Chapter 6
Assembly Language Programming for PIC18F: Part 1

Chung-Ping Young (楊中平)

Introduction

- The assembler reads each instruction of an assembly language program as ASCII characters and translates them into the respective binary op-codes.
- For example, the PIC18F assembler translates the SLEEP instruction into its 16-bit binary op-code 0000 0000 0000 0011 (0003 in hex), as depicted in Table 6.1.

Introduction

TABLE 6.1 Conversion of SLEEP instruction into its Binary Op-Code

Assembly Code	Binary Form of ASCII Codes as Seen by the Assembler	Binary Op-Code Created by the MPLAB PIC18F Assembler
S	0101 0011	
L	0100 1000	0000 0000 0000 0011
E	0100 0101	
E	0100 0101	
P	0101 0000	

Introduction

- The MPLAB assembler/simulator is used to assemble and debug all PIC18F assembly language programs in this book.
- The MPLAB, like most assemblers, allows the programmer to use a special symbol or delimiter to indicate the beginning or end of each field. Typical delimiters used are spaces, commas, and semicolons.

Introduction

- The MPLAB assembler recognizes a number without any prescript or postscript as a hexadecimal number. For example, with the MPLAB assembler, the user can define a hexadecimal number usually in three ways using:
  - 0x before the number
  - H after the number
  - default (without any prescript or postscript)

Introduction

- The MPLAB uses D before a ‘number’ to specify a decimal number. Decimal number 60 can be represented as D’60’.
- A binary number is specified by the MPLAB using B before the ‘Number’. 8-bit binary number 01011100 can be represented by the MPLAB as B’01011100’.
- ASCII characters are represented using MPLAB by the symbols ‘ ‘ or A‘ ‘. For example, PIC18F is represented as ‘PIC18F’ or A’PIC18F’ to be recognized as ASCII characters in MPLAB.

Introduction

- Assemblers use pseudoinstructions or directives to make the formatting of the edited text easier.
- ORIGIN (ORG) The directive ORG specifies the starting address of a program or data. For example, after assembling the following statements, the MPLAB will place the assembled code for MOVLW 0x50 starting at address 0x100:

```
ORG     0x100
MOVLW   0x50
```

Introduction

- Equate (EQU). The EQU assigns a value in its operand field to an address in its label field. This allows the user to assign a numerical value to a symbolic name. A typical example of EQU is TEST EQU 0x20, which assigns the value 20 in hexadecimal to the label START.

```
TEST    EQU         0x20
        MOVLW       TEST
        MOVWF       TEST
```

The instruction MOVLW TEST, in the above, moves the constant 0x20 into WREG. The instruction MOVWF TEST, on the other hand, moves the contents of WREG into the data register with address 0x20.

Introduction

- Define Byte (DB). DB is generally used to set a memory location to a certain byte value.

```
START    DB    0x45
```

will store the data value 45 hex to the address START. The DB directive can be used to generate a table of data as follows:

```
ORG    0x70
DB     0x20,0x30,0x40,0x50
```

In this case, 20 hex is the first data of the memory location 70 hex; 30 hex, 40 hex, and 50 hex occupy the next three memory locations. Therefore, the data in memory will look like

70	20
71	30
72	40
73	50

Introduction

- Define Word (DW). The directive DW is typically used to assign a 16-bit value to two memory locations. For example,

```
ORG     0x50
DW      0x4AC2
```

will assign C2 to location 50 hex and 4A to location 51 hex. It is assumed that the assembler will assign the low byte first (C2) and then the high byte (4A). The DW directive can be used to generate a table of 16-bit data as follows:

```
ORG     0x80
DW      0x5000,0x6000, 0x7000
```

Introduction

- In this case, the three 16-bit values 5000 hex, 6000 hex, and 7000 hex are assigned to memory locations starting at the address 80 hex. That is, the array would look like this:

80	00
81	50
82	00
83	60
84	00
85	70

Introduction

- INCLUDE. The directive INCLUDE includes source code or a file from the MPLAB library for a specific device. This will allow the MPLAB to assemble a program for that device.
- Using INCLUDE <P18F4321.INC> at the beginning of a program will add appropriate files from the MPLAB library required to assemble the program.

Introduction

- For example, the INCLUDE <P18F4321.INC> includes files which will insert SFR addresses during the assembly. This will allow the programmer to use variable names rather than their physical addresses. The programmer can use variable names such as PORTA in the program, and the assembler will insert its SFR address (F80H) whenever it will encounter PORTA in the program.

Introduction

```
Label Field     Mnemonic Field   Operand Field          Comment Field
-------------------------------------------------------------------------
                INCLUDE          <P18F4321.INC>
SUM             EQU              0x40
                ORG              0x100          ; STARTING  ADDRESS
                MOVLW            0x02            ; MOVING 2 INTO WREG
                ADDLW            0x05           ; ADDING WREG and 5
                MOVWF            SUM            ; STORE  RESULT IN SUM
                SLEEP                           ; HALT
                END                             ; END OF PROGRAM
```

Introduction

- SUM EQU 0x40 assigns 40 (hex) to label SUM. ORG 0x100 assembles the program at address 100 (hex). MOVLW 0x02 moves 02 (hex) into WREG. ADDLW 0x05 adds the contents of WREG with 05 (hex), and stores the 8-bit result, 07 (hex) in WREG.

Introduction

- The instruction MOVWF SUM stores the 8-bit contents of WREG (07 hex) in file register SUM with address 40 (hex). The SLEEP instruction halts the program. Finally, the MPLAB assembler directive “end” indicates the end of the program. Note that the directive “end” must be present at the end of all PIC18F assembly language programs.

Introduction

- The assembler converts the source file into an object file containing the binary codes or machine codes that the PIC18F will understand. In MPLAB, the source file must be stored with a file extension called .ASM.
- Suppose that the programmer stores the source file as SUM.ASM. To assemble the program, the source file SUM.ASM is presented as input to the assembler. The assembler typically generates two files: SUM.OBJ (object file) and SUM.LST (list file).

Introduction

- SUM.OBJ is an *object file*, a binary file containing the machine code and data that correspond to the assembly language program in the source file (SUM.ASM).
- The SUM.LST is a *list file* which shows how the assembler interprets the source file SUM.ASM. The list file may be displayed on the screen.

Introduction

- The SUM.LST file is provided below:

```
                          1:          INCLUDE <P18F4321.INC>
                          2: SUM      EQU     0x40
                          3:          ORG     0x100   ; STARTING  ADDRESS
0100 0E02 MOVLW 0x2       4:          MOVLW   0x02    ; MOVING 2 INTO WREG
0102 0F05 ADDLW 0x5       5:          ADDLW   0x05    ; ADDING WREG and 5
0104 6E40 MOVWF 0x40      6:          MOVWF   SUM     ; STORE RESULT
0106 0003 SLEEP           7:          SLEEP           ; HALT
```

ORG 0x100 (line 3) generates the starting address 0100 in hex.

The machine code (0E02 hex) for the first instruction, MOVLW 0x2, is stored at the address 0100 (hex).

Introduction

```
                                1:          INCLUDE  <P18F4321.INC>
                                2:  SUM     EQU      0x40
                                3:          ORG      0x100   ; STARTING  ADDRESS
0100  0E02  MOVLW 0x2           4:          MOVLW    0x02    ; MOVING 2 INTO WREG
0102  0F05  ADDLW 0x5           5:          ADDLW    0x05    ; ADDING WREG and 5
0104  6E40  MOVWF 0x40          6:          MOVWF    SUM     ; STORE RESULT
0106  0003  SLEEP               7:          SLEEP            ; HALT
```

Since this instruction takes 16-bit (two bytes), the machine code for the next instruction, ADDLW 0x5, starts at address 0102 (hex). Note that the comment fields are not translated by the MPLAB assembler.

Introduction

- When a large program is being developed by a group of programmers, each programmer may write only a portion of the whole program. The individual program parts must be tested and assembled to ensure their proper operation. When all portions of the program are verified for correct operation, their object files must be combined into a single object program using a *Linker*, a program that checks each object file and finds certain characteristics, such as the size in bytes and its proper location in the single object program. The linker also resolves any problems with regard to cross-references to labels.

Introduction

- A *library* of object files is typically used to reduce the size of the source file. The library files may contain frequently used subroutines and/or sections of codes. Rather than writing these codes repeatedly in the source file, a special pseudo instruction is used to tell the assembler that the code must be inserted by the linker at linking time. When linking is completed, the final object file is called an executable (.EXE) file. Finally, a program called a *Loader* loads the .EXE file in memory for execution.

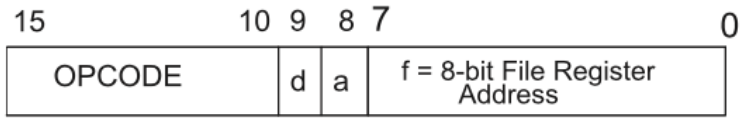
PIC18F Instruction Format

- The instruction set is grouped into four basic categories based on the operations:
  - Byte-oriented operations
  - Bit-oriented operations
  - Literal operations
  - Control operations

PIC18F Instruction Format

TABLE 6.2 General Format for Instructions

Byte-oriented file register operations



Example Instruction

ADDWF f, d, a

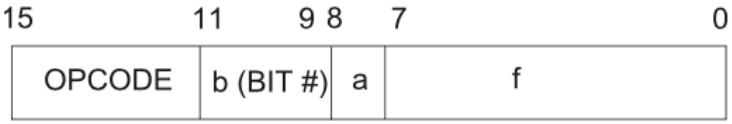
d = 0 for result destination to be WREG register

d = 1 for result destination to be file register (f)

a = 0 means Access Bank, a = 1 means BSR to select the bank

f = 8-bit file register address

Bit-oriented file register operatios



BSF f, b, a

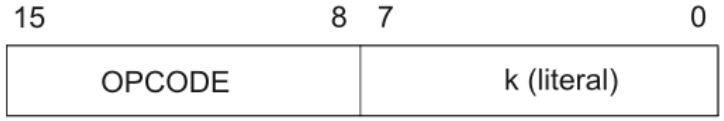
b = 3-bit position of bit in file register (f)

a = 0 to force Access Bank

a = 1 for BSR to select bank

f = 8-bit file register address

Literal operations

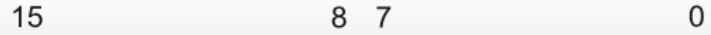


MOVLW 0x2A

k = 8-bit immediate value

Control operations

Branch operation



PIC18F Instruction Format

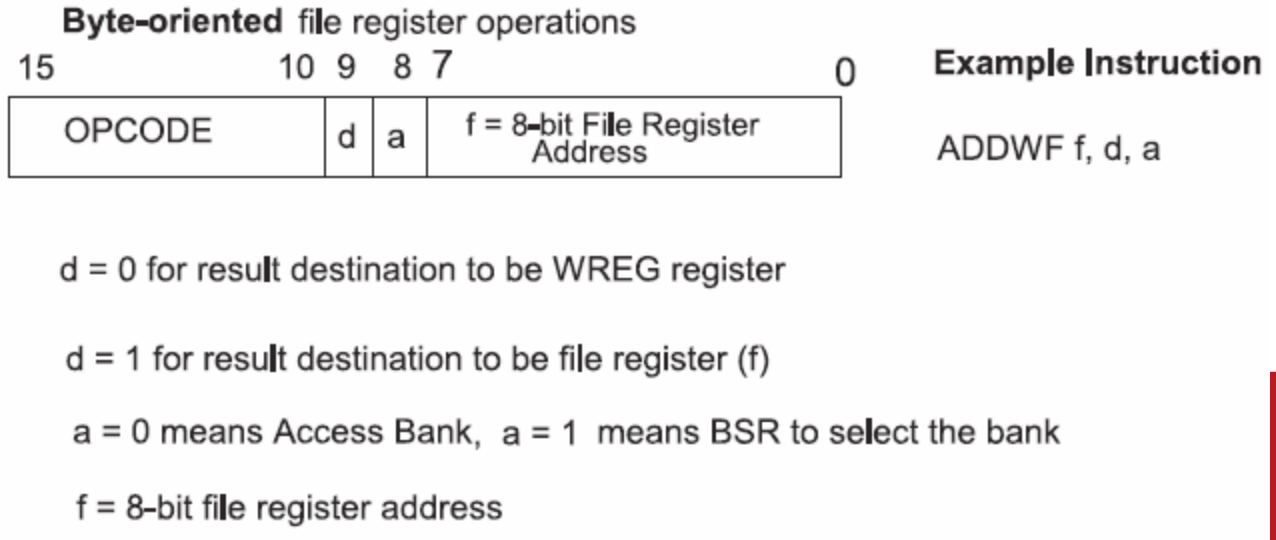
- Most byte-oriented instructions have three operands:
  - 1. The file register (specified by ‘f’)
  - 2. The destination of the result (specified by ‘d’)
  - 3. The access memory (specified by ‘a’)
- The file register designator ‘f’ specifies which file register is to be used by the instruction. The destination designator ‘d’ specifies where the result of the operation is to be placed.

PIC18F Instruction Format

- If ‘d’ is zero, the result is placed in the WREG register.
- If ‘d’ is one, the result is placed in the file register specified in the instruction.
- If a = 0, the file register is the access bank.
- If a = 1, the BSR specifies the data memory bank.

PIC18F Instruction Format

- Consider ADDWF 0x04, 0, 0. This instruction adds the contents of WREG register with the contents of File register 0x04 and stores the result in WREG.
- By inspecting ADDWF 0x04, 0, 0, and comparing with the byte-oriented format of Table 6.2, d = 0, a = 0, and f = 0x04. Since the 6-bit opcode for ADDWF is 001001, the binary 16-bit code for ADDWF 0x04, 0, 0 is 0010010000000100 (2404H).

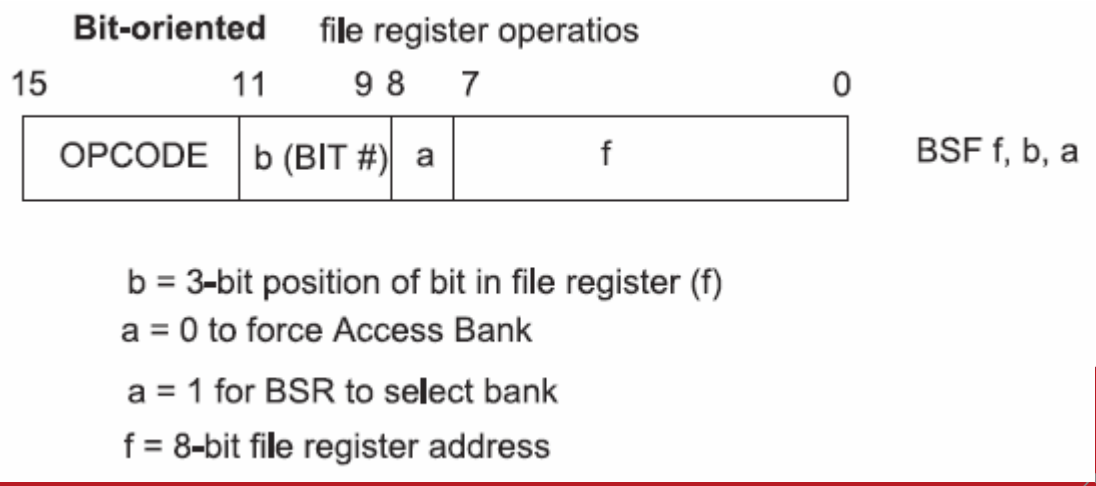


PIC18F Instruction Format

- All bit-oriented instructions have three operands:
  - 1. The file register (specified by ‘f’)
  - 2. The bit in the file register (specified by ‘b’)
  - 3. The accessed memory (specified by ‘a’)
- The bit field designator ‘b’ selects the number of the bit affected by the operation, while the file register designator ‘f’ represents the file address in which the bit is located. If a = 0, the file register is the access bank. On the other hand, if a = 1, the BSR specifies the data memory bank.

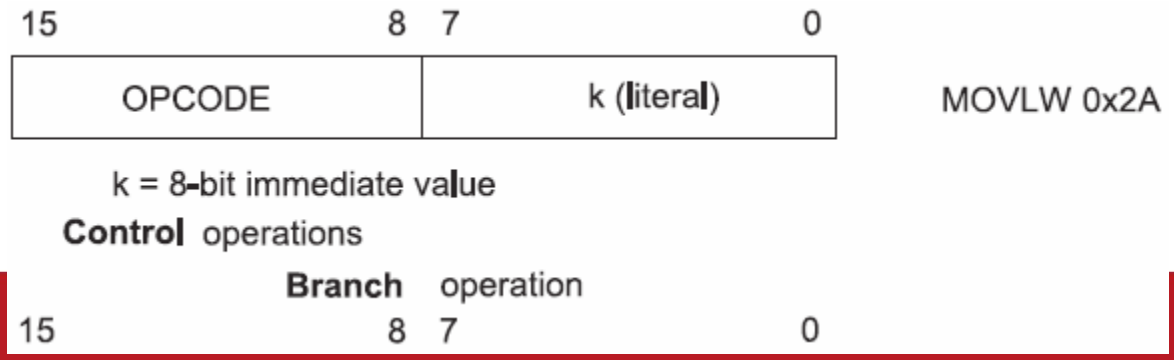
PIC18F Instruction Format

- Consider BSF 0x27, 5, 0. This instruction sets bit 5 to one in file register 0x27 in the access bank.
- Comparing with the bit-oriented format of Table 6.2, b = 101, a = 0, and f = 0x27. Since the 4-bit opcode for BSF is 1000, the binary 16-bit code for BSF 0x27, 5, 0 is 1000101000100111 (8A27H).



PIC18F Instruction Format

- Consider MOVLW 0x2A. This instruction moves 0x2A into the WREG register.
- Comparing with the literal-oriented format of Table 6.2, k = 00101010 (0x2A). Since the 8-bit opcode for MOVLW is 00001110, the binary 16-bit code for MOVLW 0x2A is 0000111000101010 (0E2AH).



PIC18F Instruction Format

- An example of the control instructions includes conditional branch instructions as follows.

  BZ n where ‘n’ is a signed 8-bit offset (specified by ‘n’)

- Consider BZ 0x04 where 04 (hex) is the offset (n). This instruction branches to an address (PC+2+ 2 x 4) if Z = 1; otherwise, it executes the next instruction. Since the 8-bit opcode for BZ is 11100000 and n = 00000100 (0x04), the binary 16-bit code for BZ 0x04 is 1110000000000100 (E004H).

PIC18F Instruction Format

- Most PIC18F instructions are 16-bit (word) wide; only four instructions are double-word instructions. These instructions were made double-word to contain the required information in 32 bits. The double-word instructions execute in two instruction cycles.

PIC18F Instruction Set

- The PIC18F instruction set contains a total of 75 core instructions.

  In addition, it provides an extended set of 8 new instructions.

1. Data movement instructions;
2. Arithmetic instructions
3. Logic instructions;
4. Rotate instructions;
5. Bit manipulation instructions;
6. Jump/Branch instructions;
7. Test, Compare, and Skip instructions;
8. Table Read/Write instructions;
9. Subroutine instructions;
10. System control instructions;

PIC18F Instruction Set

- Several PIC18F instructions that specify the destination as the WREG or a file register. Consider ADDWF F, d, a where ‘F‘ is the file register, ‘d’ defines the destination bit, and ‘a’ specifies the bank. This instruction adds the contents of the specified file register, ‘F‘ with the contents of WREG. If d = 0, then the result is stored in the WREG register. If d = 1, then the result is placed in the file register, ‘F‘ specified in the instruction.

PIC18F Instruction Set

- Note that a = 0 means that the data register is located in the access bank while a = 1 means that the contents of BSR specify the address of the bank. As mentioned before, the file (data) register can be one of the General-Purpose Registers (GPRs) or one of the Special Function Registers (SFRs) in the access bank.

PIC18F Instruction Set

- ADDWF 0x18, 0, 0 will add the contents of file register, 0x18 with the contents of WREG. The result will be placed in WREG since d = 0.
- ADDWF 0x18, 1, 0 will also add the contents of file register 0x18 with the contents of WREG. But the result will be stored in memory location 0x18 since d = 1.

PIC18F Instruction Set

- For instructions such as ADDWF F, d, a, the MPLAB assembler allows the programmer to use ‘W’ instead of ‘0’ and ‘F’ instead of ‘1’ as far as ‘d’ is concerned. The programmer can use ADDWF 0x18, W, 0 instead of ADDWF 0x18, 0, 0 and ADDWF 0x18, F, 0 instead of ADDWF 0x18, 1, 0.

PIC18F Instruction Set

- As mentioned in Chapter 5, upon power-up, the PIC18F uses the access bank. Note that the purpose of access bank in general is for fast access to any of the selected location in the access bank. This is because the BSR is not required for the access bank, and only an 8-bit rather than 12-bit data memory address is required.

PIC18F Instruction Set

- In the lower half of the access bank, a = 0, and the file (data) registers are used as the default bank with addresses 0x00 through 0x7F. Hence, we will not specify ‘a’ in the instruction (assuming the access bank with a = 0). Also, for better clarity, the MPLAB assembler uses W or F instead of 0 or 1 for ‘d’. This means that ADDWF 0x18, W will be used instead of ADDWF 0x18, 0,0 and ADDWF 0x18, F will be used instead of ADDWF 0x18,1,0.

PIC18F Instruction Set

- In the following, the brackets [ ] are used to indicate the contents of a register or a data memory location. For example, [WREG] will mean the contents of WREG.
- In the PIC18F, the term “file register” can be one of the GPR’s in the low access bank (addresses 0x00 to 0x7F in bank 0) or one of the SFR’s in high access bank (addresses 0x80 to 0xFF in bank F).

Data Movement Instructions

TABLE 6.4 PIC18F Data Movement Instructions

Instruction	*Comment*
CLRF F, a	Clear data register F to zero. F is located in access bank if a = 0 and F is located in the bank specified by BSR if a =1
LFSR F, K	Load low 12 bits of K into the specified File Select Register (F can be 0 or 1 or 2)
MOVLB K	Move 8-bit value K to BSR; low 4 bits of K are moved into low 4 bits of BSR with upper 4 bits of BSR are always cleared to zero regardless of upper 4 bits of K.
MOVLW data	Move 8-bit immediate data into WREG.
MOVWF F, a	Move data from WREG into data register, F. F is located in access bank if a = 0 and F is located in the bank specified by BSR if a =1.
MOVFF Fs, Fd	Move data from Fs (source data register) to Fd (destination data reg).
MOVF F, d, a	Move the contents of the file register into WREG (d = 0) or into the same file register (d = 1). F is located in access bank if a = 0 and F is located in the bank specified by BSR if a =1. This is the only MOVE instruction that affects N and Z status flags.
SETF F, a	Set all eight bits in the specified data register F to ones. See note for 'a'.
SWAPF F, d, a	Swaps low order 4 bits with the high-order 4 bits of the file register, F; see note for 'a' and 'd'

Data Movement Instructions

CLRF F, a (CLRF 0x20) clears the contents of the specified data register, F to zero. If a = 0, then F is located in the access bank. On the other hand, if a = 1, then F is located in the bank specified by BSR . For example, CLRF 0x20 will clear the 8-bit contents of data register with address 0x20 to zero. After clearing, the Z-flag is set to one. No other flags are affected.

LFSR F, K (LFSR 2, 0x0020) loads low 12 bits of K into one of three FSR's (File Select Registers). The specified FSR can then be used to point to a data register. For example, LFSR 2, 0x0020 will load 00H into FSR2H and 20H into FSR2L. The low 12-bit contents of FSR2 (0x020) can then be used as a pointer to data memory. Note that since FSR's are 16-bit wide registers, the 12-bit address is stored in low 12 bits (bits 0 through 11) of the FSRs with the upper four bits (bits 12 through 15) as 0's.

Data Movement Instructions

MOVLB K (MOVLB 0x01) moves 8-bit value K to BSR; low 4 bits of K are moved into low 4 bits of BSR with upper 4 bits of BSR are always cleared to zero regardless of the upper 4 bits of K. For example, MOVLB 0x01 will move 01H into BSR. This instruction is useful for bank switching.

MOVLW data8 (MOVLW 0x25) moves an 8-bit literal (constant data) into WREG. For example, MOVLW 0x25 will move 25H into the WREG register. The previous contents of WREG are lost.

MOVWF F, a (MOVWF 0x40) moves data from WREG into data register, F. F is located in the access bank if a = 0 and F is located in the bank specified by BSR if a =1. As an example, consider MOVWF 0x40.

Prior to execution of MOVWF 0x40: [0x40] = F1H, [WREG] = 53H

After execution of MOVWF 0x40: [0x40] = 53H, [WREG] = 53H (unchanged)

Data Movement Instructions

MOVFF Fs, Fd (MOVFF 0x04, 0x03) moves data from source data register Fs to destination data register Fd. WREG can be used as either Fs or Fd. Also, Fs and Fd can be any data memory location from 000H to FFFH. This is a two-word (32 bits) instruction. As an example, consider MOVFF 0x04, 0x03.

Prior to execution of MOVFF 0x04, 0x03: [0x03] = 2FH, [0x04] = 57H

After execution of MOVFF 0x04, 0x03: [0x03] = 57H, [0x04] = 57H (unchanged)

Data Movement Instructions

MOVF F, d, a (MOVF 0x30, W or MOVF 0x30, F) moves the contents of the data register F into WREG (d = 0) or into the same data register, F (d = 1). F is located in the access bank if a = 0 and F is located in the bank specified by BSR if a =1.

As an example, consider MOVF 0x30, W.

Prior to execution of MOVF 0x30, W: [WREG] = 70H, [0x30] = 2AH

After execution of MOVF 0x30, W: [WREG] = 2AH, [0x30] = 2AH (unchanged)

As another example, consider MOVF 0x30, F.

Prior to execution of MOVF 0x30, F: [WREG] = 70H, [0x30] = 2AH

After execution of MOVF 0x30, F: [WREG] = 70H, [0x30] = 2AH (unchanged). This is the same as NOP (No operation instruction), and can be used for time delays.

Data Movement Instructions

SETF F, a (SETF 0x20) sets the contents of the specified data register (F) to FFH. If a = 0, then the data register is located in the access bank while a = 1 means that the contents of BSR specify the address of the bank. As an example, consider SETF 0x20.

Prior to execution of SETF 0x20: [0x20] = 24H

After execution of SETF 0x20: [0x20] = FFH

The SETF instruction can be used to configure an I/O port. For example, the TRISB register in the PIC18F is used to configure Port B. Writing 1's to all 8 bits of TRISB will configure Port B as an input port. This can be accomplished using the SETF TRISB instruction. This topic will be discussed later.

Data Movement Instructions

SWAPF F, d, a (SWAPF 0x60, W or SWAPF 0x60, F) exchanges low order 4 bits with the high-order 4 bits of the file register, F. F is located in access bank if a = 0 and F is located in the bank specified by BSR if a =1. d = 0 means that the destination is WREG while d = 1 means that the destination is a file register.

As an example, consider SWAPF 0x60, W.

Prior to execution of the instruction, SWAPF 0x60, W: [0x60] = 48H, [WREG] = 50H.

After execution of the instruction, SWAPF 0x60, W: [0x60] = 48H, [WREG] = 84H.

As another example, consider SWAPF 0x60, F.

Prior to execution of the instruction, SWAPF 0x60, F: [0x60] = 48H, [WREG] = 50H.

After execution of the instruction, SWAPF 0x60, F: [0x60] =84H, [WREG] = 50H.

Data Movement Instructions

Example 6.1 Determine the effect of each of the following PIC18F instructions:

- CLRF PREINC1
- MOVWF INDF1
- MOVFF 0x40, 0x81,
- MOVF 0x40, F; also find the flags which are affected
- SWAPF 0x45, W

Assume the following initial configuration before each instruction is executed; also, assume that all numbers are in hex:

[FSR0] = 0044,	[FSR1] = 0075
[043] = 66,	[076] = FF
[075] = 24,	[WREG] = 33
[040] = 78,	[045] = 61
[081] = 55	

Solution See Table 6.5

Data Movement Instructions

TABLE 6. 5 Results for Example 6.1 (All numbers in hex)

Instruction	Affected Register	Net Effect (Hex)
CLRF PREINC1	Data Register Address = 076	[076]= 00
MOVWF INDF1	Data Register Address = 075	[075] = 33
MOVFF 0x40, 0x81	Data Register Address = 081	[0x81] = 78
MOVF 0x40, F	Data Register Address = 040	[0x40] = 78,N = 0, Z=0
SWAPF 0x45, W	WREG	[WREG] = 16

Data Movement Instructions

Example 6.2 Find the affected FSR, WREG, and data register contents for the following PIC18F instruction sequence:

```
LFSR    2,0x0044
MOVLW   D'20'
MOVWF   0x40
MOVFF   PLUSW2, 0x40
CLRF    POSTINC2
SETF    0x40
```

Assume [0x58] = 1AH prior to execution of the instruction sequence.

Data Movement Instructions

Solution

After execution of LFSR 2,0x0044, the File Select register FSR2 is loaded with 0044H. MOVLW D'20' moves immediate decimal data 20 (14H) into WREG. The instruction MOVWF 0x40 moves [WREG] into data register 0x40. Hence, [0x40] = 14H.

MOVFF PLUSW2, 0x40 adds [WREG] with [FSR2] and then moves the byte content of that address into data register 0x40. This means that the contents of data register 0x58 are moved to WREG. Hence, [0x40] = 1AH.

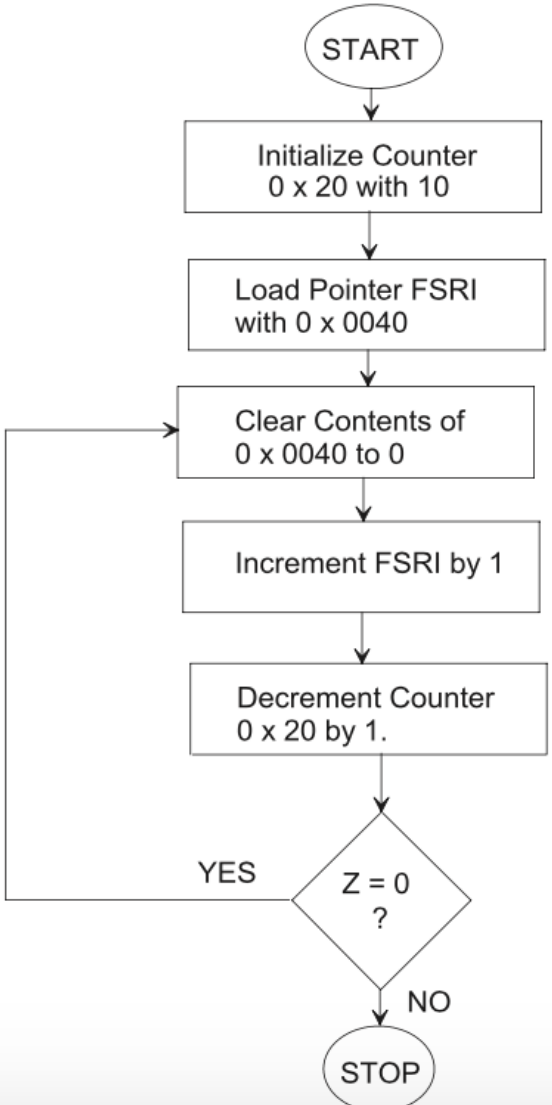
CLRF POSTINC2 clears the contents of data register 0x44 to zero, and then increments FSR2 by 1. Hence, [0x44] = 00H, and [FSR2] = 0x45.

SETF 0x40 sets all bits in data register 0x40 to ones. Hence, [0x40] = FFH.

Example 6.3 It is desired to clear 10 consecutive bytes to zero from LOW to HIGH data register addresses starting at data register 0x40 addressed by FSR1.

(a) Flowchart the problem.

(b) Convert the flowchart to a PIC18F assembly language program starting at address 0x100.



Data Movement Instructions

Solution

(a) The flowchart is provided above.

(b) The flowchart is converted to PIC18F assembly language program as follows:

```
        INCLUDE   <P18F4321.INC>
          ORG     0x100
           MOVLW   D'10'        ; Move 10 decimal into WREG
           MOVWF   0x20         ; Initialize Counter reg (20H) with 10 decimal
           LFSR    1, 0x0040    ; Initialize FSR1 with 040H one below starting
                                ; address since preincrement mode is to be used
REPEAT     CLRF    POSTINC1     ; then Clear a location to 0 and increment FSR1 by 1
           DECF    0x20, F      ; Decrement Counter by 1
           BNZ     REPEAT       ; Branch to  REPEAT if  Zero flag = 0, otherwise
                                ; go to the next instruction
           SLEEP                ; HALT
           END
```

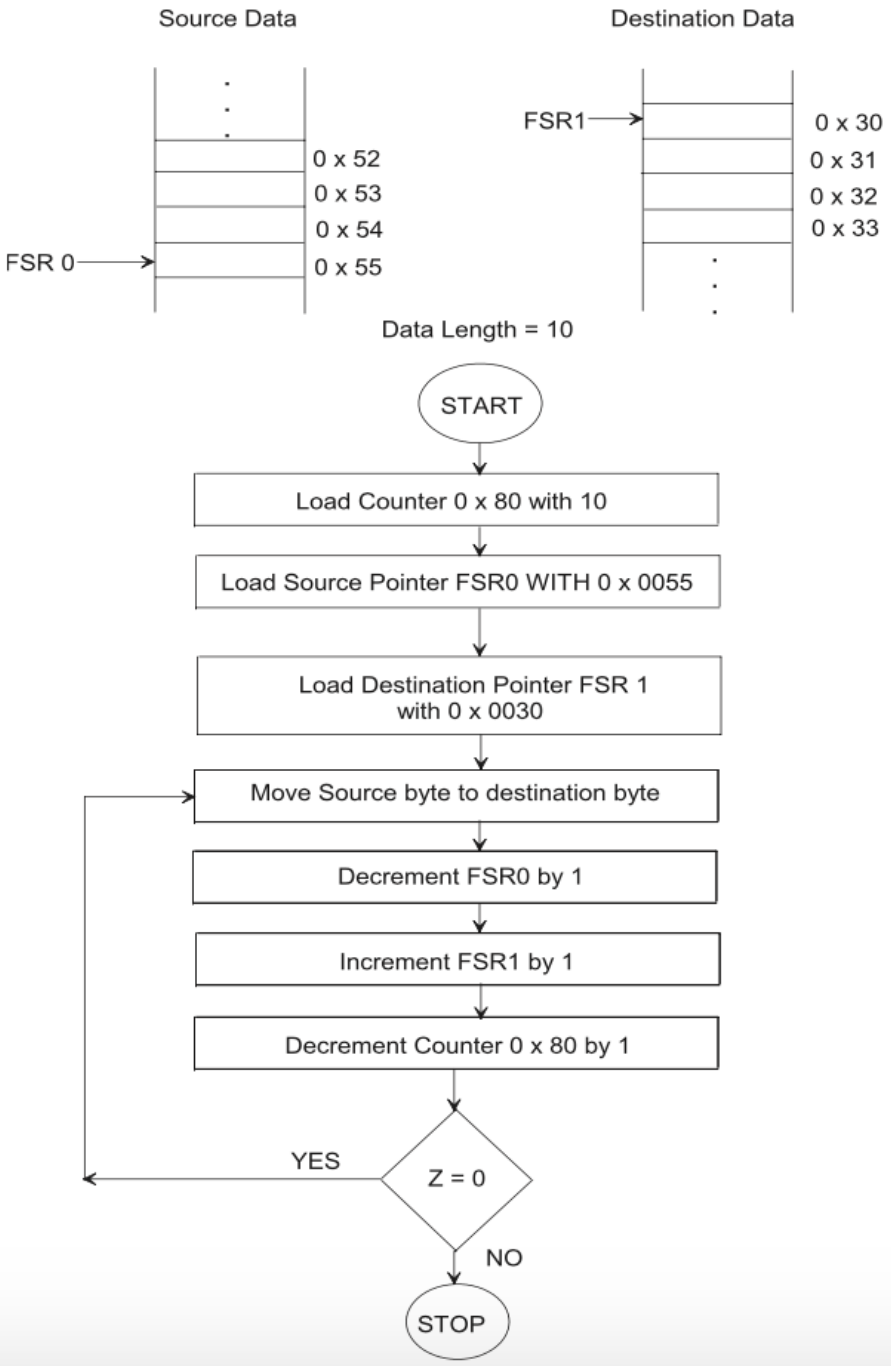
Data Movement Instructions

Example 6.4 It is desired to move a block of 8-bit data of length 10 from the source block (from HIGH to LOW address) starting at data register address 0x55 to the destination block (from LOW to HIGH addresses) starting at data register address 0x30. That is, [0x55] will be moved to [0x30], [0x54] to [0x31], and so on. Assume that data for the source block and the destination block are already stored in data memory addresses.

(a) Flowchart the problem.

(b) Convert the flowchart to PIC18F assembly language program starting at address 0x100.

Data Movement Instructions



Data Movement Instructions

(b) The flowchart can be converted to PIC18F assembly language program as follows:

```
            INCLUDE <P18F4321.INC>
COUNTER     EQU     0x80
            ORG     0x100
            MOVLW   D'10'           ; Move 10 decimal into WREG
            MOVWF   COUNTER         ; Initialize Counter reg (0x80) with 10 decimal
            LFSR    0, 0x0055       ; Initialize FSR0 with source starting address
            LFSR    1, 0x0030       ; Initialize FSR1 with destination starting address
BACK        MOVFF   POSTDEC0, POSTINC1  ; Move Source data to destination
            DECF    COUNTER, F      ; Decrement Counter by 1
            BNZ     BACK            ; Branch to BACK if Zero flag = 0, otherwise
                                    ; go to the next instruction
            SLEEP                   ; HALT
            END
```

Arithmetic Instructions

- The PIC18F arithmetic instructions allow:
  - 8-bit addition and subtraction
  - 8-bit by 8-bit unsigned multiplication. The PIC18F does not provide any signed multiplication and division (Signed and Unsigned) instructions.
  - Negate instruction
  - Decrement and Increment Instructions
  - BCD Adjust (BCD Correction)

Arithmetic Instructions

- Typical microcontrollers utilize common hardware to perform addition and subtraction operations for both unsigned and signed numbers. The instruction set of microcontrollers include the same ADD and SUBTRACT instructions for both unsigned and signed numbers. The interpretations of unsigned and signed ADD and SUBTRACT operations are performed by the programmer.

Arithmetic Instructions

- Unsigned and signed multiplication and division operations can be performed using various algorithms. The PIC18F provides only unsigned multiplication instruction. The other multiplication and division instructions can be obtained by writing PIC18F assembly language programs using appropriate algorithms.

Arithmetic Instructions

- The PIC18F arithmetic instructions are summarized in Table 6.6.
  - All instructions in Table 6.6 are executed in one cycle. The size of each instruction is one word.
  - a = 0 means that the data register is located in the access bank while a = 1 means that the contents of BSR specify the address of the bank.
  - For destination: d = 0 means that the destination is WREG while d = 1 means that the destination is file register. As mentioned before, W or F instead of 0 or 1 will be used in this book for better clarity.
  - The PIC18F does not provide any multiplication (signed) and division (signed and unsigned) instructions.

Arithmetic Instructions

TABLE 6.6 PIC18F Arithmetic Instructions

Instruction	*Operation*
	Addition and Subtraction Instructions
ADDLW data8	[WREG] + 8-bit data → [WREG]
ADDWF F, d, a	[WREG] + [F] destination; see note for 'a' and 'destination'
ADDWFC F, d, a	[WREG] + [F] + Carry→ destination; see note for 'a' and 'destination'
SUBLW data8	8-bit data - [WREG] → [WREG]
SUBWF F, d, a	[F] – [WREG] → destination; see note for 'a' and 'destination'
SUBWFB F, d, a	[F] – [WREG] – Carry → destination; see note for 'a' and 'destination'
SUBFWB F, d, a	[WREG] – [F]– Carry → destination; see note for 'a' and 'destination'
	Unsigned Multiplication Instructions
MULLW data8	[WREG] x 8-bit data → [PRODH]: [PRODL] (unsigned multiplication)
MULWF F, a	[WREG]x [F] → [PRODH]: [PRODL] (unsigned multiplication)
	Negate instruction
NEGF F, a	0 – [F] → [F]; see note for 'a'.
	Decrement and Increment instructions
DECF F, d, a	[F] – 1→ destination; see note for 'a' and 'destination'
INCF F, d, a	[F] + 1→ destination; see note for 'a' and 'destination'
	BCD Adjust (BCD Correction) *Instruction*
DAW	Decimal Adjust [WREG]

Arithmetic Instructions

ADDLW data8 (ADDLW 0x02) instruction adds the 8-bit contents of WREG with 8-bit immediate data, and stores the result in WREG. For example, Consider ADDLW 0x02.

Prior to execution of ADDLW 0x02: [WREG] = 12H.
After execution of ADDLW 0x02: [WREG] = 12H + 02H = 14H.
The flags are affected based on the result as follows:

```
                    Previous carry →  0000  010  Intermediate Carries
                  [WREG]=  12H = 0001  0010
          Add  immediate data, 02H = 0000 0010
                                     ----------
                  final carry → 0   0001 0100 = 14H
```

N = 0 (most significant bit of the result is 0), OV = 0 (no overflow since the previous carry and the final carry are the same), Z = 0 (nonzero result), DC = 0 (No carry from bit 3 to bit 4), C = 0 (no Carry).

Arithmetic Instructions

ADDWF F, d, a (ADDWF 0x50, W or ADDWF 0x50, F) instruction adds the contents of WREG with the contents of the specified data register (F). The result is stored in WREG if d = 0 or in the data register if d = 1. The data register is in access bank if a = 0 or specified by BSR if a = 1.

As an example, consider ADDWF 0x50, W.

Prior to execution of ADDWF 0x50, W: [WREG] = 73H, [0x50] = 2AH.

After execution of ADDWF 0x50, W: [WREG]= 73H + 2AH = 9DH, [0x50] = unchanged = 2AH

The flags are affected based on the result as follows:

Previous carry →1100 010 ← Intermediate Carries

[WREG] = 73H = 0111 0011

Add [0x50] = 2AH = 0010 1010

---

final carry → 0 1001 1101 = 9DH

N = 1 (most significant bit of the result is 0), OV = $C_f \oplus C_p = 0 \oplus 1 = 1$ (overflow indicating wrong result; addition of two positive numbers generates a negative result), Z = 0 (nonzero result), DC = 0 (No carry from bit 3 to bit 4), C = 0 (no Carry). Note that the correct result can be rectified by increasing the number of bits for the two signed numbers to be added (73H and 2AH).

As another example, consider ADDWF 0x50, F.

Prior to execution of ADDWF 0x50, F: [WREG] = 73H, [0x50] = 2AH

After execution of ADDWF 0x50, F: [0x50] = 73H + 2AH = 9DH, [WREG] = unchanged = 73H

Arithmetic Instructions

ADDWFC F, d, a (ADDWFC 0x60, W or ADDWFC 0x60, F) adds the contents of WREG with the contents of the specified data register and the carry flag. The result is stored in WREG if d = 0 or in the data register if d = 1. The data register is in access bank if a = 0 or specified by BSR if a = 1.

As an example, consider ADDWFC 0x60, W.

Prior to instruction execution: Carry bit = 1, [0x60] = 03H, [WREG] = 07H

After instruction execution: C = 0, [0x60] = 03H (unchanged), [WREG] = 0BH,

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

Note that ADDWFC 0x60, F performs the same operation as ADDWFC 0x60, W except that the addition result is stored in 0x60.

Arithmetic Instructions

SUBLW data8 (SUBLW 07H) subtracts [WREG] from 8-bit immediate data. The result is placed in WREG. As an example, consider SUBLW 07H.

Prior to instruction execution: [WREG] = 03H.

After Instruction execution: [WREG] = 07H – 03H = 04H.

The flags are affected as follows:

Using two's complement subtraction, 1111 111 ← Intermediate Carries

Immediate data = 07H = 0000 0111

Add two's complement of 03 = 1111 1101

---

final carry →1 0000 0100 (04H)

The final carry is one's-complemented after subtraction to reflect the correct borrow. Hence, C = 0. Also, N = 0 (most significant bit of the result is zero), OV = $C_f \oplus C_p$ = $1 \oplus 1 = 0$, Z = 0 (nonzero Result), DC = 1. Note that in the above, the final carry is 1 indicating a borrow while performing the operation (07H - 03H). The correct result should have been 04H without any borrow. But, in the above, the result is 04H with a borrow. While performing two's complement subtraction analytically using pencil and paper, the borrow is always one's complement of the true borrow. Hence, the PIC18F complements the carry to reflect the true borrow.

Arithmetic Instructions

SUBWF F, d, a (SUBWF 0x20, W or SUBWF 0x20, F) instruction subtracts the contents of WREG from the contents of the specified data register. The result is stored in WREG if d = 0 or in the data register if d = 1. The data register is in access bank if a = 0 or specified by BSR if a = 1. The flags are affected in the same way as the SUBLW instruction. Note that the carry is one's complemented to reflect the correct borrow.

SUBWFB F, d, a (SUBWFB 0x30, W or SUBWFB 0x30, F) instruction subtracts the contents of WREG and the Carry from the contents of the specified data register. The result is stored in WREG if d = 0 or in the data register if d = 1. The data register is in access bank if a = 0 or specified by BSR if a = 1.

SUBFWB F, d, a (SUBFWB 0x20, W or SUBFWB 0x20, F) instruction subtracts the contents of specified data register and the Carry from the contents of the WREG. That is, the SUBWFB performs the following operation using two's complement: [dest] ← [F] - [WREG + Carry]. The result is stored in WREG if d = 0 or in the data register if d = 1. The data register is in access bank if a = 0 or specified by BSR if a = 1 .

Arithmetic Instructions

- **Multiplication Instructions**
- The PIC18F includes an 8 x 8 hardware multiplier as part of the ALU. The multiplier performs an unsigned operation and provides a 16-bit result that is stored in the product register pair, PRODH:PRODL.
  - Because of hardware implementation, the multiplier executes the multiplication operation in a single instruction cycle.
  - None of the status flags are affected. Note that neither overflow nor carry is possible in this operation. A zero result is possible but not detected.

Arithmetic Instructions

- MULLW data8 (MULLW 0x03) instruction performs an unsigned multiplication between the 8-bit contents of WREG and 8-bit immediate data. The 16-bit result is placed in the PRODH:PRODL register pair. PRODH contains the high byte, and PRODL contains the low byte. The contents of WREG are unchanged. As an example, consider MULLW 0x03.

  Prior to instruction execution: [WREG] = 02H
  After Instruction execution: [PRODH] = 00H, [PRODL] = 06H, [WREG] = 02H = unchanged

- MULWF F, a (MULWF 0x50) instruction performs an unsigned multiplication between the 8-bit contents of WREG and 8-bit contents of the specified data register. The 16-bit result is placed in the PRODH:PRODL register pair. PRODH contains the high byte, and PRODL contains the low byte. The contents of the WREG and the data register are unchanged. The data register is in the access bank if a = 0 or specified by BSR if a = 1.

Arithmetic Instructions

Negate instruction The PIC18F Negate instruction is illustrated by means of numerical examples in the following.

- NEGF F, a (NEGF 0x70) instruction negates the contents of the specified data register using two's complement. The result is stored in the data register. The data register is in access bank if a = 0 or specified by BSR if a = 1. An example is NEGF 0x70.

  Prior to instruction execution: [0x70] = 02H.
  After instruction execution: [0x70] = FEH = -2_{10}.

Arithmetic Instructions

Decrement and Increment Instructions The PIC18F decrement and increment instructions are illustrated in the following by means of numerical examples.

- DECF F, d, a (DECF 0x50, W or DECF 0x50, F) decrements the contents of the specified data register by 1. The result is stored in WREG if d = 0 or in the data register if d = 1. The data register is in the access bank if a = 0 or specified by BSR if a = 1. All flags are affected. An example is DECF 0x50, F.

  Prior to instruction execution: [0x50] = 01H.
  After instruction execution: [0x50] = 00H.

  Note that DECF 0x50, W decrements [0x50] by 1 and the result is stored in WREG.

- INCF F, d, a (INCF 0x50, W or INCF 0x50, F) increments the contents of the specified data register by 1. The result is stored in WREG if d = 0 or in the data register if d = 1. The data register is in the access bank if a = 0 or specified by BSR if a = 1. An example is INCF 0x50, F.

  Prior to instruction execution: [0x50] = FFH = -1_{10}.
  After instruction execution: [0x50] = 00H.

  INCF 0x50, W performs the same operation as INCF 0x50, F except that the result is stored in WREG.

Arithmetic Instructions

BCD adjust (BCD correction) instruction The PIC18F contains a BCD adjust instruction which will be illustrated in the following by means of a numerical example.

- DAW instruction adjusts the eight-bit result in WREG after adding two packed BCD numbers using ADDLW or ADDWF or ADDWFC to provide the correct packed BCD result.

If, after the addition, the low 4 bits of the result in WREG are greater than 9 (or if DC = 1), the DAW adds 6 to the low 4 bits of WREG. On the other hand, if the high 4 bits of the result in WREG are greater than 9 (or if C = 1), DAW adds 6 to the high 4 bits in WREG. Consider the following instruction sequence:

```
MOVLW 0x29   ; Move 29H into WREG
ADDLW 0x54   ; Add 29H with 54H and store the result in WREG
DAW          ; Decimal adjust WREG to provide the correct packed BCD result
```

The details of the result obtained by the instruction sequence above are provided in the following:

```
[WREG] = 29H = 0010 1001  (Packed BCD 29, same as 29H)
     Add  54H = 0101 0100  (Packed BCD 54, same as 54H)
                ----------
     [WREG] = 0111 1101
                 0110   Add 6 (BCD correction by DAW since low 4 bits of the
                ----------    sum in WREG are greater than 9)
                1000 0011 = 83H correct packed BCD result since 29 + 54 = 83
```

Note that packed BCD is covered in section 1.2.3 of Chapter 1.

Arithmetic Instructions

Example 6.5 Write a PIC18F assembly language program to implement the following C segment;

```
if (x<0)
y++;
else
y--;
```

Solution

```
        MOVF   X, F     ; Move [X] into [F], MOVF affects N flag
        BNN    ELSE     ; Branch to ELSE if N = 0  (if [X] is positive)
        INCF   Y        ; else, if N = 1 ( [X] is negative), Increment [Y] by 1
        BRA    NEXT
ELSE    DECF   Y        ; Decrement [Y] by 1 if N = 0
NEXT    GOTO   NEXT     ; Halt
```

Arithmetic Instructions

Example 6.6 Write a PIC18F assembly language instruction sequence to add four numbers 1,2,3,4, and then store the result in a data register 0x40 as follows:
(a) without using a loop (b) using a loop

Solution

(a) without using a loop

```
SUM     EQU     0x40
        MOVLW   1       ; Move 1 into WREG
        ADDLW   2       ; Add 2 with [WREG] and store 3 in WREG
        ADDLW   3       ; Add 3 with [WREG] and store 6 in WREG
        ADDLW   4       ; Add 4 with [WREG] and store 10 in WREG
        MOVWF   SUM     ; Store [WREG] in SUM
```

(b) using a loop

```
SUM     EQU     0x40
COUNTER EQU     0x50
        CLRF    SUM         ; Clear  SUM to 0
        MOVLW   4           ; Move 4 into WREG
        MOVWF   COUNTER     ; Initialize COUNTER with 4
        MOVF    SUM, W      ; Move 0 to WREG
LOOP    ADDWF   COUNTER, W  ; Add [COUNTER] with [WREG], store result in W
        DECF    COUNTER     ; Decrement [COUNTER] by 1
        BNZ     LOOP        ; Branch to loop if Z is not equal to 0
        MOVWF   SUM         ; Move result (10) from WREG into SUM
```

Arithmetic Instructions

Example 6.7 Write a PIC18F instruction sequence to implement the following C statement:

c = a + b;

Assume data registers 0x50, 0x60, and 0x70 store a, b, and c respectively.

Solution

```
A       EQU     0x50
B       EQU     0x60
C       EQU     0x70
        MOVF    A, W    ; Move [A] to [WREG]
        ADDWF   B, W    ; Add [B] with [WREG], result in WREG
        MOVWF   C       ; store result in WREG in C
```

Arithmetic Instructions

Example 6.8 Write a PIC18F instruction sequence to implement the following C statement:

$$c = 2a + b;$$

Assume data registers 0x24, 0x31, and 0x50 store a, b, and c respectively.

Solution

```
A       EQU     0x24
B       EQU     0x31
C       EQU     0x50
        MOVF    A, W    ; Move [A] to [WREG]
        ADDWF   A, W    ; Add [A] with [WREG], result 2 x [A] in WREG
        ADDWF   B, W    ; Add 2x [A] with [B], store result in WREG
        MOVWF   C       ; store result in WREG in C
```

Example 6.9 Write a PIC18F assembly language program at address 0x100 that implements the following C language program segment:

```
sum = 0;
for (i = 0; i <= 9; i = i + 1)
sum = sum + a[i];
```

where sum is the address of the 8-bit result of addition. Assume sum as 0x50 and the address of the first element of the array, a[0], is stored in data register 0x20, the second element, a[1], in data register 0x21, and so on. Assume that the array is already stored in data memory. Also, assume addition of two consecutive will generate no carry.

Solution

Assume that register FSR0 holds the address of the first element of the array. The assembly language program is listed below:

```
        INCLUDE  <P18F4321.INC>
        ORG      0x100
SUM     EQU      0x50          ; Initialize SUM to 0x50 for result
        LFSR     0, 0x0020     ; Point FS0 to a[0]
        CLRF     SUM           ; Clear [SUM] to zero
        MOVLW    D'10'         ; Move  WREG with 10
        MOVWF    0x70          ; Initialize 0x70 with loop count (10)
LOOP    MOVF     SUM, W        ; Move [SUM] into WREG
        ADDWF    POSTINC0, W   ; Add and Store result in WREG
        MOVWF    SUM           ; Save [WREG] IN SUM
        DECF     0x70, F       ; Decrement counter 0x70 by 1
        BNZ      LOOP          ; Branch to LOOP if  Z not equal 0
        SLEEP                  ; Halt
        END
```

Arithmetic Instructions

Example 6.10 Write a PIC18F assembly language program at address 0x100 to add two 16-bit numbers as follows:

```
        [0x51] [0x50]
PLUS    [0x61] [0x60]
    ----------------------
        [0x61] [0x60]
```

Assume data registers 0x50 and 0x51 contain low and high bytes of the first 16-bit numbers while data registers 0x60 and 0x61 contain the second 16-bit numbers. Also, assume that data are already loaded into the data registers.

Solution

```
INCLUDE   <P18F4321.INC>
ORG       0x100
MOVF      0x50, W     ; Move low byte of first data into WREG
ADDWF     0x60, F     ; Add low 8 bits, store result in 0x60
MOVF      0x51, W     ; Move high byte of first data into WREG
ADDWFC    0x61, F     ; Add high bytes with carry, store result in 0x61
SLEEP                 ; Halt
END
```

Arithmetic Instructions

Example 6.11 Write a PIC18F assembly language program at address 0x100 to add four 8-bit numbers stored in consecutive data registers from high to low addresses starting at data register 0x78. Store the 8-bit result WREG. Assume that no carry is generated, due to the addition of two consecutive 8-bit numbers. Assume that data are already loaded into data memory addresses 0x75 through 0x78 with the first data byte at 0x78 and the last data byte at 0x75.

Solution

```
        INCLUDE   <P18F4321.INC>
        ORG       0x100
        MOVLW     4              ; Move WREG with 4
        MOVWF     0x50           ; Initialize 0x50 with loop count (4)
        MOVLW     0x00           ; Clear  WREG to store sum
        LFSR      0, 0x0078      ; Initialize pointer FSR0 with 0x0078
START   ADDWF     POSTDEC0, W    ; Add two bytes and store sum in WREG
        DECF      0x50, F        ; Decrement counter 0x50 by 1
        BNZ       START          ; Branch to START  if  Z is 0
        SLEEP                    ; Halt
        END
```

Example 6.12 Write a PIC18F assembly language program at 0x100 to subtract two 32-bit numbers as follows:

```
      [0x43] [0x42] [0x41] [0x40]
MINUS [0x75] [0x74] [0x73] [0x72]
      ---------------------------------
      [0x43] [0x42]  [0x41] [0x40]
```

Assume data registers 0x40 through 0x43 contain the first 32-bit number while data registers 0x72 through 0x75 contain the second 32-bit number. Also, assume that data are already loaded into the data registers. Store the 32-bit result in data registers 0x40 (lowest byte) through 0x43 (highest byte). Do not use a loop.

Solution

```
INCLUDE   <P18F4321.INC>
ORG       0x100
MOVF      0x72, W       ; Move byte into WREG and update pointer
SUBWF     0x40, F       ; Subtract [WRFG] from a byte, store result
                        ; in data registers
MOVF      0x73, W
SUBWFB    0x41, F       ; Subtract [WREG] and carry from byte
MOVF      0x74, W       ; and store rusult in data reg
SUBWFB    0x42, F
MOVF      0x75, W
SUBWFB    0x43, F
SLEEP                   ; Halt
END
```

Arithmetic Instructions

Example 6.13 Write a PIC18F assembly language program at address 0x100 to compute $(X^2 + Y^2)$ where X and Y are two 8-bit unsigned numbers stored in data registers 0x50 and 0x40 respectively. Store the 16-bit result in data registers 0x51 (high byte) and 0x50 (low byte). Assume X and Y are already loaded into data registers 0x50 and 0x40.

Solution

```
INCLUDE  <P18F4321.INC>
ORG      0x100
MOVF     0x40, W        ; Move X into WREG
MULWF    0x40           ; Multiply X by X, result in PRODH:PRODL
MOVFF    PRODL, 0x50    ; Save low byte of result in data reg 0x50
MOVFF    PRODH, 0x51    ; Save high byte of result in data reg 0x51
MOVF     0x40, W        ; Move Y into WREG
MULWF    0x40           ; Multiply Y by Y, result in PRODH:PRODL
MOVF     PRODL, W       ; Move PRODL into WREG
ADDWF    0x50, F        ; Add and store low byte of result in 0x50
MOVF     PRODH, W       ; Move high byte of X times X into WREG
ADDWFC   0x51, F        ; Add high bytes with carry. Store result in 0x51
SLEEP                   ; Halt
END
```

Arithmetic Instructions

Example 6.14 Write a PIC18F assembly language program at address 0x100 to add two packed BCD bytes stored in data registers 0x20 and 0x21. Store the correct packed BCD result in WREG. Load packed BCD bytes 0x72 and 0x45 into data registers 0x20 and 0x21 respectively using PIC18F instructions. Note that data are arbitrarily chosen.

Solution

```
INCLUDE  <P18F4321.INC>
ORG      0x100
MOVL     W 0x72        ; Load first packed BCD data
MOVWF    0x20          ; into 0x20
MOVL     W 0x45        ; Load second packed BCD data
MOVWF    0x21          ; into 0x21
MOVF     0x20, W       ; Move 72H into WREG
ADDWF    0x21, W       ; Add and store binary result in WREG
DAW                    ; Convert WREG to correct packed
                       ; BCD byte
SLEEP                  ; Halt
END
```

Logic Instructions

- The PIC18F logic instructions include logic AND, NOT (one's complement), OR, and Exclusive-OR operations.

TABLE 6.7 PIC18F Logic Instructions

Instruction	*Comment*
ANDLW data8	[WREG] AND 8-bit data → [WREG]
ANDWF F, d, a	[WREG] AND [F] → destination; see note for 'a' and 'destination'
COMF F, d, a	NOT [F] → destination; see note for 'a' and 'destination'
IORLW data8	[WREG] OR 8-bit data → [WREG]; Performs Inclusive OR or simply OR operation.
IORWF F, d, a	[WREG] OR [F] → destination; see note for 'a' and 'destination'. Performs inclusive OR or simply OR operation.
XORLW data8	[WREG] ⊕ 8-bit data → [WREG]
XORWF F, d, a	[WREG] ⊕ [F] → [destination] ; see note for 'a' and 'destination'

- All instructions in the above are executed in one cycle.
- The size of each instruction is one word.
- All instructions affect N and Z flags; other flags are not affected.
- a = 0 means that the data register is located in the access bank while a = 1 means that the contents of BSR specify the address of the bank.
- For destination: d = 0 means that the destination is WREG while d = 1 means that the destination is file register F.

Logic Instructions

- ANDLW data8 (ANDLW 0x8F) instruction ANDs the contents of WREG with the 8-bit literal (immediate data, data8). The result is placed in WREG. As an example, consider ANDLW 0x8F.

Prior to instruction execution: [WREG] = 0x72
After instruction execution:

```
[WREG] = 0x72 =   0111 0010
   AND  0x8F =   1000 1111
                 ---------
[WREG] =         0000 0010
```

N and Z flags are affected. Z = 0 (Result is nonzero) and N = 0 (Most Significant Bit of the result is 0). The status flags are affected in the same way after execution of other logic instructions such as OR, XOR, and NOT.

Logic Instructions

- ANDWF F, d, a (ANDWF 0x60, W or ANDWF 0x60, F) instruction ANDs the contents of WREG with register 'F'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'F'. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the bank.

As an example, consider ANDWF 0x60, F.

Prior to instruction execution: [0x60] = 0xFF, [WREG] = 0x01

After instruction execution: [0x60] = 0x01, [WREG] = 0x01 (unchanged), Z = 0, N = 0

The AND instruction can be used to perform a masking operation. If the bit value in a particular bit position is desired in data byte, the data can be logically ANDed with appropriate data to accomplish this. For example, the bit value at bit 2 of an 8-bit number 0100 1Y10 (where unknown bit value of Y is to be determined) can be obtained as follows:

```
          0 1 0 0 1 Y 1 0  -- 8-bit number
AND       0 0 0 0 0 1 0 0 -- Masking data
          -------------------------------
          0 0 0  0  0 Y 0 0 -- Result
```

Logic Instructions

- If the bit value Y at bit 2 is 1, then the result is nonzero (Flag Z=0); otherwise, the result is zero (Z=1). The Z flag can be tested using typical conditional branch instructions such as BZ (Branch if Z=1) or BNZ (Branch if Z = 0) to determine whether Y is 0 or 1. This is called masking operation. The AND instruction can also be used to determine whether a binary number is ODD or EVEN by checking the Least Significant Bit (LSB) of the number (LSB=0 for even and LSB=1 for odd).
- ANDWF 0x60, W performs the same operation as ANDWF 0x60, F except the result is stored in WREG.

Logic Instructions

- COMF F, d, a (COMF 0x50, W or COMF 0x50, F) instruction complements (one's) the contents of register 'F'. If 'd' is '0', the result is stored in WREG. If 'd' is '1', the result is stored back in register 'F'. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the bank.

For example, consider COMF 0x50, W.
Prior to instruction execution: [0x50] = 0x01, [WREG] = 0x57
After instruction execution: [WREG] = 0xFE, [0x50] = 0x01 (unchanged),
Z = 0, N = 1; No other flags are affected.
Note that COMF 0x50, F is the same as COMF 0x50, F except that the result is stored in WREG.

Logic Instructions

- IORLW data8 (IORLW 0x7F) instruction ORes the contents of WREG with the 8-bit literal (immediate data, data8). The result is placed in WREG. As an example, consider IORLW 0x7F.

Prior to instruction execution: [WREG] = 0x01

After instruction execution: [WREG] = 0x7F (unchanged), Z = 0, N = 0; other flags are not affected.

The OR instruction can typically be used to insert a 1 in a particular bit position of a binary number without changing the values of the other bits. For example, a 1 can be inserted using the OR instruction at bit number 3 of the 8-bit binary number 0 1 1 1 0 0 1 1 without changing the values of the other bits as follows:

```
        0 1 1 1 0 0 1 1 -- 8-bit number
OR      0 0 0 0 1 0 0 0 -- data for inserting a 1 at bit number 3
        -----------------
        0 1 1 1 1 0 1 1 -- Result
```

Logic Instructions

- IORWF F, d, a (IORWF 0x50, W or 0x50, F) instruction ORes [WREG] with register ‘F’. If ‘d’ is ‘0’, the result is placed in WREG. If ‘d’ is ‘1’, the result is placed back in register ‘F’. If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the bank.

As an example, consider IORWF 0x50, W.
Prior to instruction execution: [WREG] = 0xA2,[0x50] = 0x5D
After instruction execution: [WREG] = 0xFF, [0x50] = 0x5D (unchanged), Z = 0, N = 0; other flags are not affected.

IORWF 0x50, F is the same as the IORWF 0x50, W except that the result is stored in 0x50.

The OR instruction can typically be used to insert a 1 in a particular bit position of a binary number without changing the values of the other bits.

Logic Instructions

- IORWF F, d, a (IORWF 0x50, W or 0x50, F) instruction ORes [WREG] with register ‘F’. If ‘d’ is ‘0’, the result is placed in WREG. If ‘d’ is ‘1’, the result is placed back in register ‘F’. If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the bank.

For example, a 1 can be inserted using the OR instruction at bit number 3 of the 8-bit binary number 0 1 1 1 0 0 1 1 without changing the values of the other bits as follows:

```
  0 1 1 1 0 0 1 1 -- 8-bit number
OR 0 0 0 0 1 0 0 0 -- data for inserting a 1 at bit number 3
  ----------------
  0 1 1 1 1 0 1 1 -- Result
```

IORWF 0x50, F stores the result in 0x50.

Logic Instructions

- XORLW data8 (XORLW 0x02) instruction exclusive-ORes the contents of WREG with 8-bit literal (immediate data, data8). The result is placed in WREG.

As an example, consider XORLW 0x02 .
Prior to instruction execution: [WREG] = 0x42
After instruction execution: [WREG] = 0x40, Z = 0, N = 0 ; no other flags are affected.

The Exclusive-OR instruction can be used to find the ones complement of a binary number by XORing the number with all 1's as follows:

```
    0 1 0 1 1 1 0 0 - - 8-bit number
XOR 1 1 1 1 1 1 1 1 - - data
    ------------------------
    1 0 1 0 0 0 1 1 -- Result (Ones Complement of the 8-bit number 0 1 0 1 1 1 0 0)
```

Logic Instructions

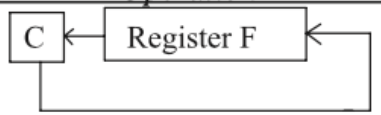
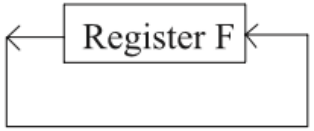
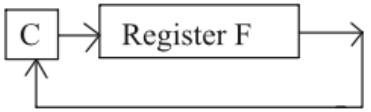
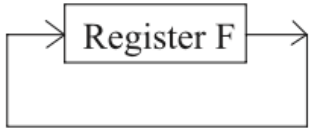
- XORWF F, d, a (XORWF 0x42, W or XORWF 0x42, F) instruction exclusive ORes the contents of WREG with register ‘F’. If ‘d’ is ‘0’, the result is stored in WREG. If ‘d’ is ‘1’, the result is stored back in the register ‘F’. If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select bank .

As an example, consider XORWF 0x42, W .
Prior to instruction execution: [WREG] = 0xFF, [0x42] = 0xFF
After instruction execution: [WREG] = 0x00, Z = 1, N = 0 ; No other flags are affected.

Note that XORWF 0x42, F stores the result in 0x42.

Rotate Instructions

TABLE 6.8 PIC18F Rotate Instructions

Instruction	*Operation*
RLCF F, d, a	 Rotate register F one bit to the left through carry. See notes for 'd' and 'a'.
RLNCF F, d, a	 Rotate register F one bit to the left without carry. See notes for 'd' and 'a'.
RRCF F, d, a	 Rotate register F one bit to the left through carry. See notes for 'd' and 'a'.
RRNCF F, d, a	 Rotate register F one bit to the left without carry. See notes for 'd' and 'a'.

- All instructions in the above are executed in one cycle.
- The size of each instruction is one word.
- a = 0 means that the data register is located in the access bank while a = 1 means that the contents of BSR specify the address of the bank.
- For destination: d = 0 means that the destination is WREG while d = 1 means that the destination is file register, F.
- RLCF and RRCF affect N, Z, and C flags according to the result while RLNCF and RRNCF affect N and Z flags based on the result.
- Note that the PIC18F does not have any shift instructions

Rotate Instructions

- RLCF F, d, a (RLCF 0x40, W or RLCF 0x40, F) instruction rotates the contents of register ‘F’ one bit to the left through the Carry flag. If ‘d’ is ‘0’, the result is placed in WREG. If ‘d’ is ‘1’, the result is stored back in register ‘F’. If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the bank.

As an example, consider RLCF 0x40, W.
Prior to instruction execution: [WREG] = 0xFF, [0x40] = 0xAF , C = 0
After instruction execution: [WREG] = 0x5E, C = 1, Z = 0 (result in WREG after rotating is nonzero), N = 0 (most significant bit of result, 0x5E is 0); no other flags are affected.

Note that RLCF 0x40, F stores the result in data register 0x40.

Rotate Instructions

- RLNCF F, d, a (RLNCF 0x70, W or RLNCF 0x70, F) instruction rotates the contents of register ‘F’ one bit to the left. If ‘d’ is ‘0’, the result is placed in WREG. If ‘d’ is ‘1’, the result is stored back in register ‘F’. If ‘a’ is ‘0’, the access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the bank .

As an example, consider RLNCF 0x70, F.
Prior to instruction execution: [WREG] = 0x89, [0x70] = 0x32
After instruction execution: [0x70] = 0x64, [WREG] = 0x89 (unchanged), Z = 0 (result in 0x70 after rotating is nonzero) , N = 0 (most significant bit of result, 0x64 is 0) ; no other flags are affected.
Note that RLNCF 0x70, W stores the result in WREG.
The RLNCF instruction can be used to multiply an unsigned number by 2^n by shifting the number, *n* times to the left using a loop as long as a ‘1’ is not shifted out of the most significant bit. In the above example, after shifting [0x70] once, the contents 0x32 of data register 0x70 are multiplied by 2. Hence, [0x70] = 0x64 after shifting.

Rotate Instructions

- RRCF F, d, a (RRCF 0x30, W or RRCF 0x30, F) instruction rotates the contents of register ‘F’ one bit to the right through the Carry flag. If ‘d’ is ‘0’, the result is placed in WREG. If ‘d’ is ‘1’, the result is stored back in register ‘F’. If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the bank .

As an example, consider RRCF 0x30, F.
Prior to instruction execution: [WREG] = 0x91, [0x30] = 0x27 , C = 0
After instruction execution: [0x30] = 0x13, [WREG] = 0x91 (unchanged), C = 1, Z = 0 (result, 0x13 is nonzero), N = 0 (most significant bit of result, 0x13 is 0) ; no other flags are affected.
Note that RRCF 0x30, W stores the result in WREG.

Rotate Instructions

- RRNCF F, d, a (RRNCF 0x60, W or RRNCF 0x60, F) instruction rotates the contents of register ‘F’ one bit to the right. If ‘d’ is ‘0’, the result is placed in WREG. If ‘d’ is ‘1’, the result is stored back in register ‘F’. If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the bank.

As an example, consider RRNCF 0x60, F.
Prior to instruction execution: [WREG] = 0xB3, [0x60] = 0x28
After instruction execution: [0x60] = 0x14, [WREG] = 0xB3 (unchanged), Z = 0 (result, 0x14 after rotating is nonzero), N = 0 (most significant bit of result, 0x14 is 0) ; no other flags are affected.
Note that RRNCF 0x60, W stores the result in WREG.
The RRNCF instruction can be used to divide an unsigned number by 2 by shifting the number *n* times to the right using a loop as long as a ‘1’ is not shifted out of the least significant bit. This means that the remainder is discarded. Since the PIC18F does not have unsigned division instruction, the RRNCF can be used for this purpose. In the above example, after shifting [0x60] once, the contents, 0x28 of data register 0x60, are divided by 2. Hence, [0x60] = 0x14 after shifting.

Rotate Instructions

- Example 6.15 Write a PIC18F logic instruction to convert a 4-bit unsigned number in low 4 bits of WREG into an 8-bit unsigned number in WREG. Assume the 4-bit unsigned number is already loaded into WREG.

Solution

```
ANDLW  0x0F
```

Note that ANDLW 0x0F instruction logically ANDs 8-bit data in WREG with 0x0F. This operation will clear the upper 4 bits to zero, and retain the lower 4 bits. Thus, the 4-bit unsigned number will be converted to an 8-bit unsigned number.

Rotate Instructions

Example 6.16 Write a PIC18F assembly language program at address 0x100 to convert a 4-bit signed number stored in the low 4 bits of WREG into an 8- bit signed number in WREG. That is, if the 4-bit signed number is positive, zero-extend to 8 bits. On the other hand, if the 4-bit signed number is negative, sign-extend to 8 bits. Assume the 4-bit signed number is already loaded into WREG. Store result in WREG. Do not use any ROTATE instructions.

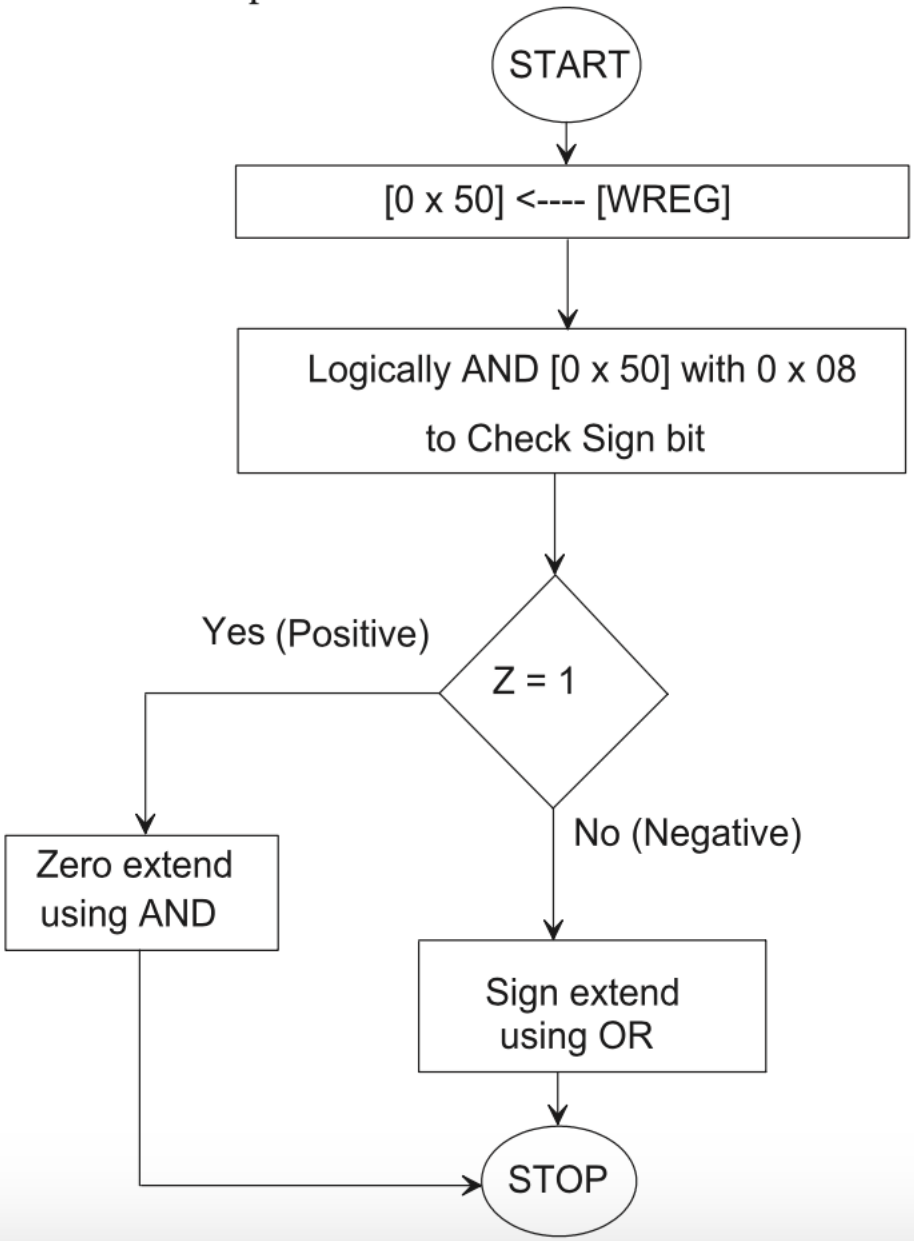
(a) Flowchart the problem.

(b) Convert the flowchart to PIC18F assembly language program starting at address 0x100.

Rotate Instruction

Solution

(a) The flowchart is provided below:



Rotate Instructions

(b) The PIC18F assembly language program is provided below:

```
            INCLUDE  <P18F4321.INC>
            ORG      0x100
            MOVWF    0x50        ; copy [WREG] into data register 0x50
            ANDLW    0x08        ; check bit 3 whether it is 0 (positive) or 1 (negative)
            BZ       POSITIVE    ; if Z = 1, the number is positive
            MOVF     0x50, W     ; else, the number is negative. Move [0x50] into W
            IORLW    0xF0        ; sign extend by inserting 1's in upper 4 bits of  W
            SLEEP                ; Halt. Result in WREG
POSITIVE    MOVF     0x50, W     ; the number is positive. Move [0x50] into WREG
            ANDLW    0x0F        ; the number is positive. hence zero extend upper 4
                                 ; bits
            SLEEP
            END
```

Rotate Instructions

Example 6.17 Write a PIC18F assembly language program at address 0x100 to multiply an 8-bit unsigned number in data register 0x50 by 16. Store the 8-bit result in WREG. Do not use any multiplication instructions. Use ROTATE instruction. Assume that a '1' is not shifted out of the most significant bit each time after rotating to the left. Also, assume that the 8-bit unsigned number is already loaded into data register 0x40.
Write the program

(a) without using a loop (b) using a loop

Rotate Instructions

Solution

(a) without using a loop

The following program will multiply [0x50] by 16 by shifting [0x50] four times to the left:

```
        INCLUDE  <P18F4321.INC>
        ORG      0x100
        BCF      STATUS, C      ; Clear Carry
        RLCF     0x50, F        ; Rotate [0x50] once  to left
        BCF      STATUS, C      ; Clear Carry
        RLCF     0x50, F        ; Rotate [0x50]  once to left
        BCF      STATUS, C      ; Clear Carry
        RLCF     0x50, F        ; Rotate [0x50] once  to left
        BCF      STATUS, C      ; Clear Carry
        RLCF     0x50, F        ; Rotate [0x50]  once to left
        MOVF     0x50, W        ; Save  result in WREG
FOREVER GOTO     FOREVER        ; Stop
        END
```

Rotate Instructions

(b) using a loop

The following program will multiply [0x50] by 16 by shifting [0x50] four times to the left in a loop:

```
            INCLUDE   <P18F4321.INC>
            ORG       0x100
COUNTER     EQU       0x70
            MOVLW     4              ; Initialize COUNTER with 4
BACK        BCF       STATUS, C      ; Clear Carry
            RLCF      0x50, F        ; Rotate [0x50] four times
            DECF      COUNTER, F     ; to left to multiply [0x50] by 16
            BNZ       BACK           ; branch to BACK if  Z = 0
            MOVF      0x50, W        ; Move result to WREG
FOREVER     GOTO      FOREVER        ; Stop
            END
```

Rotate Instructions

Example 6.18 Write a PIC18F assembly language program at address 0x100 that will multiply an 8-bit unsigned number in data register 0x50 by 4 to provide an 8-bit product, and then, perform the following operations on the contents of data register 0x50 :

- set bits 0 and 3 to one without changing other bits in data register 0x50.
- clear bit 5 to zero without changing other bits in data register 0x50.
- one's complement bit 7 without changing other bits in data register 0x50.

Use only "Logic", and "Rotate" instructions. Do not use any multiplication or any other instructions. Assume data is already in data register 0x50. Store result in WREG. Assume that a '1' is not shifted out of the most significant bit each time after rotating to the left.

Rotate Instructions

Solution

```
INCLUDE <P18F4321.INC>
ORG     0x100
BCF     STATUS, C       ; Clear Carry
RLCF    0x50, F         ; Unsigned multiply [0x50] by 2, result in 0x50
BCF     STATUS, C       ; Clear Carry
RLCF    0x50, W         ; Unsigned multiply [0x50] by 4, result in W
IORLW   0x09            ; set bits 0 and 3 in WREG to one's
ANDLW   0xDF            ; clear bit 5  in WREG to zero
XORLW   0x80            ; ones complement  bit 7 in WREG
SLEEP                   ; Stop
END
```

As mentioned before, FINISH GOTO FINISH (unconditionally jumping to the same location) and the instruction SLEEP are equivalent to HALT instruction in other processors. Either can be used in the PIC18F as HALT in the assembly language program.

Rotate Instructions

- Example 6.19: Write a PIC18F assembly language program at address 0x100 to check whether an 8-bit signed number (x) in data register 0x60 is positive or negative. If the number is positive, then compute 16-bit value, $y1 = x^2$, and store the result in PRODH:PRODL. If the number is negative, then compute the 8-bit value, y2 = 2x. Store the result in WREG. Do not use any logic instructions. Assume that the 8-bit number, x, is already loaded in data register 0x60.

Rotate Instructions

Solution

```
            INCLUDE <P18F4321.INC>
            ORG     0x100
            MOVF    0x60, W         ; Move  x into WREG
            MOVFF   0x60, 0x70      ; Save x in 0x70
            RLCF    0x60, F         ; Rotate sign bit to carry to check whether 0 or 1
            BC      NEGATIVE        ; Branch if C = 1
            MULWF   0x60            ; Compute y1 and store in PRODH:PRODL
            GOTO    FINISH          ; Jump to FINISH
NEGATIVE    ADDWF   0x70, W         ; Compute y2 by adding x to itself
            SLEEP
            END
```

Bit Manipulation Instructions

TABLE 6.9 Bit Manipulation Instructions

Instruction	*Comment*
BCF F, b, a	Clear bit number 'b' to 0 in file register F
BSF F, b, a	Set bit number 'b' to 1 in file register F
BTG F, b, a	Toggle (one's complement) bit number 'b' in file register F

- All instructions in the above are executed in one cycle.
- The size of each instruction is one word.
- No flags are affected.
- b' can be from 0 to 7.
- a = 0 means that the data register is located in the access bank while a = 1 means that the contents of BSR specify the address of the bank.

Bit Manipulation Instructions

- BCF F, b, a (BCF 0x50, 2) instruction clears the specified bit 'b' in register 'F' to zero. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the bank. As an example, consider BCF 0x50, 2.

  Prior to instruction execution: $[0x50] = 0x36 = 00110110_2$
  After Instruction execution: Bit 2 is cleared to 0. Hence, $[0x50] = 00110010_2 = 0x32$.
  The BCF instruction can be used to clear the carry flag to 0. For example, BCF STATUS, C will clear the carry flag in the status register to 0.

- BSF F, b, a (BSF 0x30, 5) instruction sets the specified bit 'b' in register 'F' to one. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the bank. As an example, consider BSF 0x30, 5.

  Prior to instruction execution: $[0x30] = 0x0F = 00001111_2$
  After instruction execution: Bit 5 is set to 1. Hence, $[0x30] = 00101111_2 = 0x2F$.
  The BSF instruction can be used to set the carry flag to 1. For example, BSF STATUS, C will set the carry flag in the status register to 1.

Bit Manipulation Instructions

- BTG F, b, a (BTG 0x40, 2) instruction one's complements (toggles) the specified bit 'b' in register 'F'. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the bank. As an example, consider BCF 0x40, 2.

  Prior to instruction execution: $[0x40] = 0x16 = 00010110_2$
  After instruction execution: Bit 2 is one's complemented from 1 to 0. Hence, $[0x40] = 00010010_2 = 0x12$.

  Note that the BTG instruction can be used to toggle a specific bit in an I/O port. For example, BTG PORTB, 1 will toggle bit 1 of Port B. This may be useful sometimes in some I/O applications.

Bit Manipulation Instructions

- Example 6.20: Write a PIC18F assembly language program at address 0x100 that will multiply an 8-bit unsigned number in data register 0x50 by 4 to provide an 8-bit product, and then perform the following operations on the contents of data register 0x50 :
  - set bits 0 and 3 to one without changing other bits in data register 0x50.
  - clear bit 5 to zero without changing other bits in data register 0x50.
  - one's complement bit 7 without changing other bits in data register 0x50.
  - Use only "rotate (RLCF instruction only)" and "Bit manipulation" instructions. Do not use any multiplication or any other instructions. Assume data is already in data register 0x50. Store the result in 0x50. Assume that a '1' is not shifted out of the most significant bit each time after rotating to the left.

Bit Manipulation Instructions

Solution

```
        INCLUDE  <P18F4321.INC>
        ORG      0x100
        BCF      STATUS, C
        RLCF     0x50, F        ; Unsigned multiply [0x50] by 2
        BCF      STATUS, C
        RLCF     0x50, F        ; Unsigned multiply [0x50] by 4, result in W
        BSF      0x50, 0        ; set bit 0 in [0x50] to one
        BSF      0x50, 3        ; set bit 3 in [0x50] to one
        BCF      0x50, 5        ; clear bit 5  in [0x50] to zero
        BTG      0x50,7         ; ones complement  bit 7 in 0x50
        SLEEP                   ; Stop
        END
```

Bit Manipulation Instructions

Example 6.21 Write a PIC18F assembly language program at address 0x100 that will perform : $5 \times X + 6 \times Y + [Y/2] \rightarrow$ [0x71][0x70] where X is an unsigned 8-bit number stored in data register 0x40 and Y is a 4-bit unsigned number stored in the upper 4 bits of data register 0x50. Discard the remainder of $Y/2$. Save the 16-bit result in 0x71 (upper byte) and in 0x70 (lower byte).

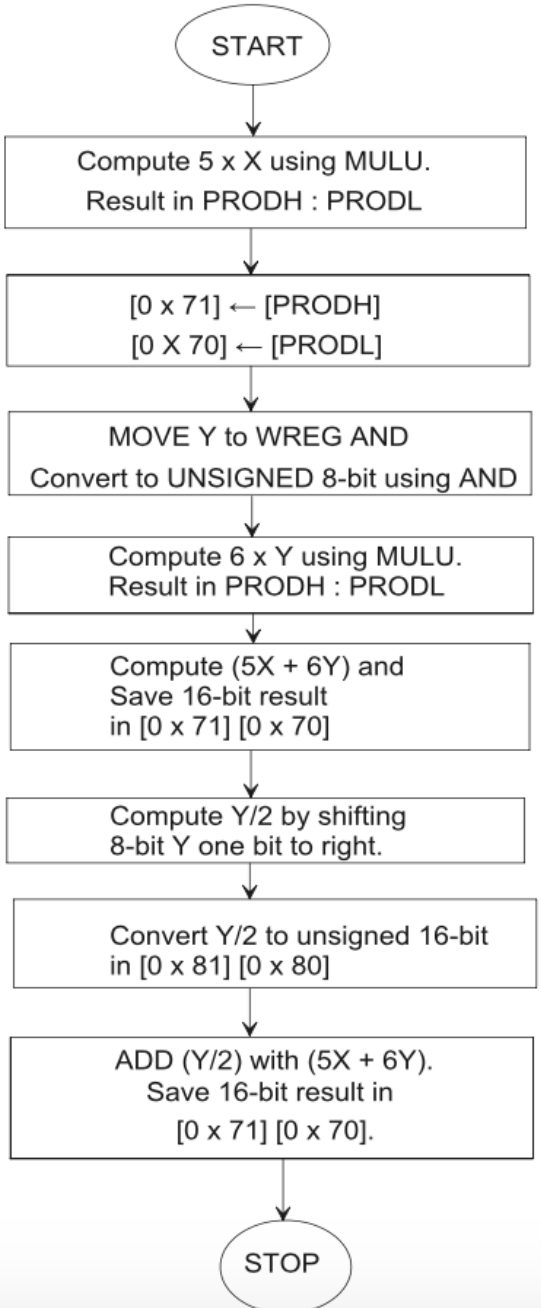
(a) Flowchart the problem.

(b) Convert the flowchart to PIC18F assembly language program starting at address 0x100.

Bit Manipulation Instructions

Solution

(a) The flowchart is provided below:



(b) The PIC18F assembly language program is provided below:

```
        INCLUDE   <P18F4321.INC>
        ORG       0x100
        MOVF      0x40, W        ; MOVE X  TO WREG
        MULLW     5              ; COMPUTE UNSIGNED 16-BIT  5xX IN
                                 ; PRODH:PRODL
        MOVFF     PRODH, 0x71    ; SAVE UPPER BYTE  OF 5xX  IN 0x71
        MOVFF     PRODL, 0x70    ; SAVE LOWER BYTE OF 5xX  IN 0x70
        SWAPF     0x50, W        ; MOVE Y TO LOW 4 BITS IN  WREG
        ANDLW     0x0F           ; CONVERT Y TO UNSIGNED 8-BIT IN WREG
        MOVWF     0x80           ; SAVE Y  FROM WREG TO 0x80
        MULLW     6              ; COMPUTE UNSIGNED 16-bit  6 x Y IN
                                 ; PRODH:PRODL
        MOVFF     PRODL, W       ; MOVE  PRODL  INTO WREG
        ADDWF     0x70, F        ; ADD  LOW BYTES OF 5*X WITH 6*Y, SAVE
                                 ; IN 0x70
        MOVFF     PRODH, W       ; MOVE  PRODH  INTO WREG
        ADDWFC    0x71, F        ; ADD  HIGH BYTES OF 5*X WITH 6*Y WITH
                                 ; CARRY, AND SAVE IN 0x71
        ADDLW     0              ; CLEAR CARRY
        RRCF      0x80, F        ; COMPUTE  Y/2 , 8-BIT RESULT IN   0x80
        CLRF      0x81           ; CONVERT  Y/2 TO UNSIGNED 16-BIT IN
                                 ; [0x81][0x80]
        MOVFF     0x80, W        ; MOVE LOW BYTE OF  Y/2 INTO WREG
        ADDWF     0x70, F        ; PERFORM 5 × X + 6 × Y + [Y/2] FOR LOW
                                 ; BYTES, RESULT IN 0x70
        MOVFF     0x81, W        ; MOVE HIGH BYTE OF  Y/2 INTO WREG
        ADDWFC    0x71, F        ; PERFORM 5 × X + 6 × Y + [Y/2] FOR HIGH
                                 ; BYTES PLUS CARRY , RESULT IN 0x71
FINISH  GOTO      FINISH         ; HALT
        END
```