Section 29. Instruction Set

HIGHLIGHTS

This section of the manual contains the following major topics:

29.1 Introduction

Each midrange 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 midrange Instruction Set Summary in Table 29-1 lists the instructions recognized by the MPASM assembler. The instruction set is highly orthogonal and is grouped into three basic categories:

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

Table 29-2 gives 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.

All instructions are executed in one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In these cases, the execution takes two instruction cycles with the second cycle executed as an 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 29-1: Midrange Instruction Set

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word MSb			LSb	Status Affected	Notes
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	$\overline{TO}$,$\overline{PD}$	
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	$\overline{TO}$,$\overline{PD}$	
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.

29.2 Instruction Formats

Figure 29-1 shows the three general formats that the instructions can have. As can be seen from the general format of the instructions, the opcode portion of the instruction word varies from 3-bits to 6-bits of information. This is what allows the midrange instruction set to have 35 instructions.

Note 1: Any unused opcode is Reserved. Use of any reserved opcode may cause unexpected operation.

Note 2: To maintain upward compatibility with future midrange products, do not use the `OPTION` and `TRIS` instructions.

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

0xhh

where h signifies a hexadecimal digit.

To represent a binary number:

`00000100b`

where b is a binary string identifier.

Figure 29-1: General Format for Instructions

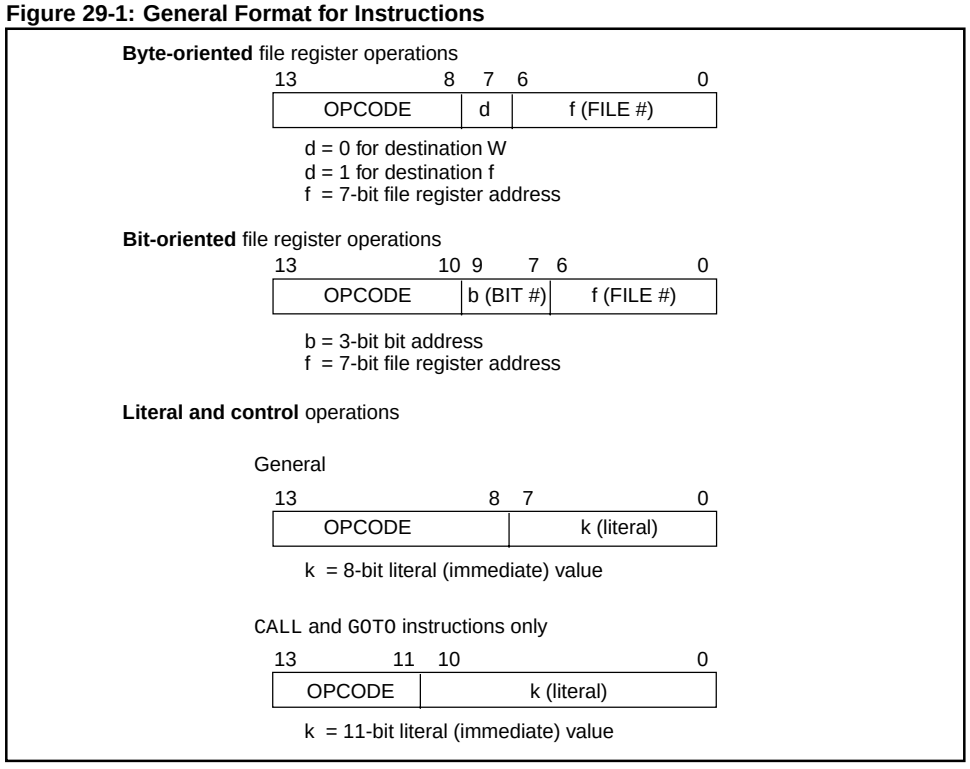


Table 29-2: Instruction Description Conventions

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register (0 to 7)
k	Literal field, constant data or label (may be either an 8-bit or an 11-bit value)
x	Don't care (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.
dest	Destination either the W register or the specified register file location
label	Label name
TOS	Top of Stack
PC	Program Counter
PCLATH	Program Counter High Latch
GIE	Global Interrupt Enable bit
WDT	Watchdog Timer
$\overline{TO}$	Time-out bit
$\overline{PD}$	Power-down bit
[ ]	Optional
( )	Contents
$\rightarrow$	Assigned to
< >	Register bit field
$\in$	In the set of
italics	User defined term (font is courier)

29.3 Special Function Registers as Source/Destination

The Section 29. Instruction Set's orthogonal instruction set allows read and write of all file registers, including special function registers. Some special situations the user should be aware of are explained in the following subsections:

29.3.1 STATUS Register as Destination

If an instruction writes to the STATUS register, the Z, C, DC and OV bits may be set or cleared as a result of the instruction and overwrite the original data bits written. For example, executing `CLRF STATUS` will clear register STATUS, and then set the Z bit leaving `0000 0100`b in the register.

29.3.2 PCL as Source or Destination

Read, write or read-modify-write on PCL may have the following results:

Read PC:	PCL → dest; PCLATH does not change;
Write PCL:	PCLATH → PCH; 8-bit destination value → PCL
Read-Modify-Write:	PCL→ ALU operand PCLATH → PCH; 8-bit result → PCL

Where PCH = program counter high byte (not an addressable register), PCLATH = Program counter high holding latch, dest = destination, W register or register file f.

29.3.3 Bit Manipulation

All bit manipulation instructions will first read the entire register, operate on the selected bit and then write the result back (read-modify-write (R-M-W)) the specified register. The user should keep this in mind when operating on some special function registers, such as ports.

Note:	Status bits that are manipulated by the device (including the interrupt flag bits) are set or cleared in the Q1 cycle. So there is no issue with executing R-M-W instructions on registers which contain these bits.

29.4 Q Cycle Activity

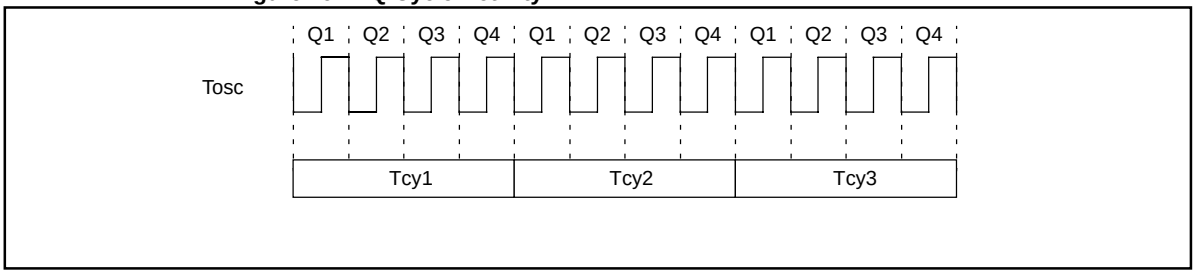
Each instruction cycle (Tcy) is comprised of four Q cycles (Q1-Q4). The Q cycle is the same as the device oscillator cycle (Tosc). The Q cycles provide the timing/designation for the Decode, Read, Process Data, Write etc., of each instruction cycle. The following diagram shows the relationship of the Q cycles to the instruction cycle.

The four Q cycles that make up an instruction cycle (Tcy) can be generalized as:

Q1: Instruction Decode Cycle or forced No Operation
Q2: Instruction Read Cycle or No Operation
Q3: Process the Data
Q4: Instruction Write Cycle or No Operation

Each instruction will show the detailed Q cycle operation for the instruction.

Figure 29-2: Q Cycle Activity



29.5 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 register

Example1

```
ADDLW   0x15
```

Before Instruction
W = 0x10
After Instruction
W = 0x25

Example 2

```
ADDLW   MYREG
```

Before Instruction
W = 0x10
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x47

Example 3

```
ADDLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0x10
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0xA3

Example 4

```
ADDLW   MYREG
```

Before Instruction
W = 0x10
Address of PCL † = 0x02
† PCL is the symbol for the Program Counter low byte location
After Instruction
W = 0x12

ADDWF Add W and f

Syntax: [*label*] ADDWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W) + (f) → 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 1

```
ADDWF   FSR, 0
```

Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0xD9
FSR = 0xC2

Example 2

```
ADDWF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x37

Example 3

```
ADDWF   PCL
```

Case 1: Before Instruction
W = 0x10
PCL = 0x37
C = x
After Instruction
PCL = 0x47
C = 0

Case 2: Before Instruction
W = 0x10
PCL = 0xF7
PCH = 0x08
C = x
After Instruction
PCL = 0x07
PCH = 0x08
C = 1

ANDLW — And Literal with W

Syntax: [*label*] ANDLW k

Operands: $0 \le k \le 255$

Operation: (W).AND. (k) → 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 register

Example 1

```
ANDLW   0x5F
```

Before Instruction
W = 0xA3

After Instruction
W = 0x03

```
; 0101 1111  (0x5F)
; 1010 0011  (0xA3)
;----------  ------
; 0000 0011  (0x03)
```

Example 2

```
ANDLW   MYREG
```

Before Instruction
W = 0xA3
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x23

Example 3

```
ANDLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0xA3
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0x83

ANDWF AND W with f

Syntax: [*label*] ANDWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W).AND. (f) → 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 1

```
ANDWF   FSR, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0x17
FSR = 0x02

```
; 0001 0111  (0x17)
; 1100 0010  (0xC2)
;----------  ------
; 0000 0010  (0x02)
```

Example 2

```
ANDWF   FSR, 0
```

Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0x02
FSR = 0xC2

```
; 0001 0111  (0x17)
; 1100 0010  (0xC2)
;----------  ------
; 0000 0010  (0x02)
```

Example 3

```
ANDWF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x5A
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x15

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 1

```
BCF     FLAG_REG, 7
```

Before Instruction
FLAG_REG = 0xC7 ; **1**100 0111

After Instruction
FLAG_REG = 0x47 ; **0**100 0111

Example 2

```
BCF     INDF, 3
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x2F

After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x27

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 1

```
BSF     FLAG_REG, 7
```

Before Instruction
FLAG_REG =0x0A ; **0**000 1010

After Instruction
FLAG_REG =0x8A ; **1**000 1010

Example 2

```
BSF     INDF, 3
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20

After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x28

BTFSC Bit Test, Skip if Clear

Syntax: [*label*] BTFSC f,b

Operands: 0 ≤ f ≤ 127
0 ≤ b ≤ 7

Operation: skip if (f<b>) = 0

Status Affected: None

Encoding:

01	10bb	bfff	ffff

Description: If bit 'b' in register 'f' is '0' then the next instruction is skipped.
If bit 'b' is '0' then the next instruction (fetched during the current instruction execution) is discarded, and a NOP is executed instead, making this a 2 cycle 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 1

```
HERE    BTFSC  FLAG, 4
FALSE   GOTO   PROCESS_CODE
TRUE    •
        •
        •
```

Case 1: Before Instruction
PC = addressHERE
FLAG= xxx0 xxxx
After Instruction
Since FLAG<4>= 0,
PC = addressTRUE

Case 2: Before Instruction
PC = addressHERE
FLAG= xxx1 xxxx
After Instruction
Since FLAG<4>=1,
PC = addressFALSE

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 '1' then the next instruction is skipped.
If bit 'b' is '1', then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle 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 1

```
HERE    BTFSS  FLAG, 4
FALSE   GOTO   PROCESS_CODE
TRUE    •
        •
        •
```

Case 1: Before Instruction
PC = address HERE
FLAG= xxx0 xxxx
After Instruction
Since FLAG<4>= 0,
PC = address FALSE

Case 2: Before Instruction
PC = address HERE
FLAG= xxx1 xxxx
After Instruction
Since FLAG<4>=1,
PC = address TRUE

CALL — Call Subroutine

Syntax: [*label*] CALL k

Operands: $0 \le k \le 2047$

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, the 13-bit 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<4:3>. CALL is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    CALL   THERE
```

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

CLRF Clear f

Syntax: [*label*] CLRF f

Operands: $0 \le f \le 127$

Operation: $00h \rightarrow f$
$1 \rightarrow Z$

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 1

```
CLRF    FLAG_REG
```

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

Example 2

```
CLRF    INDF
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR)=0xAA
After Instruction
FSR = 0xC2
Contents of Address (FSR)=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	Read register 'f'	Process data	Write register 'W'

Example 1

```
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 count,
1 → $\overline{TO}$
1 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

00	0000	0110	0100

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

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	Clear WDT Counter

Example 1

```
CLRWDT
```

Before Instruction
WDT counter= x
WDT prescaler =1:128
After Instruction
WDT counter=0x00
WDT prescaler count=0
$\overline{TO}$ = 1
$\overline{PD}$ = 1
WDT prescaler =1:128

Note: The CLRWDT instruction does not affect the assignment of the WDT prescaler.

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 1's 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 1

```
COMF    REG1, 0
```

Before Instruction
REG1= 0x13
After Instruction
REG1= 0x13
W = 0xEC

Example 2

```
COMF    INDF, 1
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR)=0xAA
After Instruction
FSR = 0xC2
Contents of Address (FSR)=0x55

Example 3

```
COMF    REG1, 1
```

Before Instruction
REG1= 0xFF
After Instruction
REG1= 0x00
Z = 1

DECF Decrement f

Syntax: [*label*] DECF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) - 1 → 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 1

```
DECF    CNT, 1
```

Before Instruction
CNT = 0x01
Z = 0
After Instruction
CNT = 0x00
Z = 1

Example 2

```
DECF    INDF, 1
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x01
Z = 0
After Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3

```
DECF    CNT, 0
```

Before Instruction
CNT = 0x10
W = x
Z = 0
After Instruction
CNT = 0x10
W = 0x0F
Z = 0

DECFSZ

Decrement f, Skip if 0

Syntax: [*label*] DECFSZ f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) - 1 → 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 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle 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 •
         •
         •
```

Case 1: Before Instruction
PC = address HERE
CNT = 0x01
After Instruction
CNT = 0x00
PC = address CONTINUE

Case 2: Before Instruction
PC = address HERE
CNT = 0x02
After Instruction
CNT = 0x01
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:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
GOTO THERE
```

After Instruction

```
PC =AddressTHERE
```

INCF Increment f

Syntax: [*label*] INCF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) + 1 → 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 1
```
INCF    CNT, 1
```
Before Instruction
CNT = 0xFF
Z = 0
After Instruction
CNT = 0x00
Z = 1

Example 2
```
INCF    INDF, 1
```
Before Instruction
FSR = 0xC2
Contents of Address (FSR) = 0xFF
Z = 0
After Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3
```
INCF    CNT, 0
```
Before Instruction
CNT = 0x10
W = x
Z = 0
After Instruction
CNT = 0x10
W = 0x11
Z = 0

INCFSZ — Increment f, Skip if 0

Syntax: [*label*] INCFSZ f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

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 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle 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 •
         •
         •
```

Case 1: Before Instruction
PC = address HERE
CNT = 0xFF
After Instruction
CNT = 0x00
PC = address CONTINUE

Case 2: Before Instruction
PC = address HERE
CNT = 0x00
After Instruction
CNT = 0x01
PC = address HERE + 1

IORLW Inclusive OR Literal with W

Syntax: [*label*] IORLW k

Operands: $0 \le k \le 255$

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 register

Example 1

```
IORLW   0x35
```

Before Instruction
W = 0x9A
After Instruction
W = 0xBF
Z = 0

Example 2

```
IORLW   MYREG
```

Before Instruction
W = 0x9A
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x9F
Z = 0

Example 3

```
IORLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0x9A
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0x9B
Z = 0

Example 4

```
IORLW   0x00
```

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

IORWF — Inclusive OR W with f

Syntax: [*label*] IORWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

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 1

```
IORWF   RESULT, 0
```

Before Instruction
RESULT=0x13
W = 0x91
After Instruction
RESULT=0x13
W = 0x93
Z = 0

Example 2

```
IORWF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x30
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x37
Z = 0

Example 3

```
IORWF   RESULT, 1
```

Case 1: Before Instruction
RESULT=0x13
W = 0x91
After Instruction
RESULT=0x93
W = 0x91
Z = 0

Case 2: Before Instruction
RESULT=0x00
W = 0x00
After Instruction
RESULT=0x00
W = 0x00
Z = 1

MOVLW Move Literal to W

Syntax: [*label*] MOVLW k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow W$

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 register

Example 1

```
MOVLW   0x5A
```

After Instruction
W = 0x5A

Example 2

```
MOVLW   MYREG
```

Before Instruction
W = 0x10
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x37

Example 3

```
MOVLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0x10
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0x93

MOVF Move f

Syntax: [*label*] MOVF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) → destination

Status Affected: Z

Encoding:

00	1000	dfff	ffff

Description: The contents of register 'f' is moved to a destination dependent 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 1
```
MOVF    FSR, 0
```
Before Instruction
W = 0x00
FSR = 0xC2
After Instruction
W = 0xC2
Z = 0

Example 2
```
MOVF    INDF, 0
```
Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3
```
MOVF    FSR, 1
```
Case 1: Before Instruction
FSR = 0x43
After Instruction
FSR = 0x43
Z = 0

Case 2: Before Instruction
FSR = 0x00
After Instruction
FSR = 0x00
Z = 1

MOVWF Move W to f

Syntax: [*label*] MOVWF f

Operands: $0 \le f \le 127$

Operation: $(W) \rightarrow f$

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 1

```
MOVWF   OPTION_REG
```

Before Instruction
OPTION_REG=0xFF
W = 0x4F
After Instruction
OPTION_REG=0x4F
W = 0x4F

Example 2

```
MOVWF   INDF
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x17

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 :

```
HERE    NOP
```

Before Instruction
PC = address HERE
After Instruction
PC = address HERE + 1

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

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. The 13-bit address at the Top of Stack (TOS) is loaded in the PC. The Global Interrupt Enable bit, GIE (INTCON<7>), is automatically set, enabling Interrupts. This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
RETFIE
```

After Instruction

```
PC  =  TOS
GIE =  1
```

RETLW — Return with Literal in W

Syntax: [*label*] RETLW k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow W$;
$TOS \rightarrow PC$

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 13-bit address at the Top of Stack (the return address). This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

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

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE    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
PC = TOS = Address `Here + 1`

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:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE    RETURN
```

After Instruction

PC = TOS

RLF — Rotate Left f through Carry

Syntax: [*label*] RLF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

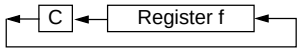
Operation: See description below

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 1

```
RLF     REG1,0
```

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

Example 2

```
RLF     INDF, 1
```

Case 1: Before Instruction
```
W   =  xxxx xxxx
FSR =  0xC2
Contents of Address (FSR) = 0011 1010
C   =  1
```
After Instruction
```
W   =  0x17
FSR =  0xC2
Contents of Address (FSR) = 0111 0101
C   =  0
```

Case 2: Before Instruction
```
W   =  xxxx xxxx
FSR =  0xC2
Contents of Address (FSR) = 1011 1001
C   =  0
```
After Instruction
```
W   =  0x17
FSR =  0xC2
Contents of Address (FSR) = 0111 0010
C   =  1
```

RRF — Rotate Right f through Carry

Syntax: [*label*] RRF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

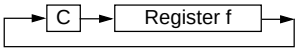
Operation: See description below

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 1

```
RRF          REG1,0
```

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

Example 2

```
RRF     INDF, 1
```

Case 1: Before Instruction
W = xxxx xxxx
FSR = 0xC2
Contents of Address (FSR) = 0011 1010
C = 1
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 1001 1101
C = 0

Case 2: Before Instruction
W = xxxx xxxx
FSR = 0xC2
Contents of Address (FSR) = 0011 1001
C = 0
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0001 1100
C = 1

SLEEP

Syntax: [*label*] SLEEP

Operands: None

Operation: 00h → WDT,
0 → WDT prescaler count,
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 count are cleared.
The processor is put into SLEEP mode with the oscillator stopped.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	Go to sleep

Example: SLEEP

Note: The SLEEP instruction does not affect the assignment of the WDT prescaler

SUBLW Subtract W from Literal

Syntax: [*label*] SUBLW k

Operands: $0 \le k \le 255$

Operation: $k - (W) \rightarrow W$

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 register

Example 1:
```
SUBLW   0x02
```
Case 1: Before Instruction
```
W = 0x01
C = x
Z = x
```
After Instruction
```
W = 0x01
C = 1       ; result is positive
Z = 0
```
Case 2: Before Instruction
```
W = 0x02
C = x
Z = x
```
After Instruction
```
W = 0x00
C = 1       ; result is zero
Z = 1
```
Case 3: Before Instruction
```
W = 0x03
C = x
Z = x
```
After Instruction
```
W = 0xFF
C = 0       ; result is negative
Z = 0
```

Example 2
```
SUBLW   MYREG
```
Before Instruction
```
W = 0x10
```
Address of MYREG † = 0x37

† MYREG is a symbol for a data memory location

After Instruction
```
W = 0x27
C = 1       ; result is positive
```

SUBWF — Subtract W from f

Syntax: [*label*] SUBWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) - (W) → 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
```
Case 1: Before Instruction
```
REG1= 3
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 1
W   = 2
C   = 1          ; result is positive
Z   = 0
```
Case 2: Before Instruction
```
REG1= 2
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 0
W   = 2
C   = 1          ; result is zero
Z   = 1
```
Case 3: Before Instruction
```
REG1= 1
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 0xFF
W   = 2
C   = 0          ; result is negative
Z   = 0
```

SWAPF — Swap Nibbles in f

Syntax: [*label*] SWAPF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f<3:0>) → destination<7:4>,
(f<7:4>) → 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 1

```
SWAPF   REG, 0
```

Before Instruction

REG1= 0xA5

After Instruction

REG1= 0xA5
W = 0x5A

Example 2

```
SWAPF   INDF, 1
```

Before Instruction

W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20

After Instruction

W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x02

Example 3

```
SWAPF   REG, 1
```

Before Instruction

REG1= 0xA5

After Instruction

REG1= 0x5A

TRIS — Load TRIS Register

Syntax:	[*label*] TRIS f
Operands:	$5 \le f \le 7$
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: $0 \le k \le 255$

Operation: (W).XOR. k → 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 register

Example 1

```
XORLW   0xAF                ; 1010 1111   (0xAF)
Before Instruction          ; 1011 0101   (0xB5)
    W   =   0xB5            ; ---------   ------
After Instruction           ; 0001 1010   (0x1A)
    W   =   0x1A
    Z   =   0
```

Example 2

```
XORLW   MYREG
```

Before Instruction
W = 0xAF
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x18
Z = 0

Example 3

```
XORLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0xAF
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0x3C
Z = 0

XORWF — Exclusive OR W with f

Syntax: [*label*] XORWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W).XOR. (f) → 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 1

```
XORWF    REG, 1                 ; 1010 1111   (0xAF)
Before Instruction              ; 1011 0101   (0xB5)
        REG= 0xAF               ; ---------   ------
        W  = 0xB5               ; 0001 1010   (0x1A)
After Instruction
        REG= 0x1A
        W  = 0xB5
```

Example 2

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

Example 3

```
XORWF   INDF, 1
```

Before Instruction
W = 0xB5
FSR = 0xC2
Contents of Address (FSR) = 0xAF

After Instruction
W = 0xB5
FSR = 0xC2
Contents of Address (FSR) = 0x1A

29.6 Design Tips

Question 1: ***How can I modify the value of W directly? I want to decrement W.***

Answer 1:

There are a few possibilities, two are:

1. For the midrange devices, there are several instructions that work with a literal and W. For instance, if it were desired to decrement W, this can be done with an `ADDLW 0xFF`. (the 0x prefix denotes hex to the assembler)
2. Notice that all of the instructions can modify a value right where it sits in the file register. This means you can decrement it right where it is. You do not even need to move it to W. If you want to decrement it AND move it somewhere else, then you make W the DESTINATION of the decrement (`DECF register,W`) then put it where you want it. It is the same number of instructions as a straight move, but it gets decremented along the way.

Question 2: ***Is there any danger in using the `TRIS` instruction for the PIC16CXXX since there is a warning in the Data book suggesting it not be used?***

Answer 2:

For code compatibility and upgrades to later parts, the use of the TRIS instruction is not recommended. You should note the `TRIS` instruction is limited to ports A, B and C. Future devices may not support these instructions.

Question 3: ***Do I have to switch to Bank1 of data memory before using the `TRIS` instruction (for parts with TRIS registers in the memory map)?***

Answer 3:

No. The `TRIS` instruction is Bank independent. Again the use of the `TRIS` instruction is not recommended.

Question 4: ***I have seen references to "Read-Modify-Write" instructions in your data sheet, but I do not know what that is. Can you explain what it is and why I need to know this?***

Answer 4:

An easy example of a Read-Modify-Write (R-M-W) instruction is the bit clear instruction `BCF`. You might think that the processor just clears the bit, which on a port output pin would clear the pin. What actually happens is the whole port (or register) is first read, THEN the bit is cleared, then the new modified value is written back to the port (or register). Actually, any instruction that depends on a value currently in the register is going to be a Read-Modify-Write instruction. This includes `ADDWF, SUBWF, BCF, BSF, INCF, XORWF`, etc... Instructions that do not depend on the current register value, like `MOVWF, CLRF`, and so on are not R-M-W instructions.

One situation where you would want to consider the affects of a R-M-W instruction is a port that is continuously changed from input to output and back. For example, say you have TRISB set to all outputs, and write all ones to the PORTB register, all of the PORTB pins will go high. Now, say you turn pin RB3 into an input, which happens to go low. A `BCF PORTB,6` is then executed to drive pin RB6 low. If you then turn RB3 back into an output, it will now drive low, even though the last value you put there was a one. What happened was that the `BCF` of the other pin (RB6) caused the whole port to be read, including the zero on RB3 when it was an input. Then, bit 6 was changed as requested, but since RB3 was read as a zero, zero will also be placed back into that port latch, overwriting the one that was there before. When the pin is turned back into an output, the new value was reflected.

Question 5: ***When I perform a BCF other pins get cleared in the port. Why?***

Answer 5:

There are a few possibilities, two are:

1. Another case where a R-M-W instruction may seem to change other pin values unexpectedly can be illustrated as follows: Suppose you make PORTC all outputs and drive the pins low. On each of the port pins is an LED connected to ground, such that a high output lights it. Across each LED is a 100 µF capacitor. Let's also suppose that the processor is running very fast, say 20 MHz. Now if you go down the port setting each pin in order; `BSF PORTC,0` then BSF PORTC,1 then `BSF PORTC,2` and so on, you may see that only the last pin was set, and only the last LED actually turns on. This is because the capacitors take a while to charge. As each pin was set, the pin before it was not charged yet and so was read as a zero. This zero is written back out to the port latch (R-M-W, remember) which clears the bit you just tried to set the instruction before. This is usually only a concern at high speeds and for successive port operations, but it can happen so take it into consideration.
2. If this is on a PIC16C7X device, you may not have configured the I/O pins properly in the ADCON1 register. If a pin is configured for analog input, any read of that pin will read a zero, regardless of the voltage on the pin. This is an exception to the normal rule that the pin state is always read. You can still configure an analog pin as an output in the TRIS register, and drive the pin high or low by writing to it, but you will always read a zero. Therefore if you execute a Read-Modify-Write instruction (see previous question) all analog pins are read as zero, and those not directly modified by the instruction will be written back to the port latch as zero. A pin configured as analog is expected to have values that may be neither high nor low to a digital pin, or floating. Floating inputs on digital pins are a no-no, and can lead to high current draw in the input buffer, so the input buffer is disabled.

29.7 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Mid-Range MCU family (that is they may be written for the Base-Line, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the instruction set are:

Currently No related Application Notes

29.8 Revision History

Revision A

This is the initial released revision of the Instruction Set description.