APPENDIX A: PIC18 Instructions: Format and Description

OVERVIEW

In the first section of this appendix, we describe the instruction format of the PIC18. Special emphasis is placed on the instructions using both WREG and file registers. This section includes a list of machine cycles (clock counts) for each of the PIC18 instructions.

In the second section of this appendix, we describe each instruction of the PIC18. In many cases, a simple programming example is given to clarify the instruction.

This Appendix deals mainly with PIC18 instructions. In Section A.1, we describe the instruction formats and categories. In Section A.2, we describe each instruction of PIC18 with some examples.

SECTION A.1: PIC18 Instruction Formats and Categories

As shown in Figure A-1, the PIC18 instructions fall into five categories:

1. Bit-oriented instructions
2. Instructions using a literal value
3. Byte-oriented instructions
4. Table read and write instructions
5. Control instructions using branch and call

In this section, we describe the format and syntax with special emphasis placed on byte-oriented instructions. For some of the instructions, the reader needs to review the concepts of access bank and bank registers in Chapter 6 (Section 6.3).

Bit-oriented Instructions

The bit-oriented instructions perform operations on a specific bit of a file register. After the operation, the result is placed back in the same file register. For example, the "BCF f,b,a" instruction clears a specific bit of fileReg. See Table A-1. In these types of instructions, the b is the specific bit of the fileReg, which can be 0 to 7, representing the D0 to D7 bits of the register. The fileReg location can be in the bank register called access bank (if a = 0) or a location within other bank registers (if a = 1). Notice that if a = 0, the assembler assumes the access bank automatically.

Mnemonic, Operands		Description	Cycles
BIT-ORIENTED FILE REGISTER OPERATIONS			
BCF	f, b, a	Bit Clear f	1
BSF	f, b, a	Bit Set f	1
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)
BTG	f, d, a	Bit Toggle f	1

Table A-1: Bit-Oriented Instructions (from Microchip datasheet)

Look at the examples that follow for clarification of bit-oriented instructions:

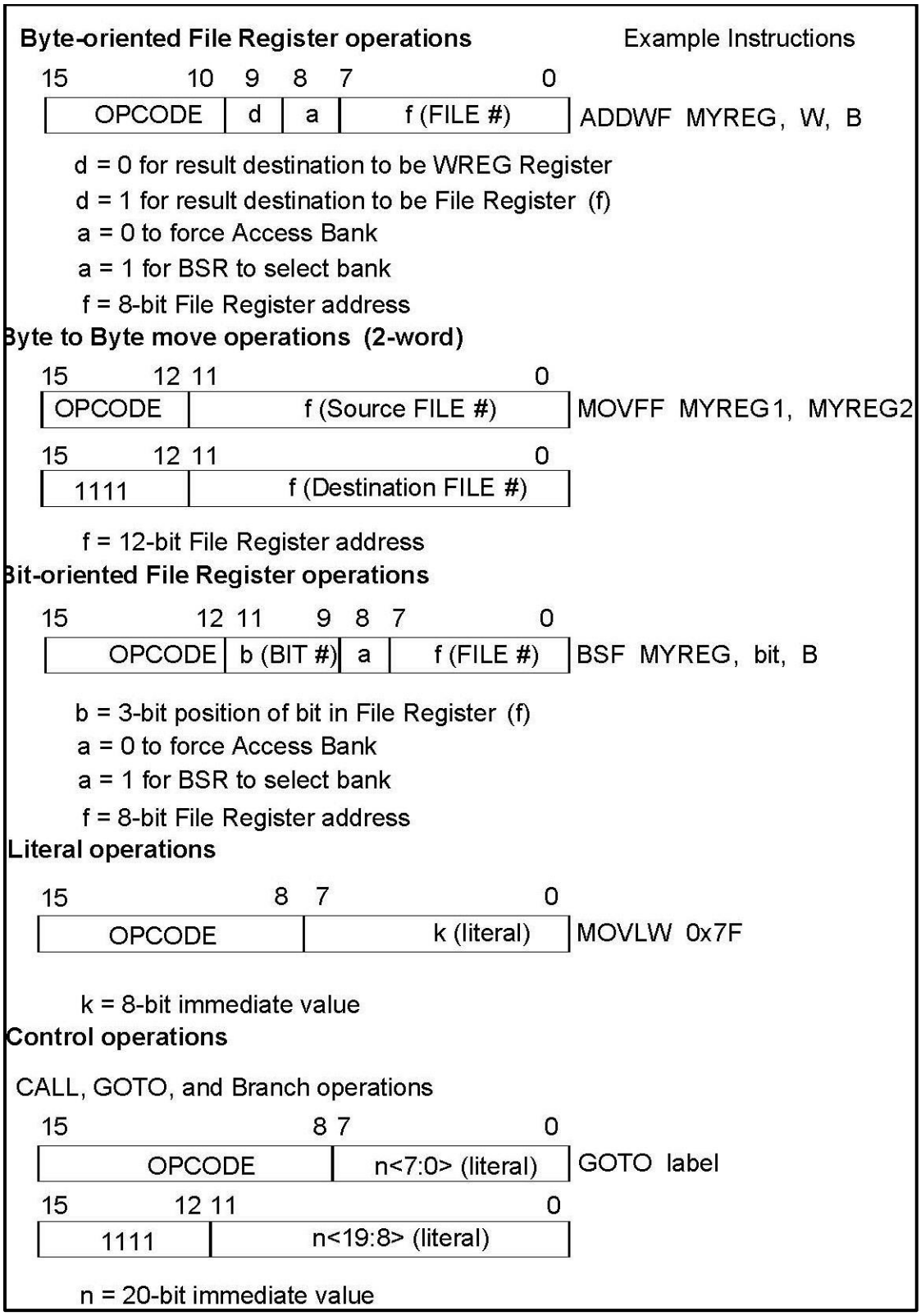


Figure A-1. General Formatting of PIC18 Instructions (From MicroChip)

```
BCF     PORTB,5         ;clear bit D5 of PORTB
```

```
BCF    TRISB,4          ;clear bit D4 of TRISC reg
BTG    PORTC,7    ;toggle bit D7 of PORTC
BTG    PORTD,0    ;toggle bit D0 of PORTD
BSF    STATUS,C   ;set carry flag to one
```

The following example uses the fileReg in the access bank:

```
MyReg   SET  0x30    ;set aside loc 30H for MyReg
MOVLW 0x0            ;WREG = 0
MOVWF MyReg          ;MyReg = 0
BTG MYReg,7          ;toggle bit D7 of MyReg
BTG MYReg,5          ;toggle bit D5 of MyReg
```

The following example uses the fileReg in the access bank:

```
MyReg   SET    0x50   ;set aside loc. 50H for MyReg
MOVLW  0x0            :WREG = 0
MOVWF  MyReg          ;MyReg = 0
BTG      MYReg,2      ;toggle bit D2 of MyReg
BTG      MYReg,4      ;toggle bit D4 of MyReg
```

As we discuss in Chapter 6, when using a bank other than the access bank, we must load the BSR (bank select register) with the desired bank number, which can go from 1 to F (in hex), depending on the family member. We do that by using the MOVLB instruction. Look at the following examples.

The example below uses a location in Bank 2 (RAM locations 200–2FFH).

```
YReg  SET     0x30          ;set aside loc 30H for YReg
MOVLB         0x2           ;use Bank 2 (address loc 230H)
MOVLW         0x0           :WREG = 0
MOVWF         YReg          ;YReg = 0
BTG           YReg,7,1      ;toggle bit D7 of YReg in bank 2
BTG           YReg,5,1      ;toggle bit D5 of YReg in bank 2
```

The example below uses a location in Bank 4 (RAM locations 400–4FFH).

```
ZReg  SET     0x10          ;set aside loc 10H for ZReg
MOVLB         0x4           ;use Bank 4 (address loc 410H)
MOVWL         0x0           ;WREG = 0
MOVWF         ZReg          ;ZReg = 0
BSF           ZReg,6,1      ;set HIGH bit D6 of ZReg in bank 4
BSF           ZReg,1,1      ;set HIGH bit D1 of ZReg in bank 4
```

Notice that all the bit-oriented instructions start with letter B (bit). The branch instructions also start with letter B, like “BZ target” for branch if zero, but they are not bit-oriented.

Mnemonic, Operands		Description	Cycles
LITERAL OPERATIONS			
ADDLW	k	Add literal and WREG	1
ANDLW	k	AND literal with WREG	1
IORLW	k	Inclusive OR literal with WREG	1
LFSR	f, k	Move literal (12-bit) 2nd word to FSRx 1st word	2
MOVLB	k	Move literal to BSR<3:0>	1
MOVLW	k	Move literal to WREG	1
MULLW	k	Multiply literal with WREG	1
RETLW	k	Return with literal in WREG	2
SUBLW	k	Subtract WREG from literal	1
XORLW	k	Exclusive OR literal with WREG	1

Table A-2: Literal Instructions (from Microchip datasheet)

Instructions Using Literal Values

In this type of instruction, an operation is performed on the WREG register and a fixed value called k. See Table A-2. Because WREG is only 8-bit, the k value cannot be greater than 8-bit. Therefore, the k value is between 0–255 (00–FF in hex). After the operation, the result is placed back in WREG. Look at the following examples for clarification:

```
MOVLW    0x45    ;WREG = 45H
ADDLW    0x24    ;WREG = 45H + 24H = 69H

MOVLW    0x35    ;WREG = 35H
ANDLW    0x0F    ;WREG = 35H ANDed with 0FH = 05H

MOVLW    0x55    ;WREG = 55H
XORLW    0xAA    ;WREG = 55H EX-ORed with AAH = FFH
```

Byte-oriented Instructions

There are two groups of instructions in this category. In the first group, the operation is performed on the file register and the result is placed back in the file register. The instruction “CLRF f,a” is an example in this group. See Table A-3. In the second group, the operation involves both fileReg and WREG. As a result, we have the options of placing the result in fileReg or in WREG. As an example in this group, examine the “ADDWF f,d,a” instruction. The destination for the result can be WREG (if d = 0) or file register (if d = 1). For the fileReg location, it can be in the access bank (if a = 0) or in other bank registers (if a = 1). Also notice that if a = 0, the assembler assumes that automatically.

Mnemonic, Operands		Description	Cycles
BYTE-ORIENTED FILE REGISTER OPERATIONS			
ADDWF	f, d, a	Add WREG and f	1
ADDWFC	f, d, a	Add WREG and Carry bit to f	1
ANDWF	f, d, a	Add WREG with f	1
CLRF	f, a,	Clear f	1
COMF	f, d, a	Complement f	1
CPFSEQ	f, a,	Compare f with WREG , skip =	1
CPFSGT	f, a,	Compare f with WREG , skip >	1
CPFSLT	f, a,	Compare f with WREG , skip <	1
DECF	f, d, a	Decrement f	1
DECFSZ	f, d, a	Decrement f, Skip if 0	1
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1
INCF	f, d, a	Increment f	1
INCFSZ	f, d, a	Increment f, Skip if 0	1
INFSNZ	f, d, a	Increment f, Skip if Not 0	1
IORWF	f, d, a	Inclusive OR WREG with f	1
MOVF	f, d, a	Move f	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word f_d (destination) 2nd word	2
MOVWF	f, a	Move WREG to f	1
MULWF	f, a	Multiply WREG with f	1
NEGF	f, a	Negate f	1
RLCF	f, d, a	Rotate Left f through Carry	1
RLNCF	f, d, a	Rotate Left f (No Carry)	1
RRCF	f, d, a	Rotate Right f through Carry	1
RRNCF	f, d, a	Rotate Right f (No Carry)	1
SETF	f, a,	Set f	1
SUBFWB	f, d, a	Subtract f from WREG with borrow	1
SUBWF	f, d, a	Subtract WREG from f	1
SUBWFB	f, d, a	Subtract WREG from f with borrow	1
SWAPF	f, d, a	Swap nibbles in f	1
TSTFSZ	f, a	Test f, Skip if 0	1
XORWF	f, d, a	Exclusive OR WREG with f	1

Table A-3: Byte-Oriented Instructions (from Microchip datasheet)

Look at the following examples.

When d = 0 and a = 0:

```
MyReg       SET  0x20      ;loc 20H for MyReg
MOVLW       0x45           ;WREG = 45H
MOVWF       MyReg          ;MyReg = 45H
MOVLW       0x23           ;WREG = 23H
ADDWF       MyReg          ;WREG = 68H (45H + 23H = 68H)
```

In the above example, the last instruction could have been coded as “ADDWF MyReg,0,0”.

When d = 1 and a = 0:

```
MyReg       SET  0x20              ;loc 20H for MyReg
MOVLW       0x45           ;WREG = 45H
MOVWF       MyReg                  ;MyReg = 45H
MOVLW       0x23                   ;WREG = 23H
ADDWF       MyReg,F     ;MyReg = 68H (45H + 23H = 68H)
```

In the above example, the last instruction could have been coded as “ADDWF MyReg,F,0” or “ADDWF MyReg,1,0”. As far as the MPLAB is concerned, they mean the same thing. Notice that the use of letter F in “ADDWF MyReg,F” is being used in place of 1.

To use banks other than the access bank, we must load the BSR register first. The following example uses a location in Bank 2 (RAM location 200–2FFH).

When d = 0 and a = 1:

```
MyReg       SET     0x30    ;set aside location 30H for MyReg
MOVLB               0x2             ;use Bank 2 (address loc 230H)
MOVLW       0x45            ;WREG = 45H
MOVWF       MyReg,1         ;MyReg = 45H (loc 230H)
MOVLW       0x23            ;WREG = 23H
ADDWF       MyReg,1         ;WREG = 68H (add loc 230H to W)
```

When d = 1 and a = 1:

```
MyReg               SET  0x20       ;loc 20H for MyReg
MOVLB       0x4             ;use bank 4
MOVLW       0x45            ;WREG = 45H
MOVWF       MyReg           ;MyReg = 45H (loc 420H)
MOVLW       0x23            ;WREG = 23H
ADDWF       MyReg,F,1       ;MyReg = 68H (loc 420)
```

Register-indirect addressing mode uses FSRx as a pointer to RAM location. We have three registers, FSR0, FSR1, and FSR2, that can be used for pointers.

Examples:

```
ADDWF POSTINC0    ;add to W data pointed to by FSR0,
                  ;also increment FSR0

ADDWF POSTINC1    ;add to W data pointed to by FSR1
                  ;also increment FSR1
```

See Example 6-6 in Chapter 6.

Table Processing Instructions

The table processing instructions allow us to read fixed data located in the program ROM of the PIC18. See Table A-4. They also allow us to write into the program ROM if it is Flash memory. Chapter 14 discusses the TBLRD and TBLWRT instructions in detail. It also shows how to use table read and write to access the EEPROM.

Mnemonic, Operands	Description	Cycles
DATA ⟷ PROGRAM MEMORY OPERATIONS		
TBLRD*	Table Read	2
TBLRD*+	Table Read with post-increment	2
TBLRD*-	Table Read with post-decrement	2
TBLRD+*	Table Read with pre-increment	2
TBLWT*	Table Write	2
TBLWT*+	Table Write with post-increment	2
TBLWT*-	Table Write with post-decrement	2
TBLWT+*	Table Write with pre-increment	2

Table A-4: Table Processing Instructions (from Microchip datasheet)

Control Instructions

The control instructions such as branch and call deal mainly with flow control. See Table A-5. We must pay special attention to the target address of the control instructions. The target address for some of the branch instructions such as BZ (branch if zero) cannot be farther than 128 bytes away from the current instruction. The CALL instruction allows us to call a subroutine located anywhere in the 2M ROM space of the PIC18. See the individual instructions in the next section for further discussion on this issue.

Mnemonic, Operands		Description	Cycles
CONTROL OPERATIONS			
BC	n	Branch if Carry	1
BN	n	Branch if Negative	1
BNC	n	Branch if Not Carry	1
BNN	n	Branch if Not Negative	1
BNOV	n	Branch if Not Overflow	1
BNZ	n	Branch if Not Zero	1
BOV	n	Branch if Overflow	1
BRA	n	Branch Unconditionally	2
BZ	n	Branch if Zero	1
CALL	n, s	Call subroutine 1st word 2nd word	2
CLRWDT	—	Clear Watchdog Timer	1
DAW	—	Decimal Adjust WREG	1
GOTO	n	Go to address 1st word 2nd word	2
NOP	—	No Operation	1
NOP	—	No Operation	1
POP	—	Pop top of return stack (TOS)	1
PUSH	—	Push top of return stack (TOS)	1
RCALL	n	Relative Call	2
RESET		Software device RESET	1
RETFIE	s	Return from interrupt enable	2
RETLW	k	Return with literal in WREG	2
RETURN	s	Return from Subroutine	2
SLEEP	—	Go into standby mode	1

Table A-5: Control Instructions (from Microchip datasheet)

SECTION A.2: The PIC18 Instruction Set

In this section we provide a brief description of each instruction with some examples.

ADDLW K **Add Literal to WREG**

Function: ADD literal value of k to WREG
Syntax: ADDLW k

This adds the literal value of k to the WREG register, and places the result back into WREG. Because register WREG is one byte in size, the operand k must also be one byte.

The ADD instruction is used for both signed and unsigned numbers. Each one is discussed separately. See Chapter 5 for discussion of signed numbers.

Unsigned Addition

In the addition of unsigned numbers, the status of C, DC, Z, N, and OV may change. The most important of these flags is C. It becomes 1 when there is a carry from D7 out in 8-bit (D0–D7) operations.

Example:

```
MOVLW 0x45      ;WREG = 45H
ADDLW 0x4F      ;WREG = 94H (45H + 4FH = 94H)
                ;C = 0
```

Example:

```
MOVLW  0xFE             ;WREG = FEH
ADDLW  0x75             ;WREG = FE + 75 = 73H
                ;C = 1
```

Example:

```
MOVLW  0x25     ;WREG = 25H
ADDLW  0x42     ;WREG = 67H (25H + 42H = 67H)
          ;C = 0
```

Notice that in all the above examples we ignored the status of the OV flag. Although ADD instructions do affect OV, it is in the context of signed numbers that the OV flag has any significance. This is discussed next.

Signed Addition and Negative Numbers

In the addition of signed numbers, special attention should be given to the overflow flag (OV) because this indicates if there is an error in the result of the addition. There are two rules for setting OV in signed number operation. The overflow flag is set to 1:

1. If there is a carry from D6 to D7 and no carry from D7 out.
1. If there is a carry from D7 out and no carry from D6 to D7.

Notice that if there is a carry both from D7 out and from D6 to D7, OV = 0.

Example:

```
MOVLW    +D'8'      ;W = 0000 1000
ADDLW    +D'4'      ;W = 0000 1100 OV = 0,
                    ;C = 0, N = 0
```

Notice that N = D7 = 0 because the result is positive, and OV = 0 because there is neither a carry from D6 to D7 nor any carry beyond D7. Because OV = 0, the result is correct [(+8) + (+4) = (+12)].

Example:

```
MOVLW   +D'66'     ;W = 0100 0010
ADDLW   +D'69'     ;W = 1000 0101 = -121
ADDWF              ;W = 1000 0111 = -121
                   ;(INCORRECT) C = 0, N = D7 = 1, OV = 1
```

In the above example, the correct result is +135 [(+66) + (+69) = (+135)], but the result was -121. OV = 1 is an indication of this error. Notice that N = 1 because the result is negative; OV = 1 because there is a carry from D6 to D7 and C = 0.

Example:

```
MOVLW   -D'12'     ;W = 1111 0100
ADDLW   +D'18'     ;W = W + (+0001 0010)
                   ;W = 0000 0110 (+6) correct
                   ;N = 0, OV = 0, and C = 1
```

Notice above that the result is correct (OV = 0), because there is a carry from D6 to D7 and a carry from D7 out.

Example:

```
MOVLW    -D'30'    ;W = 1110 0010
ADDLW    +D'14'    ;W = W + 0000 1110
                   ;W = 1111 0000 (-16, CORRECT)
              ;N = D7 = 1, OV = 0, C = 0
```

OV = 0 because there is no carry from D7 out nor any carry from D6 to D7.

Example:

```
MOVLW    -D'126'   ;W = 1000 0010
ADDLW    -D'127'   ;W = W + 1000 0001
                   ;W = 0000 0011 (+3, INCORRECT)
                   ;D7 = N = 0, OV = 1
```

C = 1 because there is a carry from D7 out but no carry from D6 to D7.

From the above discussion we conclude that while Carry is important in any addition, OV is extremely important in signed number addition because it is used to indicate whether or not the result is valid. As we will see in instruction "DAW", the DC flag is used in the addition of BCD numbers.

ADDWF **Add WREG and f**

Function:	ADD WREG and fileReg
Syntax:	ADDWF f,d,a

This adds the fileReg value to the WREG register, and places the result in WREG (if d = 0) or fileReg (if d = 1).

The ADDWF instruction is used for both signed and unsigned numbers. (See ADDLW instruction.)

Example:

```
MyReg       SET  0x20      ;loc 20H for MyReg
MOVLW       0x45           ;WREG = 45H
MOVWF       MyReg          ;MyReg = 45H
MOVLW       0x4F           ;WREG = 4FH
ADDWF       MyReg          ;WREG = 94H (45H + 4FH = 94H)
                           ;C = 0
```

We can place the result in fileReg, as shown in the following example:

```
MyReg    SET  0x20      ;loc 20H for MyReg
MOVLW       0x45        ;WREG = 45H
MOVWF       MyReg       ;MyReg = 45H
MOVLW       0x4F        ;WREG = 4FH
ADDWF       MyReg,F     ;MyReg = 94H
                        ;(45H + 4FH = 94H), C = 0
```

For cases of a = 0 and a = 1, see Section A.1 in this chapter.

ADDWFC **Add WREG and Carry flag to fileReg**

Function: ADD WREG and Carry bit to fileReg
Syntax: ADDWFC f,d,a

This will add WREG and the C flag to fileReg (Destination = WREG + fileReg + C). If C = 1 prior to this instruction, 1 is also added to destination. If C = 0 prior to the instruction, source is added to destination plus 0. This instruction is used in multibyte additions. In the addition of 25F2H to 3189H, for example, we use the ADDWFC instruction as shown below.

Example when d = 0:

Assume we have the following data in RAM locations 0x10 and 0x11
0x10 = (F2)
0x11 = (25)

```
Reg_L       SET     0x10        ;loc 0x10 for Reg_L
Reg_H       SET     0x11        ;loc 0x11 for Reg_H
BCF         STATUS,C            ;make carry = 0
MOVLW       89H                 ;WREG = 89H
ADDWFC  Reg_L,1     ;Reg_L = 89H + F2H + 0 = 7BH
                                ;and C = 1
MOVLW 0x31                      ;WREG = 31H
ADDWFC Reg_2,1      ;Reg_H = 31H + 25H + 1 = 57H
```

Therefore, the result is:

```
  25F2H
+3189H
  577BH
```

ANDLW **AND Literal byte with WREG**

Function: Logical AND literal value k with WREG
Syntax: ANDLW k

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

This performs a logical AND on the WREG and the Literal byte operand, bit by bit, storing the result in the WREG.

Example:

```
MOVLW  0x39   ;W = 39H
ANDLW  0x09   ;W = 39H ANDed with 09

        39H  0011 1001
        09H  0000 1001
        09H  0000 1001
```

Example:

```
MOVLW 32H       ;W = 32H     32H  0011 0010
ANDLW 50H       ;AND W with          50H  0101 0000
                ;(W = 10H)     10H  0001 0000
```

ANDWF **AND WREG with fileReg**

Function: Logical AND for byte variables
Syntax: ANDWF f,d,a

This performs a logical AND on the fileReg value and the WREG register, bit by bit, and places the result in WREG (if d = 0) or fileReg (if d = 1).

Example:

```
        MyReg  SET 0x40;set MyReg loc at 0x40
        MOVLW  0x39 ;W = 39H
        MOVWF  MyReg        ;MyReg = 39H
        MOVLW  0x09
ANDWF  MyReg  ;39H ANDed with 09 (W = 09)

        39H  0011 1001
        09H  0000 1001
        09H  0000 1001
```

Example:

```
MyReg    SET 0x40     ;set MyReg loc at 0x40
MOVLW    0x32                 ;W = 32H
MOVWF    MyReg        ;MyReg = 32H
MOVLW    0x0F                 ;WREG = 0FH
ANDLW    MyReg        ;32H ANDed with 0FH (W = 02)
```

```
32H  0011 0010
0FH  0000 1111
02H  0000 0010
```

We can place the result in fileReg as shown in the examples below:

```
MyReg       SET 0x40    ;set MyReg loc at 0x40
MOVLW 0x32              ;W = 32H
MOVWF MyReg     ;MyReg = 32H
MOVLW 0x50              ;WREG = 50H
ANDLW MyReg,F   ;MyReg = 09, WREG = 50H
```

The instructions below clear (mask) certain bits of the output ports, assuming the ports are configured as output ports:

```
MOVLW 0xFE
ANDWF PORTB,F   ;mask PORTB.0 (D0 of Port B)
MOVLW 0x7F
ANDWF PORTC,F   ;mask PORTC.7 (D7 of Port C)
MOVLW 0xF7
ANDWF PORTD,F   ;mask PORTD.3 (D3 of Port D)
```

Branch Condition

Function: Conditional Branch (jump)

In this type of Branch (jump), control is transferred to a target address if certain conditions are met. The following is list of branch instructions dealing with the flags:

BC	Branch if carry	jump if C = 1
BNC	Branch if no carry	jump if C = 0
BZ	Branch if zero	jump if Z = 1
BNZ	Branch if no zero	jump if Z = 0
BN	Branch if negative	jump if N = 1
BNN	Branch if no negative	jump if N = 0
BOV	Branch if overflow	jump if OV = 1
BNOV	Branch if no overflow	jump if OV = 0

Notice that all “Branch condition” instructions are short jumps, meaning that the target address cannot be more than -128 bytes backward or +127 bytes forward of the PC of the instruction following the jump. In other words, the target address cannot be more than -128 to +127 bytes away from the current PC. What happens if a programmer needs to use a “Branch condition” to go to a target address beyond the -128 to +127 range? The solution is to use the “Branch condition” along with the unconditional GOTO instruction, as shown below.

```
 ORG  0x100
MOVLW 0x87  ;WREG = 87H
ADDLW 0x95  ;C = 1 after addition
BNC   NEXT  ;branch if C = 0
GOTO  OVER  ;target more than 128 bytes away
NEXT: ...
```

```
        ...
        ...
ORG          0x5000
OVER:        MOVWF PORTD
```

BC **Branch if C = 1**

Function: Branch if Carry flag bit = 1
Syntax: BC target_address

This instruction branches if C = 1.

Example:

```
MOLW   0x0          ;WREG = 0
BACK  ADDLW 0x1     ;add 1 to WREG
BC     EXIT         ;exit if C = 1
BRA    BACK         ;keep doing it
EXIT   .....
       .....
```

Notice that this is a 2-byte instruction; therefore, the target address cannot be more than -128 to +127 bytes away from the program counter. See Branch Condition for further discussion on this issue.

BCF **Bit Clear fileReg**

Function: Clear bit of a fileReg
Syntax: BCF f,b,a

This instruction clears a single bit of a given file register. The bit can be the directly addressable bit of a port, register, or RAM location. Here are some examples of its format:

```
BCF   STATUS,C    ;C = 0
BCF   PORTB,5     ;CLEAR PORTB.5 (PORTB.5 = 0)
BCF   PORTC,7     ;CLEAR PORTC.7 (PORTC.7 = 0)
BCF   MyReg,1     ;CLEAR D1 OF File Register MyFile
```

N **Branch if N = 1**

Function: Jump if Negative flag bit = 1
Syntax: BN target_address

This instruction branches if N = 1. It is used in signed number addition. See ADDLW instruction. Notice that this is a 2-byte instruction; therefore, the target address cannot be more than -128 to +127 bytes away from the program counter. See Branch Condition for further discussion on this issue.

BNC **Branch if no Carry**

Function: Branch if Carry flag is 0
Syntax: BNC target_address

This instruction examines the C flag, and if it is zero it will jump (branch) to the target address.

Example: Find the total sum of the bytes F6H, 98H, and 8AH. Save the carries in register C_Reg.

```
C_Reg   SET   0x20    ;set aside loc 0x20 for carries

        MOVLW 0x0          ;W = 0
        MOVWF C_Reg        ;C_Reg = 0
        ADDLW 0xF6
        BNC  OVER1
        INCF C_Reg,F
OVER1:  ADDLW 0x98
        BNC    OVER2
        INCF  C_Reg,F
OVER2:  ADDWF 0x8A
        BNC    OVER3
        INCF   C_Reg
OVER3:
```

Notice that this is a 2-byte instruction; therefore, the target address cannot be more than -128 to +127 bytes away from the program counter. See Branch Condition for further discussion on this.

BNN **Branch if Not Negative**

Function: Branch if Negative flag bit = 0
Syntax: BNN target address

This instruction branches if N = 0. It is used in signed number addition. See ADDLW instruction. Notice that this is a 2-byte instruction; therefore, the target address cannot be more than -128 to +127 bytes away from the program counter. See Branch Condition for further discussion on this issue.

BNOV **Branch if No Overflow**

Function: Jump if overflow flag bit = 0
Syntax: BNOV target_address

This instruction branches if OV = 0. It is used in signed number addition. See ADDLW instruction. Notice that this is a 2-byte instruction; therefore, the target address cannot be more than -128 to +127 bytes away from the program counter. See Branch Condition for further discussion on this issue.

BNZ **Branch if No Zero**

Function: Jump if Zero flag is 0
Syntax: BNZ target_address

This instruction branches if Z = 0.

Example:

```
        CLRF TRISB    ;PORTB as output
        CLRF   PORTB;clear PORTB
OVER    INCF PORTB,F          ;INC PORTB
        BNZ    OVER  ;do it until it becomes zero
```

Example: Add value 7 to WREG five times.

```
        COUNTER      SET     0x20    ;loc 20H for COUNTER
```

```
        MOVLW 0x5          ;WREG = 5
        MOVWF COUNTER      ;COUNTER = 05
        MOVLW 0x0          ;WREG = 0
OVER    ADDLW 0x7          ;add 7 to WREG
        DECF  COUNTER,F    ;decrement counter
        BNZ     OVER       ;do it until counter is zero
```

Notice that this is a 2-byte instruction; therefore, the target address cannot be more than -128 to +127 bytes away from the program counter. See Branch Condition for further discussion on this issue.

BOV **Branch if Overflow**

Function: Jump if Overflow flag = 1
Syntax: BOV target_address

This instruction jumps if OV = 1. It is used in signed number addition. See ADDLW instruction. Notice that this is a 2-byte instruction; therefore, the target address cannot be more than -128 to +127 bytes away from the program counter. See Branch Condition for further discussion on this issue.

BRA **Branch unconditional**

Function: Branch unconditionally
Syntax: BRA target_address

BRA stands for "Branch." It transfers program execution to the target address unconditionally. The target address for this instruction must be within 1K of program memory. This is a 2-byte instruction. The first 5 bits is the opcode and the rest is the signed number displacement, which is added to the PC (program counter) of the instruction following the BRA to get the target address. Therefore, in this branch, the target address must be within -1024 to +1023 bytes of the PC (program counter) of the instruction after the BRA because the 11-bit address can take values of +1024 to -1023. This address is often referred to as a relative address because the target address is -1024 to +1023 bytes relative to the program counter (PC).

BSF **Bit Set fileReg**

Function: Set bit
Syntax: BSF f, b, a

This sets HIGH the indicated bit of a file register. The bit can be any directly addressable bit of a port, register, or RAM location.

Examples:

```
BSF  PORTB,3 ;make PORTB.3 = 1
BSF     PORTC,6        ;make PORTC.6 = 1
BSF  MyReg,2  ;make bit D2 of MyReg = 1
BSF  STATUS,C          ;set Carry Flag C = 1
```

BTFSC **Bit Test fileReg, Skip if Clear**

Function: Skip the next instruction if bit is 0
Syntax: BTFSC f, b,a

This instruction is used to test a given bit and skip the next instruction if the bit is low. The given bit can be any of the bit-addressable bits of RAM, ports, or registers of the PIC18.

Example: Monitor the PORTB.5 bit continuously and, when it becomes low, put 55H in WREG.

```
        BSF TRISB,5         ;make PORTB.5 an input bit
HERE    BTFSC PORTB,5       ;skip if PORTB.5 = 0
        BRA   HERE
        MOVLW 0x55          ;because PORTB.5 = 0,
                            ;put 55H in WREG
```

Example: See if WREG has an even number. If so, make it odd.

```
        BTFSC WREG,0        ;skip if it is odd
        BRA   NEXT
        ADDLW 0x1           ;it is even, make it odd
NEXT:       ...
```

BTFSS Bit Test fileReg, Skip if Set

Function: Skip the next instruction if bit is 1
Syntax: BTFSS f, b, a

This instruction is used to test a given bit and skip the next instruction if the bit is HIGH. The given bit can be any of the bit-addressable bits of RAM, ports, or registers of the PIC18.

Example: Monitor the PORTB.5 bit continuously and when it becomes HIGH, put 55H in WREG.

```
        BSF TRISB,5         ;make PORTB.5 an input bit
HERE    BTFSS PORTB,5       ;skip if PORTB.5 = 1
        BRA   HERE
        MOVLW 55H           ;because PORTB.5 = 0 WREG = 55H
```

Example: See if WREG has an odd number. If so, make it even.

```
        BTFSS WREG,0        ;skip if it is even
        BRA   NEXT
        ADDLW 0x01          ;it is even, make it odd
NEXT: ...
```

BTG Bit Toggle fileReg

Function: Toggle (Complement) bit
Syntax: BTG f, b, a

This instruction complements a single bit. The bit can be any bit-addressable location in the PIC18.

Example:

```
        BCF TRISB,0 ;make PORTB.0 an output
AGAINBTG PORTB,0    ;complement PORTB.0 bit
```

```
        BRA AGAIN    ;continuously forever
```

Example: Toggle PORTB.7 a total of 150 times.

```
        COUNTER    SET    0x20    ;loc 20H for COUNTER
        MOVLW 'D'150      ;WREG = 150
        MOVWF COUNTER     ;COUNTER = 150
        BCF    TRISB,7    ;make PORTB.7 an output
OVER    BTG    PORTB.7    ;toggle PORTB.7
        DECF COUNTER,F    ;decrement and put it in
                          ;COUNTER
        BNZ    OVER       ;do it 150 times
```

BZ **Branch if Zero**

Function: Branch if Z = 1
Syntax: BZ target_address

Example: Keep checking PORTB for value 99H.

```
        SETF TRISB        ;port B as input
BACK    MOVFW PORTB       ;get PORTB into WREG
        SUBLW 0x99        ;subtract 99H from it
        BZ     EXIT       ;if 0x99, exit
        BRA  BACK         ;keep checking
        ...
EXIT:   ...
```

Example: Toggle PORTB 150 times.

```
MyReg SET    0x40         ;loc 40H for MyReg
        SETF TRISB        ;port B as output
        MOVLW D'150'      ;WREG = 150
        MOVWF MyReg
BACK COMF PORTB           ;toggle PORTB
        DECF MyReg,F      ;decrement MyReg
        BZ     EXIT       ;if MyReg = 0, exit
        BRA  BACK         ;keep toggling
        ...
EXIT:   ...
```

Notice that this is a 2-byte instruction; therefore, the target address cannot be more than -128 to +127 bytes away from the program counter. See Branch Condition for further discussion on this.

CALL

Function: Transfers control to a subroutine
Syntax: CALL k,s ;s is used for fast context switching

The Call instruction is a 4-byte instruction. The first 12 bits are used for the opcode and the rest (20 bits) are set aside for the address. A 20-bit address allows us to reach the target address anywhere in the 2M ROM space of the PIC18. If calling a subroutine, the PC register (which has the address of the

instruction after the CALL) is pushed onto the stack and the stack pointer (SP) is incremented by 1. Then the program counter is loaded with the new address and control is transferred to the subroutine. At the end of the procedure, when RETURN is executed, PC is popped off the stack, which returns control to the instruction after the CALL.

Notice that CALL is a 4-byte instruction, in which 12 bits are the opcode, and the other 20 bits are the 20-bit address of an even address location. Because all the PIC18 instructions are 2 bytes in size, the lowest address bit, A0, is automatically set to zero to make sure that the CALL instruction will not land at the middle of the targeted instruction. The 20-bit address of the CALL provides the A20–A1 part of the address and with the A0 = 0, we have the 21-bit address needed to go anywhere in the 2M address space of the PIC18.

We have two options for the "CALL k,s" instruction. They are s = 0, and s = 1. When s = 0, it is simply calling a subroutine. With s = 1, we are calling a subroutine and we are also asking the CPU to save the three major registers of WREG, STATUS, and BSR in internal buffers (shadow registers) for the purpose of context-switching. This fast context-switching can be used only in the main subroutine because the depth of the shadow registers is only one. That means no nested call with the s = 1. Look at the following case:

```
                ORG    0x0
MAIN            .....
                .....
                .....
                CALL   M_SUB,1      ;call and save the registers
                MOVLW 0x55          ;address of this instruction is saved on stack
                ....

;-------------------------
                ORG    0x2000
M_SUB           ......
                .....
                CALL   Y_SUB        ;we cannot use CALL Y_SUB,1
                MOVLW 0xAA          ;address of this instruction is saved on stack
                .....
                .....
                RETURN,1            ;return to caller and restore the registers
                                    ;notice the s = 1 for RETURN
;-----------------------------------
                ORG 0x3000
Y_SUB
                .....
                .....
                RETURN

;-------------------------------
                END
```

As shown in RETURN instruction, we also have two options for the RETURN: s = 0 and s = 1. If we use s = 1 for the CALL, we must also use s = 1 for the RETURN. Notice that "CALL Target" with no

number after it is interpreted as s = 0 by the assembler. Likewise, the “RETURN” with no number after it is interpreted as s = 0 by the assembler.

CLRF **Clear fileReg**

Function: Clear
Syntax: CLRF f, a

This instruction clears the entire byte in the fileReg. All bits of the register are cleared to 0.

Example:

```
MyReg SET    0x20          ;loc 20H for MyReg
      CLRF    MyReg        ;clear MyReg
      CLRF   TRISB         ;clear TRISB (make PORTB output)
      CLRF   PORTB         ;clear PORTB
      CLRF TMR01L          ;TMR0L = 0
```

Notice that in this instruction the result can be placed in fileReg only and there is no option for the WREG to be used as the destination.

CLRWDT

Function: Clear Watchdog Timer
Syntax: CLRWDT

This instruction clears the Watchdog Timer.

COMF **Complement the fileReg**

Function: Complement a fileReg
Syntax: COMF f, d, a

This complements the contents of a given fileReg. The result is the 1's complement of the register; that is, 0s become 1s and 1s become 0s. The result can be placed in WREG (if d = 0) or fileReg (if d = 1).

Example:

```
      MOVLW 0x0          ;WREG = 0
      MOVWF TRISB        ;Make PORTB an output port
      MOVLW 0x55         ;WREG = 01010101
      MOVWF PORTB
AGAINCOMF PORTB,F        ;complement (toggle) PORTB
      CALL DELAY
      BRA    AGAIN       ;continuously (notice WREG = 55H)
```

Example:

```
      MyReg  SET 0x40    ;set MyReg loc at 0x40
       MOVLW  0x39       ;W = 39H
      MOVWF  MyReg       ;MyReg = 39H
      COMPF  MyReg,F     ;MyReg = C6H and WREG = 39H
```

Where 39H (0011 1001 bin) becomes C6H (1100 0110).

Example:

```
MyReg  SET 0x40      ;set MyReg loc at 0x40
MOVLW 0x55           ;W = 55H
MOVWF MyReg          ;MyReg = 55H
COMPF MyReg,F        ;MyReg AAH, WREG = 55H
```

where 55H (0101 0101) becomes AAH (1010 1010).

Example: Toggle PORTB 150 times.

```
        COUNTER    SET  0x40      ;loc 40H for COUNTER
          SETF TRISB               ;port B as output
        MOVLW D'150'               ;WREG = 150
        MOVWF COUNTER              ;COUNTER = 150
        MOVLW 0x55                 ;WREG = 55H
        MOVWF PORTB
BACK    COMF PORTB,F               ;toggle PORTB
        DECF   COUNTER,F           ;decrement COUNTER
        BNZ    BACK                ;toggle until counter becomes 0
```

We can place the result in WREG as shown in the examples below:

```
MyReg  SET 0x40              ;set MyReg loc at 0x40
 MOVLW  0x39                 ;W = 39H
        MOVWF  MyReg         ;MyReg = 39H
        COMPF  MyReg         ;MyReg = 39H and WREG = C6H
```

Example:

```
MyReg  SET    0x40           ;set MyReg loc at 0x40
       MOVLW 0x55           ;W = 55H
       MOVWF MyReg          ;MyReg = 55H
       COMPF MyReg          ;WREG = AA and MyReg 55H SETF
```

CPFSEQ **Compare FileReg with WREG and skip if equal (F = W)**

Function: Compare fileReg and WREG and skip if they are equal
Syntax: CPFSEQ f, a

The magnitudes of the fileReg byte and WREG byte are compared. If they are equal, it skips the next instruction.

Example: Keep monitoring PORTB indefinitely for the value of 99H. Get out only when PORTB has the value 99H.

```
SETF  TRISB              ;PORTB an input port
      MOVLW 0x99         ;WREG = 99h
BACK  CPFSEQ PORTB       ;skip if PORTB has 0x99
      BRA BACK           ;keep monitoring
```

Notice that CPFSEQ skips only when fileReg and WREG have equal values.

CPFSGT **Compare FileReg with WREG and skip if greater (F > W)**

Function: Compare fileReg and WREG and skip if fileReg > WREG.
Syntax: CPFSGT f, a

The magnitudes of the fileReg byte and WREG byte are compared. If fileReg is larger than the WREG, it skips the next instruction.

Example: Keep monitoring PORTB indefinitely for the value of 99H. Get out only when PORTB has a value greater than 99H.

```
        SETF  TRISB      ;PORTB an input port
        MOVLW 0x99       ;WREG = 99H
BACK    CPFSGT PORTB     ;skip if PORTB > 99H
        BRA BACK         ;keep monitoring
```

Notice that CPFSGT skips only if FileReg is greater than WREG.

CPFSLT **Compare FileReg with WREG and skip if less than (F < W)**

Function: Compare fileReg and WREG and skip if fileReg < WREG.
Syntax: CPFSLT f, a

The magnitudes of the fileReg byte and WREG byte are compared. If fileReg is less than the WREG, it skips the next instruction.

Example: Keep monitoring PORTB indefinitely for the value of 99H. Get out only when PORTB has a value less than 99H.

```
        SETF  TRISB      ;PORTB an input port
        MOVLW 0x99       ;WREG = 99H
BACK: CPFSEQ PORTB       ;skip if PORTB < 99H
        BRA BACK         ;keep monitoring
```

Notice that CPFSLT skips only if FileReg < WREG.

DAW

Function: Decimal-adjust WREG after addition
Syntax: DAW

This instruction is used after addition of BCD numbers to convert the result back to BCD. The data is adjusted in the following two possible cases:

1. It adds 6 to the lower 4 bits of WREG if it is greater than 9 or if DC = 1.
2. It also adds 6 to the upper 4 bits of WREG if it is greater than 9 or if C = 1.

Example:

```
MOVLW 0x47    ;WREG = 0100 0111
ADDLW 0x38    ;WREG = 47H + 38H = 7FH,
              ;invalid BCD
DAW            ;WREG = 1000 0101 = 85H, valid BCD
```

```
  47H
+ 38H
  7FH    (invalid BCD)
+ 6H     (after DAW)
  85H    (valid BCD)
```

In the above example, because the lower nibble was greater than 9, DAW added 6 to WREG. If the lower nibble is less than 9 but DC = 1, it also adds 6 to the lower nibble. See the following example:

```
MOVLW 0x29    ;WREG = 0010 1001
ADDLW 0x18    ;WREG = 0100 0001 INCORRECT
DAW           ;WREG = 0100 0111 = 47H VALID BCD
```

```
  29H
+ 18H
  41H    (incorrect result in BCD)
+ 6H
  47H    correct result in BCD
```

The same thing can happen for the upper nibble. See the following example:

```
MOVLW 0x52    ;WREG = 0101 0010
ADDLW 0x91    ;WREG = 1110 0011 INVALID BCD
DAW           ;WREG = 0100 0011 AND C = 1
```

```
  52H
+ 91H
  E3H        (invalid BCD)
+ 6          (after DAW, adding to upper nibble)
 143H        valid BCD
```

Similarly, if the upper nibble is less than 9 and C = 1, it must be corrected. See the following example:

```
MOVLW 0x94    ;W = 1001 0100
ADDLW 0x91    ;W = 0010 0101 INCORRECT
DAW           ;W = 1000 0101, VALID BCD
              ;FOR 85, C = 1
```

```
  94H
+ 91H
 125H        (incorrect BCD)
+ 6          (after DAW, adding to upper nibble)
185
```

It is possible that 6 is added to both the high and low nibbles. See the following example:

```
MOVLW  0x54   ;WREG = 0101 0100
ADDLW  0x87   ;WREG = 1101 1011 INVALID BCD
DAW           ;WREG = 0100 0001, C = 1 (BCD 141)
```

```
  54H
```

```
  + 8 7 H
    D BH        (invalid result in BCD)
  + 6 6H
    141H                valid BCD
```

DECF **Decrement fileReg**

Function: Decrement fileReg
Syntax: DECF f, d, a

This instruction subtracts 1 from the byte operand in fileReg. The result can be placed in WREG (if d = 0) or fileReg (if d = 1).

Example:

```
        MyReg SET 0x40      ;set aside loc 40H for MyReg
        MOVLW 0x99          ;WREG = 99H
        MOVWF MyReg         ;MyReg = 99H
        DECF MyReg,F        ;MyReg = 98H, WREG 99H
        DECF MyReg,F        ;MyReg = 97H, WREG 99H
        DECF MyReg,F        ;MyReg = 96H, WREG 99H
```

Example: Toggle PORTB 250 times.

```
        COUNTER   SET   0x40  ;loc 40H for COUNTER
        SETF TRISB            ;PORTB as output
        MOVLW D'250'          ;WREG = 250
        MOVWF COUNTER         ;COUNTER = 250
        MOVLW 0x55            ;WREG = 55H
        MOVWF PORTB
BACK    COMF PORTB,F          ;toggle PORTB
        DECF COUNTER,F        ;decrement COUNTER
        BNZ  BACK             ;toggle until counter becomes 0
```

We can place the result in WREG as shown in the examples below:

```
        MyReg SET 0x40      ;set aside loc for MyReg
        MOVLW 0x99          ;WREG = 99H
        MOVWF MyReg         ;MyReg = 99H
        DECF MyReg          ;WREG = 98H, MyReg = 99H
        DECF MyReg          ;WREG = 97H, MyReg = 99H
        DECF MyReg          ;WREG = 96H, MyReg = 99H
```

Example:

```
        MyReg  SET   0x50   ;set MyReg loc at 0x50
        MOVLW  0x39         ;W = 39H
        MOVWF  MyReg        ;MyReg = 39H
        DECF   MyReg        ;WREG = 38H and MyReg = 39H
        DECF   MyReg        ;WREG = 37H and MyReg = 39H
        DECF   MyReg        ;WREG = 36H and MyReg = 39H
        DECF   MyReg        ;WREG = 35H and MyReg = 39H
```

DECFSZ **Decrement fileReg and Skip if zero**

Function: Decrement fileReg and skip if fileReg has zero in it
Syntax: DECFSZ f, d, a

This instruction subtracts 1 from the byte operand of fileReg. If the result is zero, then it skips execution of the next instruction.

Example: Toggle PORTB 250 times.

```
        COUNT   SET   0x40   ;loc 40H for COUNT
        CLRF TRISB           ;PORTB an output
        MOVLW D'250'         ;WREG = 250
        MOVWF COUNT          ;COUNT = 250
        MOVLW 0x55           ;WREG = 55H
        MOVWF PORTB
BACK COMF PORTB,F            ;toggle PORTB
        DECFSZ COUNT,F        ;decrement COUNT and
                             ;skip if zero
        BRA   BACK           ;toggle until counter becomes 0
        ....
```

DECFSNZ **Decrement fileReg and skip if not zero**

Function: Decrement fileReg and skip if fileReg has other than zero
Syntax: DECFSNZ f, d, a

This instruction subtracts 1 from the byte operand of fileReg. If the result is not zero, then it skips execution of the next instruction.

Example: Toggle PORTB 250 times continuously.

```
COUNT   SET   0x40           ;loc 40H for COUNT
        CLRF TRISB           ;PORTB an output
OVER    MOVLW D'250'         ;WREG = 250
        MOVWF COUNT          ;COUNT = 250
        MOVLW 0x55           ;WREG = 55H
        MOVWF PORTB
BACK COMF PORTB,F            ;toggle PORTB
        DECFSNZ COUNT,F      ;decrement COUNT and
                             ;skip if zero
        BRA   OVER           ;start over
        BRA   BACK           ;toggle until counter becomes 0
```

GOTO **Unconditional Branch**

Function: Transfers control unconditionally to a new address.
Syntax: GOTO k

In the PIC18 there are two unconditional branches (jumps): GOTO (long jump) and BRA (short jump). Each is described next.

1. GOTO (long jump): This is a 4-byte instruction. The first 12 bits are the opcode, and the next 20 bits are an even address of the target location. Because all the PIC18 instructions are 2 bytes in size, the lowest address bit, A0, is automatically set to zero to make sure that the GOTO instruction will not land at the middle of the targeted instruction. The 20-bit address of the GOTO provides the A20–A1 part of the address and with A0 = 0, we have the 21-bit address needed to go anywhere in the 2M address space of the PIC18.
2. BRA: This is a 2-byte instruction. The first 5 bits are the opcode and the remaining 11 bits are the signed number displacement, which is added to the PC (program counter) of the instruction following the BRA to get the target address. Therefore, for the BRA instruction the target address must be within -1023 to +1024 bytes of the PC of the instruction after the BRA because a 11-bit address can take values of +1023 to -1024.

While GOTO is used to jump to any address location within the 2M code space of the PIC18, BRA is used to jump to a location within the 1K ROM space. The advantage of BRA is the fact that it takes 2 bytes of program ROM, while GOTO takes 4 bytes. BRA is widely used in chips with a small amount of program ROM and a limited number of pins.

Notice that the difference between GOTO and CALL is that the CALL instruction will return and continue execution with the instruction following the CALL, whereas GOTO will not return.

INCF **Increment fileReg**

Function: Increment
Syntax: INCF f, d, a

This instruction adds 1 to the byte operand in fileReg. The result can be placed in WREG (if d = 0) or fileReg (if d = 1).

Example:

```
        MyReg SET 0x40      ;set aside loc 40H for MyReg
        MOVLW 0x99          ;WREG = 99H
        MOVWF MyReg
        INCF MyReg,F        ;MyReg = 9AH, WREG 99H
        INCF MyReg,F        ;MyReg = 9BH, WREG 99H
        DECF MyReg,F            ;MyReg = 9CH, WREG 99H
```

Example: Toggle PORTB 5 times.

```
        COUNTER  SET     0x40   ;loc 40H for COUNTER
        SETF TRISB              ;PORTB as output
        MOVLW D'251'            ;WREG = 251
        MOVWF COUNTER   ;COUNTER = 251
        MOVLW 0x55              ;WREG = 55H
        MOVWF PORTB
BACK COMF PORTB,F               ;toggle PORTB
        INCF COUNTER,F  ;INC COUNTER
        BNC   BACK              ;toggle until counter becomes 0
```

We can place the result in fileReg as shown in the examples below:

```
        MyReg SET 0x40      ;set aside loc for MyReg
```

```
        MOVLW 0x99          ;WREG = 99H
        MOVWF MyReg                 ;MyReg = 99H
        INCF MyReg          ;WREG = 9AH, MyReg = 99H
        INCF MyReg          ;WREG = 9BH, MyReg = 99H
```

Example:

```
        MyReg  SET   0x40   ;set MyReg loc at 0x40
        MOVLW        0x5    ;W = 05H
        MOVWF        MyReg  ;MyReg = 05H
        INCF         MyReg  ;WREG = 06H and MyReg = 05H
```

INCFSZ **Increment fileReg and skip if zero**

Function: Increment
Syntax: INCFSZ f, d, a

This instruction adds 1 to fileReg and if the result is zero it skips the next instruction.

Example: Toggle PORTB 156 times.

```
        COUNTER   SET       0x40    ;loc 40H for COUNTER
        SETF TRIS           ;PORTB as output
        MOVLW D'156'                ;WREG = 156
        MOVWF COUNTER       ;COUNTER = 156
        MOVLW 0x55          ;WREG = 55H
        MOVWF PORTB
BACK COMF PORTB,F                   ;toggle PORTB
        INCFSZ COUNTER,F    ;INC COUNTER and skip if 0
        BRA    BACK         ;toggle until counter becomes 0
        .....
```

INCFSNZ **Increment fileReg and skip if not zero**

Function: Increment
Syntax: INFSNZ f, d, a

This instruction adds 1 to the register or memory location specified by the operand. If the result is not zero, it skips the next instruction.

Example: Toggle PORTB 156 times continuously.

```
        COUNTER   SET       0x40    ;loc 40H for COUNTER
        SETF TRISB          ;PORTB as output
OVER    MOVLW D'156'                ;WREG = 156
        MOVWF COUNTER       ;COUNTER = 156
        MOVLW 0x55          ;WREG = 55H
        MOVWF PORTB
BACK COMF PORTB,F                   ;toggle PORTB
        INCFSNZ COUNTER,F           ;INC COUNTER, skip if not 0
        BRA    OVER         ;start over
BRA  BACK                   ;toggle until counter becomes 0
```

IORLW **OR K value with WREG**

Function: Logical-OR WREG with value k
Syntax: IORLW k

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

This performs a logical OR on the WREG register and k value, bit by bit, and stores the result in WREG.

Example:

```
MOVLW 0x30  ;W = 30H
IORLW 0x 09  ;now W = 39H

 39H   0011 0000
 09H   0000 1001
 39    0011 1001
```

Example:

```
MOVLW 0x32          ;W = 32H
IORLW 0x50    ;(W = 72H)

32H   0011 0010
50H   0101 0000
72H   0111 0010
```

IORWF **OR FileReg with WREG**

Function: Logical-OR fileReg and WREG
Syntax: IORWF f, d, a

This performs a logical OR on the fileReg value and the WREG register, bit by bit, and places the result in WREG (if d = 0) or fileReg (if d = 1).

Example:

```
MyReg      SET 0x40    ;set MyReg loc at 0x40
MOVLW   0x39                ;WREG = 39H
MOVWF  MyReg        ;MyReg = 39H
MOVLW  0x07
IORWF     MyReg        ;39H ORed with 07 (W = 3F)

39   0011 1001
07   0000 0111
3F   0011 1111
```

Example:

```
MyReg      SET 0x40    ;set MyReg loc at 0x40
 MOVLW  0x5          ;WREG = 05H
MOVWF  MyReg        ;MyReg = 05H
```

```
MOVLW  0x30
IORWF  MyReg                ;30H ORed with 05 (W = 35H)

05   0000 0101
30   0011 0000
35   0011 0101
```

We can place the result in fileReg as shown in the examples below:

```
MOVLW 0x30  ;W = 30H

IORWF PORTB,F       ;W and PORTB are ORed and result
                    ;goes to PORTB
```

Example:

```
MyReg   SET    0x20
MOVLW 0x54          ;WREG = 54H
MOVWF MyReg
MOVLW 0x67          ;WREG = 67H
IORWF MyReg,F       ;OR WREG and MyReg
                    ;after the operation MyReg = 77H

44H  0101 0100
67H  0110 0111
77H  0111 0111Therefore MyReg will have 77H, WREG = 54H.
```

LFSR **Load FSR**

Function: Load into FSR registers a 12-bit value of k
Syntax: LFSR f,k ;k is between 000 and FFFH

This loads a 12-bit value into one of the FSR registers of FSR0, FSR1, or FSR2.

```
LFSR   0 , 0x200        ;FSR0 = 200H
LFSR   1 , 0x050        ;FSR1 = 050H
LFSR   2 , 0x160        ;FSR2 = 160H
```

This is widely used in register indirect addressing mode. See Chapter 6.

MOVF (or MOVFW) **Move fileReg to WREG**

Function: Copy byte from fileReg to WREG
Syntax MOVF f, d, a:

This instruction is widely used for moving data from a fileReg to WREG. Look at the following examples:

```
CLRF   TRISC          ;PORTC output
SETF   TRISB          ;PORTB as input
MOVFW       PORTB;copy PORTB to WREG
ANDLW       0x0F            ;mask the upper 4 bits
MOVWF       PORTC;put it in PORTC
```

Example:

```
CLRF   TRISD        ;PORTD as output
SETF   TRISB        ;PORTB as input
MOVFW       PORTB;copy PORTB to WREG
IORW        0x30    ;OR it with 30H
MOVWF       PORTD       ;put it in PORTD
```

This instruction can be used to copy the fileReg to itself in order to get the status of the N and Z flags. Look at the following example.

Example:

```
MyReg    SET 0x20    ;set aside loc 0x20 to MyReg
MOVLW   0x54             ;W = 54H
MOVWF   MyReg       ;MyReg = 54H
MOVFW   MyReg,F     ;My Reg = 54, also N = 0 and Z = 0
```

MOVFF **Move FileReg to Filereg**

Function: Copy byte from one fileReg to another fileReg
Syntax: MOVFF fs, fd

This copies a byte from the source location to the destination. The source and destination locations can be any of the file register locations, SFRs, or ports.

```
MOVFF       PORTB,MyReg
MOVFF       PORTC,PORTD
MOVFF       RCREG,PORTC
MOVFF       Reg1,REG2
```

Notice that this a 4-byte instruction because the source and destination address each take 12 bits of the instruction. That means the 24 bits of the instruction are used for the source and destination addresses. The 12-bit address allows data to be moved from any source location to any destination location within the 4K RAM space of the PIC18.

MOVLB **Move Literal 4-bit value to lower 4-bit of the BSR**

Function: Move 4-bit value k to lower 4 bits of the BSR registers
Syntax: MOVLB k ;k is between 0 and 15 (0–F in hex)

We use this instruction to select a register bank other than the access bank. With this instruction we can load into the BSR (bank selector register) a 4-bit value representing one of 16 banks supported by the PIC18. That means the values between 0000 and 1111 (0–F in hex). For examples of the MOVLB instruction, see Chapter 6 and Section A.1 in this chapter.

MOVLW K **Move Literal to WREG**

Function: Move 8-bit value k to WREG
Syntax: MOVLW k ;k is between 0 and 255 (0–FF in hex)

Example:

```
MOVLW       0x55            ;WREG = 55H
```

```
MOVLW     0x0          ;clear WREG (WREG = 0)
MOVLW     0xC2         ;WREG = C2H
MOVLW     0x7F         ;WREG = 7FH
```

This instruction, along with the MOVWF, is widely used to load fixed values into any port, SFR, or fileReg location. See the next instruction to see how it is used.

MOVWF **Move WREG to a fileReg**

Function: Copy the WREG contents to a fileReg
Syntax: MOVWF f, a

This copies a byte from WREG to fileReg. This instruction is widely used along with the MOVLW instruction to load any of the fileReg locations, SFRs, or PORTs with a fixed value. See the following examples:

Example: Toggle PORTB.

```
MOVLW     0x55         ;WREG = 55H
MOVWF     PORTB
MOVLW     0xAA         ;WREG = AAH
MOVWF     PORTB
BRA       OVER         ;keep toggling the PORTB
```

Example: Load RAM location 20H with value 50H.

```
MyReg SET 0x20         ;set aside the loc 0x20 for MyReg
MOVLW     0x50
MOVWF     MyReg        ;MyReg = 50H (loc 20H has 50H)
```

Example: Initialize the Timer0 low and high registers.

```
MOVLW     0x05         ;WREG = 05H
MOVWF     TMR0H        ;TMR0H = 0x5
MOVLW     0x30         ;WREG = 30H
MOVWF     TMR0L        ;TMR0L = 0x30
```

MULLW **Multiply Literal with WREG**

Function: Multiply k x WREG
Syntax: MULLW k

This multiplies an unsigned byte k by an unsigned byte in register WREG and the 16-bit result is placed in registers PRODH and PRODL, where PRODL has the lower byte and PRODH has the higher byte.

Example:

```
MOVLW 0x5          ;WREG = 5H
MULLW 0x07         ;PRODL = 35 = 23H, PRODH = 00
```

Example:

```
MOVLW 0x0A      ;WREG = 10
MULLW 0x0F      ;PRODL = 10 x 15 = 150 = 96H
                ;PRODH = 00
```

Example:

```
MOVLW 0x25
MULLW 0x78      ;PRODL = 58H, PRODH = 11H
                ;because 25H x 78H = 1158H
```

Example:

```
MOVLW D'100'    ;WREG = 100
MULLW D'200'    ;PRODL = 20H, PRODH = 4EH
                ;(100 x 200 = 20,000 = 4E20H)
```

MULWF **Multiply WREG with fileReg**

Function: Multiply WREG x fileReg and place the result in PRODH:PROFDL registers

Syntax: MULWF f, a

This multiplies an unsigned byte in WREG by an unsigned byte in the fileReg register and the result is placed in PRODL and PRODH, where PRODL has the lower byte and PRODH has the higher byte.

Example:

```
MyReg SET   0x20    ;MyReg has location of 0x20
MOVLW 0x5
MOVWF MyReg         ;MyReg has 0x5
MOVLW 0x7           ;WREG = 0x7
MULWF MyReg         ;PRODL = 35 = 23H, PRODH = 00
```

Example:

```
MOVLW 0x0A
MOVWF MyReg         ;MyReg = 10
MOVLW 0x0F          ;WREG = 15
MULFW MyReg         ;PRODL = 150 = 96H, PRODH = 00
```

Example:

```
MOVLW 0x25
MOVWF MyReg         ;MyReg = 0x25
MOVLW 0x78          ;WREG 78H
MULWF Myreg         ;PRODL = 58H, PRODH = 11H
                    ;(25H x 78H = 1158H)
```

Example:

```
MOVLW D'100'        ;WREG = 100
MOVWF MyReg         ;MyReg = 100
MOVLW D'200'        ;WREG = 200
MULWF MyReg         ;PRODL = 20H, PRODH = 4EH
        ;(100 x 200 = 20,000 = 4E20H)
```

NEGF **Negate fileReg**

Function: No operation

Syntax: NEGF f, a

This performs 2's complement on the value stored in fileReg and places it back in fileReg.

Example:

```
MyReg SET 0x30
MOVLW 0x98          ;WREG = 0x98
MOVWF MyReg               ;MyReg = 0x98
NEGF                ;2's complement fileReg

98H     10011000
        01100111          1's complement
    +         1
        01101000          Now FileReg = 68H
```

Example:

```
MyReg SET 0x10
MOVLW 0x75          ;WREG = 0x75
MOVWF MyReg               ;MyReg = 0x75
NEGF                ;2's complement fileReg

75H     01110101
        10001010          1's complement
    +         1
        10001011          Now FileReg = 7AH
```

Notice that in this instruction we cannot place the result in the WREG register.

NOP **No Operation**

Function: No operation
Syntax: NOP

This performs no operation and execution continues with the next instruction. It is sometimes used for timing delays to waste clock cyles. This instruction only updates the PC (program counter) to point to the next instruction following NOP. In PIC18, this a 2-byte instruction.

POP **POP Top of Stack**

Function: Pop from the stack
Syntax: POP

This takes out the top of stack (TOS) pointed to by SP (stack pointer) and discards it. It also decrements SP by 1. After the operation, the top of the stack will be the value pushed onto the stack previously.

PUSH **PUSH Top of the Stack**

Function: Push the PC onto the stack
Syntax: PUSH

This copies the program counter (PC) onto the stack and increments SP by 1, which means the previous top of the stack is pushed down.

RCALL **Relative Call**

Function: Transfers control to a subroutine within 1K space
Syntax: RCALL target_address

There are two types of CALLs: RCALL and CALL. In RCALL, the target address is within 1K of the current PC (program counter). To reach the target address in the 2M ROM space of the PIC18, we must use CALL. In calling a subroutine, the PC register (which has the address of the instruction after the RCALL) is pushed onto the stack and the stack pointer (SP) is incremented by 1. Then the program counter is loaded with the new address and control is transferred to the subroutine. At the end of the procedure, when RETURN is executed, PC is popped off the stack, which returns control to the instruction after the RCALL.

Notice that RCALL is a 2-byte instruction, in which 5 bits are used for the opcode and the remaining 11 bits are used for the target subroutine address. An 11-bit address limits the range to –1024 to +1023. See the CALL instruction for discussion of the target address being anywhere in the 2M ROM space of the PIC18. Notice that RCALL is a 2-byte instruction while CALL is a 4-byte instruction. Also notice that the RCALL does not have the option of context saving, as CALL has.

RESET **Reset (by software)**

Function: Reset by software
Syntax: RESET

This instruction is used to reset the PIC18 by way of software. After execution of this instruction, all the registers and flags are forced to their reset condition. The reset condition is created by activating the hardware pin MCLR. In other words, the RESET instruction is the software version of the MCLR pin.

RETFIE **Return from Interrupt Exit**

Function: Return from interrupt
Syntax: RETFIE s

This is used at the end of an interrupt service routine (interrupt handler). The top of the stack is popped into the program counter and program execution continues at this new address. After popping the top of the stack into the program counter (PC), the stack pointer (SP) is decremented by 1.

Notice that while the RETURN instruction is used at the end of a subroutine associated with the CALL and RCALL instructions, RETFIE must be used for the interrupt service routines (ISRs).

RETLW **Return with Literal in WREG**

Function: The k value is placed in WREG and the top of the stack is the placed in PC (program counter)
Syntax: RETLW k

After execution of this instruction, the k value is loaded into WREG and the top of the stack is popped into the program counter (PC). After popping the top of the stack into the program counter, the stack pointer (SP) is decremented by 1. This instruction is used for the implementation of a look-up table. See Section 6.3 in Chapter 6.

RETURN **Return**

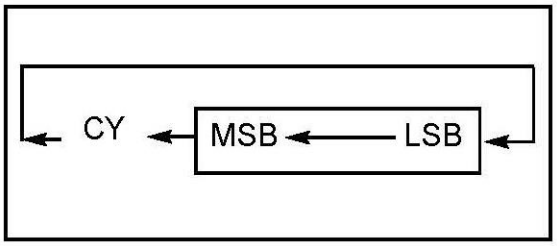
Function: Return from subroutine
Syntax: RETURN s ;where s = 0 or s = 1

This instruction is used to return from a subroutine previously entered by instructions CALL or RCALL. The top of the stack is popped into the program counter (PC) and program execution continues at this new address. After popping the top of the stack into the program counter, the stack pointer (SP) is decremented by 1. For the case of "RETURN s" where s = 1, the RETURN will also restore the context registers. See the CALL instruction for the case of s = 1. Notice that "RETURN 1" cannot be used for subroutines associated with RCALL.

RLCF **Rotate Left Through Carry the fileReg**

Function: Rotate fileReg left through carry
0Syntax: RLCF f, d, a

This rotates the bits of a fileReg register left. The bits rotated out of fileReg are rotated into C, and the C bit is rotated into the opposite end of the fileReg register.



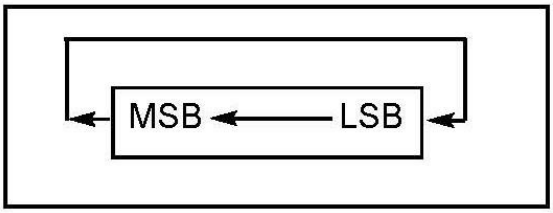
Example:

```
        MyReg SET    0x30    ;set aside loc 30H for MyReg
BCF     STATUS,C             ;C = 0
        MOVLW 0x99           ;WREG = 99H
        MOVWF MyReg          ;MyReg = 99H = 10011001
        RLCF MyReg,F         ;now MyReg = 00110010 and
                             ;C = 1
              RLCF MyReg,F;now MyReg = 01100101 and
                             ;C = 0
```

RLNCF **Rotate left not through Carry**

Function: Rotate left the fileReg
Syntax: RLNCF f, d, a

This rotates the bits of a fileReg register left. The bits rotated out of fileReg are rotated back into fileReg at the opposite end.



Example:

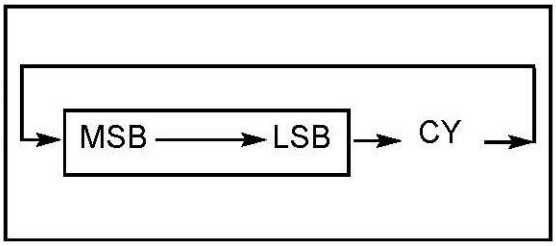
```
        MyReg    SET 0x20    ;set aside loc 20 for MyReg
        MOVLW    0x69                ;WREG = 01101001
        MOVWF    MyReg       ;MyReg = 69H = 01101001
        RLNCF     MyReg,F    ;now MyReg = 11010010
        RLNCF     MyReg,F    ;now MyReg = 10100101
        RLNCF     MyReg,F    ;now MyReg = 01001011
        RLNCF     MyReg,F    ;now MyReg = 10010110
```

Notice that after four rotations, the upper and lower nibbles are swapped.

RRCF **Rotate Right through Carry**

Function: Rotate fileReg right through carry
Syntax: RRCF f, d, a

This rotates the bits of a fileReg register right. The bits rotated out of the register are rotated into C, and the C bit is rotated into the opposite end of the register.



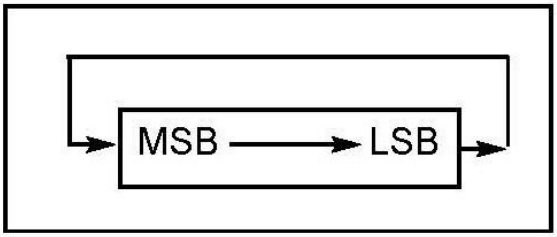
Example:

```
MyReg SET   0x20          ;set aside loc 20 for MyReg
      BSF   STATUS,C      ;C = 1
      MOVLW 0x99          ;WREG = 10011001
      MOVWF MyReg         ;MyReg = 99H = 10011001
      RRCF  MyReg,F       ;now MyReg = 11001100, C = 1
      RRCF  MyReg,F       ;now MyReg = 11100110, C = 0
```

RRNCF **Rotate Right not through Carry**

Function: Rotate fileReg right
Syntax: RRNCF f, d, a

This rotates the bits of a fileReg register right. The bits rotated out of the register are rotated back into fileReg at the opposite end.



Example:

```
      MyReg   SET 0x20    ;set aside loc 20H for MyReg
      MOVLW   0x66        ;WREG = 66H = 01100110
      MOVWF   MyReg       ;MyReg = 66H = 01100110
      RRNCF   MyReg,F     ;now MyReg = 00110011
      RRNCF   MyReg,F     ;now MyReg = 10011001
      RRNCF   MyReg,F     ;now MyReg = 11001100
      RRNCF   MyReg,F     ;now MyReg = 01100110
```

Example: We can use this instruction to swap the upper and lower nibbles.

```
      MyReg   SET0x20     ;set aside loc 20H for MyReg
      MOVLW   0x36        ;WREG = 36H = 00110110
      MOVWF   MyReg       ;MyReg = 36H = 00110110
      RRNCF    MyReg,F    ;now MyReg = 00011011
      RRNCF    MyReg,F    ;now MyReg = 10001101
      RRNCF    MyReg,F    ;now MyReg = 11000110
      RRNCF    MyReg,F    ;now MyReg = 01100011 = 63H
```

SETF **Set fileReg**

Function: Set
Syntax: SETF f, a

This instruction sets the entire byte in fileReg to HIGH. All bits of the register are set to 1.

Examples:

```
SETF    MyReg       ;MyReg = 11111111
SETF   TRISB        ;TRISB = FFH,(makes PORTB input)
SETF   PORTC        ;PORTC = 1111 1111
```

Notice that in this instruction, the result can be placed in fileReg only and there is no option for WREG to be used as the destination for the result.

SLEEP **Enter Sleep mode**

Function: Put the CPU into sleep mode
Syntax: SLEEP

This instruction stops the oscillator and puts the CPU into sleep mode. It also resets the Watchdog Timer (WDT). The WDT is used mainly with the SLEEP instruction. Upon execution of the SLEEP instruction, the entire microcontroller goes into sleep mode by shutting down the main oscillator and by stopping the Program Counter from fetching the next instruction after SLEEP. There are two ways to get out of sleep mode: (a) an external event via hardware interrupt, (b) the internal WDT interrupt. Upon wake-up from a WDT interrupt, the microcontroller resumes operation by executing the next instruction after SLEEP.

Check the Microchip Corp. website for application notes on WDT.

SUBFWB **Subtract fileReg from WREG with borrow**

Function: WREG – fileReg – #borrow ;#borrow is inverted carry
Syntax: SUBFWB f, d, a

This subtracts fileReg and the Carry (borrow) flag from WREG and puts the result in WREG (d = 0) or fileReg (d = 1). The steps for subtraction performed by the internal hardware of the CPU are as follows:

1. Take the 2's complement of the fileReg byte.
2. Add this to register WREG.
3. Add the inverted Carry (borrow) flag to the result.
4. Ignore the Carry.
5. Examine the N (negative) flag for positive or negative result.

Example:

```
MyReg   SET 0x20      ;set aside loc 0x20 for MyReg
BSF        STATUS,C  ;make Carry = 1
MOVLW   0x45                ;WREG 45H
MOVWF   MyReg       ;MYReg = 45H
MOVLW   0x23
SUBWF    MyReg      ;WREG = 45H - 23H - 0 = 22H
 45H    0100 0101                    0100 0101
```

```
-23H    0010 0011       2's comp        + 1101 1101
                    Inverted carry +              0
-------                                 ----------------
+22H                                     0010 0010
```

Because D7 (the N flag) is 0, the result is positive.

This instruction sets the negative flag according to the following:

	N	
WREG > (fileReg + #C)	0	the result is positive
WREG = (fileReg + #C)	0	the result is 0
WREG < (fileReg + #C)	1	the result is negative and in 2's comp

SUBLW **Subtract WREG from Literal value**

Function: Subtract WREG from literal value k (WREG = k – WREG)
Syntax: SUBLW k

This subtracts the WREG value from the literal value k and puts the result in WREG. The steps for subtraction performed by the internal hardware of the CPU are as follows:

1. Take the 2's complement of the WREG value.
2. Add it to literal value k.
3. Ignore the Carry.
4. Examine the N (negative) flag for positive or negative result.

```
MOVLW 0x23  ;WREG 23H
SUBLW 0x45  ;WREG = 45H - 23H = 22H

 45H    0100 0101                       0100 0101
-23H    0010 0011       2's comp        +1101 1101
-------                                 ----------------
+22H                                     0010 0010
```

Because D7 (the N flag) is 0, the result is positive.

This instruction sets the negative flag according to the following:

	N	
Literal value k > WREG	0	the result is positive
Literal value k = WREG	0	the result is 0
Literal value < WREG	1	the result is negative and in 2's comp

Example:

```
MOVLW 0x98  ;WREG 98H
SUBLW 0x66  ;WREG = 66H - 98H = CEH

 66H            0110 0110               0110 0110
-98H    1001 1000       2's comp        +0110 1000
 ------                                 ----------------
 CEH                                     1100 1110
```

Because D7 (the N flag) is 1, the result is negative and in 2's comp.

SUBWF **Subtract WREG from fileReg**

Function: Subtract WREG from fileReg (Dest = fileReg – WREG)
Syntax: SUBWF f, d, a

This subtracts the WREG value from the fileReg value and puts the result in either WREG (d = 0) or fileReg (d = 1). The steps for subtraction performed by the internal hardware of the CPU are as follows:

1. Take the 2's complement of the WREG byte.
2. Add this to the fileReg register.
3. Ignore the carry.
4. Examine the N (negative) flag for positive or negative result.

Example:

```
MyReg     SET 0x20     ;set aside loc 0x20 for MyReg
MOVLW   0x45                  ;WREG 45H
MOVWF   MyReg        ;MYReg = 45H
MOVLW   0x23                  ;WREG = 23H
SUBWF    MyReg,F     ;MyReg = 45H - 23H = 22H

 45H              0100 0101                0100 0101
-23H    0010 0011      2's comp          +1101 1101
-------                                   ----------
+22H                                      0010 0010
```

Because D7 (the N flag) is 0, the result is positive.

This instruction sets the negative flag according to the following:

	N	
fileReg > WREG	0	the result is positive
fileReg = WREG	0	the result is 0
fileReg < WREG	1	the result is negative and in 2's comp

SUBWFB **Subtract WREG from fileReg with borrow**

Function: Dest = fileReg – WREG – #borrow ;#borrow is inverted carry
Syntax: SUBWFB f, d, a

This subtracts the WREG value and the inverted borrow (carry) flag from the fileReg value and puts the result in WREG (if d = 0), or fileReg (if d = 1). The steps for subtraction performed by the internal hardware of the CPU are as follows:

1. Take the 2's complement of WREG.
2. Add this to fileReg.
3. Add the inverted Carry flag to the result.
4. Ignore the carry.
5. Examine the N (negative) flag for positive or negative result.

Example:

```
MyReg    SET 0x20    ;set aside loc 0x20 for MyReg
BSF      STATUS,C    ;C = 1
MOVLW    0x45        ;WREG 45H
MOVWF    MyReg       ;MYReg = 45H
MOVLW    0x23        ;WREG = 23H
SUBWFB   MyReg,F     ;MyReg = 45H - 23H - 0 = 22H

 45H    0100 0101                    0100 0101
-23H    0010 0011    2's comp        +1101 1101
                  Inverted carry     +         0
-----                                ----------
+22H                                 0010 0010
```

Because D7 (the N flag) is 0, the result is positive.

This instruction sets the negative flag according to the following:

	N	
fileReg > (WREG + #C)	0	the result is positive
fileReg = (WREG + #C)	0	the result is 0
fileReg < (WREG + #C)	1	the result is negative and in 2's comp

SWAPF **Swap Nibbles in fileReg**

Function: Swap nibbles within fileReg
Syntax: SAWPF f, d, a

The SWAPF instruction interchanges the lower nibble (D0–D3) with the upper nibble (D4–D7) inside fileReg. The result is placed in WREG (d = 0) or fileReg (d = 1).

Example:

```
MyReg    SET 0X20    ;set aside loc 20H for MyReg
MOVLW    0x59H       ;W = 59H (0101 1001 in binary)
MOVWF    MyReg       ;MyReg = 59H (0101 1001)
SWAPF    MyReg,F     ;MyReg = 95H (1001 0101)
```

TBLRD **Table Read**

Function: Read a byte from ROM to the TABLAT register
Syntax: TBLRD *
TBLRD *+
TBLRD *-
TBLRD +*

This instruction moves (copies) a byte of data located in program (code) ROM into the TableLatch (TABLAT) register. This allows us to put strings of data, such as look-up table elements, in the code space and read them into the CPU. The address of the desired byte in the program space (on-chip ROM) is held by the TBLPTR register. Table A-6 shows the auto-increment feature of the TBLRD instruction.

Instruction	Function
TBLRD*	Table Read After read, TBLPTR stays the same
TBLRD*+	Table Read with post-increment (Read and increment TBLPTR)
TBLRD*-	Table Read with post-decrement (Read and decrement TBLPTR)
TBLRD+*	Table Read with pre-increment (increment TBLPTR and read)
Note: A byte of data is read into the TABLAT register from code space pointed to by TBLPTR.	

Table A-6: PIC18 Table Read Instructions

Example: Assume that an ASCII character string is stored in the on-chip ROM program memory starting at address 500H. Write a program to bring each character into the CPU and send it to PORTB.

```
        ORG  0000H              ;burn into ROM starting at 0
        MOVLW LOW(MESSAGE)      ;WREG = 00 low-byte addr.
        MOVWF TBLPTRL           ;look-up table low-byte addr
        MOVLW HIGH(MESSAGE)     ;WREG = 05 = high-byte addr
        MOVWF TBLPTRH           ;look-up table high-byte addr
        CLRF    TBLPTRU         ;clear upper 5 bits

B8      TBLRD*+                 ;read the table,then increment TBLPTR
        MOVF  TABLAT,W          ;copy to WREG (Z = 1 if null)
        BZ      EXIT            ;exit if end of string
        MOVWF PORTB             ;copy WREG to PORTB
        BRA     B8
EXIT    GOTO  EXIT
;--------------------message
        ORG 0x500               ;data burned starting at 0x500
                ORG     0x500
MESSAGE         DB      "The earth is but one country and "
                DB      "mankind its citizens","Baha'u'llah",0
                END
```

In the program above, the TBLPTR holds the address of the desired byte. After the execution of the TBLRD*+ instruction, register TABLAT has the character. Notice that TBLPTR is incremented automatically to point to the next character in the MRESSAGE table.

TBLWT **Table Write**

Function: Write to Flash a block of data
Syntax: TBLWT*
TBLWT*+
TBLWT*-
TBLWT+*

This instruction writes a block of data to the program (code) space assuming that the on-chip program ROM is of Flash type. The address of the desired location in Flash ROM is held by the TBLPTR register. The process of writing to Flash ROM using the TBLWT instruction is discussed in Section 14.3 of Chapter 14.

TSTFSZ **Test fileReg, Skip if Zero**

Function: Test fileReg for zero value and skip if it is zero
Syntax: TSTFSZ f, a

This instruction tests the entire contents of fileReg for value zero and skips the next instruction if fileReg has zero in it.

Example: Test PORTB for zero continuously.

```
        SETF  TRISB  ;make PORTB an input
        CLRF  TRISD  ;make PORTD an output
BACK    TSTFSZ PORTB
        BRA    BACK
        MOVFF  PORTB,PORTD
```

Example: Toggle PORTB 250 times.

```
        COUNTER  SET      0x40    ;loc 40H for COUNTER
        SETF     TRISB    ;PORTB as output
        MOVLW    D'250'   ;WREG = 250
        MOVWF    COUNTER  ;COUNTER = 250
        MOVLW    0x55     ;WREG = 55H
        MOVWF    PORTB
BACK    COMF     PORTB,F  ;toggle PORTB
        DECF     COUNTER,F      ;decrement COUNTER
        TSTFSZ   COUNTER  ;test counter for 0
        BRA      BACK     ;keep doing it
        ......
```

XORLW **Ex-Or Literal with WREG**

Function: Logical exclusive-OR Literal k and WREG
Syntax: XORLW k

This performs a logical exclusive-OR on the Literal value and WREG operands, bit by bit, storing the result in WREG.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Example:

```
MOVLW 0x39    ;WREG = 39H
XORLW 0x09    ;WREG = 39H ORed with 09
                ;now, WREG = 30H
39H   0011 1001
09H   0000 1001
30    0011 0000
```

Example:

```
MOVLW 0x32          ;WREG = 32H
XORLW 0x50  ;(now, WREG = 62H)

32H   0011 0010
50H   0101 0000
62H   0110 0010
```

XORWF **Ex-Or WREG with fileReg**

Function: Logical exclusive-OR fileReg and WREG
Syntax: XORWF f, d, a

This performs a logical exclusive-OR on the operands, bit by bit, storing the result in the destination. The destination can be WREG (d = 0), or fileReg (d = 1).

Example:

```
MyReg       SET         0x20     ;set aside loc 20h for MyReg
MOVLW     0x39          ;WREG = 39H
MOVWF    MyReg          ;MyReg = 39H
MOVLW     0x09                   ;WREG = 09H
XORWF      MyReg,F      ;MyReg = 39H ORed with 09
                        ;MyReg = 30H
39H   0011 1001
09H   0000 1001
30    0011 0000
```

Example:

```
MyReg       SET         0x15     ;set aside loc 15 for MyReg
 MOVLW     0x32          ;WREG = 32H
MOVWF     MyReg          ;MyReg = 32H
MOVLW     0x50                   ;WREG = 50H
XORWF      MyReg,F       ;now W = 62H

32H   0011 0010
50H   0101 0000
62H   0110 0010.
```

We can place the result in WREG.

Example:

```
MyReg      SET0x15     ;set aside loc 15 for MyReg
MOVLW     0x44         ;WREG = 44H
MOVWF     MyReg        ;MyReg = 44H
MOVLW     0x67                  ;WREG = 67H
XORWF     MyReg        ;now W = 23H, and MyReg = 44H

44H  0100 0100
67H  0110 0111
23H  0010 0011
```