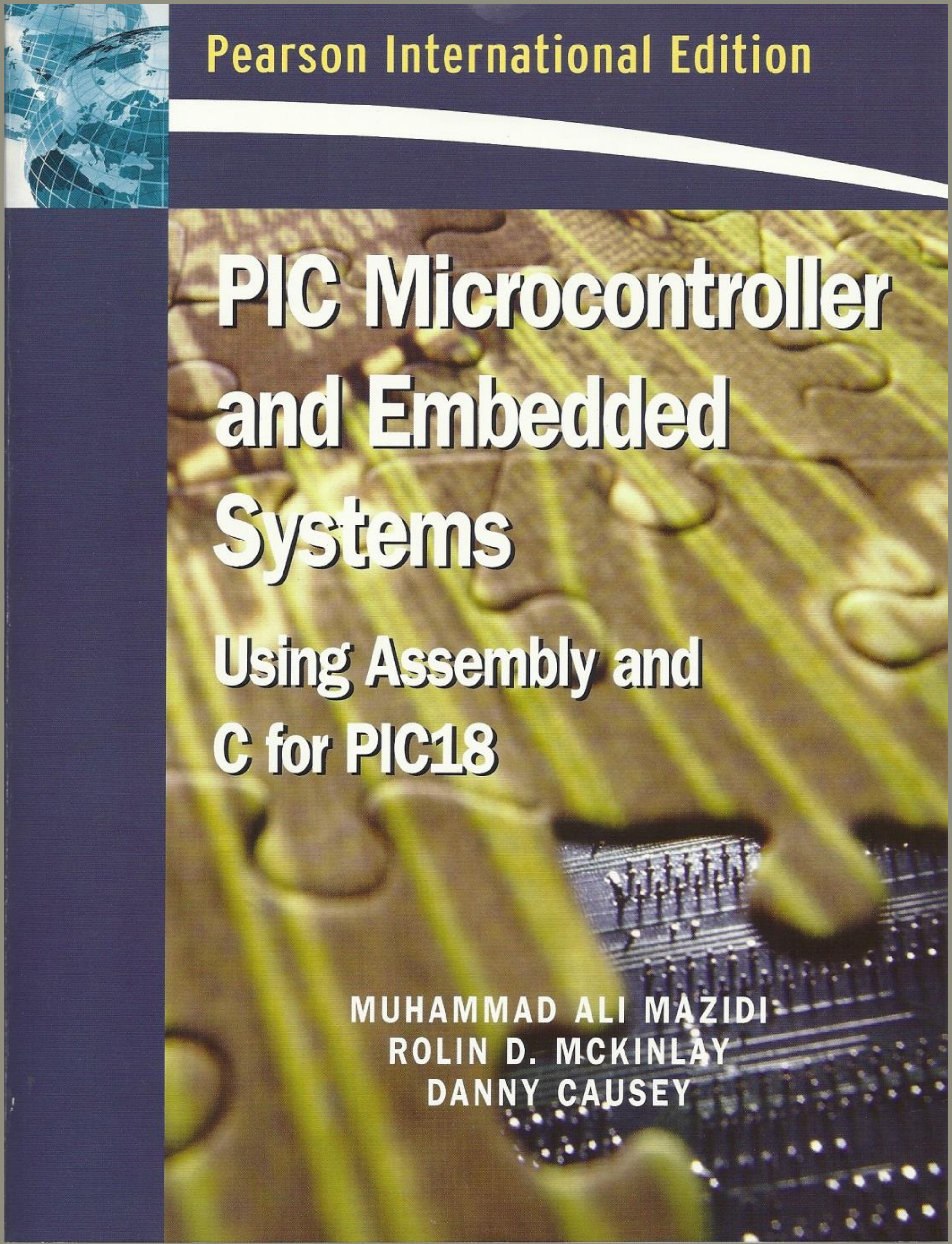
Chapter 3

Branch, Call, and Time Delay Loop



DECFSZ fileReg, d

- Decrement fileReg and skip next instruction if 0.
- Loop – repeating a sequence of instructions or an operation a certain number of times.

Example 3-1

Write a program to (a) clear WREG, and (b) add 3 to WREG ten times and place the result in SFR of PORTB. Use the DECFSZ instruction to perform looping.

Solution:

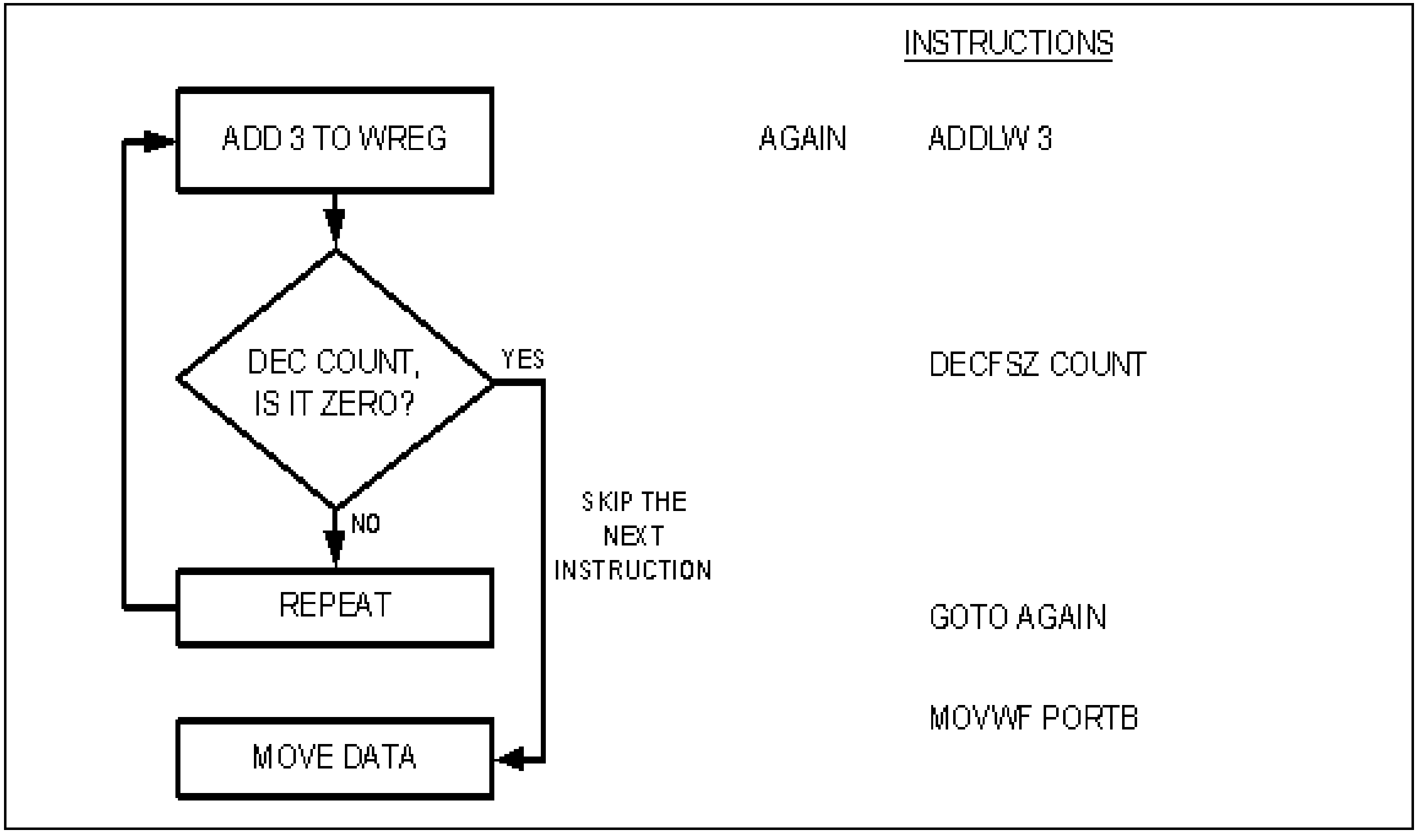
```
;this program adds value 3 to WREG ten times

COUNT   EQU 0x25              ;use loc 25H for counter

        MOVLW   d'10'         ;WREG = 10 (decimal) for counter
        MOVWF   COUNT         ;load the counter
        MOVLW   0             ;WREG = 0
AGAIN   ADDLW   3             ;add 03 to WREG (WREG = sum)
        DECFSZ  COUNT,F       ;decrement counter, skip if count = 0
        GOTO    AGAIN         ;repeat until count becomes 0
        MOVWF   PORTB         ;send sum to PORTB SFR
```

Notice that the DECFSZ instruction will decrement the counter (fileReg loc 0x25), which has 10 in it. It becomes 9. Because it is not zero, it will execute the "GOTO AGAIN" instruction. The "GOTO AGAIN" goes back to the start of the loop. Next, it decrements, our counter becomes 8, and, because it is not zero, it executes the GOTO. It goes on like that until the counter becomes zero. Upon the counter becoming zero, it skips the GOTO, which gets it out of the loop, and executes the "MOVWF PORTB" instruction. Notice that we use "DECFSZ COUNT, F" and not "DECFSZ COUNT, W" because we want the count value to change for the next iteration. We will never get out of the loop if we use "DECFSZ COUNT, W" because COUNT = 9 and the decrement value is placed in WREG.

Figure 3-1. Flowchart for the DECFSZ Instruction



BNZ – Branch if not zero

Example 3-2

Write a program to (a) clear WREG, then (b) add 3 to WREG ten times.

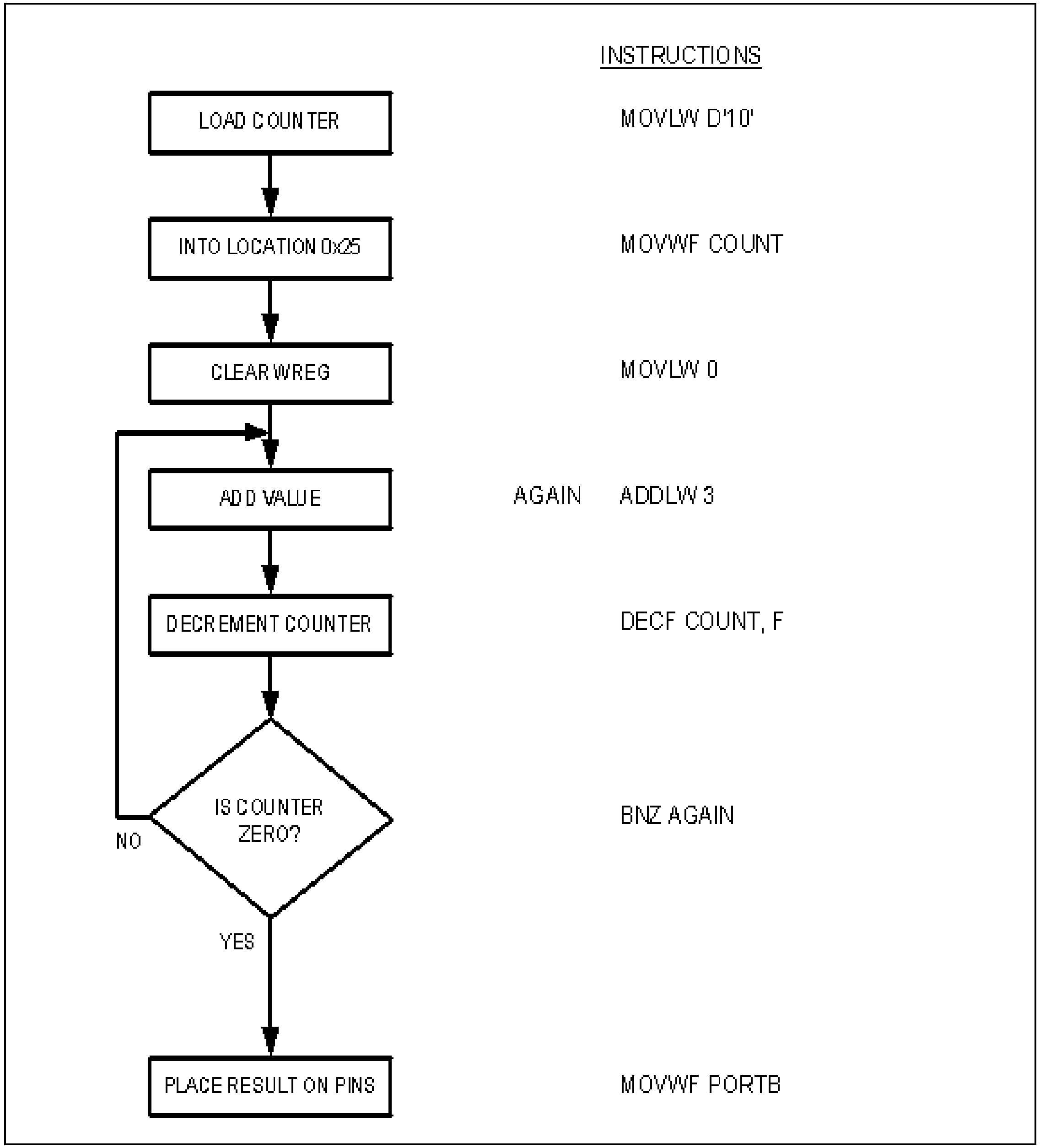
Use the zero flag and BNZ.

Solution:

```
;this program adds value 3 to the WREG ten times

      COUNT EQU 0x25      ;use loc 25H for counter

      MOVLW d'10'         ;WREG = 10 (decimal) for counter
      MOVWF COUNT         ;load the counter
      MOVLW 0             ;WREG = 0
AGAIN ADDLW 3             ;add 03 to WREG (WREG = sum)
      DECF COUNT, F       ;decrement counter
      BNZ AGAIN           ;repeat until COUNT = 0
      MOVWF PORTB         ;send sum to PORTB SFR
```



INSTRUCTIONS
LOAD COUNTER
MOVLW D'10'
INTO LOCATION 0x25
MOVWF COUNT
CLEAR WREG
MOVLW 0
ADD VALUE
AGAIN
ADDLW 3
DECREMENT COUNTER
DECF COUNT, F
IS COUNTER ZERO?
NO
BNZ AGAIN
YES
PLACE RESULT ON PINS
MOVWF PORTB

Maximum Number of Times for loop

Example 3-3

What is the maximum number of times that the loop in Example 3-2 can be repeated?

Solution:

Because location COUNT in fileReg is an 8-bit register, it can hold a maximum of FFH (255 decimal); therefore, the loop can be repeated a maximum of 255 times. See Example 3-4 to bypass this limitation.

Loop Inside a Loop – Nested Loop

Example 3-4

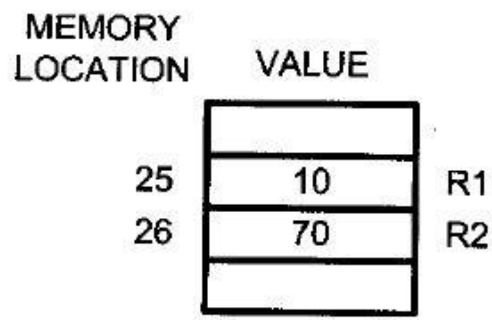
Write a program to (a) load the PORTB SFR register with the value 55H, and (b) complement Port B 700 times.

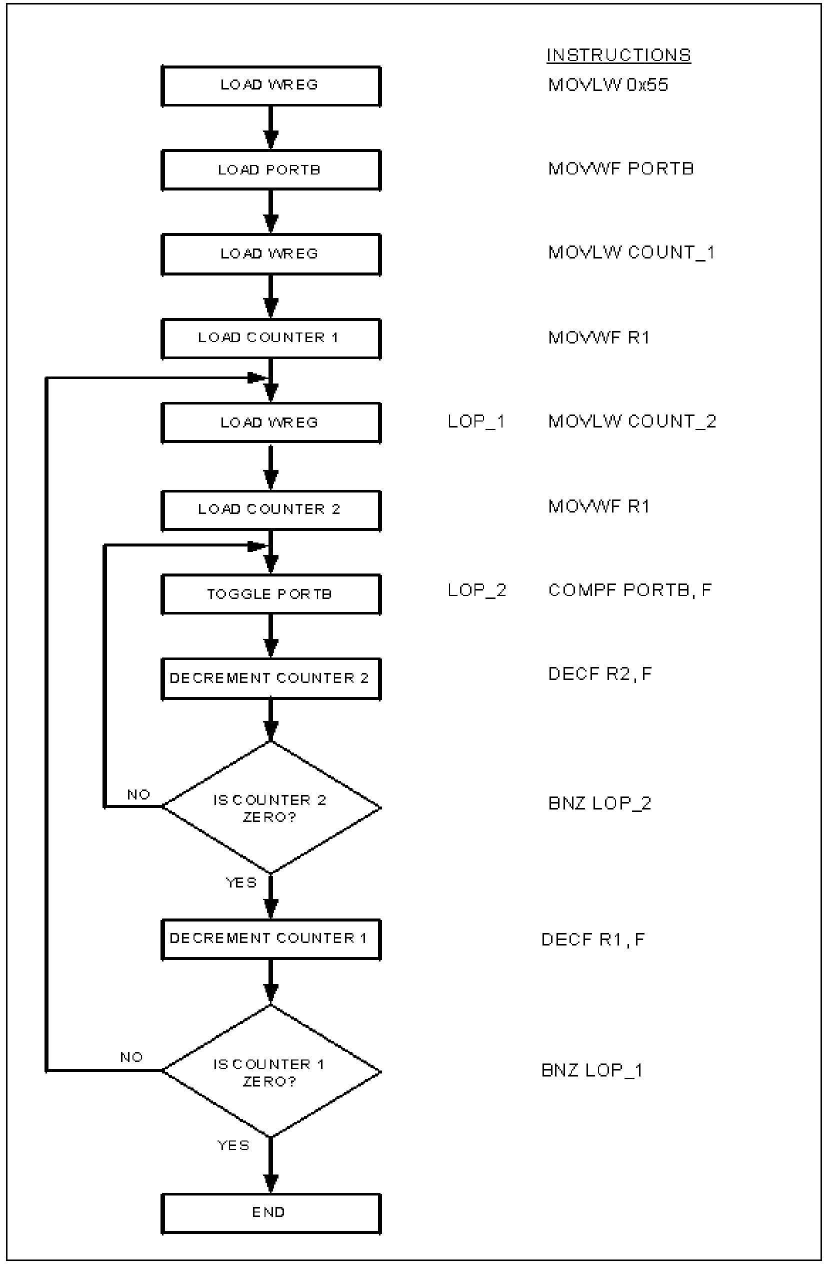
Solution:

Because 700 is larger than 255 (the maximum capacity of any register), we use two registers to hold the count. The following code shows how to use fileReg locations 25H and 26H as a register for counters.

```
        R1  EQU  0x25
        R2  EQU  0x26
        COUNT_1  EQU  d'10'
        COUNT_2  EQU  d'70'
        MOVLW  0x55          ;WREG = 55h
        MOVWF  PORTB         ;PORTB = 55h
        MOVLW  COUNT_1       ;WREG = 10, outer loop count value
        MOVWF  R1            ;load 10 into loc 25H (outer loop count)
LOP_1   MOVLW  COUNT_2       ;WREG = 70, inner loop count value
        MOVWF  R2            ;load 70 into loc 26H
LOP_2   COMPF  PORTB, F      ;complement Port B SFR
        DECF  R2, F          ;dec fileReg loc 26 (inner loop)
        BNZ  LOP_2           ;repeat it 70 times
        DECF  R1, F          ;dec fileReg loc 25 (outer loop)
        BNZ  LOP_1           ;repeat it 10 times
```

In this program, fileReg location 0x26 is used to keep the inner loop count. In the instruction "`BNZ LOP_2`", whenever location 26H becomes 0 it falls through and "`DECF R1, F`" is executed. This instruction forces the CPU to load the inner count with 70 if it is not zero, and the inner loop starts again. This process will continue until location 25 becomes zero and the outer loop is finished.




INSTRUCTIONS
LOAD WREG
MOVLW 0x55
LOAD PORTB
MOVWF PORTB
LOAD WREG
MOVLW COUNT_1
LOAD COUNTER 1
MOVWF R1
LOAD WREG
LOP_1
MOVLW COUNT_2
LOAD COUNTER 2
MOVWF R1
TOGGLE PORTB
LOP_2
COMPF PORTB, F
DECREMENT COUNTER 2
DECF R2, F
NO
IS COUNTER 2 ZERO?
BNZ LOP_2
YES
DECREMENT COUNTER 1
DECF R1, F
NO
IS COUNTER 1 ZERO?
BNZ LOP_1
YES
END

Other Conditional Jumps

Table 3-1: PIC Conditional Branch (Jump) Instructions

Instruction	Action
BC	Branch if C = 1
BNC	Branch if C ≠ 0
BZ	Branch if Z = 1
BNZ	Branch if Z ≠ 0
BN	Branch if N = 1
BNN	Branch if N ≠ 0
BOV	Branch if OV = 1
BNOV	Branch if OV ≠ 0

BZ – Branch if Z = 1

Example 3-5

Write a program to determine if fileReg location 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 zero
        MOVLW 0x55
        MOVWF MYLOC                 ;put 0x55 if MYLOC has zero value
NEXT    ...
```

BNC – Branch If No Carry (CY = 0)

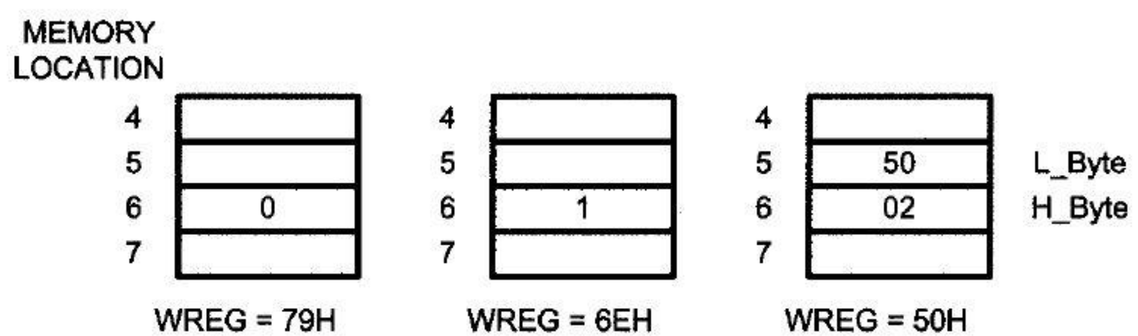
Example 3-6

Find the sum of the values 79H, F5H, and E2H. Put the sum in fileReg locations 5 (low byte) and 6 (high byte).

Solution:

```
L_Byte EQU 0x5            ;assign RAM loc 5 to L_byte of sum
H_Byte EQU 0x6            ;assign RAM loc 6 to H_byte of sum

       ORG  0h
       MOVLW 0x0          ;clear WREG(WREG = 0)
       MOVWF H_Byte       ;H_Byte = 0
       ADDLW 0x79         ;WREG = 0 + 79H = 79H, C = 0
       BNC   N_1          ;if C = 0, add next number
       INCF  H_Byte,F     ;C = 1, increment (now H_Byte = 0)
N_1    ADDLW 0xF5         ;WREG = 79 + F5 = 6E and C = 1
       BNC   N_2          ;branch if CY = 0
       INCF  H_Byte,F     ;C = 1, increment (now H_Byte = 1)
N_2    ADDLW 0xE2         ;WREG = 6E + E2 = 50 and C = 1
       BNC   OVER         ;branch if C = 0
       INCF  H_Byte,F     ;C = 1, increment (now H_Byte = 2)
OVER   MOVWF L_Byte       ;now L_Byte = 50H, and H_Byte = 02
       END
```



Short Jumps

- All conditional jumps are short jumps.
- Meaning that the address of the target must be within 256 bytes of the contents of the PC.
- 2-byte instructions.
- Target address = (2^{nd} byte of instructions×2)+PC
- The 2^{nd} byte can be a value from -127 to +128

Example 3-7

Using the following list file of Example 3-6, verify the jump forward address calculation.

```
Line  PC          Opcode                          Mnemonic Operand

LOC    OBJECT LINE   SOURCE TEXT
       CODE
  VALUE

  00000005    00001 L_Byte EQU 0x5           ;assign RAM Loc 5 to L_byte of sum
  00000006    00002 H_Byte EQU 0x6           ;assign RAM Loc 6 to H_byte of sum
              00003
000000        00004          ORG 0h
000000 0E00   00005          MOVLW  0x0      ;clear WREG(WREG=0)
000002 6E06   00006          MOVWF  H_Byte   ;H_Byte = 0
000004 0F79   00007          ADDLW  0x79     ;WREG = 0 + 79H = 79H, C = 0
000006 E301   00008          BNC    N_1      ;if C = 0, add next number
000008 2A06   00009          INCF   H_Byte,F ;C = 1, increment (now H_Byte = 0)
00000A 0FF5   00010 N_1      ADDLW  0xF5     ;WREG = 79 + F5 = 6E and C = 1
00000C E301   00011          BNC    N_2      ;branch if CY = 0
00000E 2A06   00012          INCF   H_Byte,F ;C = 1, increment (now H_Byte = 1)
000010 0FE2   00013 N_2      ADDLW  0xE2     ;WREG = 6E + E2 = 50 and C = 1
000012 E301   00014          BNC    OVER     ;branch if C = 0
000014 2A06   00015          INCF   H_Byte,F ;C = 1, increment (now H_Byte = 2)
000016 6E05   00016 OVER     MOVWF  L_Byte   ;now L_Byte = 50H, and H_Byte = 02
              00017          END
```

Solution:

First notice that the BNC instruction jumps forward. The target address for a forward jump is calculated by adding the PC of the following instruction to the second byte of the branch instruction times 2. Recall that each instruction takes 2 bytes. In line 6 the instruction "`BNC N_1`" has an opcode of E3 and an operand of 01 at the addresses of 000006 and 000007. The 01 × 02 = 02 is the relative address, relative to the address of the next instruction INCF, which is 000008. By adding 000002 to 000008, the target address of the label N_1, which is 00000A, is generated. In the same way for line 000011, the "`BNC N_2`" instruction, and line 000014, the "`BNC OVER`" instruction jumps forward because the relative value is positive.

Example 3-8

Verify the calculation of backward jumps for the listing of Example 3-2, shown below.

Solution:

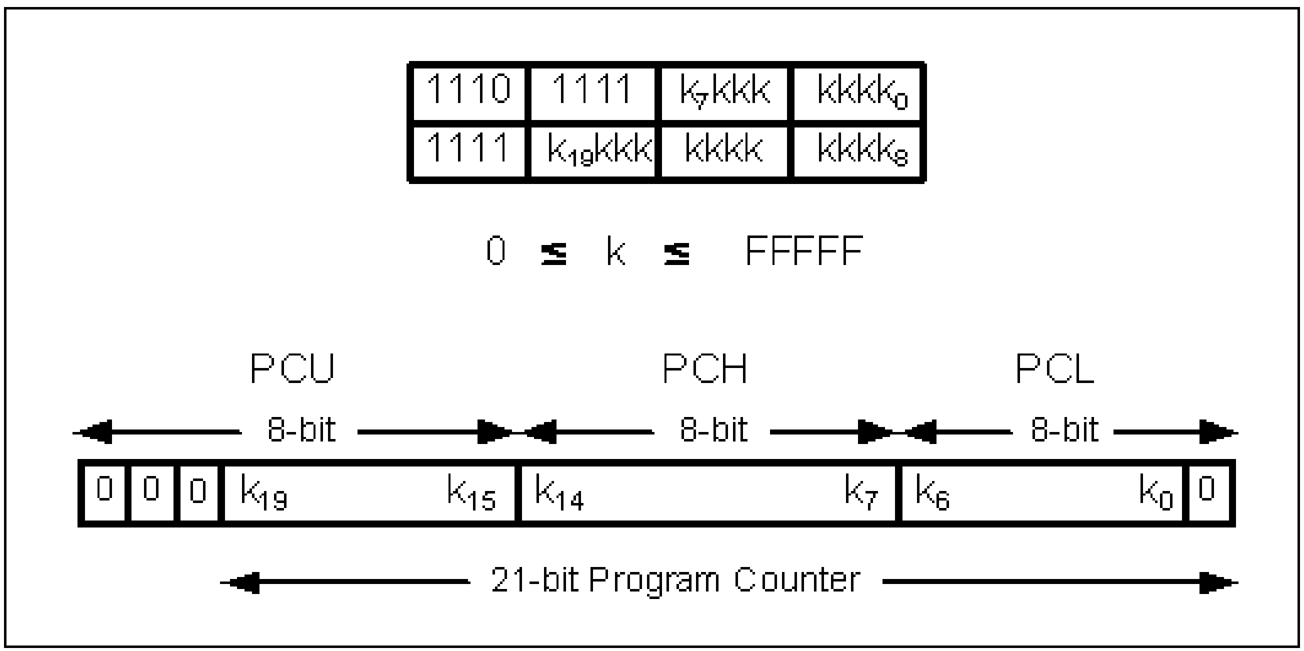
```
LOC    OBJECT LINE   SOURCE TEXT
       CODE
  VALUE

  00000025    00001  COUNT  EQU 0x25        ;use loc 25H for counter
000000        00002         ORG 0h
000000 0E0A   00003         MOVLW d'10'     ;WREG = 10 (decimal) for counter
000002 6E25   00004         MOVWF COUNT     ;load the counter
000004 0E00   00005         MOVLW 0         ;WREG = 0
000006 0F03   00006  AGAIN  ADDLW 3         ;add 03 to WREG (WREG = sum)
000008 0625   00007         DECF COUNT, F   ;decrement counter
00000A E1FD   00008         BNZ AGAIN       ;repeat until COUNT = 0
00000C 6E81   00009         MOVWF PORTB     ;send sum to PORTB SFR
              00010         END
```

In the program list, "BNZ AGAIN" has opcode E1 and relative address FDH. The FDH gives us –3, which means the displacement is $-3 \times 2 = -6$. When the relative address of –6 is added to 00000CH, the address of the instruction below the byte, we have –6 + 0CH = 06H (the carry is dropped). Notice that 000006 is the address of the label AGAIN. FDH is a negative number and that means it will branch backward. For further discussion of the addition of negative numbers, see Chapter 5.

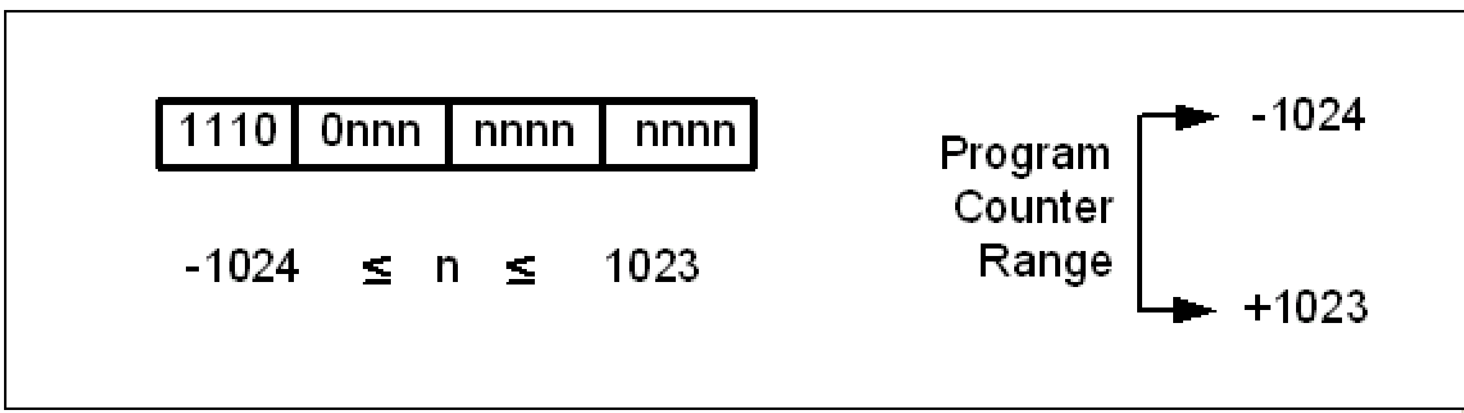
Unconditional Branch Instruction – GOTO

- Long jump – can go any memory locations in the 2M address space of the PIC18.
- A 4-byte instruction.



Unconditional Branch Instruction – BRA (Branch)

- A 2-byte instruction.
- The first 5 bits are the opcode.
- The rest lower 11 bits are the relative address of the target address



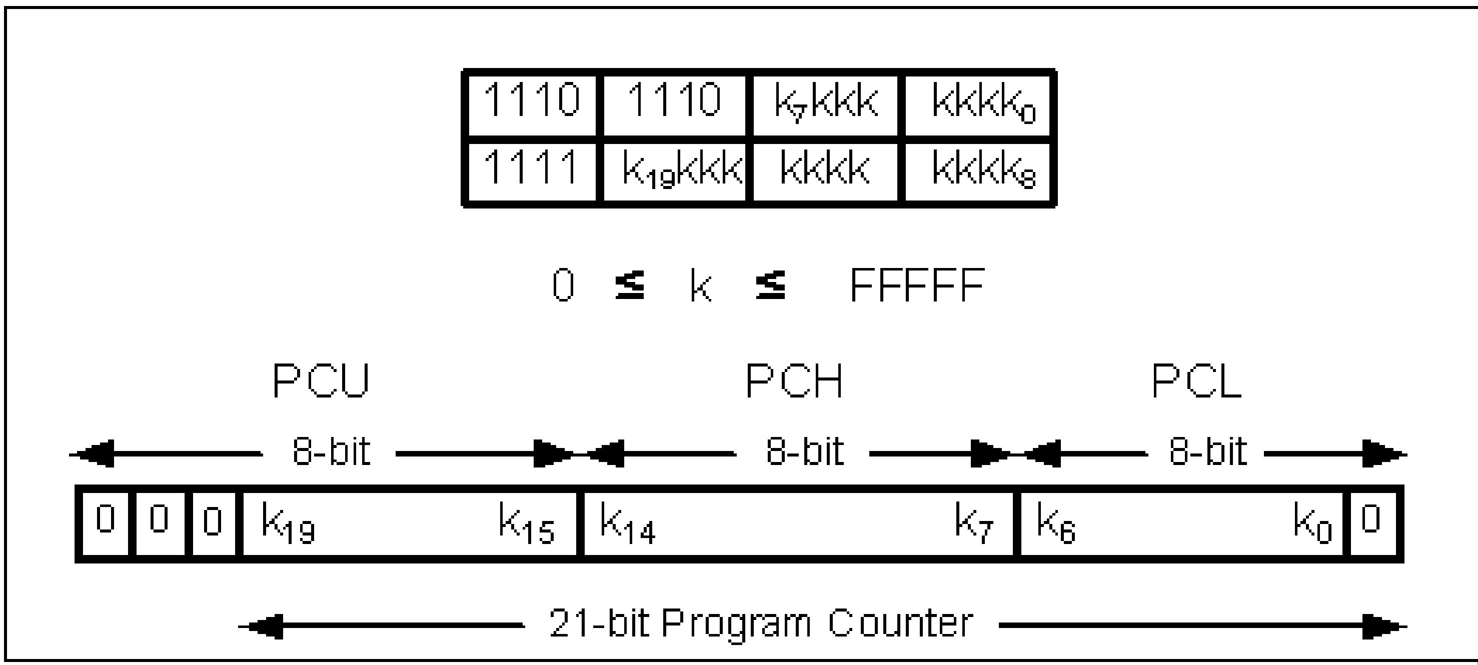
- GOTO to itself using $ sign.
- HERE GOTO HERE

  GOTO $
- OVER BRA OVER

  BRA $

CALL Instructions and Stack

- A 4-byte instruction.
- Long call.



Simplified view of a PIC microcontroller

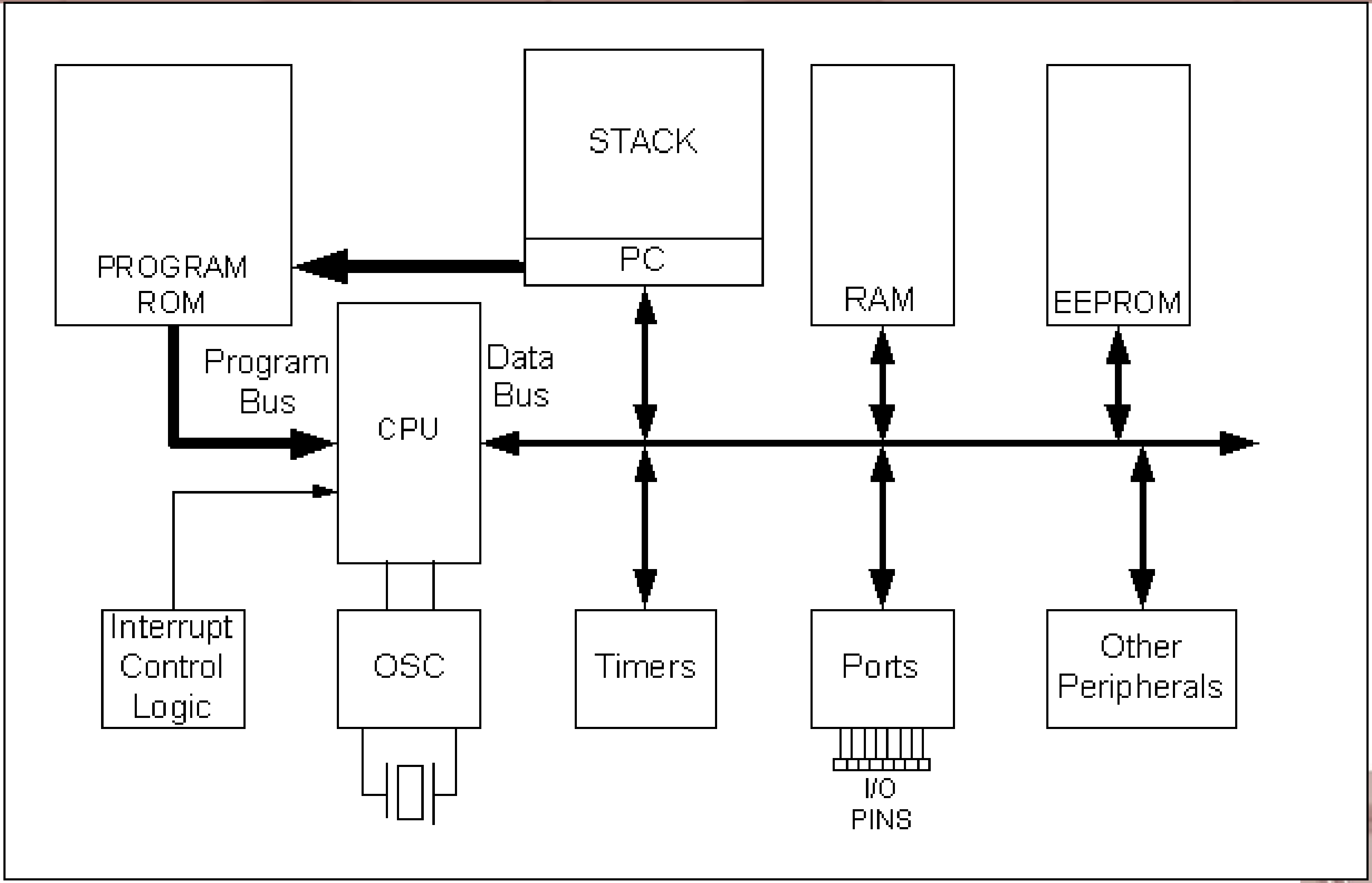
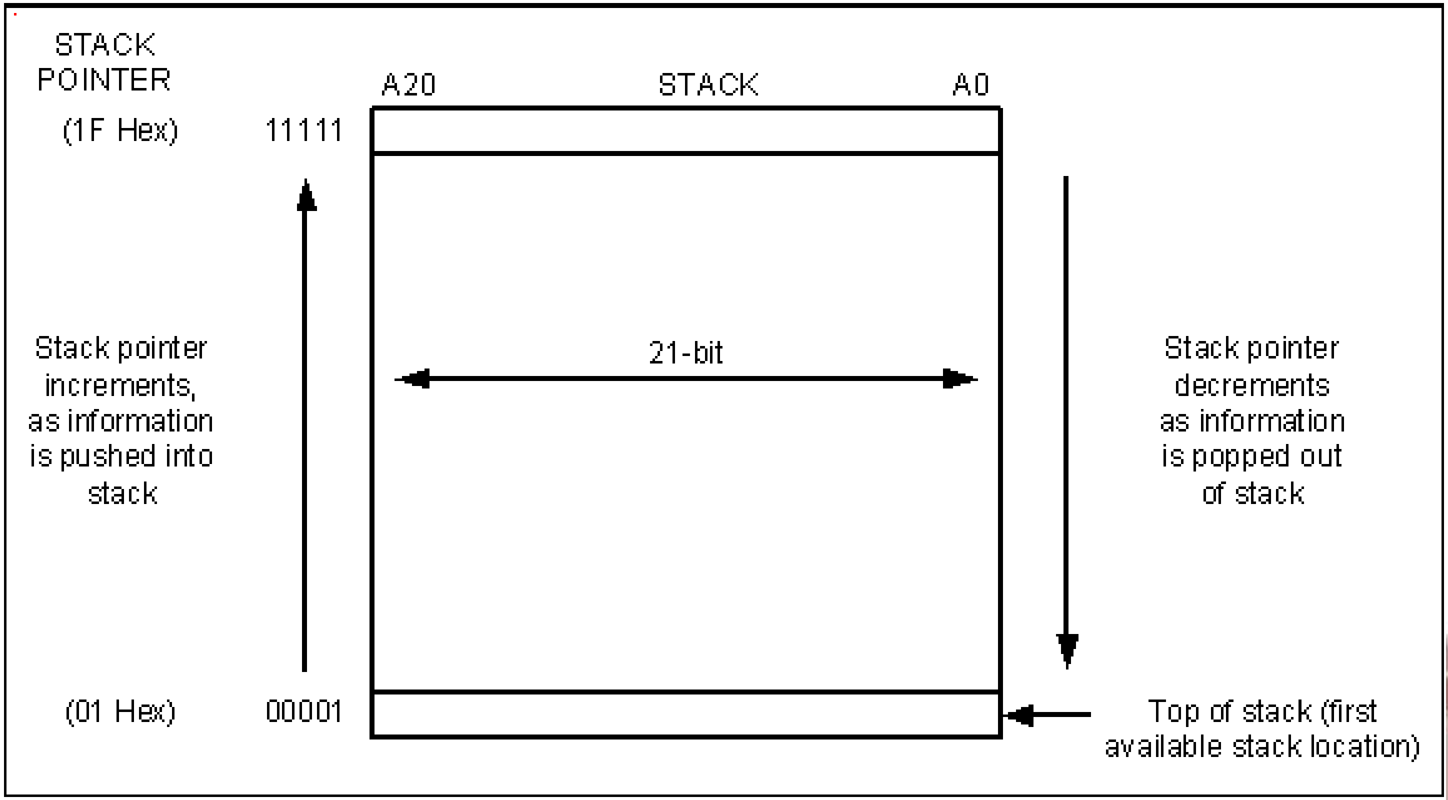


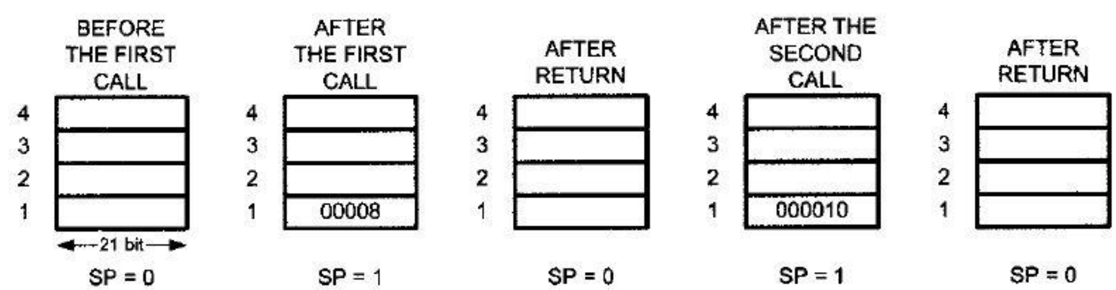
Figure 3-7. PIC Stack 31 × 21

- A 5-bit stack pointer (SP) with initial value 0.
- Last-In-First-Out (LIFO)



```
LOC   OBJECT CODE      LINE SOURCE TEXT
 VALUE

                      00001 #DEFINE PORTB 0xF81
  00000008            00002 MYREG EQU 0x08        ;use location 08 as counter
                      00003
                      00004
  000000              00005        ORG    0
000000 0E55           00006  BACK  MOVLW  0x55    ;load WREG with 55H
000002 6E81           00007        MOVWF  PORTB   ;send 55H to port B
000004 EC80 F001      00008        CALL   DELAY   ;time delay
000008 0EAA           00009        MOVLW  0xAA    ;load WREG with AA (in hex)
00000A 6E81           00010        MOVWF  PORTB   ;send AAH to port B
00000C EC80 F001      00011        CALL   DELAY
000010 EF00 F000      00012        GOTO   BACK    ;keep doing this indefinitely
                      00013
                      00014 ;------- this is the delay subroutine
                      00015
  000300              00016        ORG    300H    ;put delay at address 300H
000300 0EFF           00017  DELAY MOVLW  0xFF    ;WREG = 255,the counter
000302 6E08           00018        MOVWF  MYREG
000304 0000           00019  AGAIN NOP            ;no op wastes clock cycles
000306 0000           00020        NOP
000308 0608           00021        DECF   MYREG, F
00030A E1FC           00022        BNZ    AGAIN   ;repeat until MYREG becomes 0
00030C 0012           00023        RETURN         ;return to caller
                      00024        END            ;end of asm file
```

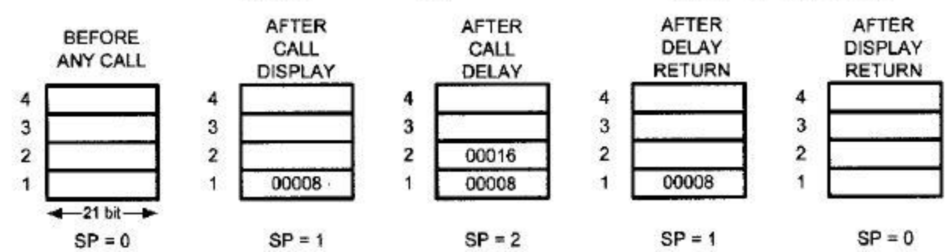


Example 3-11

Write a program to count up from 00 to FFH and send the count to SFR of Port B. Use one CALL subroutine for sending the data to Port B and another one for time delay. Put a time delay in between each issuing of data to Port B.

Solution:

```
LOC    OBJECT CODE  LINE   SOURCE TEXT
  VALUE
                    00001         list P=PIC18F458
                    00002  #include P18F458.INC
                    00003
  00000007          00004  COUNT  EQU    0x07   ;use location 07 for count-up
  00000008          00005  MYREG  EQU    0x08   ;use location 08 for delay
                    00006
000000              00007  ORG    0
000000 0E00         00008         MOVLW  0                 ;WREG = 0
000002 6E07         00009         MOVWF  COUNT             ;count = 0
000004 EC06 F000    00010  BACK   CALL   DISPLAY
000008 EF02 F000    00011         GOTO   BACK
                    00012
                    00013  ;------- increment and put it in PORTB
00000C 2A07         00014  DISPLAY INCF  COUNT,F       ;increment count
00000E C007 FF81    00015         MOVFF  COUNT,PORTB   ;send it to PORTB
000012 EC80 F001    00016         CALL   DELAY
000016 0012         00017         RETURN               ;return to caller
                    00018
                    00019  ;------- this is the delay subroutine
000300              00020  ORG    300H   ;put time delay at address 300H
000300 0EFF         00021  DELAY  MOVLW  0xFF          ;WREG = 255, the counter
000302 6E08         00022         MOVWF  MYREG
000304 0000         00023  AGAIN  NOP    ;no operation wastes clock cycles
000306 0000         00024         NOP
000308 0000         00025         NOP
00030A 0608         00026         DECF   MYREG,F
00030C E1FB         00027         BNZ    AGAIN  ;repeat until MYREG becomes 0
00030E 0012         00028         RETURN               ;return to caller
                    00029         END                  ;end of asm file
```



RCALL (Relative Call)

- A 2-byte instruction.
- Only 11 bits of the 2 bytes are used for the address.
- The target address of RCALL must be within a 2K range.

Example 3-12

Rewrite the main part of Example 3-9 as efficiently as you can.

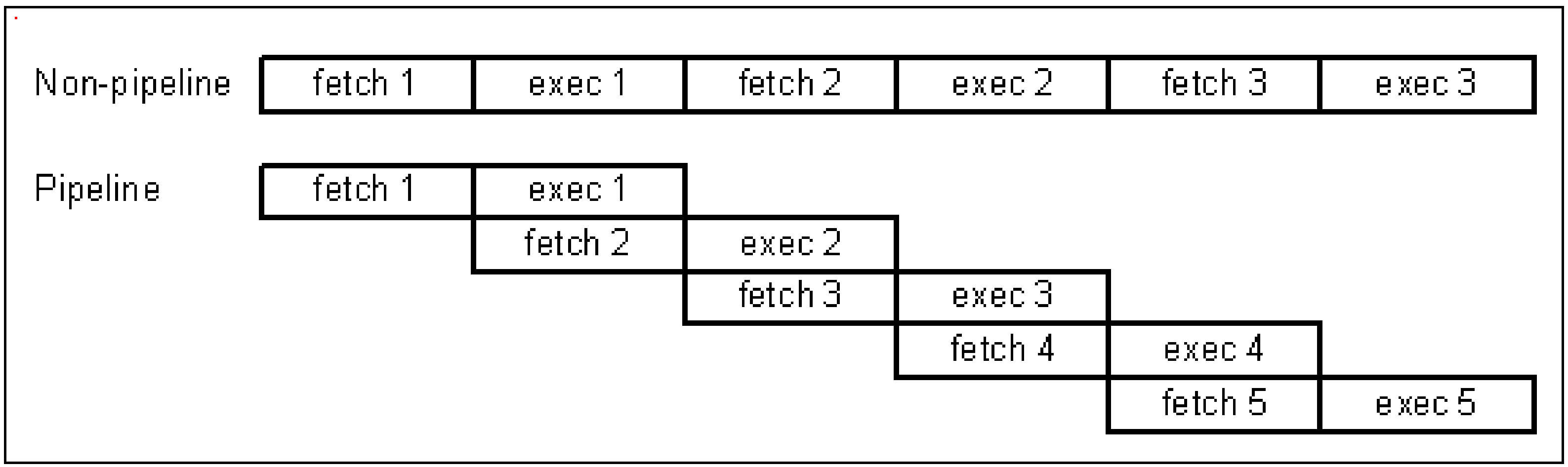
Solution:

```
       MYREG  EQU 0x08
       ORG    0
       MOVLW  0x55          ;load WREG with 55H
BACK   MOVWF  PORTB               ;issue value in PORTB SFR
       RCALL  DELAY         ;time delay
       COMPF  PORTB,F             ;complement Port B SFR
       BRA    BACK          ;keep doing this indefinitely
;————————this is the delay subroutine
DELAY  MOVLW  0xFF          ;WREG = 255, the counter
       MOVWF  MYREG
AGAIN  NOP                  ;no operation wastes clock cycles
       NOP
       DECF   MYREG,F
       BNZ    AGAIN         ;repeat until MYREG becomes 0
       RETURN               ;return to caller (MYREG = 0)
       END                  ;end of asm file
```

Delay Calculation

- Two factors that affect the accuracy of the delay:
  - The crystal frequency
  - The PIC design
- An instruction in one cycle
  - Use Harvard architecture
  - Use RISK features
  - Use piplining

Figure 3-9. Pipeline vs. Non-pipeline



- Instruction cycles (machine cycles) – In PIC18, one instruction cycle consists of four oscillator periods.
- Brach penalty – CPU flushes out the instruction queue.

Example 3-14

The following shows the crystal frequency for three different PIC-based systems. Find the period of the instruction cycle in each case.

(a) 4 MHz (b) 16 MHz (c) 20 MHz

Solution:

(a) 4/4 = 1 MHz; instruction cycle is 1/1 MHz = 1 μs (microsecond)
(b) 16 MHz/4 = 4 MHz; instruction cycle = 1/4 MHz = 0.25 μs = 250 ns (nanosecond)
(c) 20 MHz/4 = 5 MHz; instruction cycle = 1/5 MHz = 0.2 μs = 200 ns

Example 3-16

Find the size of the delay of the code snippet below if the crystal frequency is 4 MHz:

Solution:

From Appendix A, we have the following machine cycles for each instruction of the DELAY subroutine:

```
                                        Instruction Cycle
MYREG EQU   0x08          ;use location 08 as counter

DELAY     MOVLW     0xFF                   1
          MOVWF     MYREG                  1

AGAIN     NOP                              1
          NOP                              1
          DECF      MYREG,F                1
          BNZ       AGAIN                  2

          RETURN                           1
```

Therefore, we have a time delay of $[(255 \times 5) + 1 + 1 + 1] \times 1\ \mu s = 1278\ \mu s$.
Notice that BNZ takes two instruction cycles if it jumps back, and takes only one when falling through the loop. That means the above number should be 1277 μs.

Example 3-18

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

```
MYREG EQU   0x08               ;use location 08 as counter

        ORG     0
BACK    MOVLW   0x55           ;load WREG with 55H
        MOVWF   PORTB          ;send 55H to port B
        CALL    DELAY          ;time delay
        MOVLW   0xAA           ;load WREG with AA (in hex)
        MOVWF   PORTB          ;send AAH to port B
        CALL    DELAY
        GOTO    BACK           ;keep doing this indefinitely

;------- this is the delay subroutine
        ORG     300H           ;put time delay at address 300H
DELAY   MOVLW   0xFA           ;WREG = 250, the counter
        MOVWF   MYREG
AGAIN   NOP                    ;no operation wastes clock cycles
        NOP
        NOP
        DECF    MYREG, F
        BNZ     AGAIN          ;repeat until MYREG becomes 0
        RETURN                 ;return to caller
        END                    ;end of asm file
```

Solution:

From Appendix A, we have the following machine cycles for each instruction of the DELAY subroutine:

```
                                   Instruction Cycle

DELAY   MOVLW   0xFA                      1
        MOVWF   MYREG                     1

AGAIN   NOP                               1
        NOP                               1
        NOP                               1
        DECF    MYREG, F                  1
        BNZ     AGAIN                     2

        RETURN                            1
```

Therefore, we have a time delay of $[(250 \times 6) + 1 + 1 + 1] \times 1\ \mu s = 1503\ \mu s$.

Example 3-18

For a instruction cycle of 1 μs, find the time delay in the following subroutine:

```
R2      EQU   0x7
R3      EQU   0x8
DELAY                          Instruction Cycle
          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
```

Solution:

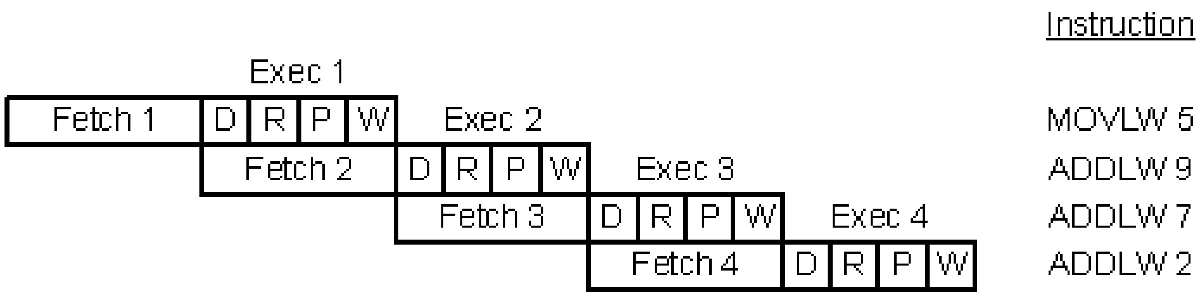
For the HERE loop, we have (5 × 250) 1 μs = 1250 μs. The AGAIN loop repeats the HERE loop 200 times; therefore, we have 200 × 1250 μs = 250000 μs, if we do not include the overhead. However, the following instructions of the outer loop add to the delay:

```
AGAIN     MOVLW     D'250'        1
          MOVWF     R3            1
          .....
          DECF      R2, F         1
          BNZ       AGAIN         2
```

The above instructions at the beginning and end of the AGAIN loop add 5 × 200 × 1 μs = 1000 μs to the time delay. We should also subtract 200 μs for the times BNZ HERE falls through. As a result we have 250000 + 1000 – 200 = 250800 μs = 250.8 milliseconds for the total time delay associated with the above DELAY subroutine. Notice that in the case of a nested loop, as in all other time delay loops, the time is approximate because we have ignored the first few instructions and the last instruction, RETURN, in the subroutine. NOP is a 2-byte instruction. There are 11 instructions in the above DELAY program, and all the instructions are 2-byte instructions. That means that the loop delay takes 22 bytes of ROM code space.

Pipeline Activity

Q1	Q2	Q3	Q4
DECODE	READ	PROCESS	WRITE TO



D = Decode the instruction

R = Read the operand

P = Process (eg. ADDLW)

W = Write the result to destination register