td>0012</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>11</td>
<td>0014</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>12</td>
<td>0016</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>13</td>
<td>0018</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>14</td>
<td>001A</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
</tbody>
</table>
Question?

Which is better, to use BNZ along with DECF or DCFSNZ??
Unconditional branch instruction

- Control is transferred unconditionally to the target location (at ROM)
- Tow unconditional branches
  - GOTO
  - BRA
Figure 3-4. GOTO Instruction
BRA Instruction

Figure 3-5. BRA (Branch Unconditionally)
Instruction Address Range
### BRA Instruction

#### Program Memory

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Opcode</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>0EOA</td>
<td><strong>MOVWLW 0xa</strong></td>
</tr>
<tr>
<td>2</td>
<td>0002</td>
<td>6E25</td>
<td><strong>MOVWF COUNT, ACCESS</strong></td>
</tr>
<tr>
<td>3</td>
<td>0004</td>
<td>D001</td>
<td><strong>BRA AGAIN</strong></td>
</tr>
<tr>
<td>4</td>
<td>0006</td>
<td>0EO0</td>
<td><strong>MOVWLW 0</strong></td>
</tr>
<tr>
<td>5</td>
<td>0008</td>
<td>0F03</td>
<td><strong>ADDLW 0x3</strong></td>
</tr>
<tr>
<td>6</td>
<td>000A</td>
<td>0625</td>
<td><strong>DEF CF COUNT, F, ACCESS</strong></td>
</tr>
<tr>
<td>7</td>
<td>000C</td>
<td>E1FD</td>
<td><strong>BNZ AGAIN</strong></td>
</tr>
<tr>
<td>8</td>
<td>000E</td>
<td>6E81</td>
<td><strong>MOVWF PORTB, ACCESS</strong></td>
</tr>
<tr>
<td>9</td>
<td>0010</td>
<td>D7FB</td>
<td><strong>BRA AGAIN</strong></td>
</tr>
<tr>
<td>10</td>
<td>0012</td>
<td>FFFF</td>
<td><strong>NOP</strong></td>
</tr>
<tr>
<td>11</td>
<td>0014</td>
<td>FFFF</td>
<td><strong>NOP</strong></td>
</tr>
<tr>
<td>12</td>
<td>0016</td>
<td>FFFF</td>
<td><strong>NOP</strong></td>
</tr>
</tbody>
</table>

The PIC uCs
GOTO to itself

- Label and $ can be used to keep uC busy (jump to the same location)
- HERE GOTO HERE
- GOTO $ 
- OVER BRA OVER
- BRA $
PIC18 Call instruction

Section 3-2
Call instruction

Call a subroutine

Call
- 4-byte instruction
- Long Call

Rcall
- 2-byte instruction
- Relative Call
CALL Instruction

Figure 3-6. CALL Instruction
CALL Instruction

- Control is transferred to subroutine
- Current PC value, the instruction address just below the CALL instruction, is stored in the stack
  - push onto the stack

- Return instruction is used to transfer the control back to the caller,
  - the previous PC is popped from the stack
Stack and Stack Pointer (SP)

- Read/Write Memory
- Store the PC Address
  - 21-bit (000000 to 1FFFFFF)
- 5-bit stack, total of 32 locations
- SP points to the last used location of the stack
  - Location 0 doesn’t used
  - Incremented pointer
Figure 3-7. PIC Stack 31 × 21
**Return from Subroutine**

- The stack is popped and the top of the stack (TOS) is loaded into the program counter.
- If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, W, STATUS and BSR.
- If 's' = 0, no update of these registers occurs (default).

| 0000 | 0000 | 0001 | 001s |
Example 3-9

- Toggle all bits of to SFR register of PORTB by sending to it values 55H and AAH continuously. Put a delay in between issuing of data to PORTB.

- Analyze the stack for the CALL instructions
Solution

MYREG EQU 0x08
PORTB EQU 0x0F8

ORG 0
BACK
MOVLW 0x55
MOVWF PORTB
CALL DELAY

MOVLW 0xAA
MOVWF PORTB
CALL DELAY
GOTO BACK

ORG 20H
DELAY
MOVLW 0xFF
MOVWF MYREG
AGAIN
NOP
NOP
DECF MYREG, F
BNZ AGAIN
RETURN
END
### Example 3-10

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>000008</td>
</tr>
</tbody>
</table>

#### Program Memory

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Opcode</th>
<th>Label</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>OE55</td>
<td>BACK</td>
<td>MOVLW 0x55</td>
</tr>
<tr>
<td>2</td>
<td>0002</td>
<td>6E81</td>
<td></td>
<td>MOVWF PORTB, ACCESS</td>
</tr>
<tr>
<td>3</td>
<td>0004</td>
<td>EC10</td>
<td></td>
<td>CALL DELAY, 0</td>
</tr>
<tr>
<td>4</td>
<td>0006</td>
<td>F000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>5</td>
<td>0008</td>
<td>0EAA</td>
<td></td>
<td>MOVLW 0xaa</td>
</tr>
<tr>
<td>6</td>
<td>000A</td>
<td>6E81</td>
<td></td>
<td>MOVWF PORTB, ACCESS</td>
</tr>
<tr>
<td>7</td>
<td>000C</td>
<td>EC10</td>
<td></td>
<td>CALL DELAY, 0</td>
</tr>
<tr>
<td>8</td>
<td>000E</td>
<td>F000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>9</td>
<td>0010</td>
<td>EF00</td>
<td></td>
<td>GOTO BACK</td>
</tr>
<tr>
<td>10</td>
<td>0012</td>
<td>F000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>11</td>
<td>0014</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>12</td>
<td>0016</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>13</td>
<td>0018</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>14</td>
<td>001A</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>15</td>
<td>001C</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>16</td>
<td>001E</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>17</td>
<td>0020</td>
<td>0EFF</td>
<td>DELAY</td>
<td>MOVLW 0xff</td>
</tr>
<tr>
<td>18</td>
<td>0022</td>
<td>6E08</td>
<td></td>
<td>MOVWF MYREG, ACCESS</td>
</tr>
<tr>
<td>19</td>
<td>0024</td>
<td>0000</td>
<td>AGAIN</td>
<td>NOP</td>
</tr>
<tr>
<td>20</td>
<td>0026</td>
<td>0000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>21</td>
<td>0028</td>
<td>0608</td>
<td></td>
<td>DECF MYREG, F, ACCESS</td>
</tr>
<tr>
<td>22</td>
<td>002A</td>
<td>E1FC</td>
<td></td>
<td>BNZ AGAIN</td>
</tr>
<tr>
<td>23</td>
<td>002C</td>
<td>0012</td>
<td></td>
<td>RETURN 0</td>
</tr>
<tr>
<td>24</td>
<td>002E</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>25</td>
<td>0030</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
</tbody>
</table>
;MAIN program calling subroutines
ORG 0
MAIN
CALL SUBR_1
CALL SUBR_2
CALL SUBR_3

HERE
BRA HERE ;stay here
;---------end of MAIN
;
SUBR_1
....
....
RETURN
;---------end of subroutine 1
;
SUBR_2
....
....
RETURN
;---------end of subroutine 2

SUBR_3
....
....
RETURN
;---------end of subroutine 3
END ;end of the asm file

Figure 3-8. PIC Assembly Main Program That Calls Subroutines
RCALL (Relative Call)

- 2-Byte instruction
- The target address must be within 2K
  - 11 bits of the 2 Byte is used
- Save a number of bytes.

```
1101   lnnn   nnnn   nnnn
```
### Example 3-12

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Opcode</th>
<th>Label</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>0E55</td>
<td></td>
<td>MOVLW 0x55</td>
</tr>
<tr>
<td>2</td>
<td>0002</td>
<td>6E81</td>
<td>BACK</td>
<td>MOVWF PORTB, ACCESS</td>
</tr>
<tr>
<td>3</td>
<td>0004</td>
<td>D802</td>
<td></td>
<td>RCALL DELAY</td>
</tr>
<tr>
<td>4</td>
<td>0006</td>
<td>1E81</td>
<td></td>
<td>COMF PORTB, F, ACCESS</td>
</tr>
<tr>
<td>5</td>
<td>0008</td>
<td>D7FC</td>
<td></td>
<td>BRA BACK</td>
</tr>
<tr>
<td>6</td>
<td>000A</td>
<td>0EFF</td>
<td>DELAY</td>
<td>MOVLW 0xff</td>
</tr>
<tr>
<td>7</td>
<td>000C</td>
<td>6E08</td>
<td></td>
<td>MOVWF MYREG, ACCESS</td>
</tr>
<tr>
<td>8</td>
<td>000E</td>
<td>0000</td>
<td>AGAIN</td>
<td>NOP</td>
</tr>
<tr>
<td>9</td>
<td>0010</td>
<td>0000</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>10</td>
<td>0012</td>
<td>0608</td>
<td></td>
<td>DECF MYREG, F, ACCESS</td>
</tr>
<tr>
<td>11</td>
<td>0014</td>
<td>E1FC</td>
<td></td>
<td>BNZ AGAIN</td>
</tr>
<tr>
<td>12</td>
<td>0016</td>
<td>0012</td>
<td></td>
<td>RETURN 0</td>
</tr>
<tr>
<td>13</td>
<td>0018</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>14</td>
<td>001A</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>15</td>
<td>001C</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>16</td>
<td>001E</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>17</td>
<td>0020</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>18</td>
<td>0022</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>19</td>
<td>0024</td>
<td>FFFF</td>
<td></td>
<td>NOP</td>
</tr>
</tbody>
</table>
PIC18 Time Delay and instruction pipeline

Section 3-3
Delay Calculating for PIC18

- Two factors can affect the accuracy of the delay
  1. The duration of the clock period, which is a function of the Crystal freq
     - Connected to OSC! And OSC2
  2. The instruction cycle duration
     - Most of the PIC18 instructions consumes 1 cycle
       - Use Harvard Architecture
       - Use RISC Architecture
       - Use the pipeline concept between fetch and execute.
Figure 3-9. Pipeline vs. Non-pipeline
**PIC multistage pipeline**

- Superpipeline is used to speed up execution.
- The process of executing instructions is split into small steps.
- Limited to the slowest step.
Figure 3-10. Pipeline Activity After the Instruction Has Been Fetched
Figure 3-11. Pipeline Activity for Both Fetch and Execute

D = Decode the instruction
R = Read the operand
P = Process (e.g. ADDLW)
W = Write the result to destination register

Instruction
MOV LW 5
ADDLW 9
ADDLW 7
ADDLW 2
Instruction Cycle time for the PIC

- What is the Instruction Cycle?
- Most instructions take one or two cycles
  - BTFSS can take up to 3 cycles
- Instruction Cycle depends on the freq. of oscillator
- Clock source: Crystal oscillator and on-chip circuitry
- One instruction cycle consists of four oscillator period.
Example 3-14

Find the period of the instruction cycle you chose 4 MHz crystal? And what is required time for fetching an instruction?

Solution

4 MHz/4 =1 MHz

Instruction Cycle = 1/1MHz = 1 usec

Fetch cycle = 4 * 1 usec = 4 usec
Branch penalty

- Queue is needed for prefetched instruction
- If the prefetched instruction is incorrect, the CPU must flush the memory. When??

<table>
<thead>
<tr>
<th>If Jump:</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Decode</td>
<td>Read literal 'n'</td>
<td>Process Data</td>
<td>Write to PC</td>
</tr>
<tr>
<td>No</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>If No Jump:</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Decode</td>
<td>Read literal 'n'</td>
<td>Process Data</td>
<td>No operation</td>
</tr>
</tbody>
</table>

The PIC uCs
Branch penalty

<table>
<thead>
<tr>
<th>TCY0</th>
<th>TCY1</th>
<th>TCY2</th>
<th>TCY3</th>
<th>TCY4</th>
<th>TCY5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fetch 1</td>
<td>Execute 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch 2</td>
<td>Execute 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch 3</td>
<td>Execute 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch 4</td>
<td>Flush (NOP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch SUB_1</td>
<td>Execute SUB_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. MOV LW 55h
2. MOVWF PORTB
3. BRA SUB_1
4. BSF PORTA, BIT3 (Forced NOP)
5. Instruction @ address SUB_1

All instructions are single-cycle except for any program branches. These take two cycles since the fetch instruction is “flushed” from the pipeline, while the new instruction is being fetched and then executed.
### BTFSC and BTFSS

#### Q Cycle Activity:

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode</td>
<td>Read register ‘f’</td>
<td>Process Data</td>
<td>No operation</td>
</tr>
</tbody>
</table>

If skip:

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
</tr>
</tbody>
</table>

If skip and followed by 2-word instruction:

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
</tr>
<tr>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
</tr>
</tbody>
</table>
Example 3-15

Find how long it take to execute each of the following instructions for a PIC18 with 4 MHz

- MOVWLW
- ADDLW
- DECF
- NOP
- MOVWF
- CALL
- GOTO
- BNZ
Delay calculation for PIC18
Example 3-16

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

```
DELAY   MOVVLW 0xFF
MOVF   MOVWF MYREG
AGAIN  NOP
       NOP
       DECF MYREG, F
       BNZ AGAIN
RET
```
Example 3-17

MYREG  EQU 0x08
ORG  0
BACK  MOVLW 0x55
MOVWF  PORTB
CALL DELAY
MOVLW 0xAA
MOVWF PORTB
CALL DELAY
GOTO BACK

ORG 300H
DELAY MOVLW 0xFA
MOVWF MYREG
AGAIN NOP
NOP
DECF MYREG, F
BNZ AGAIN
RETURN
Example 3-20

R2 EQU 0x2
R3 EQU 0x3
R4 EQU 0x4

MOVLW 0x55
MOVWF PORTB

BACK

CALL DELAY_500MS

COMF PORTB

GOTO BACK

DELAY_500MS

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
Chapter 3: Summary

- Looping in PIC Assembly language is performed using an instruction to decrement a counter and to jump to the top of the loop if the counter is not zero.

- Assembly language includes conditional and unconditional, and call instructions.

- PIC18 uses Superpipeline is used to speed up execution.

Next: Chapter 4
PIC I/O Port Programming