



# Instruction Set in PIC16Cxx MC Family

- Complete set: 35 instructions.
- MC Architecture: RISC microcontroller.
- Instruction Types:
  1. **Data Processing Operations:**
    - Copy data between registers.
    - Manipulate data in a single register.
    - Arithmetic operations.
    - Logic operations.
  2. **Program Sequence Control Operations:**
    - Unconditional Jump.
    - Conditional Jump.
    - Call.
    - Control.



## Word list

**f** any memory location in a microcontroller

**W** work register

**b** bit position in 'f' register

**d** destination bit

*label* group of eight characters which marks the beginning of a part of the program

**TOS** top of stack

**[ ]** option

**< >** bit position inside register

Example:

<b>008C</b>	<b>movwf 0C</b>
-------------	-----------------



## Data transfer

Transfer of data in a MC is done between W register and an 'f' register.

Mnemonic	Description	Operation	Flag	Cycle	Notes
<b>Data transfer</b>					
MOVLW k	Move constant to W	$k \rightarrow W$		1	
MOVWF f	Move W to f	$W \rightarrow f$		1	
MOVF f, d	Move f	$f \rightarrow d$	Z	1	1,2
CLRW -	Clear W	$0 \rightarrow W$	Z	1	
CLRF f	Clear f	$0 \rightarrow f$	Z	1	2
SWAPF f, d	Swap nibbles in f	$f(7:4), (3:0) \rightarrow f(3:0), (7:4)$		1	1,2

These instructions provide for:

- a constant being written in W register (MOVLW)
- data to be copied from W register onto RAM.
- data from RAM to be copied onto W register (or on the same RAM location, at which point only the status of Z flag changes).
- Instruction CLRF writes constant 0 in 'f' register,
- Instruction CLRW writes constant 0 in register W.
- SWAPF instruction exchanges places of the 4-bit nibbles field inside a register.





## Arithmetic and logic

PIC like most MCs supports only subtraction and addition.

Flags C, DC and Z are set depending on a result of addition or subtraction.

Logic unit performs AND, OR, EX-OR, complement (COMF) and rotation (RLF & RRF).

Arithmetic and logic						
ADDLW	k	Add constant and W	$W+1 \rightarrow W$	C,DC,Z	1	
ADDWF	f, d	Add W and f	$W+f \rightarrow d$	C,DC,Z	1	1,2
SUBLW	k	Subtract W from constant	$W-k \rightarrow W$	C,DC,Z	1	
SUBWF	f, d	Subtract W from f	$W-f \rightarrow d$	C,DC,Z	1	1,2
ANDLW	k	AND constant with W	$W.AND.k \rightarrow W$	Z	1	
ANDWF	f, d	AND W with f	$W.AND.f \rightarrow d$	Z	1	1,2
IORLW	k	OR constant with W	$W.OR.k \rightarrow W$	Z	1	
IORWF	f, d	OR W with f	$W.OR.f \rightarrow d$	Z	1	1,2
XORLW	k	Exclusive OR constant with W	$W.XOR.k \rightarrow W$	Z	1	1,2
XORWF	f, d	Exclusive OR W with f	$W.XOR.f \rightarrow d$	Z	1	
INCF	f, d	Increment f	$f+1 \rightarrow f$	Z	1	1,2
DECF	f, d	Decrement f	$f-1 \rightarrow f$	Z	1	1,2
RLF	f, d	Rotate Left f through carry		C	1	1,2
RRF	f, d	Rotate Right f through carry		C	1	1,2
COMF	f, d	Complement f	$F \rightarrow d$	Z	1	1,2



## Bit operations

Instructions BCF and BSF do setting or cleaning of one bit anywhere in the memory. The CPU first reads the whole byte, changes one bit in it and then writes in the entire byte at the same place.

### Bit operations

BCF	f, b	Bit Clear f	$0 \rightarrow f(b)$		1	1,2
BSF	f, b	Bit Set f	$1 \rightarrow f(b)$		1	1,2



# Directing a program flow

- Instructions **GOTO**, **CALL** and **RETURN** are executed the same way as on all other microcontrollers, only stack is independent of internal RAM and limited to eight levels.
- '**RETLW k**' instruction is identical with **RETURN** instruction, except that before coming back from a subprogram a constant defined by instruction operand is written in W register.

## Directing a program flow

BTFSC	f, b	Bit Test f, Skip if Clear	jump if f(b)=0		1 (2)	3
BTFSS	f, b	Bit Test f, Skip if Set	jump if f(b)=1		1 (2)	3
DECFSZ	f, d	Decrement f, Skip if 0	f-1 → d, jump if Z=1		1(2)	1,2,3
INCFSZ	f, d	Increment f, Skip if 0	f+1 → d, jump if Z=0		1(2)	1,2,3
GOTO	k	Go to address	W.AND.k → W		2	
CALL	k	Call subroutine	W.AND.f → d		2	
RETURN	-	Return from Subroutine	W.OR.k → W		2	
RETLW	k	Return with constant in W	W.OR.f → d		2	
RETFIE	-	Return from interrupt	W.XOR.k → W		2	



## Look-up tables Design:

- This instruction '**RETLW k**' enables us to design easily the Look-up tables (lists).
- We use them by determining data position on our table adding it to the address at which the table begins, and then we read data from that location (which is usually found in program memory).
- Table can be formed as a subprogram which consists of a series of '**RETLW k**' instructions, where 'k' constants are members of the table.
- We write the position of a member of our table in W register, and using CALL instruction to call a subprogram which creates the table.
- The instruction **ADDWF PCL, f** adds the position of a W register member to the starting address of our table, found in PCL register, and so we get the real data address in program memory.
- When returning from a subprogram we will have in W register the contents of an addressed table member.

```
Main      molov 2
           call Lookup
Lookup     addwf PCL, f
           retlw k
           retlw k1
           retlw k2
           :
           :
           retlw kn
```

### Other instructions

NOP	-	No Operation			1	
CLRWDI	-	Clear Watchdog Timer	$0 \rightarrow \text{WDT}, 1 \rightarrow \text{PD}$	$\overline{\text{TO}}, \overline{\text{PD}}$	1	
SLEEP	-	Go into standby mode	$0 \rightarrow \text{WDT}, 0 \rightarrow \text{PD}$	$\overline{\text{TO}}, \overline{\text{PD}}$	1	



# Instruction Execution Period

All instructions are executed in one cycle except for conditional branch instructions if condition was true, or if the contents of program counter was changed by some instruction. In that case, execution requires two instruction cycles, and the second cycle is executed as NOP (No Operation).

Four oscillator clocks make up one instruction cycle. If we are using an oscillator with 4MHz frequency, the normal time for executing an instruction is 1  $\mu$ s, and in case of conditional branching, execution period is 2  $\mu$ s.





**Syntax:** `[label] MOVLW k`  
**Description:** 8-bit constant **k** is written in **W** register.  
**Operation:**  $k \Rightarrow (W)$   
**Operand:**  $0 \leq k \leq 255$   
**Flag:** -  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `MOVLW 0x5A`

After instruction:  $W=0x5A$

**Example 2** `MOVLW REGISTAR`

Before instruction:  $W=0x10$  and  $REGISTAR=0x40$

After instruction:  $W=0x40$

**Syntax:** `[label] MOVWF f`  
**Description:** Contents of **W** register is copied to **f** register.  
**Operation:**  $W \Rightarrow (f)$   
**Operand:**  $0 \leq f \leq 127$   
**Flag:** -  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `MOVWF OPTION_REG`

Before instruction:  $OPTION\_REG=0x20$   
 $W=0x40$

After instruction:  $OPTION\_REG=0x40$   
 $W=0x40$

**Example 2** `MOVWF INDF`

Before instruction:  $W=0x17$   
 $FSR=0xC2$   
address contents  $0xC2=0x00$

After instruction:  $W=0x17$   
 $FSR=0xC2$   
address contents  $0xC2=0x17$

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

Contents of **f** register is stored in location determined by **d** operand.

If **d=0**, destination is **W** register.

If **d=1**, destination is **f** register itself.

Option **d=1** is used for testing the contents of **f** register because execution of this instruction affects Z flag in STATUS register.

**Operation:** $f \Rightarrow (d)$ **Operand:** $0 \leq f \leq 127$  $d \in [0,1]$ **Flag:****Z****Number of words:****1****Number of cycles:****1****Example 1**    `MOVF FSR, 0`

Before instruction:    `FSR=0xC2`

`W=0x00`

After instruction:    `W=0xC2`

`Z=0`**Example 2**    `MOVF INDF, 0`

Before instruction:    `W=0x17`

`FSR=0xC2``address contents 0xC2=0x00`

After instruction:    `W=0x17`

`FSR=0xC2``address contents 0xC2=0x00``Z=1`

**Syntax:**

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

**Description:**

Contents of **f** register is stored in location determined by **d** operand.

If **d=0**, destination is **W** register.

If **d=1**, destination is **f** register itself.

Option **d=1** is used for testing the contents of **f** register because execution of this instruction affects Z flag in STATUS register.

**Operation:**

$f \Rightarrow (d)$

**Operand:**

$0 \leq f \leq 127$

$d \in [0,1]$

**Flag:**

Z

**Number of words:**

1

**Number of cycles:**

1

**Example 1**    MOVF   FSR, 0

Before instruction:    FSR=0xC2

W=0x00

After instruction:    W=0xC2

Z=0

**Example 2**    MOVF   INDF, 0

Before instruction:    W=0x17

FSR=0xC2

address contents 0xC2=0x00

After instruction:    W=0x17

FSR=0xC2

address contents 0xC2=0x00

Z=1



**Syntax:** `[label] CLRW`  
**Description:** Contents of **W** register evens out to zero, and Z flag in STATUS register is set to one.  
**Operation:**  $0 \Rightarrow (W)$   
**Operand:** -  
**Flag:** Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example** CLRW

Before instruction:  $W=0 \times 55$   
After instruction:  $W=0 \times 00$   
Z=1  
Z=1

**Example 2** CLRF INDF

Before instruction:  $FSR=0 \times C2$   
address contents  $0 \times C2=0 \times 33$   
After instruction:  $FSR=0 \times C2$   
address contents  $0 \times C2=0 \times 00$   
Z=1

**Syntax:** `[label] CRLF f`  
**Description:** Contents of **f** register evens out to zero, and Z flag in status register is set to one.  
**Operation:**  $0 \Rightarrow f$   
**Operand:**  $0 \leq f \leq 127$   
**Flag:** Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** CRLF STATUS

Before instruction:  $STATUS=0 \times C2$   
After instruction:  $STATUS=0 \times 00$   
Z=1

**Example 2** CLRF INDF

Before instruction:  $FSR=0 \times C2$   
address contents  $0 \times C2=0 \times 33$   
After instruction:  $FSR=0 \times C2$   
address contents  $0 \times C2=0 \times 00$   
Z=1



**Syntax:** `[label] SWAPF f, d`  
**Description:** Upper and lower half of **f** register exchange places.  
If **d=0**, result is stored in **W** register.  
If **d=1**, result is stored in **f** register.

**Operation:**  $f<0:3> \Rightarrow d<4:7>, f<4:7> \Rightarrow d<0:3>;$   
**Operand:**  $0 \leq f \leq 127$   
 $d \in [0,1]$   
**Flag:** -  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `SWAP REG, 0`

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

**Example 2** `SWAP REG, 1`

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



**Syntax:** `[label] ADDLW k`  
**Description:** Contents of **W** register is added to 8-bit constant **k** and result is stored in **W** register.  
**Operation:**  $(W) + k \Rightarrow W$   
**Operand:**  $0 \leq k \leq 255$   
**Flag:** C, DC, Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `ADDLW 0x15`

Before instruction:  $W=0x10$   
After instruction:  $W=0x25$

**Example 2** `ADDLW REG`

Before instruction:  $W=0x10$   
register contents  $REG=0x37$   
After instruction:  $W=0x47$

**Syntax:** `[label] ADDWF f, d`  
**Description:** Add contents of register **W** to register **f**.  
If **d=0**, result is stored in **W** register.  
If **d=1**, result is stored in **f** register.  
**Operation:**  $(W) + (f) \Rightarrow d$   
 $d \in [0,1]$   
**Operand:**  $0 \leq f \leq 127$   
**Flag:** C, DC, Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `ADDWF FSR, 0`

Before instruction:  $W=0x17$   
 $FSR=0xC2$   
After instruction:  $W=0xD9$   
 $FSR=0xC2$

**Example 2** `ADDLW INDF, 1`

Before instruction:  $W=0x17$   
 $FSR=0xC0$   
address contents  $0xC2=0x20$   
After instruction:  $W=0x17$   
 $FSR=0xC2$   
address contents  $0xC2=0x37$



**Syntax:** `[label] SUBLW k`  
**Description:** Contents of **W** register is subtracted from **k** constant, and result is stored in **W** register.  
**Operation:**  $k - (W) \Rightarrow W$   
**Operand:**  $0 \leq k \leq 255$   
**Flag:** C, DC, Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `SUBLW 0x03`

Before instruction: `W=0x01, C=x, Z=x`  
 After instruction: `W=0x02, C=1, Z=0`      Result > 0

Before instruction: `W=0x03, C=x, Z=x`  
 After instruction: `W=0x00, C=1, Z=1`      Result = 0

Before instruction: `W=0x04, C=x, Z=x`  
 After instruction: `W=0xFF, C=0, Z=0`      Result < 0

**Example 2** `SUBLW REG`

Before instruction: `W=0x10`  
                           `contents REG=0x37`  
 After instruction: `W=0x27`  
                           `C=1`      Result > 0



**Syntax:** `[label] SUBWF f, d`  
**Description:** Contents of **W** register is subtracted from the contents of **f** register.  
 If **d=0**, result is stored in **W** register.  
 If **d=1**, result is stored in **f** register.  
**Operation:**  $(f) - (W) \Rightarrow d$   
**Operand:**  $0 \leq f \leq 127$   
                    $d \in [0,1]$   
**Flag:** C, DC, Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `SUBWF REG, 1`

Before instruction: `REG=3, W=2, C=x, Z=x`  
 After instruction: `REG=1, W=2, C=1, Z=0`      Result > 0

Before instruction: `REG=2, W=2, C=x, Z=x`  
 After instruction: `REG=0, W=2, C=1, Z=1`      Result = 0

Before instruction: `REG=1, W=2, C=x, Z=x`  
 After instruction: `REG=0xFF, W=2, C=0, Z=0`      Result < 0



**Syntax:** `[label] ANDLW k`  
**Description:** Performs operation logic AND over the contents of **W** register and constant **k**. Result is stored in **W** register.  
**Operation:**  $(W) \text{ .AND. } k \Rightarrow W$   
**Operand:**  $0 \leq k \leq 255$   
**Flag:** Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `ANDLW 0x5F`

Before instruction:	W=0xA3	; 0101 1111 (0x5F)
After instruction:	W=0x03	; 1010 0011 (0xA3)
		-----
		; 0000 0011 (0x03)

**Example 2** `ANDLW REG`

Before instruction:	W=0xA3	; 1010 0011 (0xA3)
	REG=0x37	; 0011 0111 (0x37)
After instruction:	W=0x23	-----
		; 0010 0011 (0x23)

**Syntax:** `[label] ANDWF f, d`  
**Description:** Performs operation of logic AND over the contents of **W** and **f** registers. If **d=0**, result is stored in **W** register. If **d=1**, result is stored in **f** register.  
**Operation:**  $(W) \text{ .AND. } f \Rightarrow d$   
**Operand:**  $0 \leq f \leq 127$   
**Flag:** Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `ANDWF FSR, 1`

Before instruction:	W=0x17, FSR=0xC2	; 0001 0111 (0x17)
After instruction:	W=0x17, FSR=02	; 1100 0010 (0xC2)
		-----
		; 0000 0010 (0x02)

**Example 2** `ANDWF FSR, 0`

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





**Syntax:** `[label] IORLW k`

**Description:** Operation logic OR is performed over the contents of **W** register and over 8-bit constant **k**, and result is stored in **W** register.

**Operation:**  $(W) .OR. (k) \Rightarrow W$

**Operand:**  $0 \leq k \leq 255$

**Flag:** Z

**Number of words:** 1

**Number of cycles:** 1

**Example 1** `IORLW 0x35`

Before instruction: `W=0x9A`

After instruction: `W=0xBF`  
`Z=0`

**Example 2** `IORLW REG`

Before instruction: `W=0x9A`  
`contentst REG=0x37`

After instruction: `W=0x9F`  
`Z=0`

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

**Description:** Operation logic OR is performed over the contents of **W** and **f** registers. If **d=0**, result is stored in **W** register. If **d=1**, result is stored in **f** register.

**Operation:**  $(W) .OR. (f) \Rightarrow d$

**Operand:**  $0 \leq f \leq 127$

$d \in [0,1]$

**Flag:** Z

**Number of words:** 1

**Number of cycles:** 1

**Example 1** `IORWF REG, 0`

Before instruction: `REG=0x13, W=0x91`

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

**Example 2** `IORWF REG, 1`

Before instruction: `REG=0x13, W=0x91`

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



Before instruction:	W=0xAF	; 1010 1111	(0xA3)
	REG=0x37	; 0011 0111	(0x37)
After instruction:	W=0x18	-----	
	Z=0	; 0001 1000	(0x18)

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



**Syntax:** `[label] INCF f, d`  
**Description:** Increments f register by one.  
If **d=0**, result is stored in **W** reg  
If **d=1**, result is stored in **f** reg  
**Operation:**  $(f) + 1 \Rightarrow d$   
**Operand:**  $0 \leq f \leq 127$   
 $d \in [0,1]$   
**Flag:** Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `INCF REG, 1`

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

**Example 2** `INCF REG, 0`

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

**Syntax:** `[label] DECF f, d`  
**Description:** Decrements f register by one.  
If **d=0**, result is stored in **W** reg  
If **d=1**, result is stored in **f** reg  
**Operation:**  $(f) - 1 \Rightarrow d$   
**Operand:**  $0 \leq f \leq 127$   
 $d \in [0,1]$   
**Flag:** Z  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** `DECF REG, 1`

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

**Example 2** `DECF REG, 0`

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



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

**Description:** Contents of **f** register is rotated by one space to the left through **C** flag.  
If **d=0**, result is stored in **W** register.  
If **d=1**, result is stored in **f** register.

**Operation:**  $(f\langle n \rangle) \Rightarrow d\langle n+1 \rangle, f\langle 7 \rangle \Rightarrow C, C \Rightarrow d\langle 0 \rangle;$

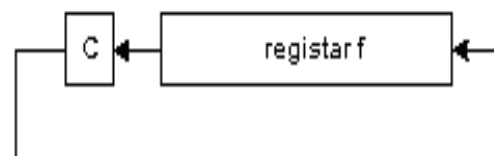
**Operand:**  $0 \leq f \leq 127$

$d \in [0,1]$

**Flag:** **C**

**Number of words:** 1

**Number of cycles:** 1



**Example 1** `RLF REG, 0`

Before instruction: `REG=1110 0110`  
`C=0`

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

**Example 2** `RLF REG, 1`

Before instruction: `REG=1110 0110`  
`C=0`

After instruction: `REG=1100 1100`  
`C=1`

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

**Description:** Contents of **f** register is rotated by one space to the right through **C**

If **d=0**, result is stored in **W** register.

If **d=1**, result is stored in **f** register.

**Operation:**  $(f\langle n \rangle) \Rightarrow d\langle n-1 \rangle, f\langle 0 \rangle \Rightarrow C, C \Rightarrow d\langle 7 \rangle;$

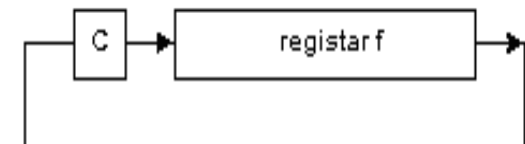
**Operand:**  $0 \leq f \leq 127$

$d \in [0,1]$

**Flag:** **C**

**Number of words:** 1

**Number of cycles:** 1



**Example 1** `RRF REG, 0`

Before instruction: `REG=1110 0110`  
`W=x`  
`C=0`

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

**Example 2** `RRF REG, 1`

Before instruction: `REG=1110 0110`  
`C=0`

After instruction: `REG=0111 0011`  
`C=0`



Before instruction:	FSR=0xC2 address contents (FSR)=0xAA
After instruction:	FSR=0xC2 address contents (FSR)=0x55

Before instruction:	W=0x17
	FSR=0xC2
	address contents (FSR)=0x2F
After instruction:	W=0x17
	FSR=0xC2
	address contents (FSR)=0x27



**Syntax:** `[label] BSF f, b`  
**Description:** Set bit **b** in **f** register.  
**Operation:**  $1 \Rightarrow f \langle b \rangle$   
**Operand:**  $0 \leq f \leq 127$   
 $0 \leq b \leq 7$   
**Flag:** -  
**Number of words:** 1  
**Number of cycles:** 1

**Example 1** BSF REG, 7

Before instruction: REG=0x07 ; 0000 0111 (0x07)  
After instruction: REG=0x17 ; 1000 0111 (0x17)

**Example 2** BCF INDF, 3

Before instruction: W=0x17  
FSR=0xC2  
address contents (FSR)=0x20  
After instruction: W=0x17  
FSR=0xC2  
address contents (FSR)=0x28

**Syntax:** `[label] BTFSC f, b`  
**Description:** If bit **b** in **f** register equals zero, then we skip the next instruction.  
If bit **b** equals zero, during execution of the current instruction, execution of the next one is disabled, and NOP instruction executes instead thus making the current one a two-cycle instruction.  
**Operation:** Skip next instruction if  $(f \langle b \rangle) = 0$   
**Operand:**  $0 \leq f \leq 127$   
 $0 \leq b \leq 7$   
**Flag:** -  
**Number of words:** 1  
**Number of cycles:** 1 or 2 depending on a **b** bit

**Example**

```
LAB_01    BTFSC REG,1      ;Test bit no.1 in REG
LAB_02    .....          ;Skip this line if =0
LAB_03    .....          ;Skip here if =1
```

Before instruction, program counter was at address LAB\_01.

After instruction, if the first bit in REG register was zero, program counter points to address LAB\_03.

If the first bit in REG register was one, program counter points to address LAB\_02.



**Syntax:** [label] BTFSS f, b

**Description:** If bit **b** in **f** register equals one, then skip over the next instruction. If bit **b** equals one, during execution of the current instruction, the next one is disabled, and NOP instruction is executed instead, thus making the current one a two-cycle instruction.

**Operation:** Skip next instruction if (**f**<**b**>)=1

**Operand:**  $0 \leq f \leq 127$   
 $0 \leq b \leq 7$

**Flag:** -

**Number of words:** 1

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

### Example

```
LAB_01    BTFSS REG,1      ;Test bit no.1 in REG
LAB_02    .....          ;Skip this line if =1
LAB_03    .....          ;Skip here if =0
```

Before instruction, program counter was at address LAB\_01

After instruction, if the first bit in REG register was one, program counter points to address LAB\_03.

If the first bit in REG register was zero, program counter points to address LAB\_02.

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

**Description:** Contents of **f** register is incremented by one.

If **d**=0, result is stored in **W** register.

If **d**=1, result is stored in **f** register.

If result =0, the next instruction is executed as NOP making the current one a two-cycle instruction.

**Operation:**  $(f) + 1 \Rightarrow d$

**Operand:**  $0 \leq f \leq 127$   
 $d \in [0,1]$

**Flag:** -

**Number of words:** 1

**Number of cycles:** 1 or 2 depending on a result

### Example

```
LAB_01    INCFSZ REG, 1    ; Increase the contents REG by one.
LAB_02    .....          ; Skip this line if =0
LAB_03    .....          ; Skip here if =0
```

The contents of program counter before instruction, PC=address LAB\_01

The contents of REG after executing an instruction  $REG=REG+1$ , if  $REG=0$ , program counter points to label address LAB\_03. Otherwise, pc points to address of the next instruction or to LAB\_02.



**Syntax:** `[label] DECFSZ f, d`  
**Description:** Contents of **f** register is decr by one.  
 If **d=0**, result is stored in **W** register.  
 If **d=1**, result is stored in **f** register.  
 If result = 0, next instruction is executed as NOP, thus making the current one, a two-cycle instruction.

**Operation:**  $(f) - 1 \Rightarrow d$

**Operand:**  $0 \leq f \leq 127$   
 $d \in [0,1]$

**Flag:** -

**Number of words:** 1

**Number of cycles:** 1 or 2 depending on a result

### Example

```
LAB_01  DECFSZ  CNT, 1 ; Decr the contents REG by one
LAB_02  . . . . . ; Skip this line if = 0
LAB_03  . . . . . ; Skip here if = 1
```

The contents of program counter before instruction,  
 PC=address LAB\_01

The contents of CNT register after executing an instruction  
 CNT=CNT-1, if CNT=0,  
 program counter points to address of label LAB\_03.

Otherwise, program counter points to  
 address of the following instruction, or to LAB\_02.

**Syntax:** `[label] GOTO k`

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

**Operation:**  $k \Rightarrow PC<10:0>, (PCLATH<4:3>) \Rightarrow PC<12:11>$

**Operand:**  $0 \leq k \leq 2048$

**Flag:** -

**Number of words:** 1

**Number of cycles:** 2

### Example

```
LAB_00      GOTO LAB_01      ; Jump to LAB_01
             :
             :
LAB_01      . . . . .
```

Before instruction: PC=address LAB\_00

After instruction: PC=address LAB\_01





**Syntax:** `[label] CALL k`

**Description:** Instruction calls a subprogram. First, return address (PC+1) is stored on stack, then 11-bit direct operand **k**, which contains the subprogram address, is stored in program counter.

**Operation:**  $(PC) + 1 \Rightarrow \text{Top Of Stack (TOS)}$   
 $k \Rightarrow PC\langle 10:0 \rangle, (PCLATH\langle 4:3 \rangle) \Rightarrow PC\langle 12:11 \rangle$

**Operand:**  $0 \leq k \leq 2048$

**Flag:** -

**Number of words:** 1

**Number of cycles:** 2

#### Example

```
LAB_01      CALL LAB_02 ; Call subrutine LAB_02
             :
             :
LAB_02      . . . . .
```

Before instruction: PC=address LAB\_01  
TOS=x

After instruction: PC=address LAB\_02  
TOS=LAB\_01

**Syntax:** `[label] RETURN`

**Description:** Contents from the top of a stack is stored in program counter.

**Operation:** TOS  $\Rightarrow$  program counter PC

**Operand:** -

**Flag:** -

**Number of words:** 1

**Number of cycles:** 2

#### Example RETURN

Before instruction: PC=x  
TOS=x

After instruction: PC=TOS  
TOS=TOS-1



**Syntax:** [label] RETLW k

**Description:** 8-bit constant k is stored in W register.  
Value off the top of a stack is stored in pc

**Operation:** (k)  $\Rightarrow$  W; TOS  $\Rightarrow$  PC

**Operand:**  $0 \leq k \leq 255$

**Flag:** -

**Number of words:** 1

**Number of cycles:** 2

**Example** RETLW 0x43

Before instruction: W=x

PC=x

TOS=x

After instruction: W=0x43

PC=TOS

TOS=TOS-1

**Syntax:** [label] RETFIE

**Description:** Return from a subprogram. Value from TOS is stored in PC. Interrupts are enabled by setting a GIE bit.

**Operation:** TOS  $\Rightarrow$  PC; 1  $\Rightarrow$  GIE

**Operand:** -

**Flag:** -

**Number of words:** 1

**Number of cycles:** 2

**Example** RETFIE

Before instruction: PC=x

GIE=0

After instruction: PC=TOS



**Syntax:** `[label] NOP`

**Description:** Does not execute any operation or affect any flag.

**Operation:** -

**Operand:** -

**Flag:** -

**Number of words:** 1

**Number of cycles:** 1

**Example** NOP

Before instruction: PC=X

After instruction: PC=X+1

**Syntax:** `[label] CLRWDT`

**Description:** Watchdog timer is reset. Prescaler of the Watchdog timer is also reset, and status bits  $\overline{TO}$  and  $\overline{PD}$  are set also.

**Operation:** 0  $\Rightarrow$  WDT  
0  $\Rightarrow$  WDT prescaler  
1  $\Rightarrow$   $\overline{TO}$   
1  $\Rightarrow$   $\overline{PD}$

**Operand:** -

**Flag:**  $\overline{TO}$ ,  $\overline{PD}$

**Number of words:** 1

**Number of cycles:** 1

**Example** CLRWDT

Before instruction: WDT counter=X  
WDT prescaler=1:128

After instruction: WDT counter=0x00  
WDT prescaler counter=0  
 $\overline{TO}$ =1  
 $\overline{PD}$ =1  
WDT prescaler=1:128



**Syntax:** `[label] SLEEP`

**Description:** Processor goes into low consumption mode. Oscis stopped.  $\overline{PD}$  (Power Down) status bit is reset.  $\overline{TO}$  bit is set. WDT (Watchdog) timer and its prescaler are reset.

**Operation:**  
 $0 \Rightarrow$  WDT  
 $0 \Rightarrow$  WDT prescaler  
 $1 \Rightarrow \overline{TO}$   
 $0 \Rightarrow \overline{PD}$

**Operand:** -

**Flag:**  $\overline{TO}$ ,  $\overline{PD}$

**Number of words:** 1

**Number of cycles:** 1

**Example**     SLEEP

Before instruction: WDT counter=x  
WDT prescaler=x

After instruction: WDT counter=0x00  
WDT prescaler=0  
 $\overline{TO}=1$   
 $\overline{PD}=0$