9.0 INSTRUCTION SET SUMMARY

Each PIC16CXX instruction is a 14-bit word divided into an OPCODE which specifies the instruction type and one or more operands which further specify the operation of the instruction. The PIC16CXX instruction set summary in Table 9-2 lists **byte-oriented**, **bit-oriented**, and **literal and control** operations. Table 9-1 shows the opcode field descriptions.

For **byte-oriented** instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For **bit-oriented** instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the number of the file in which the bit is located.

For **literal and control** operations, 'k' represents an eight or eleven bit constant or literal value.

TABLE 9-1 OPCODE FIELD DESCRIPTIONS

Field	Description
`f`	Register file address (0x00 to 0x7F)
`W`	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1
`label`	Label name
`TOS`	Top of Stack
`PC`	Program Counter
`PCLATH`	Program Counter High Latch
`GIE`	Global Interrupt Enable bit
`WDT`	Watchdog Timer/Counter
$\overline{\text{TO}}$	Time-out bit
$\overline{\text{PD}}$	Power-down bit
`dest`	Destination either the W register or the specified register file location
[ ]	Options
( )	Contents
$\rightarrow$	Assigned to
< >	Register bit field
$\in$	In the set of
italics	User defined term (font is courier)

The instruction set is highly orthogonal and is grouped into three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

All instructions are executed within one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In this case, the execution takes two instruction cycles with the second cycle executed as a NOP. One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 µs. If a conditional test is true or the program counter is changed as a result of an instruction, the instruction execution time is 2 µs.

Table 9-2 lists the instructions recognized by the MPASM assembler.

Figure 9-1 shows the general formats that the instructions can have.

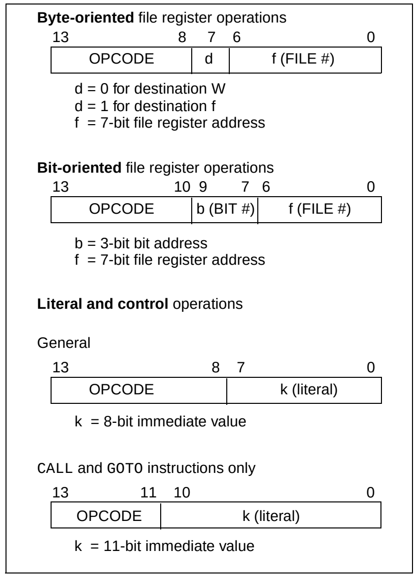
Note: To maintain upward compatibility with future PIC16CXX products, do not use the `OPTION` and `TRIS` instructions.

All examples use the following format to represent a hexadecimal number:

0xhh

where h signifies a hexadecimal digit.

FIGURE 9-1: GENERAL FORMAT FOR INSTRUCTIONS



Byte-oriented file register operations
13 8 7 6 0
OPCODE
d
f (FILE #)
d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address
Bit-oriented file register operations
13 10 9 7 6 0
OPCODE
b (BIT #)
f (FILE #)
b = 3-bit bit address
f = 7-bit file register address
Literal and control operations
General
13 8 7 0
OPCODE
k (literal)
k = 8-bit immediate value
`CALL` and `GOTO` instructions only
13 11 10 0
OPCODE
k (literal)
k = 11-bit immediate value

TABLE 9-2 PIC16FXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	**f, d**	Add W and f	1	`00`	`0111`	`dfff`	`ffff`	C,DC,Z	1,2
ANDWF	**f, d**	AND W with f	1	`00`	`0101`	`dfff`	`ffff`	Z	1,2
CLRF	**f**	Clear f	1	`00`	`0001`	`lfff`	`ffff`	Z	2
CLRW	**-**	Clear W	1	`00`	`0001`	`0xxx`	`xxxx`	Z	
COMF	**f, d**	Complement f	1	`00`	`1001`	`dfff`	`ffff`	Z	1,2
DECF	**f, d**	Decrement f	1	`00`	`0011`	`dfff`	`ffff`	Z	1,2
DECFSZ	**f, d**	Decrement f, Skip if 0	1(2)	`00`	`1011`	`dfff`	`ffff`		1,2,3
INCF	**f, d**	Increment f	1	`00`	`1010`	`dfff`	`ffff`	Z	1,2
INCFSZ	**f, d**	Increment f, Skip if 0	1(2)	`00`	`1111`	`dfff`	`ffff`		1,2,3
IORWF	**f, d**	Inclusive OR W with f	1	`00`	`0100`	`dfff`	`ffff`	Z	1,2
MOVF	**f, d**	Move f	1	`00`	`1000`	`dfff`	`ffff`	Z	1,2
MOVWF	**f**	Move W to f	1	`00`	`0000`	`lfff`	`ffff`		
NOP	**-**	No Operation	1	`00`	`0000`	`0xx0`	`0000`		
RLF	**f, d**	Rotate Left f through Carry	1	`00`	`1101`	`dfff`	`ffff`	C	1,2
RRF	**f, d**	Rotate Right f through Carry	1	`00`	`1100`	`dfff`	`ffff`	C	1,2
SUBWF	**f, d**	Subtract W from f	1	`00`	`0010`	`dfff`	`ffff`	C,DC,Z	1,2
SWAPF	**f, d**	Swap nibbles in f	1	`00`	`1110`	`dfff`	`ffff`		1,2
XORWF	**f, d**	Exclusive OR W with f	1	`00`	`0110`	`dfff`	`ffff`	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	**f, b**	Bit Clear f	1	`01`	`00bb`	`bfff`	`ffff`		1,2
BSF	**f, b**	Bit Set f	1	`01`	`01bb`	`bfff`	`ffff`		1,2
BTFSC	**f, b**	Bit Test f, Skip if Clear	1 (2)	`01`	`10bb`	`bfff`	`ffff`		3
BTFSS	**f, b**	Bit Test f, Skip if Set	1 (2)	`01`	`11bb`	`bfff`	`ffff`		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	**k**	Add literal and W	1	`11`	`111x`	`kkkk`	`kkkk`	C,DC,Z	
ANDLW	**k**	AND literal with W	1	`11`	`1001`	`kkkk`	`kkkk`	Z	
CALL	**k**	Call subroutine	2	`10`	`0kkk`	`kkkk`	`kkkk`		
CLRWDT	**-**	Clear Watchdog Timer	1	`00`	`0000`	`0110`	`0100`	$\overline{\text{TO}}$,$\overline{\text{PD}}$	
GOTO	**k**	Go to address	2	`10`	`1kkk`	`kkkk`	`kkkk`		
IORLW	**k**	Inclusive OR literal with W	1	`11`	`1000`	`kkkk`	`kkkk`	Z	
MOVLW	**k**	Move literal to W	1	`11`	`00xx`	`kkkk`	`kkkk`		
RETFIE	**-**	Return from interrupt	2	`00`	`0000`	`0000`	`1001`		
RETLW	**k**	Return with literal in W	2	`11`	`01xx`	`kkkk`	`kkkk`		
RETURN	**-**	Return from Subroutine	2	`00`	`0000`	`0000`	`1000`		
SLEEP	**-**	Go into standby mode	1	`00`	`0000`	`0110`	`0011`	$\overline{\text{TO}}$,$\overline{\text{PD}}$	
SUBLW	**k**	Subtract W from literal	1	`11`	`110x`	`kkkk`	`kkkk`	C,DC,Z	
XORLW	**k**	Exclusive OR literal with W	1	`11`	`1010`	`kkkk`	`kkkk`	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

9.1 Instruction Descriptions

ADDLW — Add Literal and W

Syntax:	[*label*] ADDLW k
Operands:	$0 \le k \le 255$
Operation:	$(W) + k \rightarrow (W)$
Status Affected:	C, DC, Z
Encoding:	`11` `111x` `kkkk` `kkkk`
Description:	The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example:
```
ADDLW   0x15
```
Before Instruction
W = 0x10
After Instruction
W = 0x25

ADDWF — Add W and f

Syntax:	[*label*] ADDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	$(W) + (f) \rightarrow$ (destination)
Status Affected:	C, DC, Z
Encoding:	`00` `0111` `dfff` `ffff`
Description:	Add the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example
```
ADDWF   FSR, 0
```
Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0xD9
FSR = 0xC2

ANDLW — AND Literal with W

Syntax:	[*label*] ANDLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .AND. (k) $\rightarrow$ (W)
Status Affected:	Z
Encoding:	`11` `1001` `kkkk` `kkkk`
Description:	The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal "k"	Process data	Write to W

Example
```
ANDLW   0x5F
```
Before Instruction
W = 0xA3
After Instruction
W = 0x03

ANDWF — AND W with f

Syntax:	[*label*] ANDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) .AND. (f) $\rightarrow$ (destination)
Status Affected:	Z
Encoding:	`00` `0101` `dfff` `ffff`
Description:	AND the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example
```
ANDWF   FSR, 1
```
Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0x17
FSR = 0x02

BCF — Bit Clear f

Syntax:	[*label*] BCF f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	$0 \rightarrow$ (f<b>)
Status Affected:	None
Encoding:	`01` `00bb` `bfff` `ffff`
Description:	Bit 'b' in register 'f' is cleared.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example
```
BCF     FLAG_REG, 7
```
Before Instruction
FLAG_REG = 0xC7
After Instruction
FLAG_REG = 0x47

BSF — Bit Set f

Syntax:	[*label*] BSF f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	$1 \rightarrow$ (f<b>)
Status Affected:	None
Encoding:	`01` `01bb` `bfff` `ffff`
Description:	Bit 'b' in register 'f' is set.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example
```
BSF     FLAG_REG,   7
```
Before Instruction
FLAG_REG = 0x0A
After Instruction
FLAG_REG = 0x8A

BTFSC — Bit Test, Skip if Clear

Syntax:	[*label*] BTFSC f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	skip if (f<b>) = 0
Status Affected:	None
Encoding:	`01` `10bb` `bfff` `ffff`
Description:	If bit 'b' in register 'f' is '1' then the next instruction is executed. If bit 'b', in register 'f', is '0' then the next instruction is discarded, and a NOP is executed instead, making this a $2T_{CY}$ instruction.
Words:	1
Cycles:	1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	No-Operation

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No-Operation	No-Operation	No-Operation	No-Operation

Example
```
HERE    BTFSC  FLAG,1
FALSE   GOTO   PROCESS_CODE
TRUE    •
        •
        •
```
Before Instruction
PC = address `HERE`
After Instruction
if FLAG<1> = 0,
PC = address `TRUE`
if FLAG<1>=1,
PC = address `FALSE`

BTFSS	**Bit Test f, Skip if Set**
Syntax:	[*label*] BTFSS f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b < 7
Operation:	skip if (f<b>) = 1
Status Affected:	None

Encoding:

01	11bb	bfff	ffff

Description: If bit 'b' in register 'f' is '0' then the next instruction is executed.
If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead, making this a 2TCY instruction.

Words:	1
Cycles:	1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	No-Operation

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE     BTFSC  FLAG,1
FALSE    GOTO   PROCESS_CODE
TRUE     •
         •
         •
```

Before Instruction
PC = address `HERE`
After Instruction
if FLAG<1> = 0,
PC = address `FALSE`
if FLAG<1> = 1,
PC = address `TRUE`

CALL	**Call Subroutine**
Syntax:	[*label*] CALL k
Operands:	0 ≤ k ≤ 2047
Operation:	(PC)+ 1→ TOS, k → PC<10:0>, (PCLATH<4:3>) → PC<12:11>
Status Affected:	None

Encoding:

10	0kkk	kkkk	kkkk

Description: Call Subroutine. First, return address (PC+1) is pushed onto the stack. The eleven bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. `CALL` is a two cycle instruction.

Words:	1
Cycles:	2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	Read literal 'k', Push PC to Stack	Process data	Write to PC
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE    CALL   THERE
```

Before Instruction
PC = Address `HERE`
After Instruction
PC = Address `THERE`
TOS = Address `HERE+1`

CLRF	**Clear f**
Syntax:	[*label*] CLRF f
Operands:	0 ≤ f ≤ 127
Operation:	00h → (f) 1 → Z
Status Affected:	Z

Encoding:

00	0001	1fff	ffff

Description: The contents of register 'f' are cleared and the Z bit is set.

Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example

```
CLRF      FLAG_REG
```

Before Instruction
FLAG_REG = 0x5A
After Instruction
FLAG_REG = 0x00
Z = 1

CLRW	**Clear W**
Syntax:	[*label*] CLRW
Operands:	None
Operation:	00h → (W) 1 → Z
Status Affected:	Z

Encoding:

00	0001	0xxx	xxxx

Description: W register is cleared. Zero bit (Z) is set.

Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No-Operation	Process data	Write to W

Example

```
CLRW
```

Before Instruction
W = 0x5A
After Instruction
W = 0x00
Z = 1

CLRWDT	**Clear Watchdog Timer**
Syntax:	[*label*] CLRWDT
Operands:	None
Operation:	00h → WDT 0 → WDT prescaler, 1 → $\overline{\text{TO}}$ 1 → $\overline{\text{PD}}$
Status Affected:	$\overline{\text{TO}}$, $\overline{\text{PD}}$

Encoding:

00	0000	0110	0100

Description: `CLRWDT` instruction resets the Watchdog Timer. It also resets the prescaler of the WDT. Status bits $\overline{\text{TO}}$ and $\overline{\text{PD}}$ are set.

Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No-Operation	Process data	Clear WDT Counter

Example

```
CLRWDT
```

Before Instruction
WDT counter = ?
After Instruction
WDT counter = 0x00
WDT prescaler = 0
$\overline{\text{TO}}$ = 1
$\overline{\text{PD}}$ = 1

COMF	**Complement f**
Syntax:	[*label*] COMF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	$(\bar{f}) \rightarrow$ (destination)
Status Affected:	Z

Encoding:

`00`	`1001`	`dfff`	`ffff`

Description:	The contents of register 'f' are complemented. If 'd' is 0 the result is stored in W. If 'd' is 1 the result is stored back in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example `COMF REG1,0`

Before Instruction

REG1	=	0x13

After Instruction

REG1	=	0x13
W	=	0xEC

DECF	**Decrement f**
Syntax:	[*label*] DECF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) - 1 $\rightarrow$ (destination)
Status Affected:	Z

Encoding:

`00`	`0011`	`dfff`	`ffff`

Description:	Decrement register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example `DECF CNT, 1`

Before Instruction

CNT	=	0x01
Z	=	0

After Instruction

CNT	=	0x00
Z	=	1

DECFSZ	**Decrement f, Skip if 0**
Syntax:	[*label*] DECFSZ f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) - 1 $\rightarrow$ (destination); skip if result = 0
Status Affected:	None

Encoding:

`00`	`1011`	`dfff`	`ffff`

Description:	The contents of register 'f' are decremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'. If the result is 1, the next instruction, is executed. If the result is 0, then a NOP is executed instead making it a 2TCY instruction.
Words:	1
Cycles:	1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE      DECFSZ   CNT, 1
          GOTO     LOOP
CONTINUE  •
          •
          •
```

Before Instruction

PC	=	address `HERE`

After Instruction

CNT	=	CNT - 1
if CNT	=	0,
PC	=	address `CONTINUE`
if CNT	≠	0,
PC	=	address `HERE+1`

GOTO	**Unconditional Branch**
Syntax:	[*label*] GOTO k
Operands:	$0 \le k \le 2047$
Operation:	k $\rightarrow$ PC<10:0> PCLATH<4:3> $\rightarrow$ PC<12:11>
Status Affected:	None

Encoding:

`10`	`1kkk`	`kkkk`	`kkkk`

Description:	`GOTO` is an unconditional branch. The eleven bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. `GOTO` is a two cycle instruction.
Words:	1
Cycles:	2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	Read literal 'k'	Process data	Write to PC
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example `GOTO THERE`

After Instruction

PC	=	Address `THERE`

INCF	**Increment f**
Syntax:	[*label*] INCF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) + 1 $\rightarrow$ (destination)
Status Affected:	Z

Encoding:

`00`	`1010`	`dfff`	`ffff`

Description:	The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example `INCF CNT, 1`

Before Instruction

CNT	=	0xFF
Z	=	0

After Instruction

CNT	=	0x00
Z	=	1

INCFSZ — Increment f, Skip if 0

Syntax:	[*label*] INCFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) + 1 → (destination), skip if result = 0
Status Affected:	None
Encoding:	`00` `1111` `dfff` `ffff`
Description:	The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'. If the result is 1, the next instruction is executed. If the result is 0, a NOP is executed instead making it a 2TCY instruction.
Words:	1
Cycles:	1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE     INCFSZ    CNT, 1
         GOTO      LOOP
CONTINUE •
         •
         •
```

Before Instruction
- PC = address HERE

After Instruction
- CNT = CNT + 1
- if CNT= 0,
- PC = address CONTINUE
- if CNT≠ 0,
- PC = address HERE +1

IORLW — Inclusive OR Literal with W

Syntax:	[*label*] IORLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .OR. k → (W)
Status Affected:	Z
Encoding:	`11` `1000` `kkkk` `kkkk`
Description:	The contents of the W register is OR'ed with the eight bit literal 'k'. The result is placed in the W register.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example

```
IORLW   0x35
```

Before Instruction
- W = 0x9A

After Instruction
- W = 0xBF
- Z = 1

IORWF — Inclusive OR W with f

Syntax:	[*label*] IORWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) .OR. (f) → (destination)
Status Affected:	$\overline{Z}$
Encoding:	`00` `0100` `dfff` `ffff`
Description:	Inclusive OR the W register with register 'f'. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example

```
IORWF        RESULT, 0
```

Before Instruction
- RESULT = 0x13
- W = 0x91

After Instruction
- RESULT = 0x13
- W = 0x93
- Z = 1

MOVF — Move f

Syntax:	[*label*] MOVF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) → (destination)
Status Affected:	Z
Encoding:	`00` `1000` `dfff` `ffff`
Description:	The contents of register f is moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register since status flag Z is affected.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example

```
MOVF    FSR,  0
```

After Instruction
- W = value in FSR register
- Z = 1

MOVLW — Move Literal to W

Syntax:	[*label*] MOVLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → (W)
Status Affected:	None
Encoding:	`11` `00xx` `kkkk` `kkkk`
Description:	The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example

```
MOVLW   0x5A
```

After Instruction
- W = 0x5A

MOVWF — Move W to f

Syntax:	[*label*] MOVWF f
Operands:	0 ≤ f ≤ 127
Operation:	(W) → (f)
Status Affected:	None
Encoding:	`00` `0000` `1fff` `ffff`
Description:	Move data from W register to register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example

```
MOVWF    OPTION_REG
```

Before Instruction
- OPTION = 0xFF
- W = 0x4F

After Instruction
- OPTION = 0x4F
- W = 0x4F

NOP	No Operation
Syntax:	[*label*] NOP
Operands:	None
Operation:	No operation
Status Affected:	None
Encoding:	`00` `0000` `0xx0` `0000`
Description:	No operation.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No-Operation	No-Operation	No-Operation

Example `NOP`

OPTION	Load Option Register
Syntax:	[*label*] OPTION
Operands:	None
Operation:	(W) → OPTION
Status Affected:	None
Encoding:	`00` `0000` `0110` `0010`
Description:	The contents of the W register are loaded in the OPTION register. This instruction is supported for code compatibility with PIC16C5X products. Since OPTION is a readable/writable register, the user can directly address it.
Words:	1
Cycles:	1
Example	

To maintain upward compatibility with future PIC16CXX products, do not use this instruction.

RETFIE	Return from Interrupt
Syntax:	[*label*] RETFIE
Operands:	None
Operation:	TOS → PC, 1 → GIE
Status Affected:	None
Encoding:	`00` `0000` `0000` `1001`
Description:	Return from Interrupt. Stack is POPed and Top of Stack (TOS) is loaded in the PC. Interrupts are enabled by setting Global Interrupt Enable bit, GIE (INTCON<7>). This is a two cycle instruction.
Words:	1
Cycles:	2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	No-Operation	Set the GIE bit	Pop from the Stack
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example `RETFIE`

After Interrupt

PC = TOS

GIE = 1

RETLW	Return with Literal in W
Syntax:	[*label*] RETLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → (W); TOS → PC
Status Affected:	None
Encoding:	`11` `01xx` `kkkk` `kkkk`
Description:	The W register is loaded with the eight bit literal 'k'. The program counter is loaded from the top of the stack (the return address). This is a two cycle instruction.
Words:	1
Cycles:	2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	Read literal 'k'	No-Operation	Write to W, Pop from the Stack
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example

```
      CALL TABLE ;W contains table
                 ;offset value
      •          ;W now has table value
      •
      •
TABLE ADDWF PC   ;W = offset
      RETLW k1   ;Begin table
      RETLW k2   ;
      •
      •
      •
      RETLW kn   ; End of table
```

Before Instruction

W = 0x07

After Instruction

W = value of k8

RETURN	Return from Subroutine
Syntax:	[*label*] RETURN
Operands:	None
Operation:	TOS → PC
Status Affected:	None
Encoding:	`00` `0000` `0000` `1000`
Description:	Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two cycle instruction.
Words:	1
Cycles:	2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	No-Operation	No-Operation	Pop from the Stack
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example `RETURN`

After Interrupt

PC = TOS

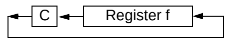
RLF **Rotate Left f through Carry**

Syntax:	[*label*] RLF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C

Encoding:

00	1101	dfff	ffff

Description: The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is stored back in register 'f'.



Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example `RLF REG1,0`

Before Instruction
```
REG1  =  1110 0110
C     =  0
```
After Instruction
```
REG1  =  1110 0110
W     =  1100 1100
C     =  1
```

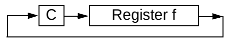
RRF **Rotate Right f through Carry**

Syntax:	[*label*] RRF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C

Encoding:

00	1100	dfff	ffff

Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.



Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example `RRF REG1,0`

Before Instruction
```
REG1  =  1110 0110
C     =  0
```
After Instruction
```
REG1  =  1110 0110
W     =  0111 0011
C     =  0
```

SLEEP

Syntax:	[*label*] SLEEP
Operands:	None
Operation:	00h → WDT, 0 → WDT prescaler, 1 → $\overline{TO}$, 0 → $\overline{PD}$
Status Affected:	$\overline{TO}$, $\overline{PD}$

Encoding:

00	0000	0110	0011

Description: The power-down status bit, $\overline{PD}$ is cleared. Time-out status bit, $\overline{TO}$ is set. Watchdog Timer and its prescaler are cleared.
The processor is put into SLEEP mode with the oscillator stopped. See Section 14.8 for more details.

Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No-Operation	No-Operation	Go to Sleep

Example: `SLEEP`

SUBLW **Subtract W from Literal**

Syntax:	[*label*] SUBLW k
Operands:	$0 \le k \le 255$
Operation:	k - (W) → (W)
Status Affected:	C, DC, Z

Encoding:

11	110x	kkkk	kkkk

Description: The W register is subtracted (2's complement method) from the eight bit literal 'k'. The result is placed in the W register.

Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example 1: `SUBLW 0x02`

Before Instruction
```
W  =  1
C  =  ?
Z  =  ?
```
After Instruction
```
W  =  1
C  =  1; result is positive
Z  =  0
```

Example 2: Before Instruction
```
W  =  2
C  =  ?
Z  =  ?
```
After Instruction
```
W  =  0
C  =  1;  result is zero
Z  =  1
```

Example 3: Before Instruction
```
W  =  3
C  =  ?
Z  =  ?
```
After Instruction
```
W  =  0xFF
C  =  0; result is negative
Z  =  0
```

SUBWF **Subtract W from f**

Syntax:	[*label*] SUBWF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	(f) - (W) $\rightarrow$ (destination)
Status Affected:	C, DC, Z
Encoding:	`00` `0010` `dfff` `ffff`
Description:	Subtract (2's complement method) W register from register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1: `SUBWF REG1,1`

Before Instruction

```
REG1   =   3
W      =   2
C      =   ?
Z      =   ?
```

After Instruction

```
REG1   =   1
W      =   2
C      =   1; result is positive
Z      =   0
```

Example 2: Before Instruction

```
REG1   =   2
W      =   2
C      =   ?
Z      =   ?
```

After Instruction

```
REG1   =   0
W      =   2
C      =   1; result is zero
Z      =   1
```

Example 3: Before Instruction

```
REG1   =   1
W      =   2
C      =   ?
Z      =   ?
```

After Instruction

```
REG1   =   0xFF
W      =   2
C      =   0; result is negative
Z      =   0
```

SWAPF **Swap Nibbles in f**

Syntax:	[*label*] SWAPF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	(f<3:0>) $\rightarrow$ (destination<7:4>), (f<7:4>) $\rightarrow$ (destination<3:0>)
Status Affected:	None
Encoding:	`00` `1110` `dfff` `ffff`
Description:	The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0 the result is placed in W register. If 'd' is 1 the result is placed in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example `SWAPF REG, 0`

Before Instruction

```
REG1   =   0xA5
```

After Instruction

```
REG1   =   0xA5
W      =   0x5A
```

TRIS **Load TRIS Register**

Syntax:	[*label*] TRIS f
Operands:	$5 \leq f \leq 7$
Operation:	(W) $\rightarrow$ TRIS register f;
Status Affected:	None
Encoding:	`00` `0000` `0110` `0fff`
Description:	The instruction is supported for code compatibility with the PIC16C5X products. Since TRIS registers are readable and writable, the user can directly address them.
Words:	1
Cycles:	1
Example	

To maintain upward compatibility with future PIC16CXX products, do not use this instruction.

XORLW **Exclusive OR Literal with W**

Syntax:	[*label*] XORLW k
Operands:	$0 \leq k \leq 255$
Operation:	(W) .XOR. k $\rightarrow$ (W)
Status Affected:	Z
Encoding:	`11` `1010` `kkkk` `kkkk`
Description:	The contents of the W register are XOR'ed with the eight bit literal 'k'. The result is placed in the W register.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example: `XORLW 0xAF`

Before Instruction

```
W   =   0xB5
```

After Instruction

```
W   =   0x1A
```

XORWF **Exclusive OR W with f**

Syntax:	[*label*] XORWF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	(W) .XOR. (f) $\rightarrow$ (destination)
Status Affected:	Z
Encoding:	`00` `0110` `dfff` `ffff`
Description:	Exclusive OR the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example `XORWF REG 1`

Before Instruction

```
REG   =   0xAF
W     =   0xB5
```

After Instruction

```
REG   =   0x1A
W     =   0xB5
```