Book: PIC Microcontrollers

Chapter 9: Instruction Set

It has been already mentioned that microcontrollers differs from other integrated circuits. Most of them are ready for installation into the target device just as they are, this is not the case with the microcontrollers. In order that the microcontroller may operate, it needs precise instructions on what to do. In other words, a program that the microcontroller should execute must be written and loaded into the microcontroller. This chapter covers the commands which the microcontroller "understands". The instruction set for the 16FXX includes 35 instructions in total. Such a small number of instructions is specific to the RISC microcontroller because they are well-optimized from the aspect of operating speed, simplicity in architecture and code compactness. The only disadvantage of RISC architecture is that the programmer is expected to cope with these instructions.

INSTRUCTION	DESCRIPTION	OPERATION	FLAG	CLK	*
Data Transfer Instructions					
MOVLW k	Move constant to W	k -> w		1	
MOVWF f	Move W to f	W -> f		1	
MOVF f,d	Move f to d	f -> d	Z	1	1, 2
CLRW	Clear W	0 -> W	Z	1	
CLRF f	Clear f	0 -> f	Z	1	2
SWAPF f,d	Swap nibbles in f	f(7:4),(3:0) -> f(3:0),(7:4)		1	1, 2
Arithmetic-logic Instructions					
ADDLW k	Add W and constant	W+k -> W	C, DC, Z	1	
ADDWF f,d	Add W and f	W+f -> d	C, DC ,Z	1	1, 2
SUBLW k	Subtract W from constant	k-W -> W	C, DC, Z	1	
SUBWF f,d	Subtract W from f	f-W -> d	C, DC, Z	1	1, 2
ANDLW k	Logical AND with W with constant	W AND k -> W	Z	1	
ANDWF f,d	Logical AND with W with f	W AND f -> d	Z	1	1, 2
ANDWF f,d	Logical AND with W with f	W AND f -> d	Z	1	1, 2
IORLW k	Logical OR with W with constant	W OR k -> W	Z	1	
IORWF f,d	Logical OR with W with f	W OR f -> d	Z	1	1, 2
XORLW k	Logical exclusive OR with W with constant	W XOR k -> W	Z	1	1, 2
XORWF f,d	Logical exclusive OR with W with f	W XOR f -> d	Z	1	
INCF f,d	Increment f by 1	f+1 -> f	Z	1	1, 2
DECF f,d	Decrement f by 1	f-1 -> f	Z	1	1, 2
RLF f,d	Rotate left f through CARRY bit		C	1	1, 2
RRF f,d	Rotate right f through CARRY bit		C	1	1, 2
COMF f,d	Complement f	f -> d	Z	1	1, 2
Bit-oriented Instructions					
BCF f,b	Clear bit b in f	0 -> f(b)		1	1,2
BSF f,b	Set bit b in f	1 -> f(b)		1	1,2
Program Control Instructions					
BTFSC f,b	Test bit b of f. Skip the following instruction if clear.	Skip if f(b) = 0		1 (2)	3
BTFSS f,b	Test bit b of f. Skip the following instruction if set.	Skip if f(b) = 1		1 (2)	3
DECFSZ f,d	Decrement f. Skip the following instruction if clear.	f-1 -> d skip if Z = 1		1 (2)	1, 2, 3
INCFSZ f,d	Increment f. Skip the following instruction if set.	f+1 -> d skip if Z = 0		1 (2)	1, 2, 3
GOTO k	Go to address	k -> PC		2	
CALL k	Call subroutine	PC -> TOS, k -> PC		2	
RETURN	Return from subroutine	TOS -> PC		2	
RETLW k	Return with constant in W	k -> W, TOS -> PC		2	
RETFIE	Return from interrupt	TOS -> PC, 1 -> GIE		2	
Other instructions					

NOP	No operation	TOS -> PC, 1 -> GIE		1	
CLRWDT	Clear watchdog timer	0 -> WDT, 1 -> TO, 1 -> PD	TO, PD	1	
SLEEP	Go into sleep mode	0 -> WDT, 1 -> TO, 0 -> PD	TO, PD	1	

Table 9-1 16Fxx Instruction Set

*1 When an I/O register is modified as a function of itself, the value used will be that value present on the pins themselves.
*2 If the instruction is executed on the TMR register and if d=1, the prescaler will be cleared.
*3 If the PC is modified or test result is logic one (1), the instruction requires two cycles.

Data Transfer Instructions

Data Transfer within the microcontroller takes place between working register W (called accumulator) and a register which represents any location of internal RAM regardless of whether it is about special function or general purpose registers.

First three instructions move literal to W register (MOVLW stands for **move Literal to W**), move data from W register to RAM and from RAM to W register (or to the same RAM location with change on flag Z only). Instruction CLRF clears f register, whereas CLRW clears W register. SWAPF instruction swaps nibbles within f register (one nibble contains four bits).

Arithmetic-logic Instructions

Similar to most microcontrollers, PIC supports only two arithmetic instructions- addition and subtraction. Flags C, DC, Z are automatically set depending on the results of addition or subtraction. The only exception is the flag C. Since subtraction is performed as addition with negative value, the flag C is inverted after subtraction. It means that the flag C is set if it is possible to perform operation and cleared if the larger number is subtracted from smaller one. Logic one (1) of the PIC is able to perform operations AND, OR, EX-OR, inverting (COMF) and rotation (RLF and RRF).

Instructions which rotate a register actually rotate its bits through the flag C by one bit left (toward bit 7) or right (toward bit 0). The bit shifted from the register is moved to the flag C which is automatically moved to the bit on the opposite side of the register.

Bit-oriented Instructions

Instructions BCF and BSF clear or set any bit in memory. Although it seems to be a simple operation, it is not like that. CPU first reads the entire byte, changes one its bit and rewrites the whole byte to the same location.

Program Control Instructions

The PIC16F887 executes instructions GOTO, CALL, RETURN in the same way as all other microcontrollers do. A difference is that stack is independent from internal RAM and has 8 levels. The ‘RETLW k’ instruction is identical to RETURN instruction, with exception that a constant defined by instruction operand is written to the W register prior to return from subroutine. This instruction enables **Lookup** tables to be easily created by creating a table as a subroutine consisting of ‘RETLWk‘ instructions, where the literals ‘k’ belong to the table. The next step is to write the position of the literals k (0, 1, 2, 3...n) to W register and call the subroutine (table) using the CALL instruction. Table below consists of the following literals: k0, k1, k2...kn.

```
Main   movlw 2     ;write number 2 to accumulator
call   Lookup      ;jump to the lookup table
Lookup addwf PCL,f ;add accumulator and program cur
                   ;rent address (PCL)
retlw  k0          ;return from subroutine (accumulator contains k0)
retlw  k1          ;...
retlw  k2          ;...
...                ;...
...                ;...
retlw  kn          ;return from subroutine (accumulator contains kn)
```

The first line of the subroutine (instruction `ADDWF PCL,f`)simply adds a literal "k" from W register and table start address which is stored in the PCL register. The result is real data address in program memory. Upon return from the subroutine, the W register will contain the addressed literal k. In this case, it is the "k2" literal.

RETFIE (**RETurn From IntErrupt**) represents a return from interrupt routine. In contrast to the RETURN instruction, it may automatically set the GIE bit (**Global Interrupt Enable**). When an interrupt occurs this bit is automatically cleared. Only the program counter is pushed to the stack, which means that there is no auto save of registers’ status and the current status either. The problem is solved by saving status of all important registers at the beginning of interrupt routine. These values are retrieved to these registers immediately before leaving the interrupt routine.

Conditional jumps are executed by two instructions: BTFSC and BTFSS. Depending on the state of bit being tested in the ‘f’ register, the following instruction will be skipped or not.

Instruction Execution Time

All instructions are single-cycle instructions. The only exception may be conditional branch instructions (if condition is met) or instructions being executed upon the program counter. In both cases, two cycles are required for instruction execution where the second cycle is executed as a NOP (**No Operation**). A single-cycle instruction consists of four clock cycles. If 4MHz oscillator is used, a nominal time for instruction execution is 1µS. In case of jump, the instruction execution time is 2µS.

Instructions

Legend

f - Any memory location (register);
W - Working register (accumulator);
b - Bit address within an 8-bit register;
d - Destination bit;
[label] - Set of 8 characters indicating start of particular address in the program;
TOS - Top of stack;
[] - Option;
<> - bit field in register (several bit addresses);
C - Carry/Borrow bit of the STATUS register;
DC - Digit Carry bit of the STATUS register; and
Z - Zero bit of the STATUS register.

ADDLW - Add literal and W

Syntax: [label] ADDLW k

Description: The content of the register **W** is added to the 8-bit literal **k**. The result is stored in the **W** register.

Operation: (**W**) + **k** -> **W**

Operand: 0 ≤ **k** ≤ 255

Status affected: C, DC, Z

Number of cycles: 1

EXAMPLE:

```
....
[label] ADDLW 0x15
```

```
Before instruction execution: W=0x10
After instruction: W=0x25
C=0 (the result is not greater than 0xFF, which means that Carry has not occurred).
```

ADDWF - Add W and f

Syntax: [label] ADDWF f, d

Description: Add the contents of the **W** and **f** registers.
If **d = w** or **d = 0** the result is stored in the **W** register.
If **d = f** or **d = 1** the result is stored in register **f**.

Operation: (**W**) + (**f**) -> **d**

Operand: 0 ≤ **f** ≤ 127, d [0,1]

Status affected: C, DC, Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] ADDWF REG,w
```

```
Before instruction execution: W = 0x17
REG = 0xC2
After instruction: W = 0xD9
REG = 0xC2
C=0 (No carry occurs, i.e. the result is maximum 8-bit long).
```

EXAMPLE 2:

```
....
[label] ADDWF INDF,f
```

```
Before instruction execution: W=0x17
```

```
FSR = 0xC2 Register at address 0xC2 contains the value 0x20
After instruction: W = 0x17
FSR=0xC2, Register at address 0xC2 contains the value 0x37
```

ANDLW - AND literal with W

Syntax: [label] ANDLW k

Description: The content of the register **W** is AND'ed with the 8-bit literal **k**. It means that the result will contain one (1) only if both corresponding bits of operand are ones (1). The result is stored in the **W** register.

Operation: (**W**) AND **k** -> **W**

Operand: 0 ≤ **k** ≤ 255

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] ANDLW 0x5F
```

```
Before instruction execution: W = 0xA3 ; 1010 0011 (0xA3)
                              ; 0101 1111 (0x5F)
                              ------------------
After instruction:  W = 0x03 ; 0000 0011 (0x03)
Z = 0 (result is not 0)
```

EXAMPLE 2:

```
....
[label] ANDLW 0x55
```

```
Before instruction execution: W = 0xAA ; 1010 1010 (0xAA)
                              ; 0101 0101 (0x55)
                              ------------------
After instruction:  W = 0x00 ; 0000 0000 (0x00)
Z = 1( result is 0)
```

ANDWF - AND W with f

Syntax: [label] ANDWF f,d

Description: AND the **W** register with register **f**.
If **d = w** or **d = 0**, the result is stored in the **W** register.
If **d = f** or **d = 1**, the result is stored in register **f**.

Operation: (**W**) AND (**f**) -> **d**

Operand: 0 ≤ **f** ≤ 127, d[0,1]

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] ANDWF REG,f
```

```
Before instruction execution: W = 0x17, REG = 0xC2 ; 0001 0111 (0x17)
                                                    ; 1100 0010 (0xC2)
                                                    ------------------
After instruction: W = 0x17, REG = 0x02  ; 0000 0010 (0x02)
```

EXAMPLE 2:

```
....
[label] ANDWF FSR,w
```

```
Before instruction execution: W = 0x17, FSR = 0xC2 ; 0001 0111 (0x17)
                                                    ; 1100 0010 (0xC2)
                                                    ------------------
After instruction: W = 0x02, FSR = 0xC2  ; 0000 0010 (0x02)
```

BCF - Bit Clear f

Syntax: [label] BCF f, b

Description: Bit **b** of register **f** is cleared.

Operation: (0) -> f(b)

Operand: 0 ≤ **f** ≤ 127, 0 ≤ **b** ≤ 7

Status affected: -

Number of cycles: 1

EXAMPLE 1:

```
....
[label] BCF REG,7
```

```
Before instruction execution: REG = 0xC7 ; 1100 0111 (0xC7)
After instruction:  REG = 0x47 ; 0100 0111 (0x47)
```

EXAMPLE 2:

```
....
[label] BCF INDF,3
```

```
Before instruction execution: W = 0x17
                    FSR = 0xC2
                    Register at address (FSR)contains the value 0x2F
After instruction:  W = 0x17
                    FSR = 0xC2
                    Register at address (FSR)contains the value 0x27
```

BSF - Bit set f

Syntax: [label] BSF f,b

Description: Bit **b** of register **f** is set.

Operation: 1 -> f (b)

Operand: 0 ≤ **f** ≤ 127, 0 ≤ **b** ≤ 7

Status affected: -

Number of cycles: 1

EXAMPLE 1:

```
....
[label] BSF REG,7
```

```
Before instruction execution: REG = 0x07 ; 0000 0111 (0x07)
After instruction:  REG = 0x87 ; 1000 0111 (0x87)
```

EXAMPLE 2:

```
....
[label] BSF INDF,3
```

```
Before instruction execution: W = 0x17
                    FSR = 0xC2
                    Register at address (FSR)contains the value 0x20
After instruction:  W = 0x17
                    FSR = 0xC2
                    Register at address (FSR)contains the value 0x28
```

BTFSC - Bit test f, Skip if Clear

Syntax: [label] BTFSC f, b

Description: If bit **b** of register **f** is 0, the next instruction is discarded and a NOP is executed instead, making this a two-cycle instruction.

Operation: Discard the next instruction if f(b) = 0

Operand: 0 ≤ **f** ≤ 127, 0 ≤ **b** ≤ 7

Status affected: -

Number of cycles: 1 or 2 depending on bit **b**

EXAMPLE:

```
       ....
LAB_01 BTFSC REG,1 ; Test bit 1 of REG
LAB_02 ....        ; Skip this line if bit = 1
LAB_03 ....        ; Jump here if bit = 0
```

```
Before instruction execution: The program counter was at address LAB_01.
After instruction:
- if bit 1 of REG is cleared, program counter points to address LAB_03.
- if bit 1 of REG is set, program counter points to address LAB_02.
```

BTFSS - Bit test f, Skip if Set

Syntax: [label] BTFSS f, b

Description: If bit **b** of register **f** is 1, the next instruction is discarded and a NOP is executed instead, making this a two-cycle instruction.

Operation: Discard the next instruction if f(b) = 1

Operand: 0 ≤ **f** ≤ 127, 0 ≤ **b** ≤ 7

Status affected: -

Number of cycles: 1 or 2 depending on bit **b**

EXAMPLE:

```
       ....
LAB_01 BTFSS REG,3 ; Test bit 3 of REG
LAB_02 ....        ; Skip this line if bit = 0
LAB_03 ....        ; Jump here if bit = 1
```

```
Before instruction execution: The program counter was at address LAB_01
After instruction:
- if bit 3 of REG is cleared, program counter points to address LAB_03.
- if bit 3 of REG is cleared, program counter points to address LAB_02.
```

CALL - Calls Subroutine

Syntax: [label] CALL k

Description: Calls subroutine. First the address of the next instruction to execute is pushed onto the stack. It is the PC+1 address. Afterwards, the subroutine address is written to the program counter.

Operation: (PC) + 1 -> (Top Of Stack - TOS) k -> PC (10 : 0), (PCLATH (4 : 3)) -> PC (12 : 11)

Operand: 0 ≤ **k** ≤ 2047

Flag: -

Status affected: 2

EXAMPLE:

```
       ....
LAB_01 CALL LAB_02 ; Call subroutine LAB_02
       ....
       ....
LAB_02 ....
```

```
Before instruction execution: PC = address LAB_01
                 TOS (top of stack) = x
After instruction:  PC = address LAB_02
                 TOS (top of stack) = LAB_01
```

CLRF - Clear f

Syntax: [label] CLRF **f**

Description: The content of register **f** is cleared and the Z flag of the STATUS register is set.

Operation: 0 -> f

Operand: 0 ≤ **f** ≤ 127

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] CLRF TRISB
```

```
Before instruction execution: TRISB=0xFF
After instruction:  TRISB=0x00
Z = 1
```

EXAMPLE 2:

```
Before instruction execution: FSR=0xC2
                 Register at address 0xC2 contains the value 0x33
After instruction:  FSR=0xC2
                 Register at address 0xC2 contains the value 0x00
                 Z = 1
```

CLRW - Clear W

Syntax: [label] CLRW

Description: Register **W** is cleared and the Z flag of the STATUS register is set.

Operation: 0 -> W

Operand: -

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] CLRW
```

```
Before instruction: W=0x55
After instruction:  W=0x00
                    Z = 1
```

CLRWDT - Clear Watchdog Timer

Syntax: [label] CLRWDT

Description: Resets the **watchdog** timer and the WDT prescaler. Status bits TO and PD are set.

Operation: 0 -> WDT 0 -> WDT prescaler 1 -> TO 1 -> PD

Operand: -

Status affected: TO, PD

Number of cycles: 1

EXAMPLE :

```
....
[label] CLRWDT
```

```
Before instruction execution: WDT counter = x
                    WDT prescaler = 1:128
After instruction:  WDT counter = 0x00
                    WDT prescaler = 0
                    TO = 1
                    PD = 1
                    WDT prescaler = 1: 128
```

COMF - Complement f

Syntax: [label] COMF f, d

Description: The content of register **f** is complemented (logic zeros (0) are replaced by ones (1) and vice versa). If **d** = **w** or **d** = 0 the result is stored in **W**. If **d** = **f** or **d** = 1 the result is stored in register **f**.

Operation: (f) -> d

Operand: 0 ≤ **f** ≤ 127, d[0,1]

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] COMF REG,w
```

```
Before instruction execution: REG = 0x13 ; 0001 0011 (0x13)
                               ; complementing
                               ------------------
After instruction:  REG = 0x13 ; 1110 1100 (0xEC)
                    W = 0xEC
```

EXAMPLE 2:

```
....
[label] COMF INDF, f
```

```
Before instruction execution: FSR = 0xC2
                    Register at address (FSR)contains the value 0xAA
After instruction:  FSR = 0xC2
                    Register at address (FSR)contains the value 0x55
```

DECF - Decrement f

Syntax: [label] DECF f, d

Description: Decrement register **f** by one. If **d** = **w** or **d** = 0, the result is stored in the **W** register. If **d** = **f** or **d** = 1, the result is stored in register **f**.

Operation: (f) - 1 -> d

Operand: 0 ≤ **f** ≤ 127, d[0,1]

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] DECF REG,f
```

```
Before instruction execution: REG = 0x01
                              Z = 0
After instruction:            REG = 0x00
                              Z = 1
```

EXAMPLE 2:

```
....
[label] DECF REG,w
```

```
Before instruction execution: REG = 0x13
                              W = x, Z = 0
After instruction:            REG = 0x13
                              W = 0x12, Z = 0
```

DECFSZ - Decrement f, Skip if 0

Syntax: [label] DECFSZ f, d

Description: Decrement register f by one. If **d** = **w** or **d** = 0, the result is stored in the **W** register. If **d** = **f** or **d** = **1**, the result is stored in register **f**. If the result is 0, then a NOP is executed instead, making this a two-cycle instruction.

Operation: (f) - 1 -> d

Operand: 0 ≤ **f** ≤ 127, d[0,1]

Status affected: -

Number of cycles: 1 or 2 depending on the result.

EXAMPLE 1:

```
       ....
       MOVLW  .10
       MOVWF  CNT       ;10 -> CNT
Loop   ......
       ......           ;Instruction block
       ......
       DECFSZ CNT,f     ; decrement REG by one
       GOTO   Loop      ; Skip this line if = 0
LAB_03 .......          ; Jump here if = 0
```

In this example, instruction block is executed as many times as the initial value of the variable CNT is, which in this example is 10.

GOTO - Unconditional Branch

Syntax: [label] GOTO k

Description: Unconditional jump to the address **k**.

Operation: (k) -> PC(10:0), (PCLATH(4:3)) -> PC(12:11)

Operand: 0 ≤ **k** ≤ 2047

Status affected: -

Number of cycles: 2

EXAMPLE :

```
       ....
LAB_00 GOTO LAB_01 ; Jump to LAB_01
       .....
       .....
LAB_01 .....       ; Program continues from here
```

```
Before instruction execution: PC = LAB_00 address
After instruction:  PC = LAB_01 address
```

INCF - Increment f

Syntax: [label] INCF **f, d**

Description: Increment register **f** by one.
If **d** = **w** or **d** = 0, the result is stored in register **W**.
If **d** = **f** or **d** = **1**, the result is stored in register **f**.

Operation: (**f**) + 1 -> d

Operand: 0 ≤ **f** ≤ 127, d[0,1]

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] INCF REG,w
```

```
Before instruction execution: REG = 0x10
                    W = x, Z = 0
After instruction:  REG = 0x10
                    W = 0x11, Z = 0
```

EXAMPLE 2:

```
....
[label] INCF REG,f
```

```
Before instruction execution: REG = 0xFF
                    Z = 0
After instruction:  REG = 0x00
                    Z = 1
```

INCFSZ - Increment f, Skip if 0

Syntax: [label] INCFSZ **f**, **d**

Description: Register **f** is incremented by one. If **d** = **w** or **d** = 0, the result is stored in register **W**. If **d** = **f** or **d** = 1, the result is stored in register **f**. If the result is 0, then a NOP is executed instead, making this a two-cycle instruction.

Operation: (f) + 1 -> d

Operand: 0 ≤ **f** ≤ 127, d[0,1]

Status affected: -

Number of cycles: 1 or 2 depending on the result.

EXAMPLE :

```
       ....
LAB_01 INCFSZ REG,f ; Increment REG by one
LAB_02 .......      ; Skip this line if result is 0
```

```
LAB_03 .......      ; Jump here if result is 0
```

The content of program counter Before instruction execution, PC= LAB_01address.

The content of REG after instruction, REG = REG+1. If REG=0, the program counter points to the address of label LAB_03. Otherwise, the program counter points to address of the next instruction, i.e. to LAB_02 address.

IORLW - Inclusive OR literal with W

Syntax: [label] IORLW **k**

Description: The content of the **W** register is OR'ed with the 8-bit literal **k**. The result is stored in register **W**.

Operation: (W) OR (k) -> W

Operand: 0 ≤ **k** ≤ 255

Status affected: -

Number of cycles: 1

EXAMPLE :

```
....
[label] IORLW 0x35
```

```
Before instruction execution: W = 0x9A
After instruction:  W = 0xBF
                    Z = 0
```

IORWF - Inclusive OR W with f

Syntax: [label] IORWF **f**, **d**

Description: The content of register **f** is OR'ed with the content of **W** register. If **d** = **w** or **d** = **0**, the result is stored in the **W** register. If **d** = **f** or **d** = 1, the result is stored in register **f**.

Operation: (**W**) OR (**f**) -> **d**

Operand: 0 ≤ **f** ≤ 127, **d** -> [0,1]

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] IORWF REG,w
```

```
Before instruction execution: REG = 0x13,
                    W = 0x91
After instruction:  REG = 0x13,
                    W = 0x93 Z = 0
```

EXAMPLE 2:

```
....
[label] IORWF REG,f
```

```
Before instruction execution: REG = 0x13,
                    W = 0x91
After instruction:  REG = 0x93,
                    W = 0x91 Z = 0
```

MOVF - Move f

Syntax: [label] MOVF **f**, **d**

Description: The content of register **f** is moved to a destination determined by the operand d. If **d** = **w** or **d** = 0, the content is moved to register **W**. If **d** = **f** or **d** = 1, the content remains in register **f**. Option **d** = 1 is used to test the content of register **f** because this instruction affects the Z flag of the STATUS register.

Operation: (f) -> d

Operand: 0 ≤ **f** ≤ 127, **d** -> [0,1]

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] MOVF FSR,w
```

```
Before instruction execution: FSR=0xC2
                    W=0x00
After instruction:  W=0xC2
                    Z = 0
```

EXAMPLE 2:

```
....
[label] MOVF INDF,f
```

```
Before instruction execution: W=0x17
                    FSR=0xC2, register at address 0xC2 contains the value 0x00
After instruction:  W=0x17
                    FSR=0xC2, register at address 0xC2 contains the value 0x00,
                    Z = 1
```

MOVLW - Move literal to W

Syntax: [label] MOVLW k

Description: 8-bit literal **k** is moved to register **W**.

Operation: k -> (W)

Operand: 0 ≤ **k** ≤ 255

Status affected: -

Number of cycles: 1

EXAMPLE 1:

```
....
[label] MOVLW 0x5A
```

```
After instruction: W=0x5A
```

EXAMPLE 2:

```
Const equ 0x40
[label] MOVLW Const
```

```
Before instruction execution: W=0x10
After instruction:  W=0x40
```

MOVWF - Move W to f

Syntax: [label] MOVWF **f**

Description: The content of register W is moved to register **f**.

Operation: (W) -> f

Operand: 0 ≤ **f** ≤ 127

Status affected: -

Number of cycles: 1

EXAMPLE 1:

```
....
[label] MOVWF OPTION_REG
```

```
Before instruction execution: OPTION_REG=0x20
                              W=0x40
After instruction:            OPTION_REG=0x40
                              W=0x40
```

EXAMPLE 2:

```
....
[label] MOVWF INDF
```

```
Before instruction execution: W=0x17
                              FSR=0xC2, register at address 0xC2 contains the value 0x00
After instruction:            W=0x17
                              FSR=0xC2, register at address 0xC2 contains the value 0x17
```

NOP - No Operation

Syntax: [label] NOP

Description: No operation.

Operation: -

Operand: -

Status affected: -

Number of cycles: 1

EXAMPLE :

```
....
[label] NOP ; 1us delay (oscillator 4MHz)
```

```
Before instruction execution: PC = x
After instruction:            PC = x + 1
```

RETFIE - Return from Interrupt

Syntax: [labels] RETFIE

Description: Return from subroutine. The value is popped from the stack and loaded to the program counter. Interrupts are enabled by setting the bit GIE of the INTCON register.

Operation: TOS -> PC, 1 -> GIE

Operand: -

Status affected: -

Number of cycles: 2

EXAMPLE :

```
....
[label] RETFIE
```

```
Before instruction execution: PC = x
```

```
                     GIE (interrupt enable bit of the SATUS register) = 0
After instruction:   PC = TOS (top of stack)
                     GIE = 1
```

RETLW - Return with literal in W

Syntax: [label] RETLW **k**

Description: 8-bit literal **k** is loaded into register **W**. The value from the top of stack is loaded to the program counter.

Operation: (k) -> W; top of stack (TOP) -> PC

Operand: -

Status affected: -

Number of cycles: 2

EXAMPLE :

```
....
[label] RETLW 0x43
```

```
Before instruction execution: W = x
                     PC = x
                     TOS (top of stack) = x
After instruction:   W = 0x43
                     PC = TOS (top of stack)
                     TOS (top of stack) = TOS - 1
```

RETURN - Return from Subroutine

Syntax: [label] RETURN

Description: Return from subroutine. The value from the top of stack is loaded to the program counter. This is a two-cycle instruction.

Operation: TOS -> program counter PC.

Operand: -

Status affected: -

Number of cycles: 2

EXAMPLE :

```
....
[label] RETURN
```

```
Before instruction execution: PC = x
                     TOS (top of stack) = x
After instruction:   PC = TOS (top of stack)
                     TOS (top of stack) = TOS - 1
```

RLF - Rotate Left f through Carry

Syntax: [label] RLF **f**, **d**

Description: The content of register f is rotated one bit to the left through the Carry flag. If **d** = **w** or **d** = 0, the result is stored in register **W**. If **d** = **f** or **d** = 1, the result is stored in register **f**.

Operation: (f(n)) -> d(n+1), f(7) -> C, C -> d(0);

Operand: 0 ≤ **f** ≤ 127, d[0,1]

Status affected: C

Number of cycles: 1

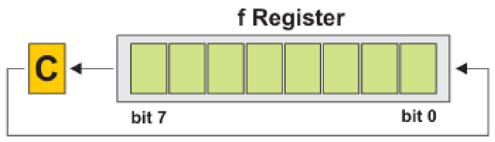

Fig. 9-1 f Register

EXAMPLE 1:

```
....
[label] RLF REG,w
```

```
Before instruction execution: REG = 1110 0110
                    C = 0
After instruction:  REG = 1110 0110
                    W = 1100 1100
                    C = 1
```

EXAMPLE 2:

```
....
[label] RLF REG,f
```

```
Before instruction execution: REG = 1110 0110
                    C = 0
After instruction:  REG = 1100 1100
                    C = 1
```

RRF - Rotate Right f through Carry

Syntax: [label] RRF **f**, **d**

Description: The content of register **f** is rotated one bit right through the Carry flag. If **d** = **w** or **d** = 0, the result is stored in register **W**. If **d** = **f** or **d** = 1, the result is stored in register **f**.

Operation: (f(n)) -> d(n-1), f(0) -> C, C -> d(7);

Operand: 0 ≤ **f** ≤ 127, d -> [0,1]

Status affected: C

Number of cycles: 1

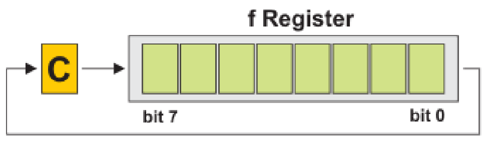

Fig. 9-2 f Register

EXAMPLE 1:

```
....
[label] RRF REG,w
```

```
Before instruction execution: REG = 1110 0110
                    W = x
                    C = 0
After instruction:  REG = 1110 0110
                    W = 0111 0011
                    C = 0
```

EXAMPLE 2:

```
....
```

```
[label] RRF REG,f
```

```
Before instruction execution: REG = 1110 0110, C = 0
After instruction:  REG = 0111 0011, C = 0
```

SLEEP - Enter Sleep mode

Syntax: [label] SLEEP

Description: The processor enters sleep mode. The oscillator is stopped. PD bit (**Power Down**) of the STATUS register is cleared. TO bit of the same register is set. The WDT and its prescaler are cleared.

Operation: 0 -> WDT, 0 -> WDT prescaler, 1 -> TO, 0 -> PD

Operand: -

Status affected: TO, PD

Number of cycles: 1

EXAMPLE :

```
....
[label] SLEEP
```

```
Before instruction execution: WDT counter = x
                              WDT prescaler = x
After instruction:            WDT counter = 0x00
                              WDT prescaler = 0
                              TO = 1
                              PD = 0
```

SUBLW - Subtract W from literal

Syntax: [label] SUBLW **k**

Description: The content of register **W** is subtracted from the literal **k**. The result is stored in register **W**.

Operation: k - (W) -> W

Operand: 0 ≤ **k** ≤ 255

Status affected: C, DC, Z

Number of cycles: 1

EXAMPLE :

```
....
[label] SUBLW 0x03
```

```
Before instruction execution: W = 0x01, C = x, Z = x
After instruction:  W = 0x02, C = 1, Z = 0 result is positive

Before instruction execution: W = 0x03, C = x, Z = x
After instruction:  W = 0x00, C = 1, Z = 1 result is 0

Before instruction execution: W = 0x04, C = x, Z = x
After instruction:  W = 0xFF, C = 0, Z = 0 result is negative
```

SUBWF - Subtract W from f

Syntax: [label] SUBWF **f**, **d**

Description: The content of register **W** is subtracted from register **f**.
If **d** = **w** or **d** = **0**, the result is stored in register **W**. If **d** = **f** or **d** = **1**, the result is stored in register **f**.

Operation: (f) - (W) -> d

Operand: 0 ≤ **f** ≤ 127, d [0,1]

Status affected: C, DC, Z

Number of cycles: 1

EXAMPLE :

```
....
[label] SUBWF REG,f
```

```
Before instruction execution: REG = 3, W = 2, C = x, Z = x
After instruction:  REG = 1, W = 2, C = 1, Z = 0 result is positive

Before instruction execution: REG = 2, W = 2, C = x, Z = x
After instruction:  REG = 0, W = 2, C = 1, Z = 1 result is 0

Before instruction execution: REG = 1, W = 2, C = x, Z = x
After instruction:  REG = 0xFF, W = 2, C = 0, Z = 0 result is negative
```

SWAPF - Swap Nibbles in f

Syntax: [label] SWAPF **f**, **d**

Description: The upper and lower nibbles of register **f** are swapped. If **d** = **w** or **d** = **0**, the result is stored in register **W**. If **d** = **f** or **d** = **1**, the result is stored in register **f**.

Operation: f(0:3) -> d(4:7), f(4:7) -> d(0:3);

Operand: 0 ≤ **f** ≤ 127, d [0,1]

Status affected: -

Number of cycles: 1

EXAMPLE 1:

```
....
[label] SWAPF REG,w
```

```
Before instruction execution: REG=0xF3
After instruction:  REG=0xF3
                    W = 0x3F
```

EXAMPLE 2:

```
....
[label] SWAPF REG,f
```

```
Before instruction execution: REG=0xF3
After instruction:  REG=0x3F
```

XORLW - Exclusive OR literal with W

Syntax: [label] **XORLW k**

Description: The content of register **W** is XOR'ed with the 8-bit literal **k** . The result is stored in register **W**.

Operation: (**W**) .XOR. **k** -> **W**

Operand: 0 ≤ **k** ≤ 255

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] XORLW 0xAF
```

```
Before instruction execution: W = 0xB5 ; 1011 0101 (0xB5)
```

```
                              ; 1010 1111 (0xAF)
                              ------------------
After instruction:  W = 0x1A ; 0001 1010 (0x1A)
                    Z = 0
```

EXAMPLE 2:

```
Const equ 0x37
[label] XORLW Const
```

```
Before instruction execution: W=0xAF       ; 1010 1111 (0xAF)
                    Const = 0x37 ; 0011 0111 (0x37)
                    -------------------------------
After instruction:  W = 0x98     ; 1001 1000 (0x98)
                    Z = 0
```

XORWF - Exclusive OR W with f

Syntax: [label] XORWF **f**, **d**

Description: The content of register **f** is XOR'ed with the content of register **W**. A bit of result is set only if the corresponding bits of operands are different. If **d** = **w** or **d** = 0, the result is stored in register **W**. If **d** = **f** or **d** = **1**, the result is stored in register **f**.

Operation: (**W**) .XOR. **k** -> **d**

Operand: 0 ≤ **f** ≤ 127, d[0,1]

Status affected: Z

Number of cycles: 1

EXAMPLE 1:

```
....
[label] XORWF REG,f
```

```
Before instruction execution: REG = 0xAF, W = 0xB5 ; 1010 1111 (0xAF)
                                                    ; 1011 0101 (0xB5)
                                                    ------------------
After instruction:  REG = 0x1A, W = 0xB5 ; 0001 1010 (0x1A)
```

EXAMPLE 2:

```
....
[label] XORWF REG,w
```

```
Before instruction execution: REG = 0xAF, W = 0xB5 ; 1010 1111 (0xAF)
                                                    ; 1011 0101 (0xB5)
                                                    ------------------
After instruction:  REG = 0xAF, W = 0x1A ; 0001 1010 (0x1A)
```

In addition to the preceding instructions, *Microchip* has also introduced some other instructions. To be more precise, they are not instructions as such, but macros supported by MPLAB. *Microchip* calls them "Special Instructions" since all of them are in fact obtained by combining already existing instructions.

INSTRUCTION		DESCRIPTION	EQUIVALENT INSTRUCTION	STATUS AFFECTED
ADDCF	f,d	Add with carry	BTFSC INCF	STATUS,C
ADDDCF	f,d	Add with Digit Carry	BTFSC INCF	STATUS,DC
B	k	Branch	GOTO	
BC	k	Branch on Carry	BTFSC GOTO	STATUS,C
BDC	k	Branch on Digit Carry	BTFSC GOTO	STATUS,DC

BNC	k	Branch on No Carry	BTFSS GOTO	STATUS,C
BNDC	k	Branch on No Digit Carry	BTFSS GOTO	STATUS,DC
BNZ	k	Branch on No Zero	BTFSS GOTO	STATUS,Z
BZ	k	Branch on Zero	BTFSC GOTO	STATUS,Z
CLRC		Clear Carry	BCF	STATUS,C
CLRDC		Clear Digit Carry	BCF	STATUS,DC
CLRZ		Clear Zero	BCF	STATUS,Z
MOVFW	f	Move File to W	MOVF	
SETC	f	Set Carry	BSF	STATUS,C
SETDC		Set Digit Carry	BSF	STATUS,DC
SETZ		Set Zero	BSF	STATUS,Z
SKPC		Skip on Carry	BTFSS	STATUS,C
SKPDC		Skip on Digit Carry	BTFSS	STATUS,DC
SKPNC		Skip on No Carry	BTFSC	STATUS,Z
SKPNDC		Skip on No Digit Carry	BTFSC	STATUS,DC
SKPNZ		Skip on Non Zero	BTFSC	STATUS,Z
SKPZ		Skip on Zero	BTFSS	STATUS,Z
SUBCF	f, d	Subtract Carry from File	BTFSC DECF	STATUS,C
SUBDCF	f, d	Subtract Digit Carry from File	BTFSC DECF	STATUS,DC
TSTF	f	Test File	MOVF	

← previous chapter | table of contents | next chapter →