Embedded Systems Design (630414)

Lecture: 5-6

PIC Instruction Set

Prof. Kasim M. Al-Aubidy
Computer Eng. Dept.

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:

008C	movwf 0C

Data transfer

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

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

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).

		Arritmetic and logic				
ADDLW	k	Add constant and W	W+1 → W	C,DC,Z	1	
ADDWF	f, d	Add W and f	W+f → d	C,DC,Z	1	1,2
SUBLW	k	Subtract W from constant	W-k → W	C,DC,Z	1	
SUBWF	f, d	Subtract W from f	W-f → d	C,DC,Z	1	1,2
ANDLW	k	AND constant with W	W.AND.k → W	Z	1	
ANDWF	f, d	AND W with f	W.AND.f → d	Z	1	1,2
IORLW	k	OR constant with W	W.OR.k → W	Z	1	
IORWF	f, d	OR W with f	W.OR.f → d	Z	1	1,2
XORLW	k	Exclusive OR constant with W	W.XOR.k → W	Z	1	1,2
XORWF	f, d	Exclusive OR W with f	W.XOR.f → d	Z	1	
INCF	f, d	Increment f	f+1 → f	Z	1	1,2
DECF	f, d	Decrement f	f-1 → f	Z	1	1,2
RLF	f, d	Rotate Left f trough carry		C	1	1,2
RRF	f, d	Rotate Right f trough carry		C	1	1,2
COMF	f, d	Complement f	F → 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 → f(b)		1	1,2
BSF	f, b	Bit Set f	1 → 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    movlw 2
        call Lookup
Lookup  addwf PCL, f
        retlw k
        retlw k1
        retlw k2
          :
          :
        retlw kn
```

Other instructions

NOP	-	No Operation			1	
CLRWDT	-	Clear Watchdog Timer	0 → WDT,1↦TO,1→ PD	$\overline{TO}$,$\overline{PD}$	1	
SLEEP	-	Go into standby mode	0 → WDT,1↦TO,0→ PD	$\overline{TO}$,$\overline{PD}$	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 μs, and in case of conditional branching, execution period is 2 μs.

Syntax: [*label*] MOVLW **k**
Description: 8-bit constant **k** is written in **W** register.
Operation: **k** ⇒ (**W**)
Operand: 0 ≤ **k** ≤ 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** ⇒ (**f**)
Operand: 0 ≤ **f** ≤ 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** ⇒ (**d**)

Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]

Flag: Z

Number of words: 1

Number of cycles: 1

Example 1 MOVF FSR, 0

Before instruction: FSR=0×C2
W=0×00
After instruction: W=0×C2
Z=0

Example 2 MOVF INDF, 0

Before instruction: W=0×17
FSR=0×C2
address contents 0×C2=0×00
After instruction: W=0×17
FSR=0×C2
address contents 0×C2=0×00
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** ⇒ (**d**)

Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]

Flag: Z

Number of words: 1

Number of cycles: 1

Example 1 MOVF FSR, 0

Before instruction: FSR=0×C2
W=0×00
After instruction: W=0×C2
Z=0

Example 2 MOVF INDF, 0

Before instruction: W=0×17
FSR=0×C2
address contents 0×C2=0×00
After instruction: W=0×17
FSR=0×C2
address contents 0×C2=0×00
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 ⇒ (**W**)
Operand: -
Flag: Z
Number of words: 1
Number of cycles: 1

Example CLRW

Before instruction: W=0×55
After instruction: W=0×00
Z=1
Z=1

Example 2 CLRF INDF

Before instruction: FSR=0×C2
address contents 0×C2=0×33
After instruction: FSR=0×C2
address contents 0×C2=0×00
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 ⇒ **f**
Operand: 0 ≤ **f** ≤ 127
Flag: Z
Number of words: 1
Number of cycles: 1

Example 1 CRLF STATUS

Before instruction: STATUS=0×C2
After instruction: STATUS=0×00
Z=1

Example 2 CLRF INDF

Before instruction: FSR=0×C2
address contents 0×C2=0×33
After instruction: FSR=0×C2
address contents 0×C2=0×00
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> ⇒ **d**<4:7>, **f**<4:7> ⇒ **d**<0:3>;
Operand:	0 ≤ **f** ≤ 127 **d** ∈ [0,1]
Flag:	-
Number of words:	1
Number of cycles:	1

Example 1 SWAP REG, 0

Before instruction:	REG=0×F3
After instruction:	REG=0×F3 W=0×3F

Example 2 SWAP REG, 1

Before instruction:	REG=0×F3
After instruction:	REG=0×3F

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** ⇒ **W**

Operand: 0 ≤ **k** ≤ 255

Flag: C, DC, Z

Number of words: 1

Number of cycles: 1

Example 1 ADDLW 0×15

Before instruction: W=0×10
After instruction: W=0×25

Example 2 ADDLW REG

Before instruction: W=0×10
register contents REG=0×37
After instruction: W=0×47

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**) ⇒ **d**
d ∈ [0,1]

Operand: 0 ≤ **f** ≤ 127

Flag: C, DC, Z

Number of words: 1

Number of cycles: 1

Example 1 ADDWF FSR, 0

Before instruction: W=0×17
FSR=0×C2
After instruction: W=0×D9
FSR=0×C2

Example 2 ADDLW INDF, 1

Before instruction: W=0×17
FSR=0×C0
address contents 0×C2=0×20
After instruction: W=0×17
FSR=0×C2
address contents 0×C2=0×37

Syntax: [*label*] SUBLW **k**

Description: Contents of **W** register is subtracted from **k** constant, and result is stored in **W** register.

Operation: **k** - (**W**) ⇒ **W**

Operand: 0 ≤ **k** ≤ 255

Flag: C, DC, Z

Number of words: 1

Number of cycles: 1

Example 1 SUBLW 0×03

Before instruction:	W=0×01, C=×, Z=×	
After instruction:	W=0×02, C=1, Z=0	Result > 0
Before instruction:	W=0×03, C=×, Z=×	
After instruction:	W=0×00, C=1, Z=1	Result = 0
Before instruction:	W=0×04, C=×, Z=×	
After instruction:	W=0×FF, C=0, Z=0	Result < 0

Example 2 SUBLW REG

Before instruction:	W=0×10	
	contents REG=0×37	
After instruction:	W=0×27	
	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**) ⇒ **d**

Operand: 0 ≤ **f** ≤ 127

d ∈ [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=×, Z=×	
After instruction:	REG=1, W=2, C=1, Z=0	Result > 0
Before instruction:	REG=2, W=2, C=×, Z=×	
After instruction:	REG=0, W=2, C=1, Z=1	Result = 0
Before instruction:	REG=1, W=2, C=×, Z=×	
After instuction:	REG=0×FF, 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**) .AND. **k** ⇒ **W**

Operand: 0 ≤ **k** ≤ 255

Flag: Z

Number of words: 1

Number of cycles: 1

Example 1 ANDLW 0×5F

```
Before instruction:  W=0×A3   ; 0101 1111  (0×5F)
After instruction:   W=0×03   ; 1010 0011  (0×A3)
                              ---------------------
                              ; 0000 0011  (0×03)
```

Example 2 ANDLW REG

```
Before instruction:  W=0×A3   ; 1010 0011  (0×A3)
                     REG=0×37 ; 0011 0111  (0×37)
After instruction:   W=0×23   ---------------------
                              ; 0010 0011  (0×23)
```

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**) .AND. **f** ⇒ **d**

Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]

Flag: Z

Number of words: 1

Number of cycles: 1

Example 1 ANDWF FSR, 1

```
Before instruction: W=0×17, FSR=0×C2 ; 0001 0111  (0×17)
After instruction:  W=0×17, FSR=02   ; 1100 0010  (0×C2)
                                     ---------------
                                     ; 0000 0010  (0×02)
```

Example 2 ANDWF FSR, 0

```
Before instruction: W=0×17, FSR=0×C2 ; 0001 0111  (0×17)
After instruction:  W=0×02, FSR=0×C2 ; 1100 0010  (0×C2)
                                     ---------------------
                                     ; 0000 0010  (0×02)
```

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**) ⇒ **W**
Operand:	0 ≤ **k** ≤ 255
Flag:	Z
Number of words:	1
Number of cycles:	1

Example 1 IORLW 0×35

Before instruction:	W=0×9A
After instruction:	W=0×BF
	Z=0

Example 2 IORLW REG

Before instruction:	W=0×9A
	contenst REG=0×37
After instruction:	W=0×9F
	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**) ⇒ **d**
Operand:	0 ≤ **f** ≤ 127 **d** ∈ [0,1]
Flag:	Z
Number of words:	1
Number of cycles:	1

Example 1 IORWF REG, 0

Before instruction:	REG=0×13, W=0×91
After instruction:	REG=0×13, W=0×93
	Z=0

Example 2 IORWF REG, 1

Before instruction:	REG=0×13, W=0×91
After instruction:	REG=0×93, W=0×91
	Z=0

Syntax: [*label*] XORLW **k**

Description: Operation exclusive OR (XOR) is done over the contents of **W** register and constant **k**, and result is stored in **W** register.

Operation: (**W**) .XOR. **k** ⇒ **W**

Operand: 0 ≤ **k** ≤ 255

Flag: Z

Number of words: 1

Number of cycles: 1

Example 1 XORLW 0×AF

```
Before instruction:   W=0×B5    ; 1010 1111  (0×AF)
After instruction:    W=0×1A    ; 1011 0101  (0×B5)
                                ---------------------
                                ; 0001 1010  (0×1A)
```

Example 2 XORLW REG

```
Before instruction:   W=0×AF    ; 1010 1111  (0×A3)
                      REG=0×37  ; 0011 0111  (0×37)
After instruction:    W=0×18    ---------------------
                      Z=0       ; 0001 1000  (0×18)
```

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

Description: Operation exclusive 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**) .XOR. (**f**) ⇒ **d**

Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]

Flag: Z

Number of words: 1

Number of cycles: 1

Example 1 XORWF REG, 1

```
Before instruction:   REG=0×AF, W=0×B5 ; 1010 1111  (0×AF)
After instruction:    REG=0×1A, W=0×B5 ; 1011 0101  (0×B5)
                                       ---------------------
                                       ; 0001 1010  (0×1A)
```

Example 2 XORWF REG, 0

```
Before instruction:   REG=0×AF, W=0×B5 ; 1010 1111  (0×AF)
After instruction:    REG=0×AF, W=0×1A ; 1011 0101  (0×B5)
                                       ---------------------
                                       ; 0001 1010  (0×1A)
```

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 ⇒ **d**
Operand:	0 ≤ **f** ≤ 127 **d** ∈ [0,1]
Flag:	Z
Number of words:	1
Number of cycles:	1

Example 1 INCF REG, 1

Before instruction:	REG=0×FF Z=0
After instruction:	REG=0×00 Z=1

Example 2 INCF REG, 0

Before instruction:	REG=0×10 W=× Z=0
After instruction:	REG=0×10 W=0×11 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 ⇒ **d**
Operand:	0 ≤ **f** ≤ 127 **d** ∈ [0,1]
Flag:	Z
Number of words:	1
Number of cycles:	1

Example 1 DECF REG, 1

Before instruction:	REG=0×01 Z=0
After instruction:	REG=0×00 Z=1

Example 2 DECF REG, 0

Before instruction:	REG=0×13 W=× Z=0
After instruction:	REG=0×13 W=0×12 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**<n>) ⇒ **d**<n+1>, **f**<7> ⇒ C, C ⇒ **d**<0>;

Operand: 0 ≤ **f** ≤ 127
d ∈ [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.

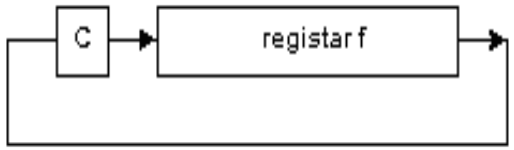
Operation: (**f**<n>) ⇒ **d**<n-1>, **f**<0> ⇒ C, C ⇒ **d**<7>

Operand: 0 ≤ **f** ≤ 127
d ∈ [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

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

Description: Contents of **f** register is complemented
If **d=0**, result is stored in **W** register.
If **d=1**, result is stored in **f** register.

Operation: $(\overline{f}) \Rightarrow d$

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

Flag: Z

Number of words: 1

Number of cycles: 1

Example 1 COMF REG, 0

```
Before instruction:   REG=0×13 ; 0001 0011  (0×13)
After instruction:    REG=0×13 ; complement
                      W=0×EC   ----------------------
                               ; 1110 1100  (0×EC)
```

Example 2 COMF INDF, 1

```
Before instruction:   FSR=0×C2
                      address contents (FSR)=0×AA
After instruction:    FSR=0×C2
                      address contents (FSR)=0×55
```

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

Description: Reset bit **b** in **f** register.

Operation: (0) ⇒ **f<b>**

Operand: $0 \le f \le 127$
$0 \le b \le 7$

Flag: -

Number of words: 1

Number of cycles: 1

Example 1 BCF REG, 7

```
Before instruction:   REG=0×C7 ; 1100 0111  (0×C7)
After instruction:    REG=0×47 ; 0100 0111  (0×47)
```

Example 2 BCF INDF, 3

```
Before instruction:   W=0×17
                      FSR=0×C2
                      address contents (FSR)=0×2F
After instruction:    W=0×17
                      FSR=0×C2
                      address contents (FSR)=0×27
```

Syntax: [*label*] BSF **f**, **b**
Description: Set bit **b** in **f** register.
Operation: 1 ⇒ **f**<**b**>
Operand: 0 ≤ **f** ≤ 127
0 ≤ **b** ≤ 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**<**b**>)=0
Operand: 0 ≤ **f** ≤ 127
0 ≤ **b** ≤ 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 ≤ **f** ≤ 127
0 ≤ **b** ≤ 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 ⇒ **d**

Operand: 0 ≤ **f** ≤ 127
d ∈ [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 ⇒ **d**

Operand: 0 ≤ **f** ≤ 127
d ∈ [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** ⇒ PC<10:0>, (PCLATH<4:3>) ⇒ PC<12:11>

Operand: 0 ≤ **k** ≤ 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 ⇒ Top Of Stack (TOS)
k ⇒ PC<10:0>, (PCLATH<4:3>) ⇒ PC<12:11>

Operand: 0 ≤ **k** ≤ 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 ⇒ 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**) ⇒ **W**; TOS ⇒ PC

Operand: 0 ≤ **k** ≤ 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 suboroaram. Value from TOS is stored in PC. Interrupts are enabled by setting a GIE bit.

Operation: TOS ⇒ PC; 1 ⇒ 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=×
After instruction: PC=×+1

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

Operation: 0 ⇒ WDT
0 ⇒ WDT prescaler
1 ⇒ $\overline{TO}$
1 ⇒ $\overline{PD}$
Operand: -
Flag: $\overline{TO}$, $\overline{PD}$
Number of words: 1
Number of cycles: 1

Example CLRWDT

Before instruction: WDT counter=×
WDT prescaler=1:128
After instruction: WDT counter=0×00
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 ⇒ WDT 0 ⇒ WDT prescaler 1 ⇒ $\overline{TO}$ 0 ⇒ $\overline{PD}$
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 TO=1 PD=0