PIC Assembly Language and Instruction set

PIC Assembly Code

Label OpCode f, F(W) ; comments

Instruction from to

f = Source: name of special-purpose register or RAM variable

F= Destination is f

W=Destination is Working register

Assembly – Guideline (Microchip)

- ✔ Mnemonics (Opcodes) --- lower case
  - – Examples,   movf, addwf
- ✔ File registers --- Upper case
  - – Examples, FSR, STATUS, RP1
- ✔ Label --- mixed case
  - – Examples, Mainline, LoopTime.

PIC Macro Assembler

- ✔ Create your own instructions from sequence of PIC instructions
  - – Format

```
Name   macro                  ;declare macro name
       PIC instruction…       ;sequence of PIC
       PIC instruction…       ;instructions
       ….
       PIC instruction
       endm                   ;end of this macro
```

PIC Macro Assembler -Examples

✔ Create macro instruction to select Bank 0 and Bank 1 of register files

```
Bank0     macro
          bcf STATUS, RP0 ;clear RP0 bit
          endm
Bank1     macro
          bsf STATUS, RP0 ;set RP0 bit
          endm
```

PIC Macro Support

- ✔ Macro definitions can be inserted into a source file at any place before it is invoked
- ✔ Macro definitions can also be place in a separate file called MACROS.INC and then "included" into the source file with

```
Include "C:\MPLAB\MACROS.INC"
```

PIC Macro Support

- ✔ Macro definitions can be inserted into a source file at any place before it is invoked
- ✔ Macro definitions can also be place in a separate files and then "included" into the source file, e.g.

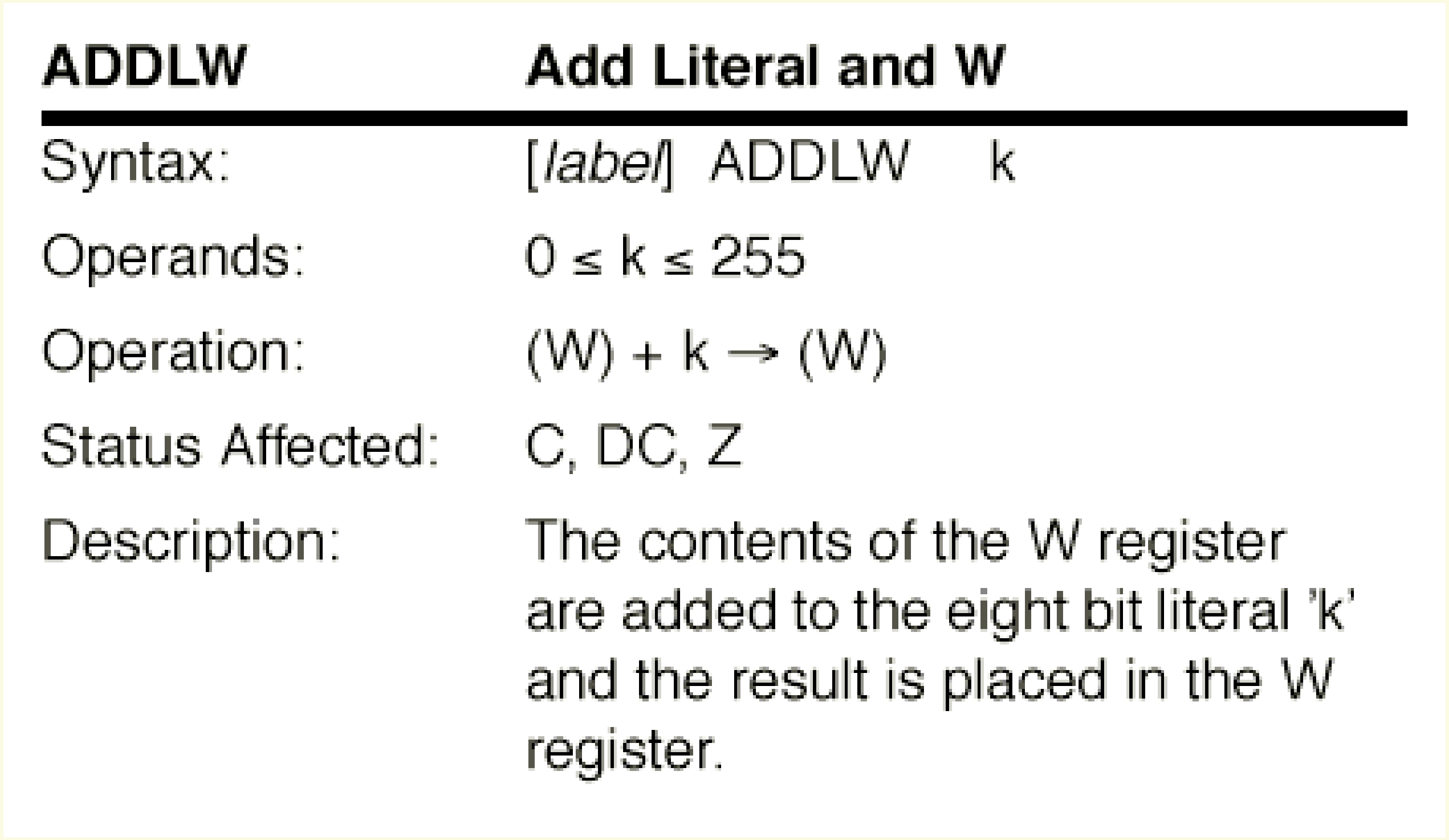
  Include "C:\MPLAB\MACROS.INC"

  Note: each macro will add code to the object file only if it is invoked one or more times.

PIC Macro Support (Cont.)

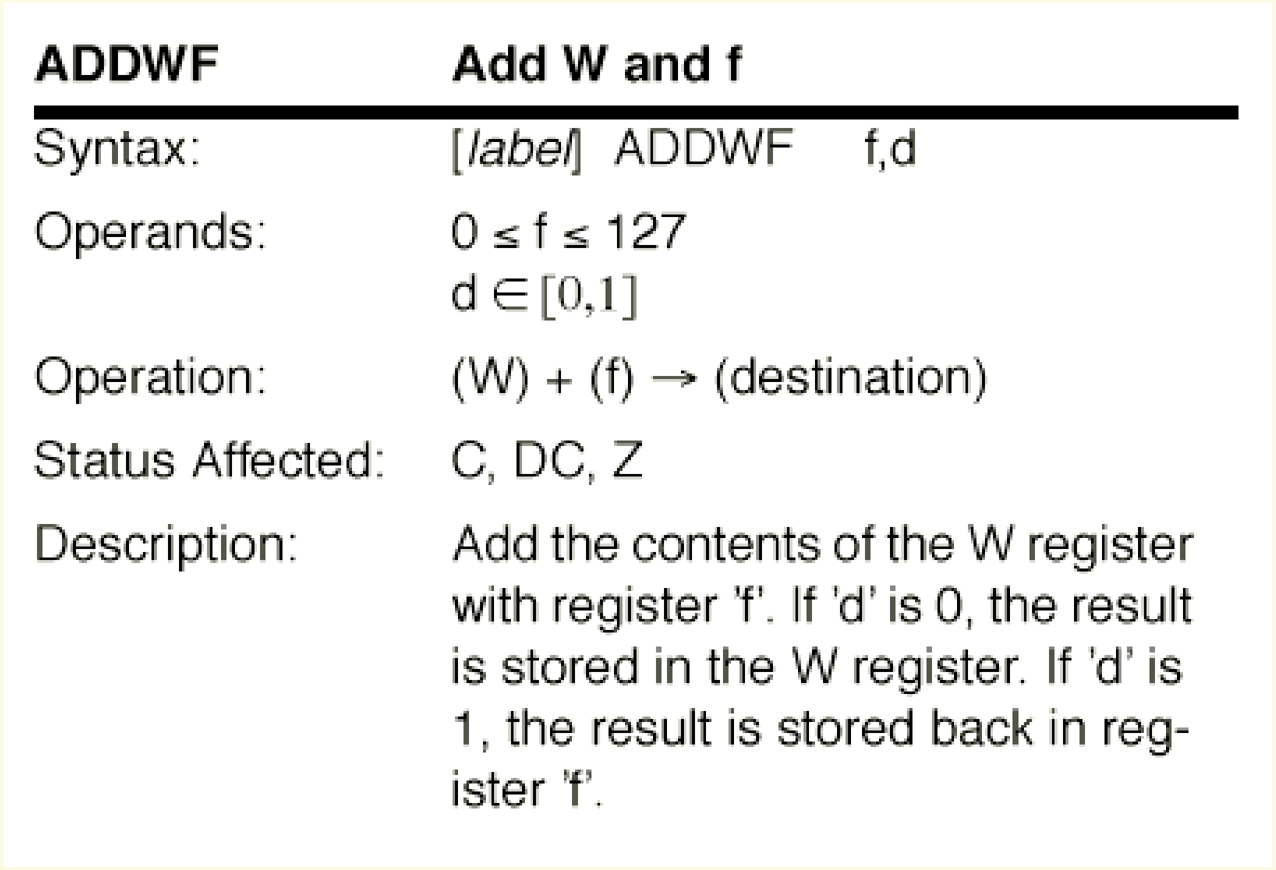
- ✔ Microchip supports multiplication and division operations with dozens of macros for both signed and unsigned integers having lengths of 8,16,24 and 32 bits.
- ✔ Where? ….WWW.Microchip.com

Instruction set



ADDLW	**Add Literal and W**
Syntax:	[*label*] ADDLW k
Operands:	$0 \leq k \leq 255$
Operation:	$(W) + k \rightarrow (W)$
Status Affected:	C, DC, Z
Description:	The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.

Instruction set



ADDWF	**Add W and f**
Syntax:	[*label*] ADDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	$(W) + (f) \rightarrow$ (destination)
Status Affected:	C, DC, Z
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'.

STATUS register: C, Carry bit

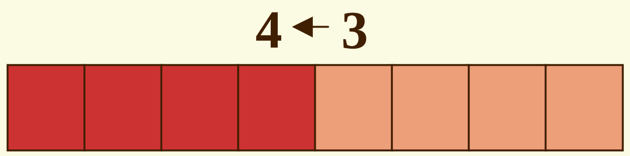
- ✔ Add Instruction
  - – PIC ALU performs 8-bit unsigned addition
    - Examples

```
movlw  25     ;load W with 25

addlw  112  ;add 112 to W


   0001 1001  ; 25   } +
   0111 0000  ; 112  }
0 1000 1001  ; 137
  Carry bit

W = 137  and C bit =0
```

```
movlw  25     ;load W with 25

addlw  250  ;add 250 to W


   0001 1001  ; 25   } +
   1111 1010  ; 250  }
1 0001 0011  ; 19!
  Carry bit

W = 19  and C bit =1
```

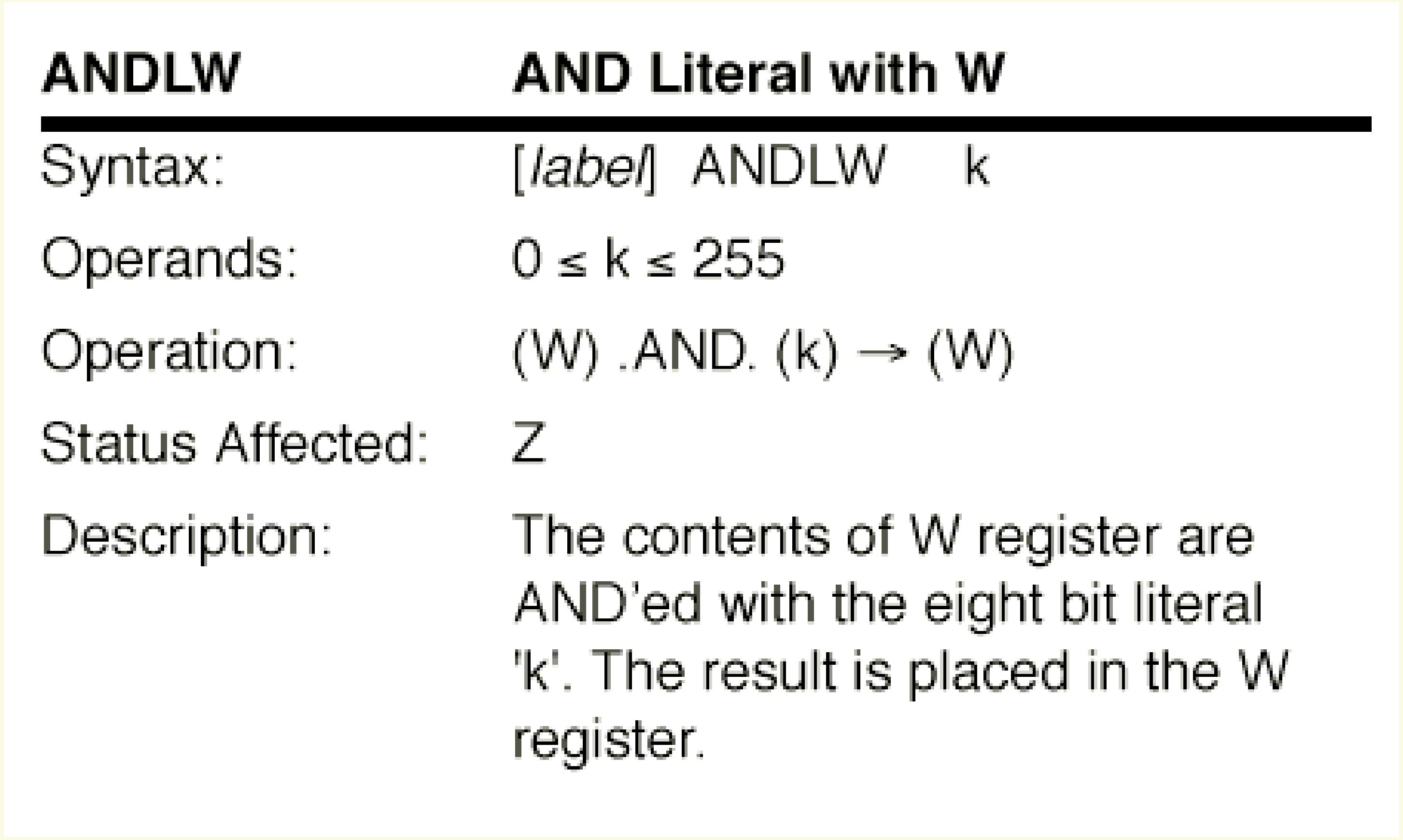
STATUS register: DC(Digit Carry)

- ✔ DC, or Digit Carry, bit indicates a carry from bit 3 to the bit 4 during an 8-bit addition/subtraction
- ✔ Useful when adding/subtracting BCD numbers
  - Can be use as a signal to adjust the BCD
  - Example



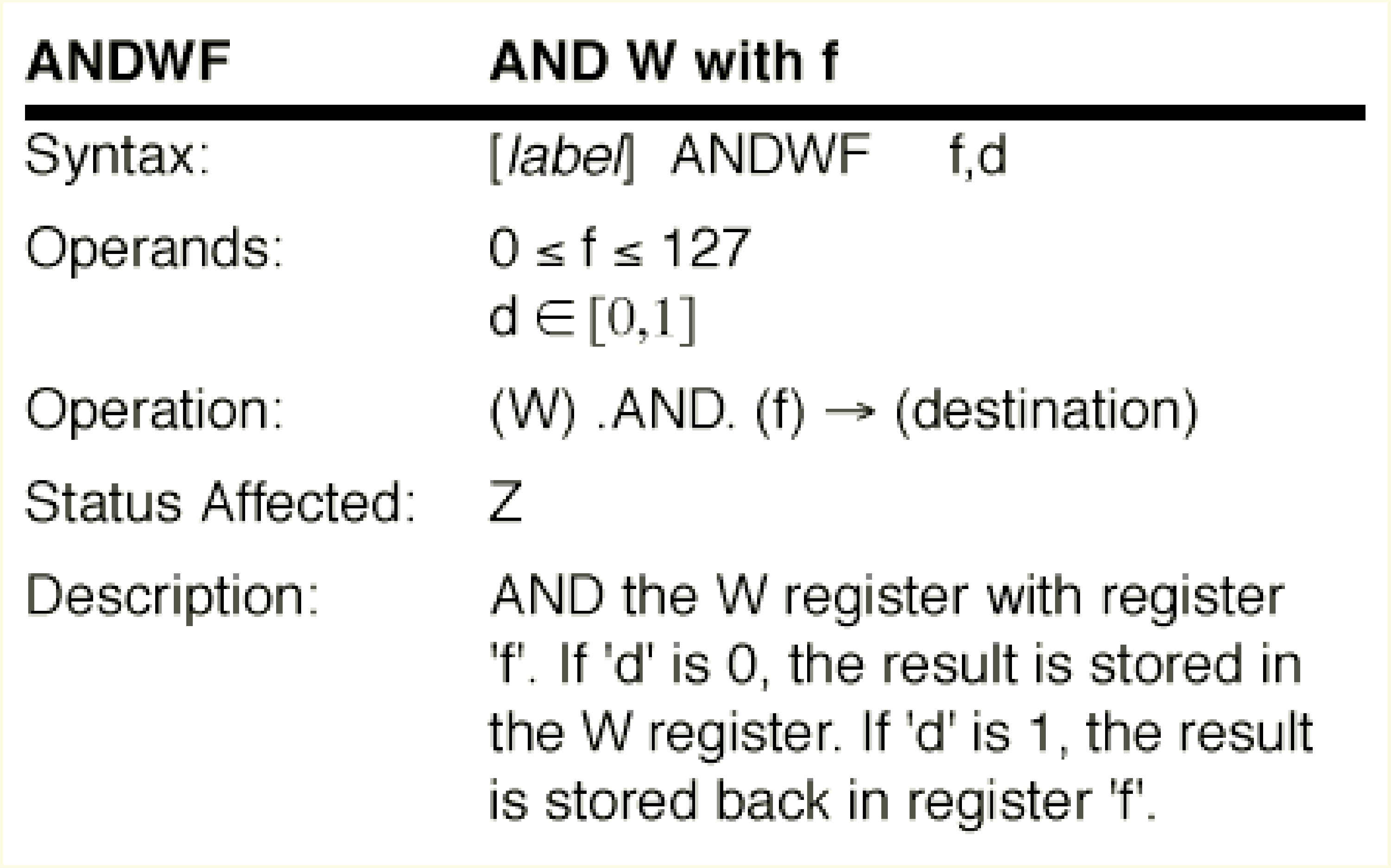
```
DC = 1
B’0011 1000’ ; 38 BCD  }
B’0011 1000’ ; 38 BCD  }  + (Binary)
B’0111 0000’ ; 70 (result from binary addition)
B’      0110’ ; Add 6 to correct the result
B’0111 0110’ ; 76 BCD!
```

Instruction set



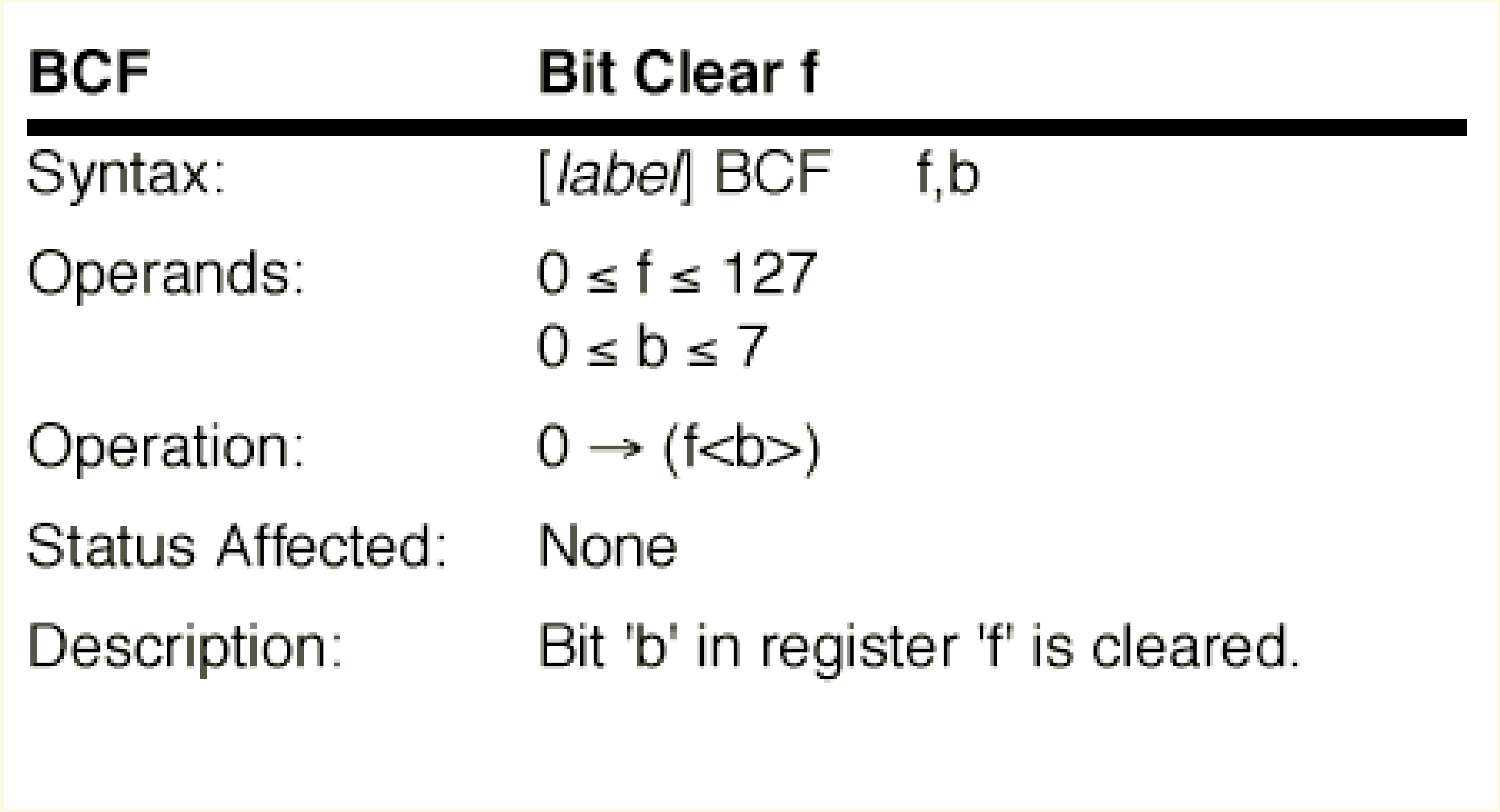
ANDLW	**AND Literal with W**
Syntax:	[*label*] ANDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .AND. (k) → (W)
Status Affected:	Z
Description:	The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register.

Instruction set



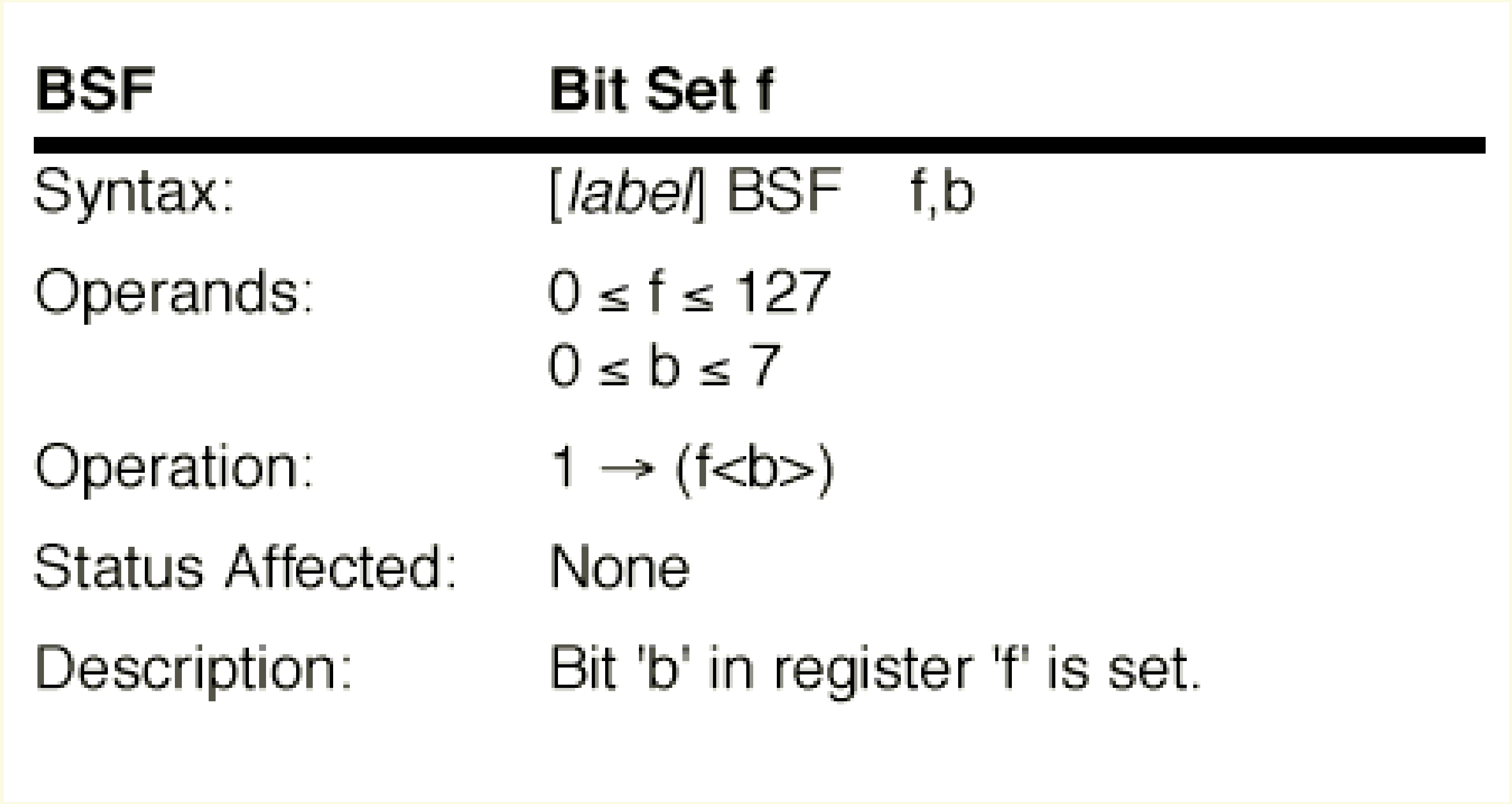
ANDWF	**AND W with f**
Syntax:	[*label*] ANDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) .AND. (f) → (destination)
Status Affected:	Z
Description:	AND the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

Instruction set



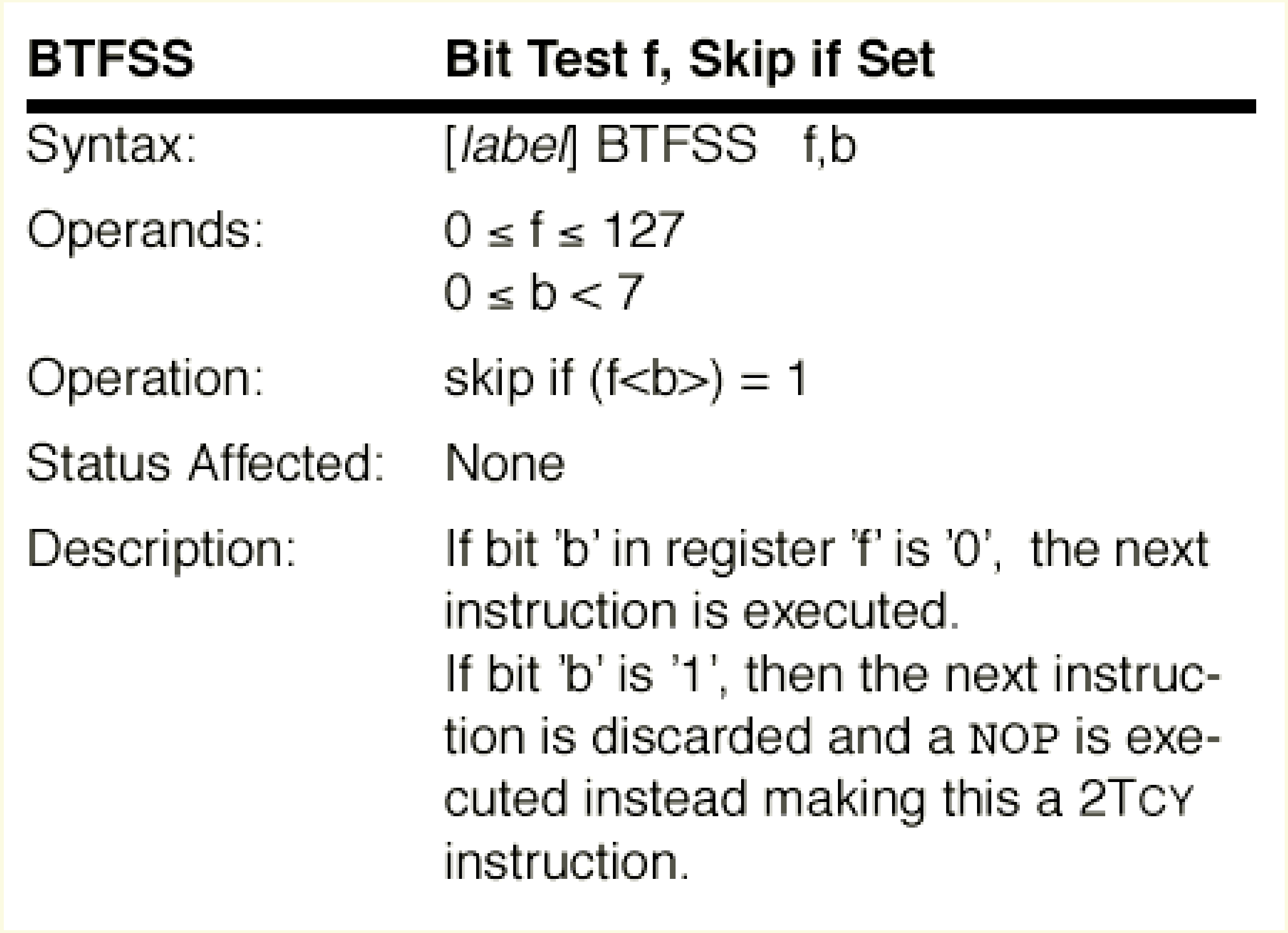
BCF	**Bit Clear f**
Syntax:	[*label*] BCF f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	$0 \rightarrow (f<b>)$
Status Affected:	None
Description:	Bit 'b' in register 'f' is cleared.

Instruction set



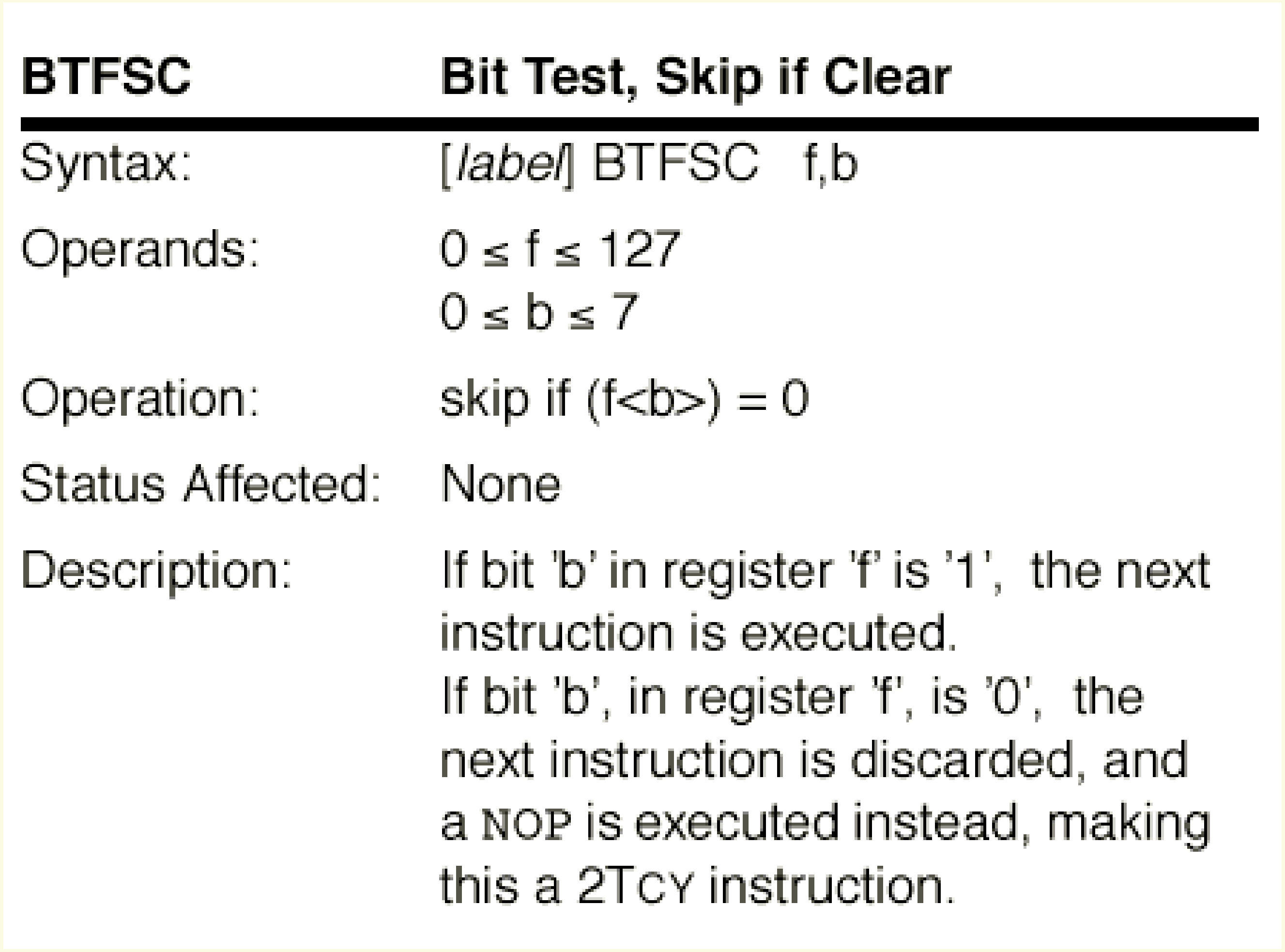
BSF	**Bit Set f**
Syntax:	[*label*] BSF f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	$1 \rightarrow (f<b>)$
Status Affected:	None
Description:	Bit 'b' in register 'f' is set.

Instruction set



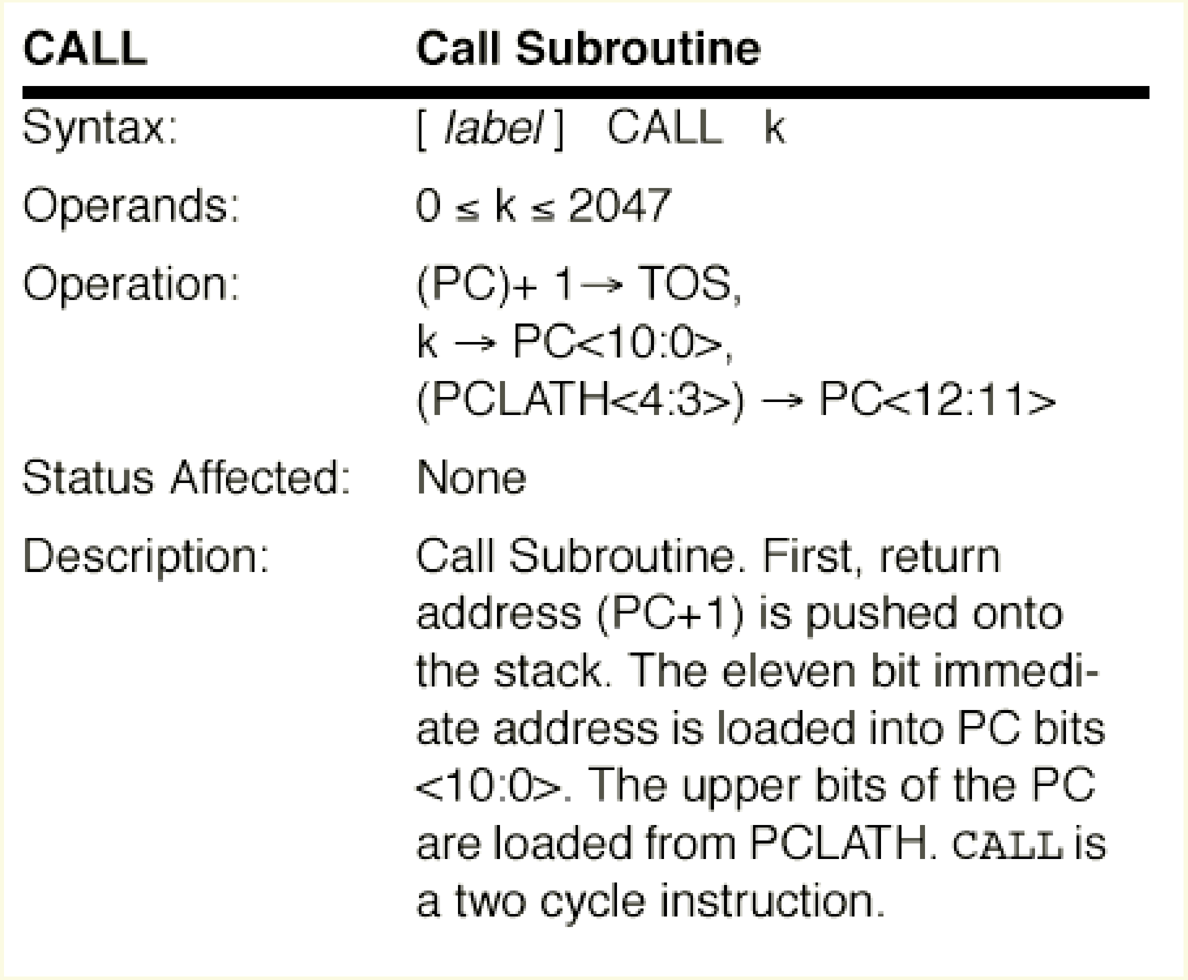
BTFSS	**Bit Test f, Skip if Set**
Syntax:	[*label*] BTFSS f,b
Operands:	$0 \leq f \leq 127$ $0 \leq b < 7$
Operation:	skip if (f<b>) = 1
Status Affected:	None
Description:	If bit 'b' in register 'f' is '0', the next instruction is executed. If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead making this a 2TCY instruction.

Instruction set



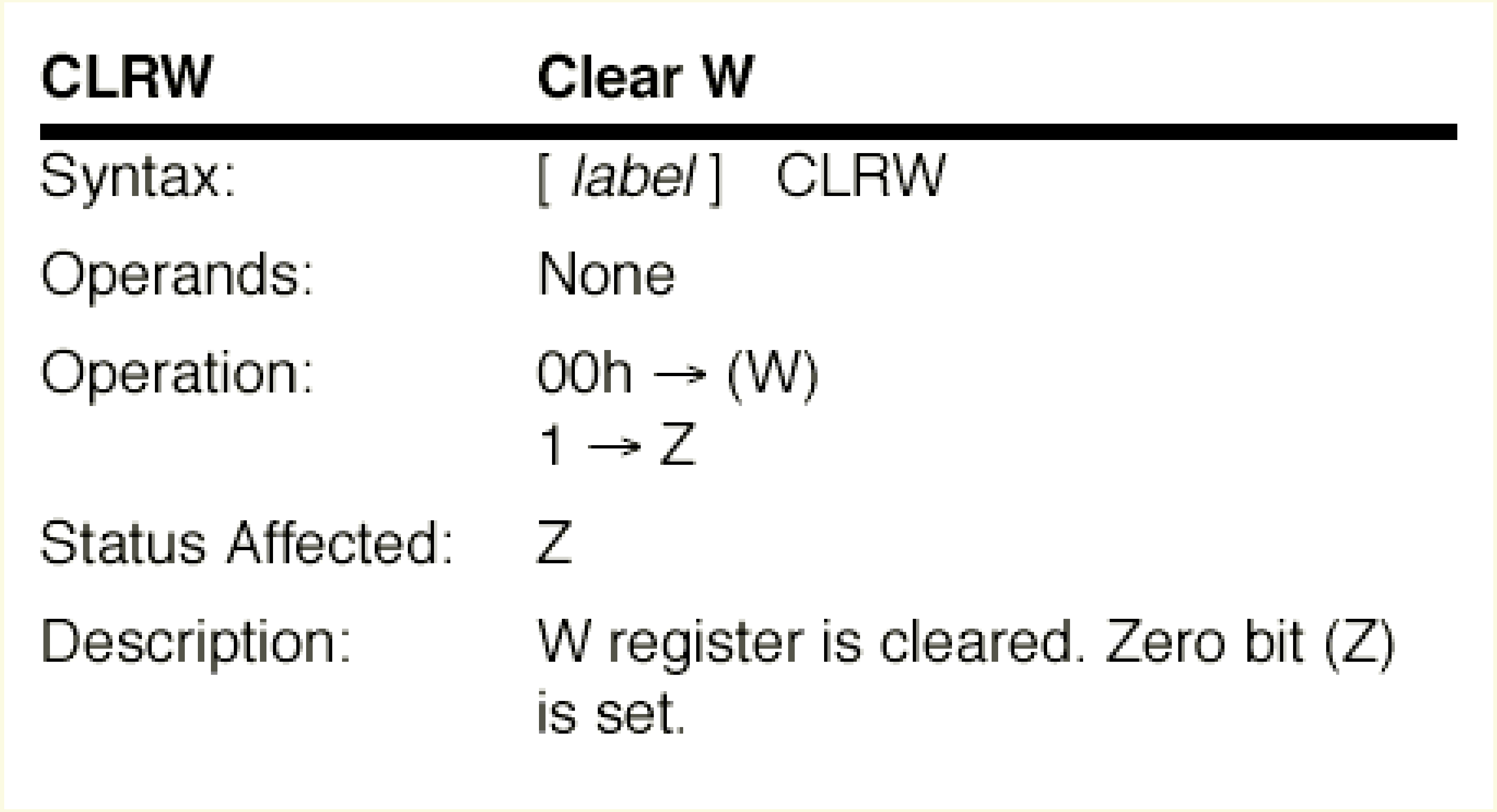
BTFSC	**Bit Test, Skip if Clear**
Syntax:	[*label*] BTFSC f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	skip if (f<b>) = 0
Status Affected:	None
Description:	If bit 'b' in register 'f' is '1', the next instruction is executed. If bit 'b', in register 'f', is '0', the next instruction is discarded, and a NOP is executed instead, making this a 2TCY instruction.

Instruction set



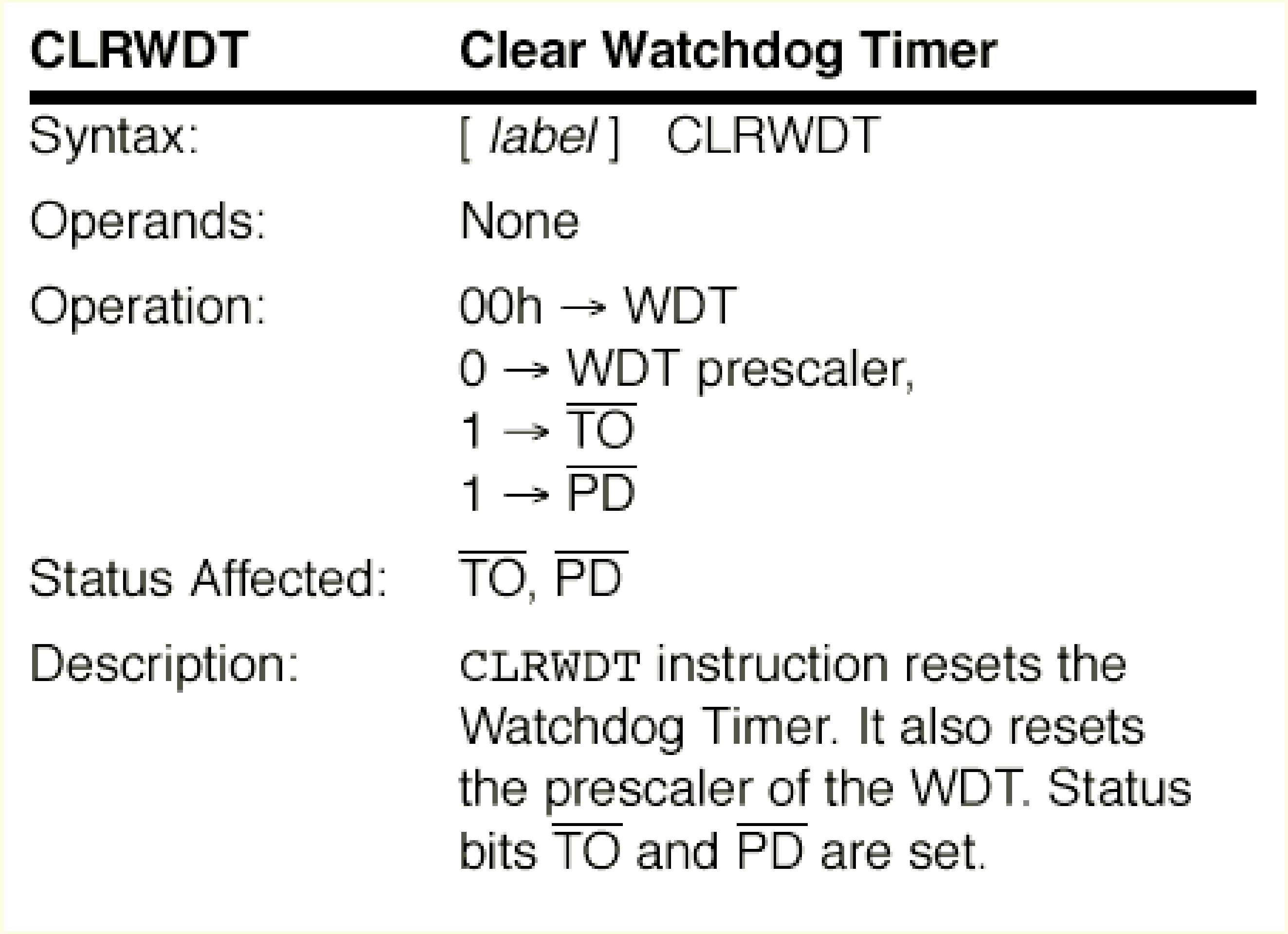
CALL	**Call Subroutine**
Syntax:	[*label*] CALL k
Operands:	$0 \le k \le 2047$
Operation:	$(PC)+1 \rightarrow TOS$, $k \rightarrow PC<10:0>$, $(PCLATH<4:3>) \rightarrow PC<12:11>$
Status Affected:	None
Description:	Call Subroutine. First, return address (PC+1) is pushed onto the stack. The eleven bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. `CALL` is a two cycle instruction.

Instruction set



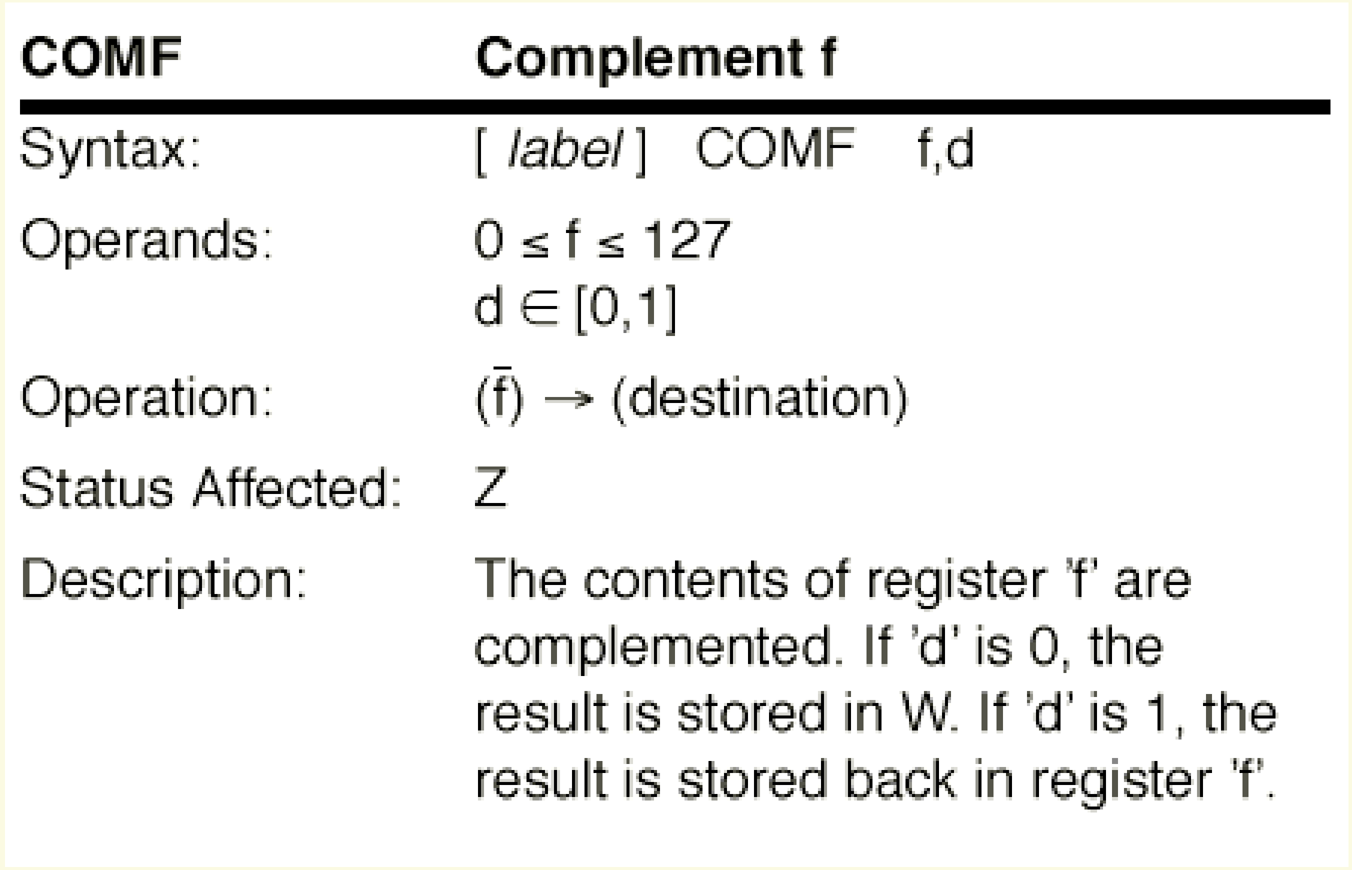
CLRW	**Clear W**
Syntax:	[*label*] CLRW
Operands:	None
Operation:	00h → (W) 1 → Z
Status Affected:	Z
Description:	W register is cleared. Zero bit (Z) is set.

Instruction set



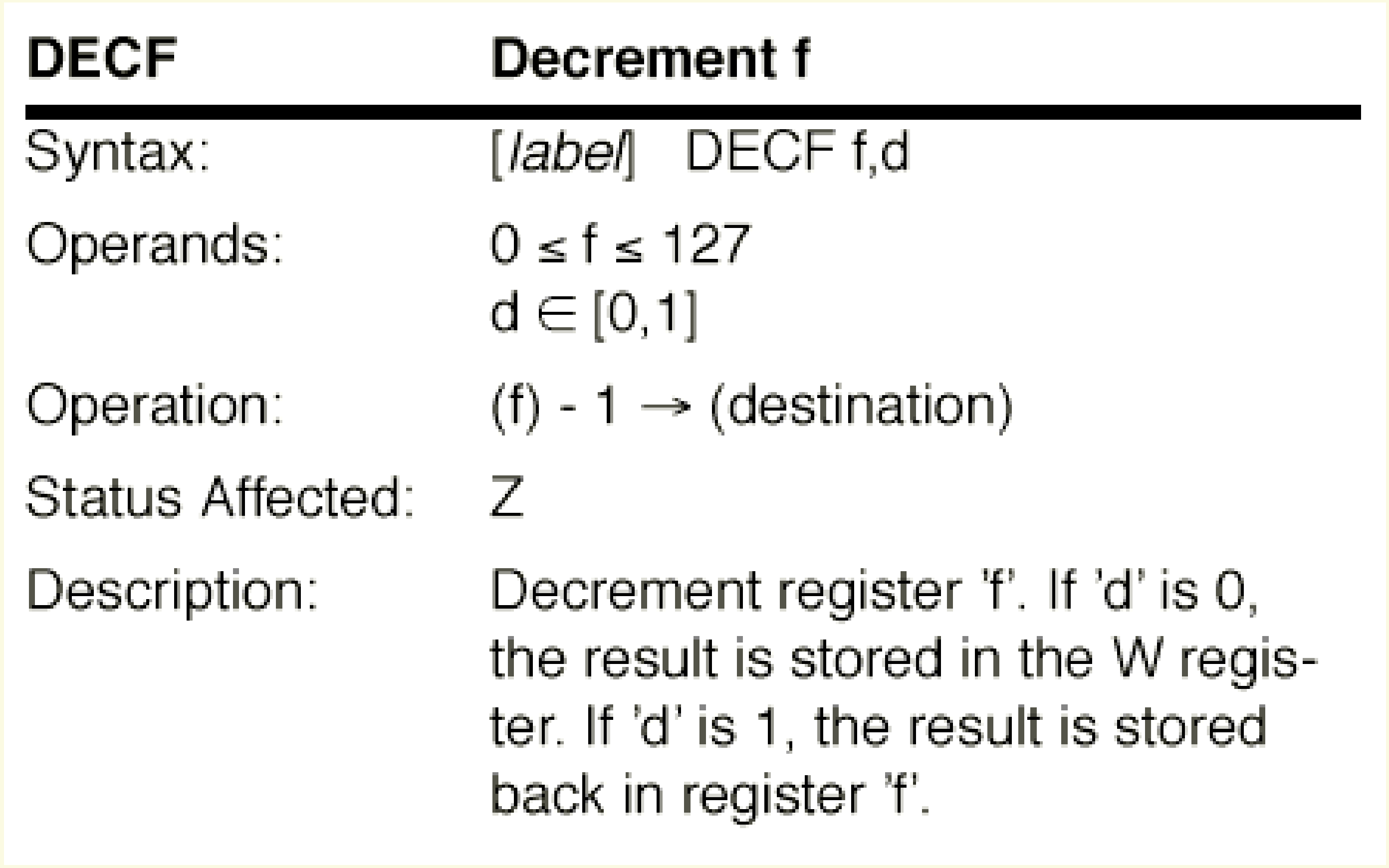
CLRWDT	**Clear Watchdog Timer**
Syntax:	[*label*] CLRWDT
Operands:	None
Operation:	00h → WDT 0 → WDT prescaler, 1 → $\overline{TO}$ 1 → $\overline{PD}$
Status Affected:	$\overline{TO}$, $\overline{PD}$
Description:	`CLRWDT` instruction resets the Watchdog Timer. It also resets the prescaler of the WDT. Status bits $\overline{TO}$ and $\overline{PD}$ are set.

Instruction set



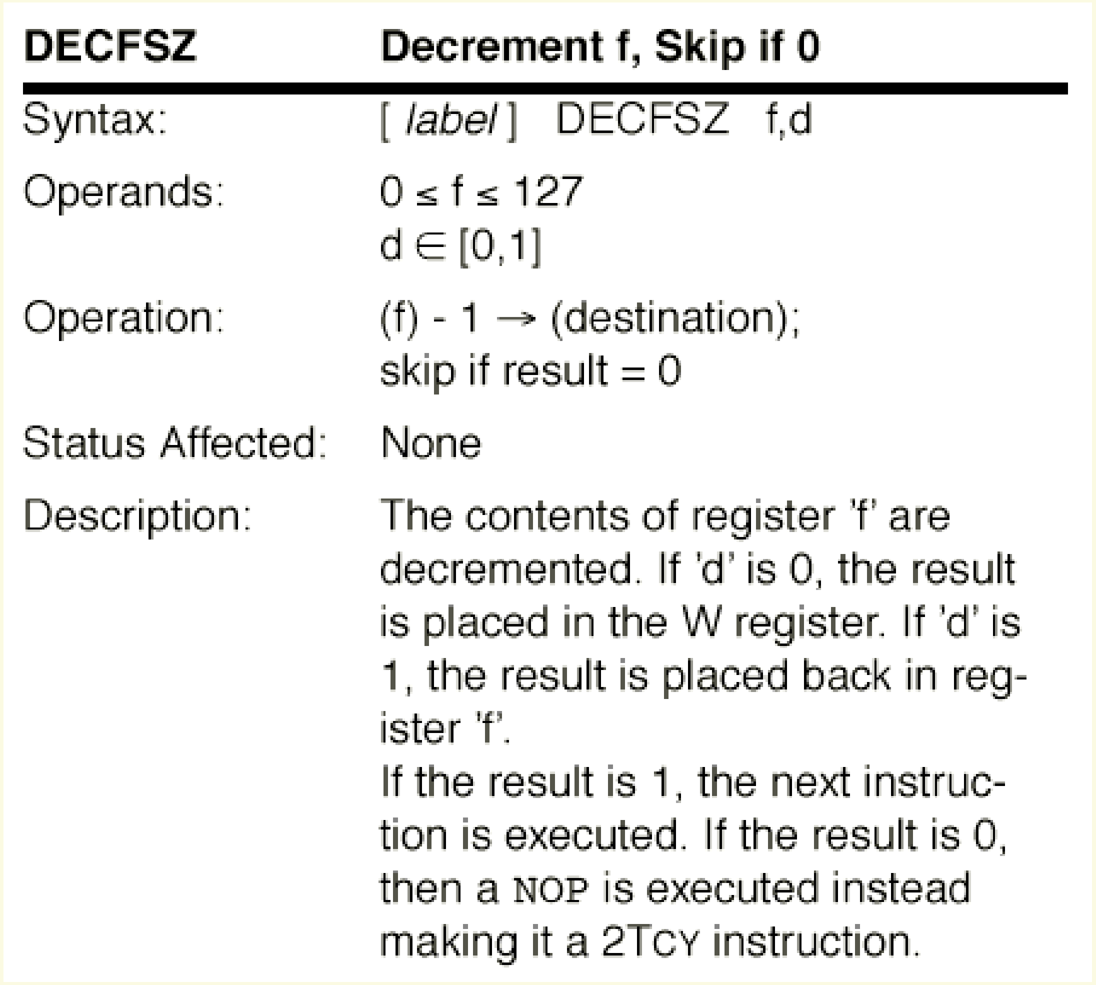
COMF	Complement f
Syntax:	[*label*] COMF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	$(\bar{f}) \rightarrow$ (destination)
Status Affected:	Z
Description:	The contents of register 'f' are complemented. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f'.

Instruction set



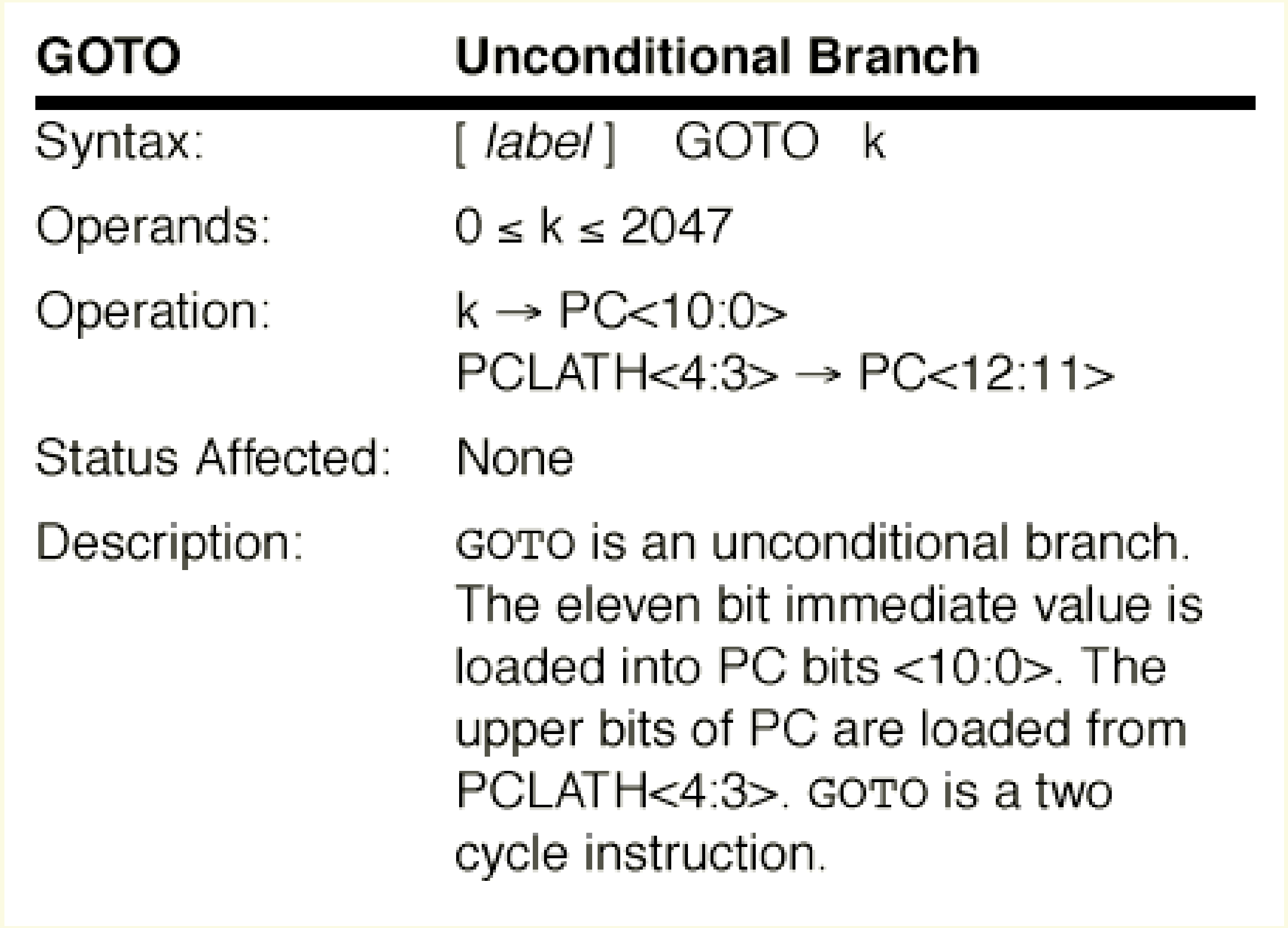
DECF	Decrement f
Syntax:	[*label*] DECF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) - 1 → (destination)
Status Affected:	Z
Description:	Decrement register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

Instruction set



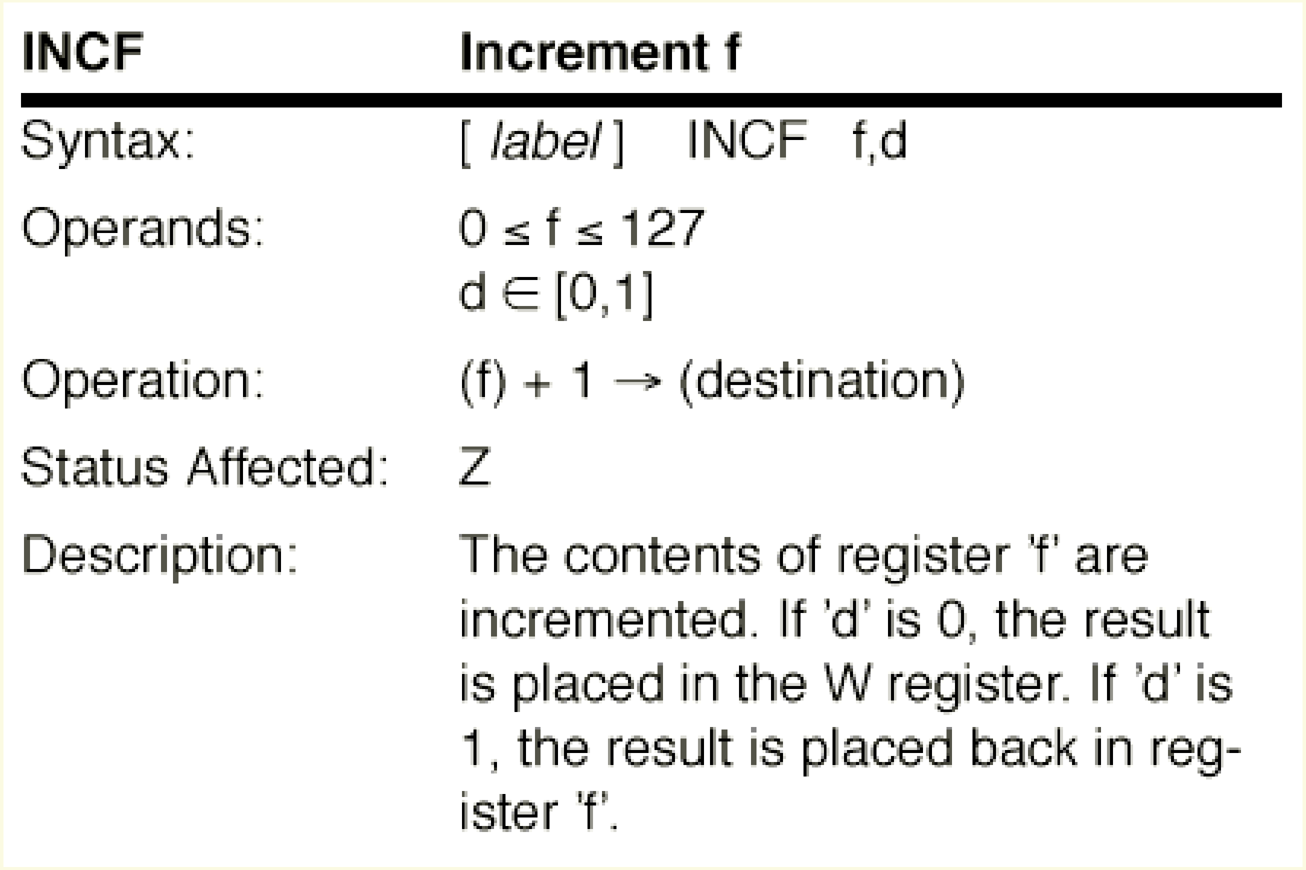
DECFSZ	**Decrement f, Skip if 0**
Syntax:	[*label*] DECFSZ f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) - 1 $\rightarrow$ (destination); skip if result = 0
Status Affected:	None
Description:	The contents of register 'f' are decremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'. If the result is 1, the next instruction is executed. If the result is 0, then a NOP is executed instead making it a 2TCY instruction.

Instruction set



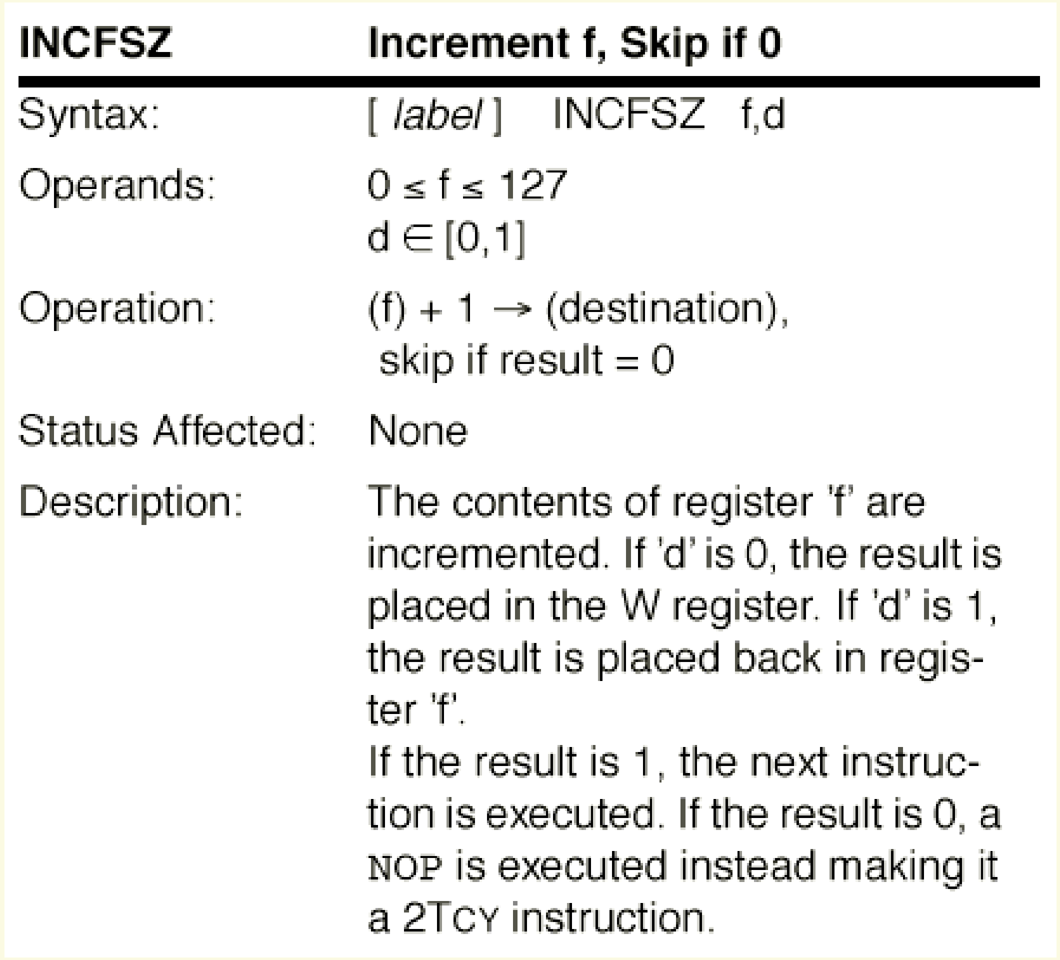
GOTO	**Unconditional Branch**
Syntax:	[*label*] GOTO k
Operands:	0 ≤ k ≤ 2047
Operation:	k → PC<10:0> PCLATH<4:3> → PC<12:11>
Status Affected:	None
Description:	GOTO is an unconditional branch. The eleven bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two cycle instruction.

Instruction set



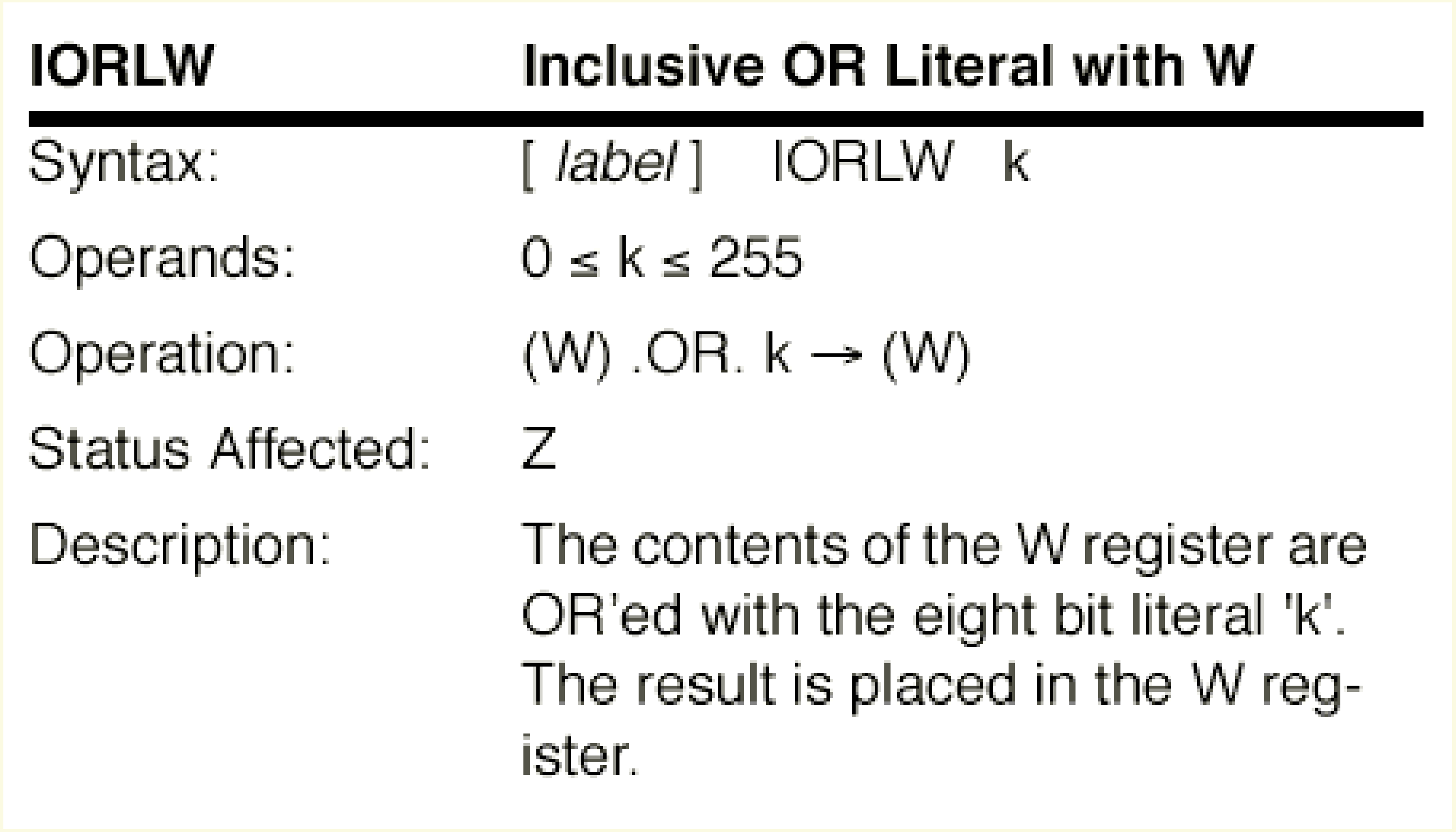
INCF	**Increment f**
Syntax:	[*label*] INCF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	$(f) + 1 \rightarrow$ (destination)
Status Affected:	Z
Description:	The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.

Instruction set



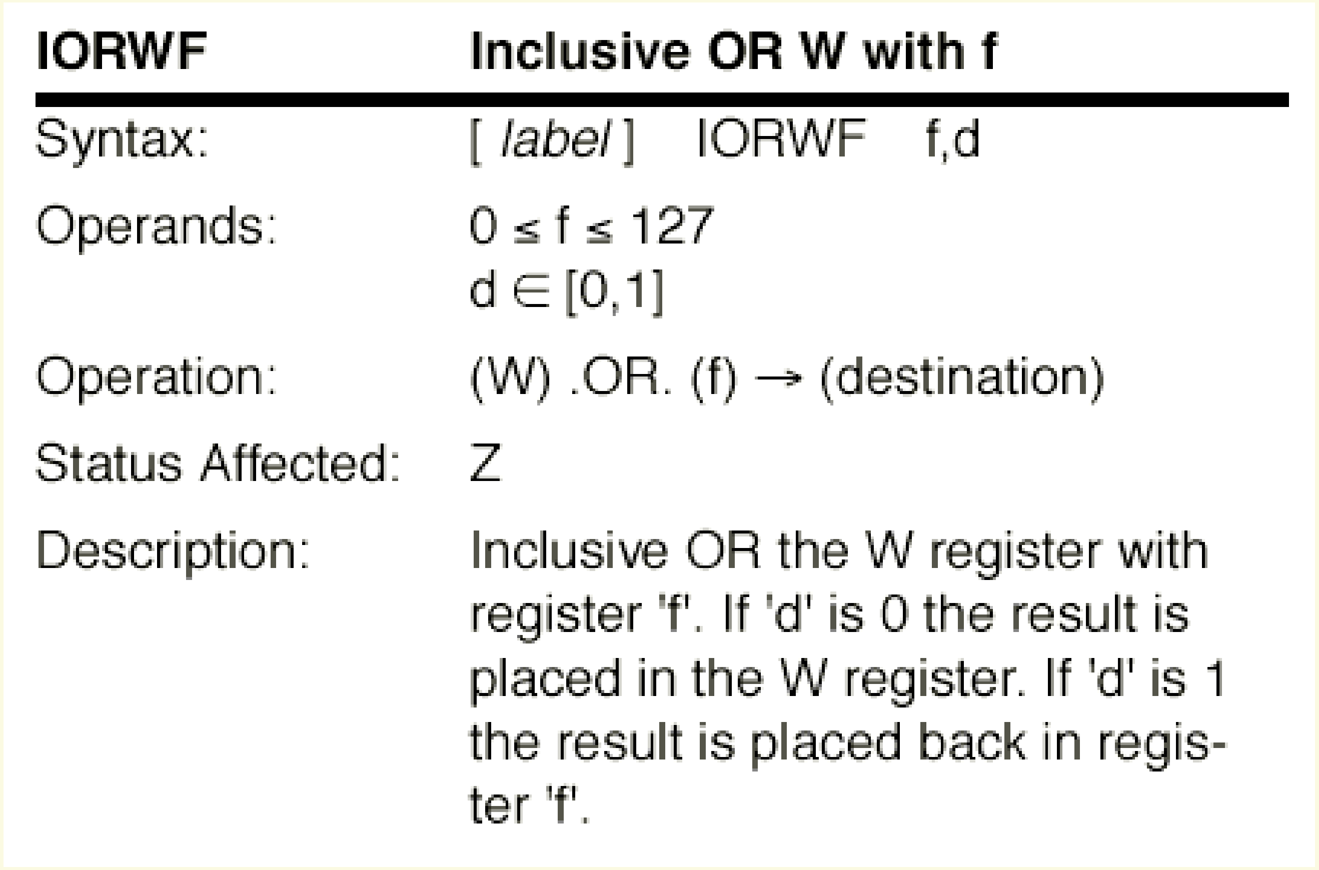
INCFSZ	**Increment f, Skip if 0**
Syntax:	[*label*] INCFSZ f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	$(f) + 1 \rightarrow$ (destination), skip if result = 0
Status Affected:	None
Description:	The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'. If the result is 1, the next instruction is executed. If the result is 0, a NOP is executed instead making it a 2TCY instruction.

Instruction set



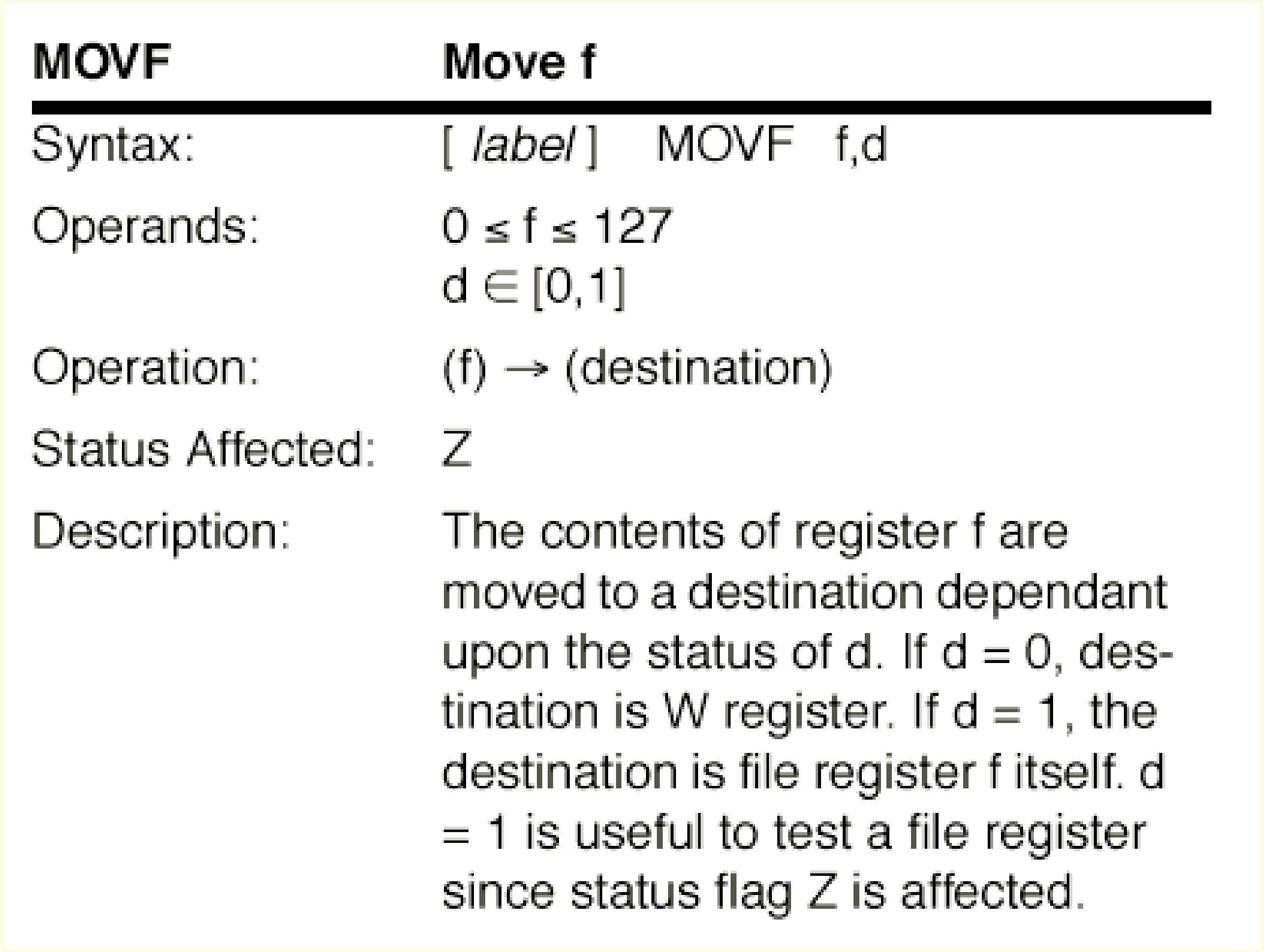
IORLW	Inclusive OR Literal with W
Syntax:	[*label*] IORLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .OR. k $\rightarrow$ (W)
Status Affected:	Z
Description:	The contents of the W register are OR'ed with the eight bit literal 'k'. The result is placed in the W register.

Instruction set



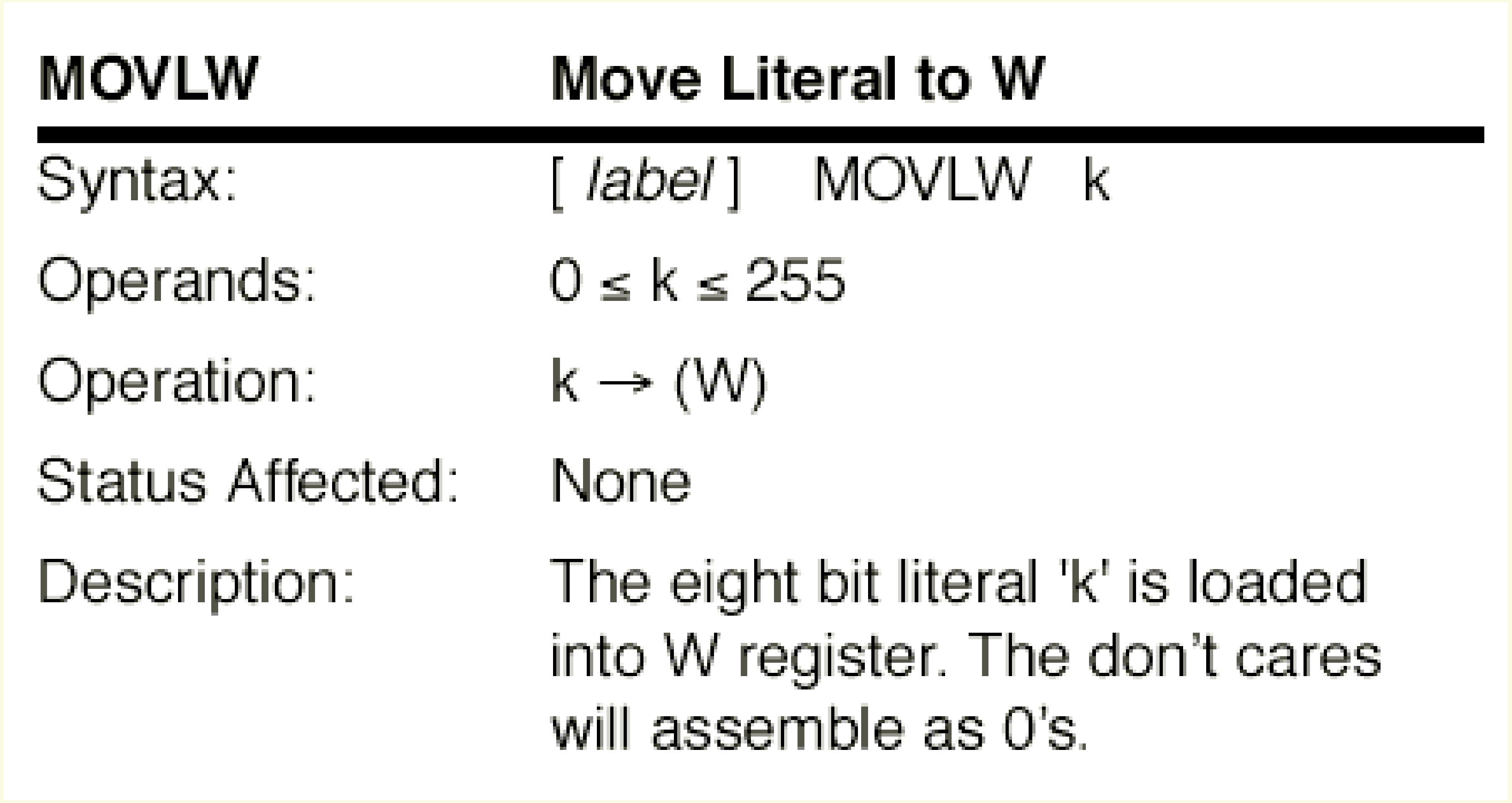
IORWF	**Inclusive OR W with f**
Syntax:	[*label*] IORWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) .OR. (f) $\rightarrow$ (destination)
Status Affected:	Z
Description:	Inclusive OR the W register with register 'f'. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Instruction set



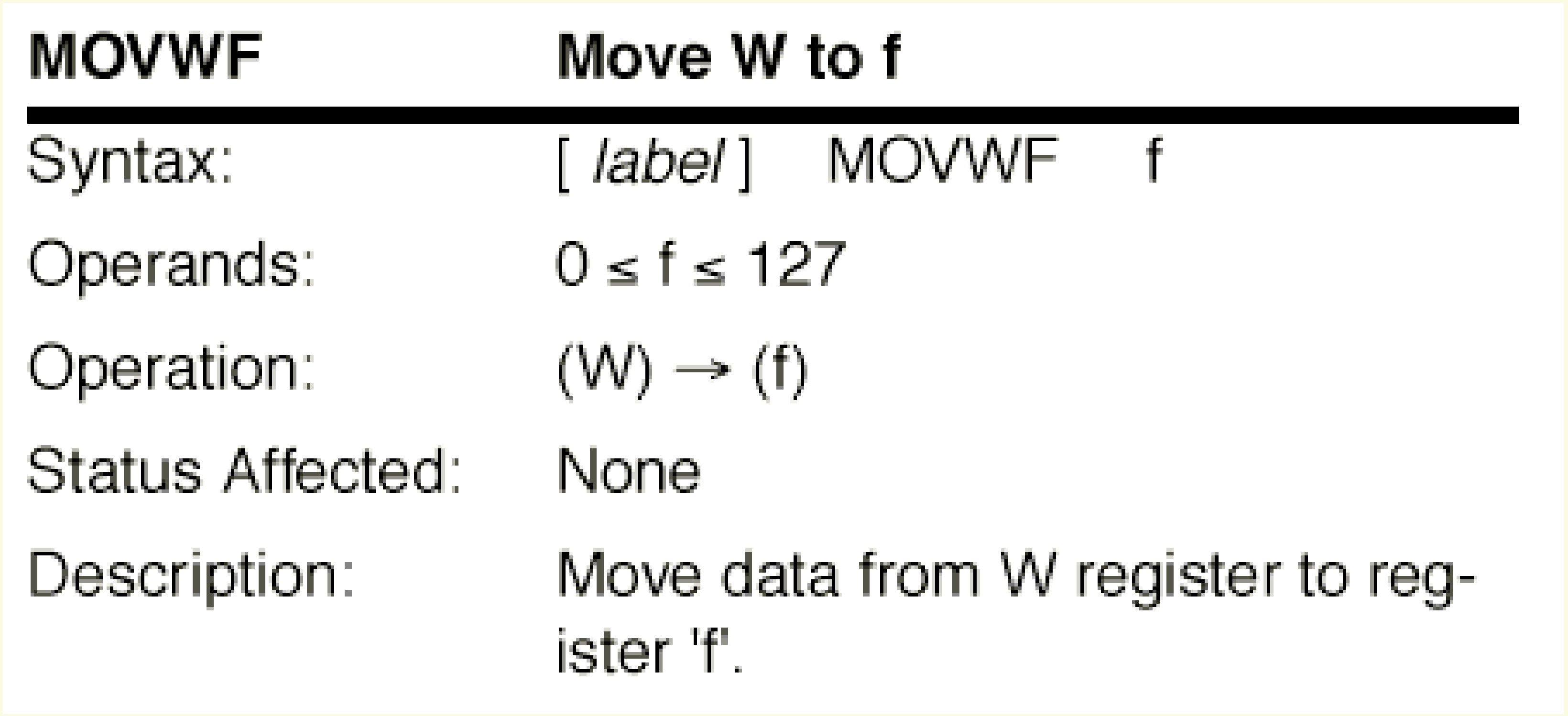
MOVF	**Move f**
Syntax:	[*label*] MOVF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	$(f) \rightarrow$ (destination)
Status Affected:	Z
Description:	The contents of register f are moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register since status flag Z is affected.

Instruction set



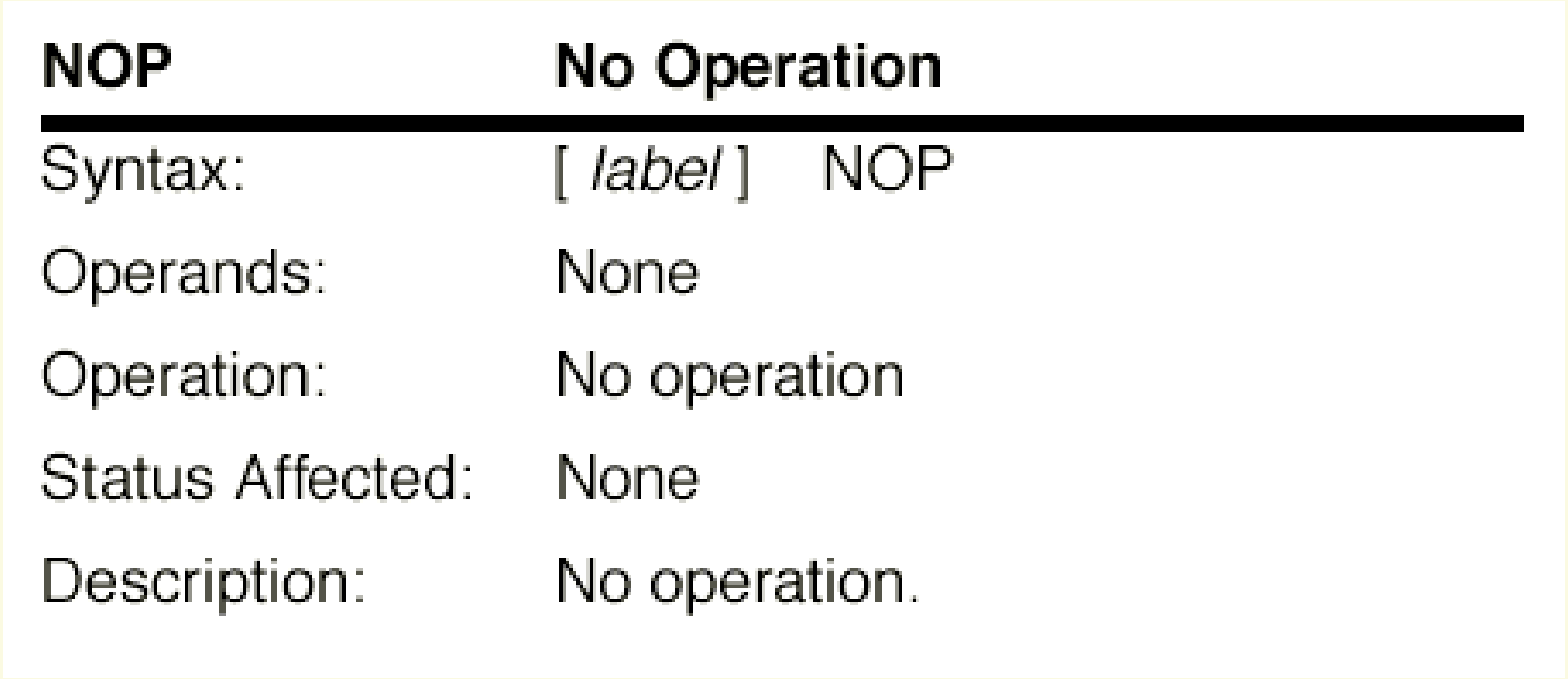
MOVLW	Move Literal to W
Syntax:	[*label*] MOVLW k
Operands:	$0 \le k \le 255$
Operation:	$k \rightarrow (W)$
Status Affected:	None
Description:	The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

Instruction set



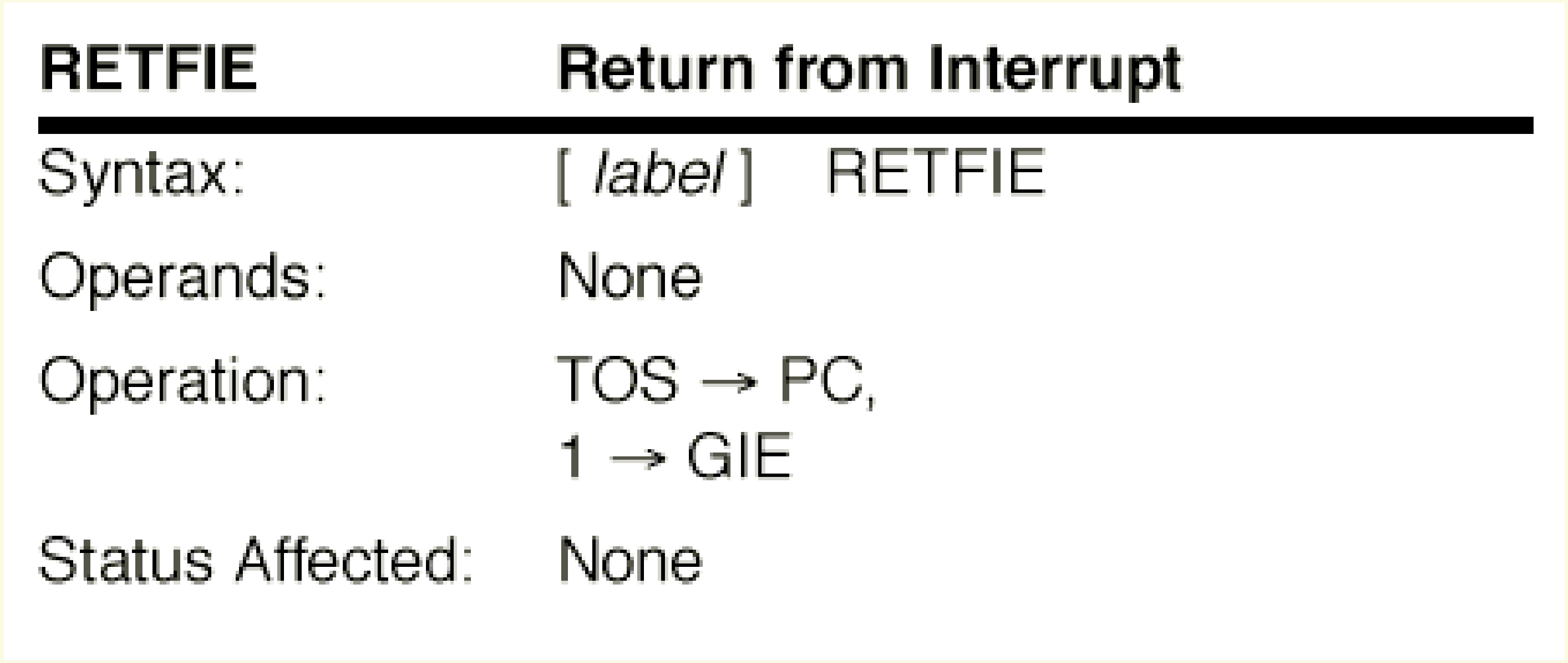
MOVWF	**Move W to f**
Syntax:	[*label*] MOVWF f
Operands:	$0 \le f \le 127$
Operation:	$(W) \rightarrow (f)$
Status Affected:	None
Description:	Move data from W register to register 'f'.

Instruction set



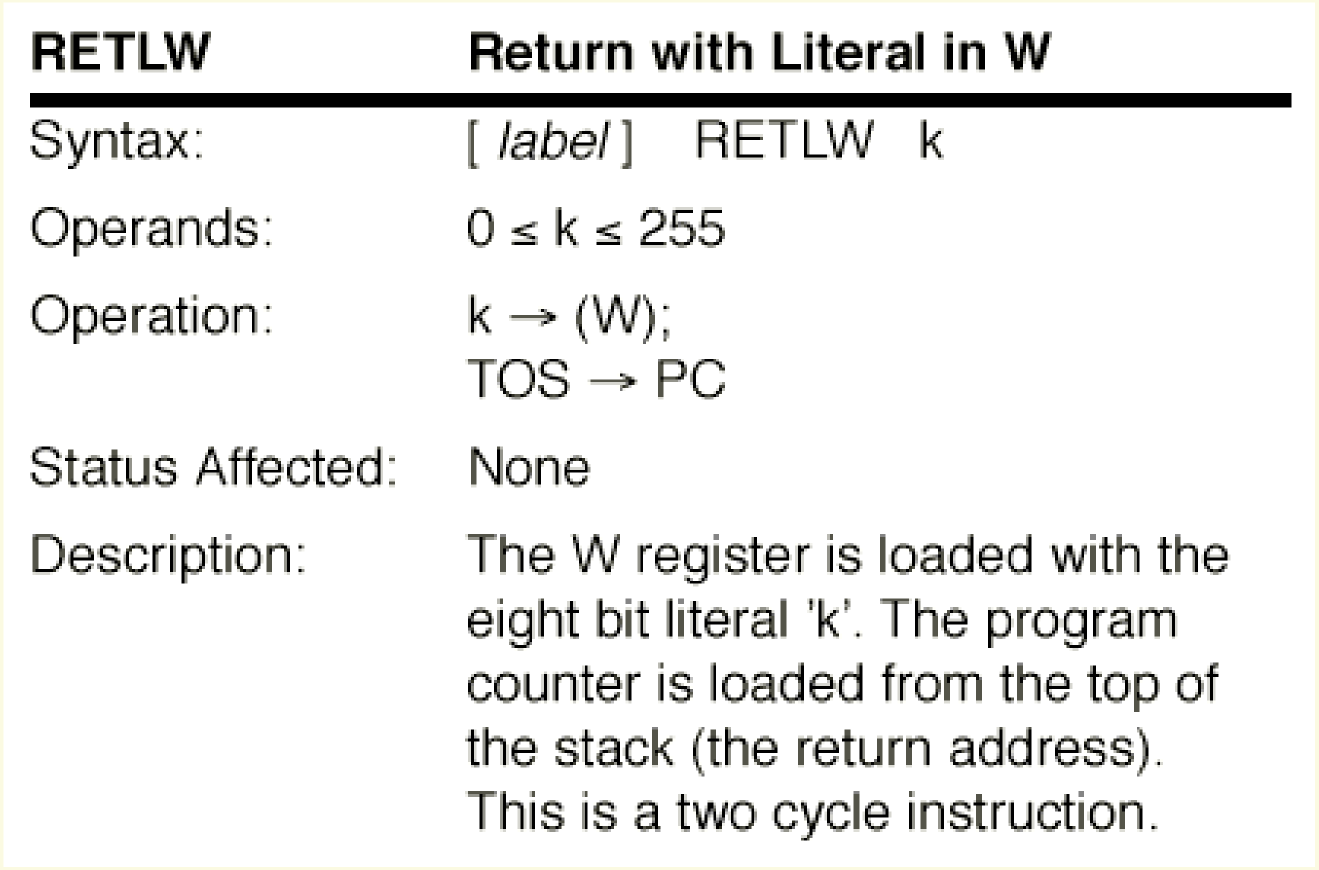
NOP	No Operation
Syntax:	[*label*] NOP
Operands:	None
Operation:	No operation
Status Affected:	None
Description:	No operation.

Instruction set



RETFIE	Return from Interrupt
Syntax:	[*label*] RETFIE
Operands:	None
Operation:	TOS $\rightarrow$ PC, 1 $\rightarrow$ GIE
Status Affected:	None

Instruction set



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
Description:	The W register is loaded with the eight bit literal 'k'. The program counter is loaded from the top of the stack (the return address). This is a two cycle instruction.

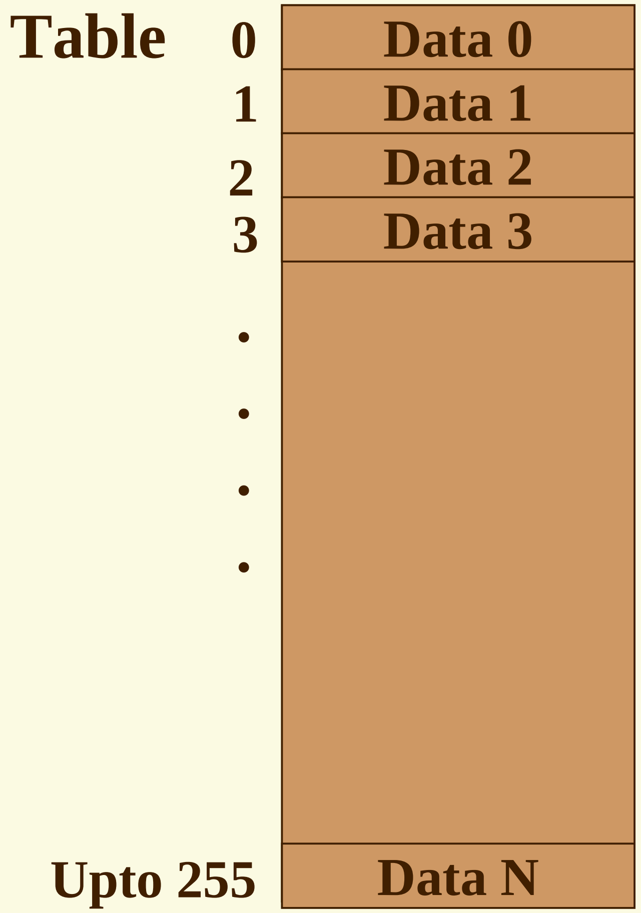
LookupTable - Program Memory

- ✔ Limited Data Memory Space for lookup table?
  - PIC16F877 - Data Memory - 368 bytes!
  - PIC16F877 - Program Memory - upto 8K
- ✔ Put LookupTable in Program Memory
  - How?
    - CALL
    - RETLW

Setup LookupTable in Program memory

- ✔ Setup the Table in Program Memory

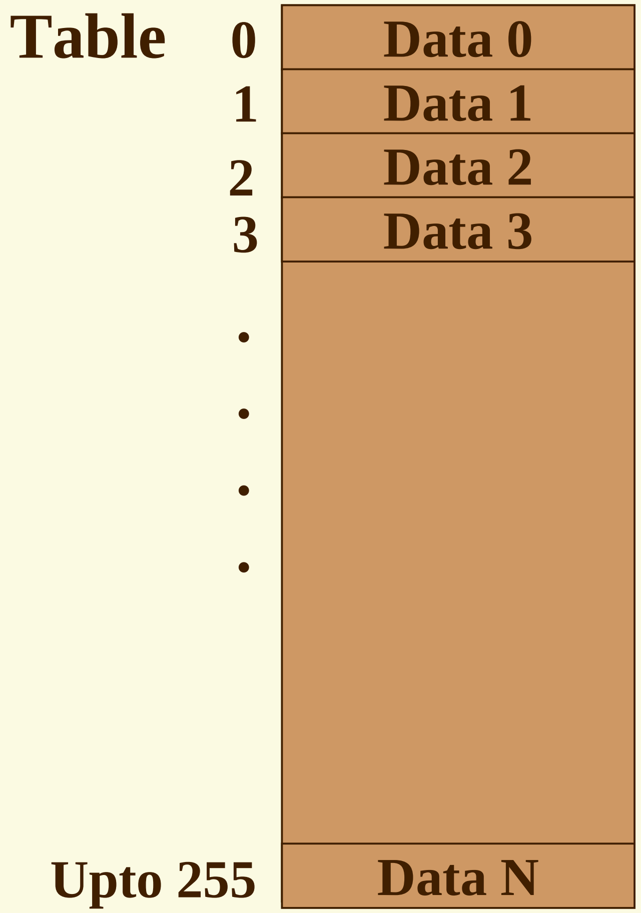
```
TableName  addwf  PCL,F
           retlw Data0
           retlw Data1
           retlw Data2
           .
           .
           retlw DataN
```



Using LookupTable

- ✔ Load W register with table index
- ✔ Call TableName
- ✔ The corresponding data will return in the W register.

```
movlw 2          ;table index =2
call TableName   ;go get data
                 ; from the stored
                 ; table
…….              ; W register will
                 ; have Data2
```



Using LookupTable: Example

- **Unpacked BCD Addition**

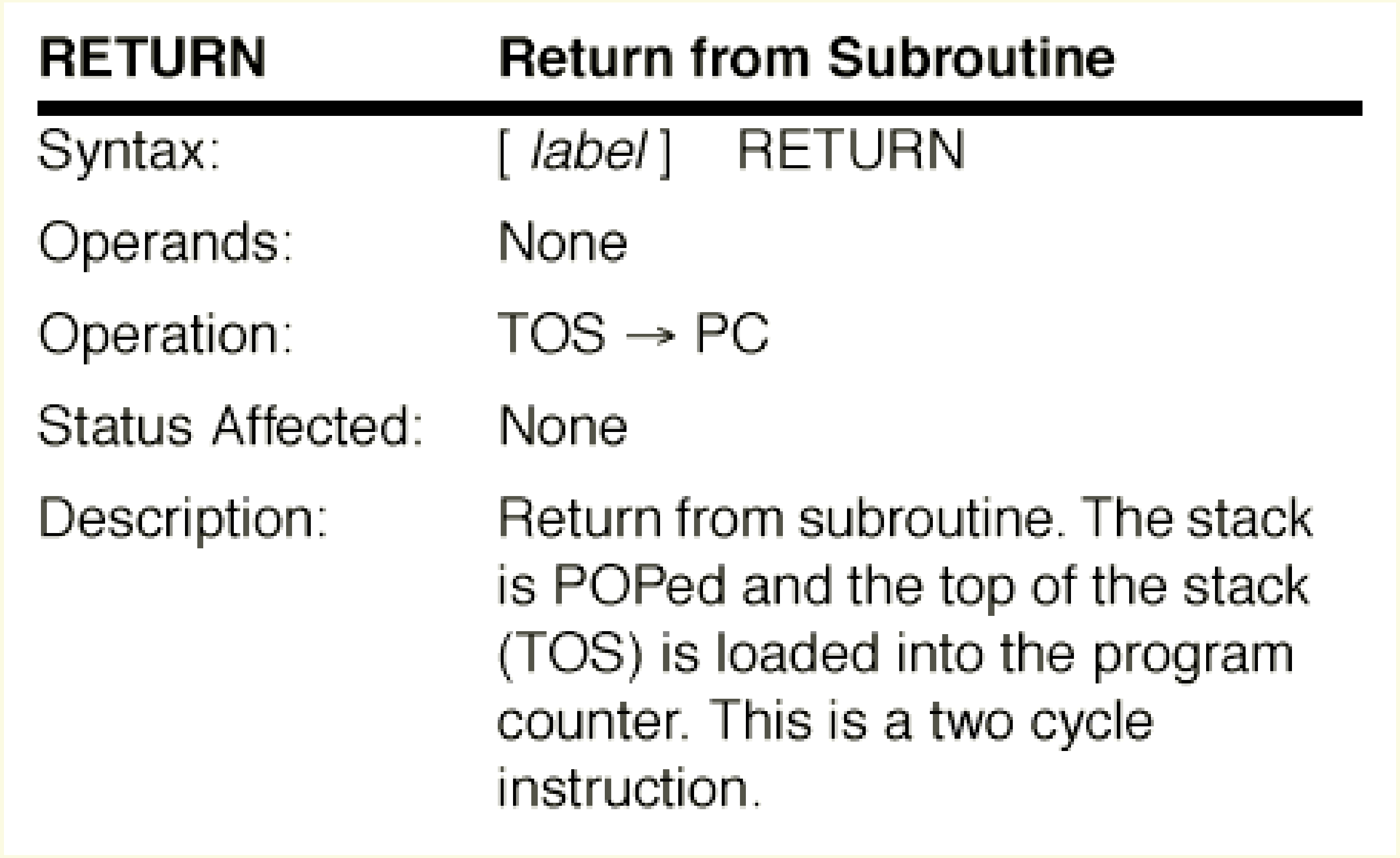
```
movf    X,W              ;get X
addwf Y,W                ;W = X+Y
call     Adjust_BCD  ;W= BCD_result
```

```
Adjust_BCD  addwf PCL,F
            retlw 0x00  }  0-9 can
            retlw 0x01  }  use return
            .
            .
            retlw 0x18
```

BCD_Table

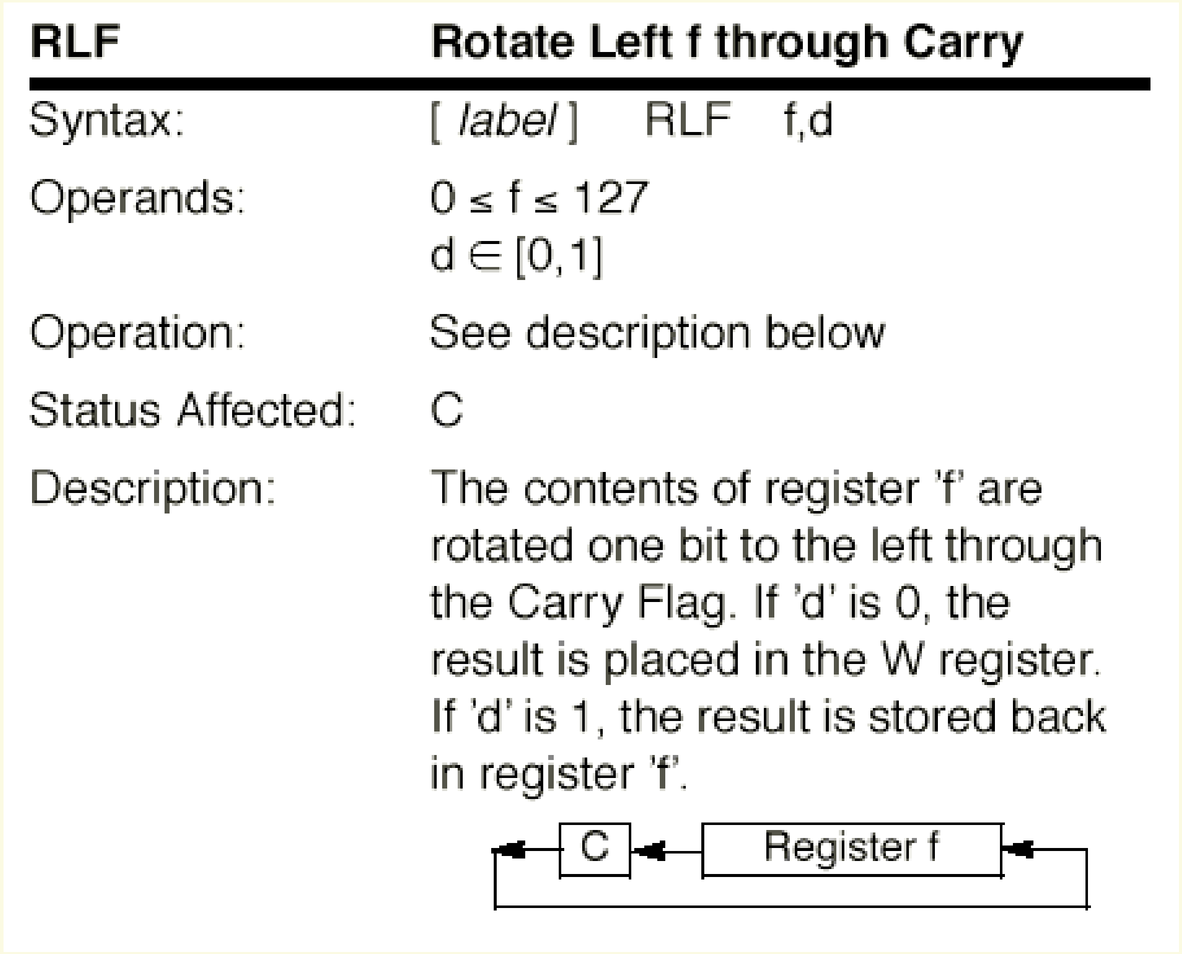
0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
.	
9	0000 1001
10	0001 0000
11	0001 0001
12	0001 0010
.	
17	0001 0111
18	0001 1000

Instruction set

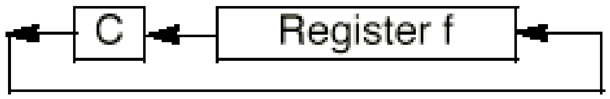


RETURN	**Return from Subroutine**
Syntax:	[*label*] RETURN
Operands:	None
Operation:	TOS → PC
Status Affected:	None
Description:	Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two cycle instruction.

Instruction set

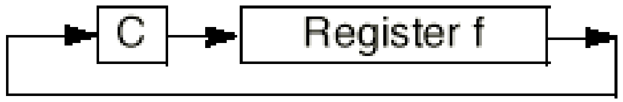


RLF	**Rotate Left f through Carry**
Syntax:	[*label*] RLF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C
Description:	The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is stored back in register 'f'.

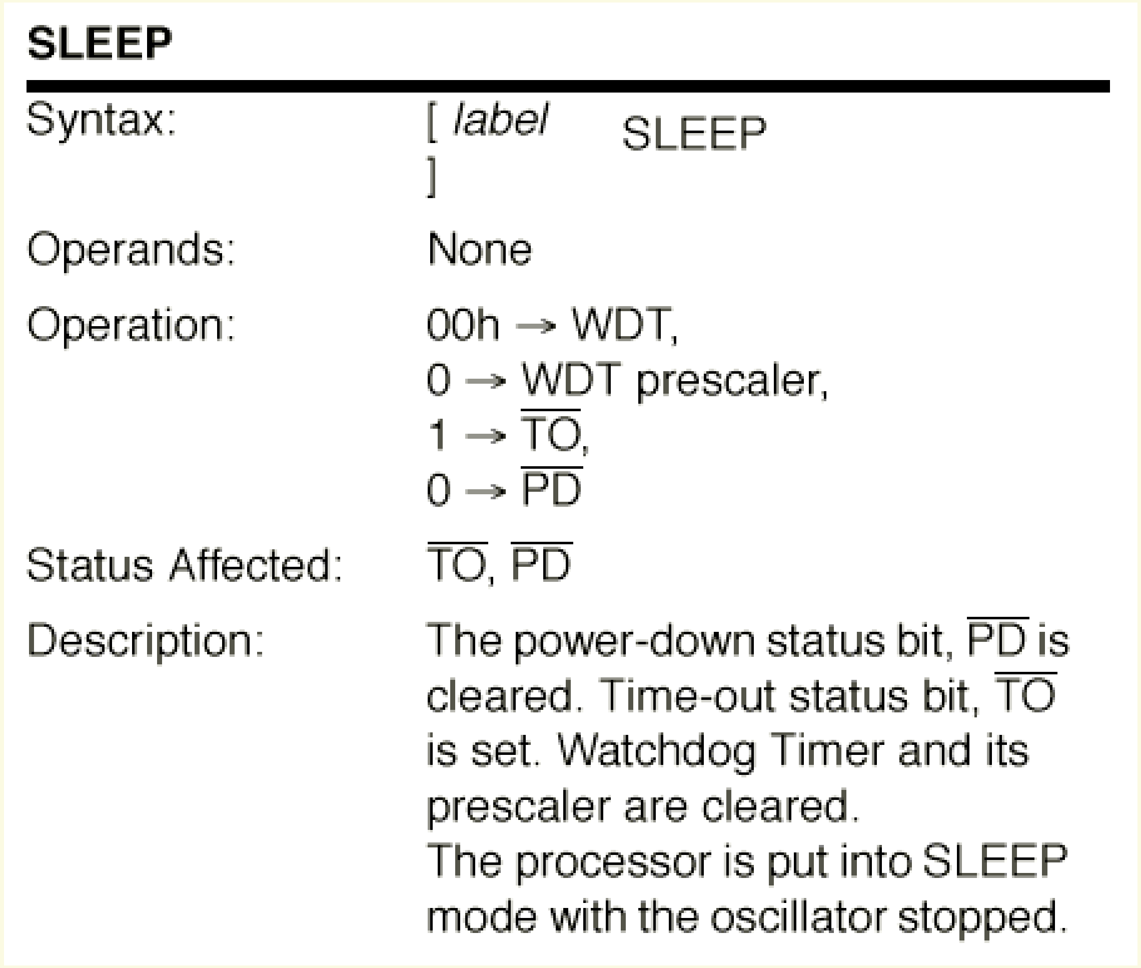


Instruction set

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
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'.



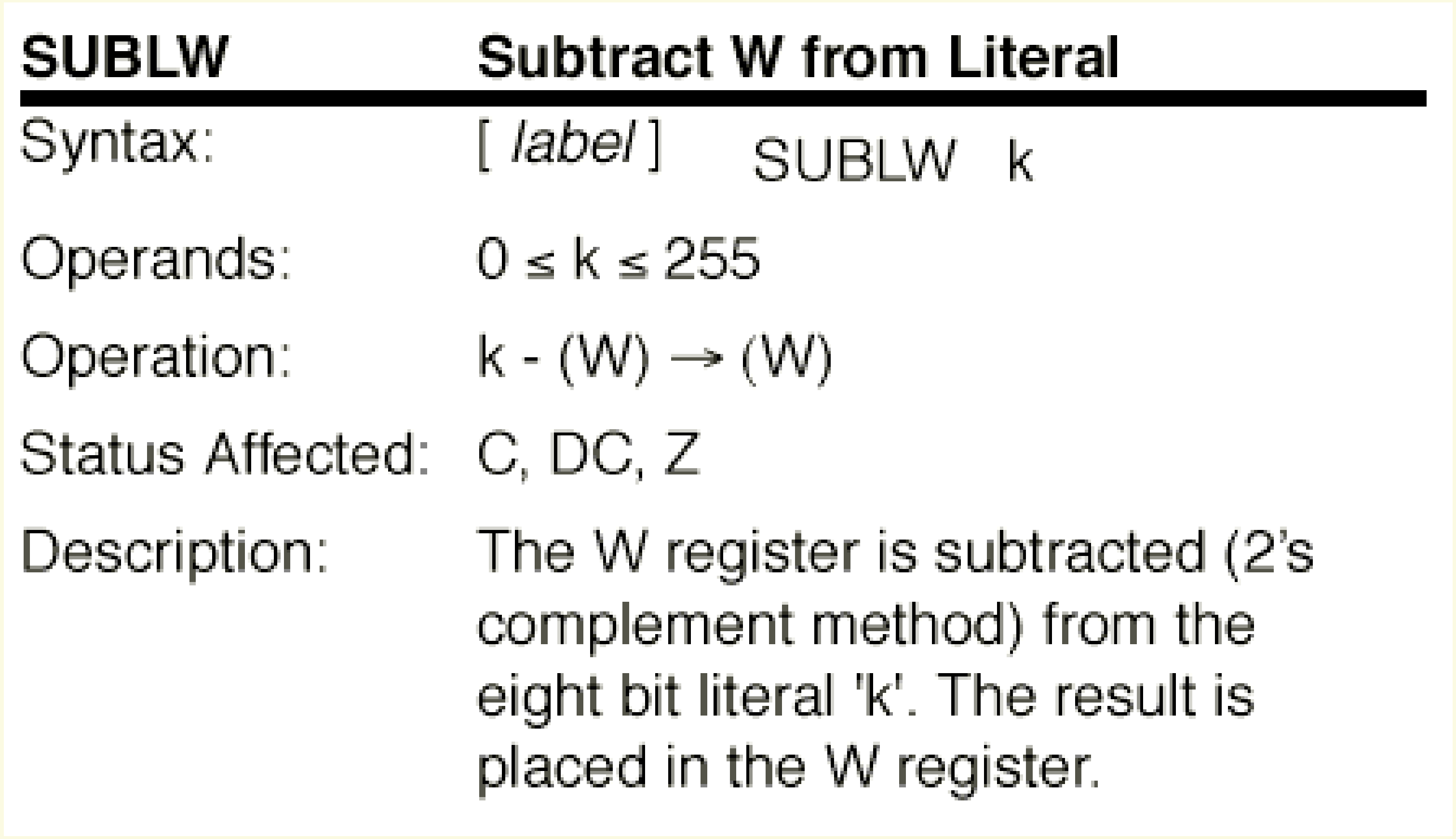
Instruction set



SLEEP

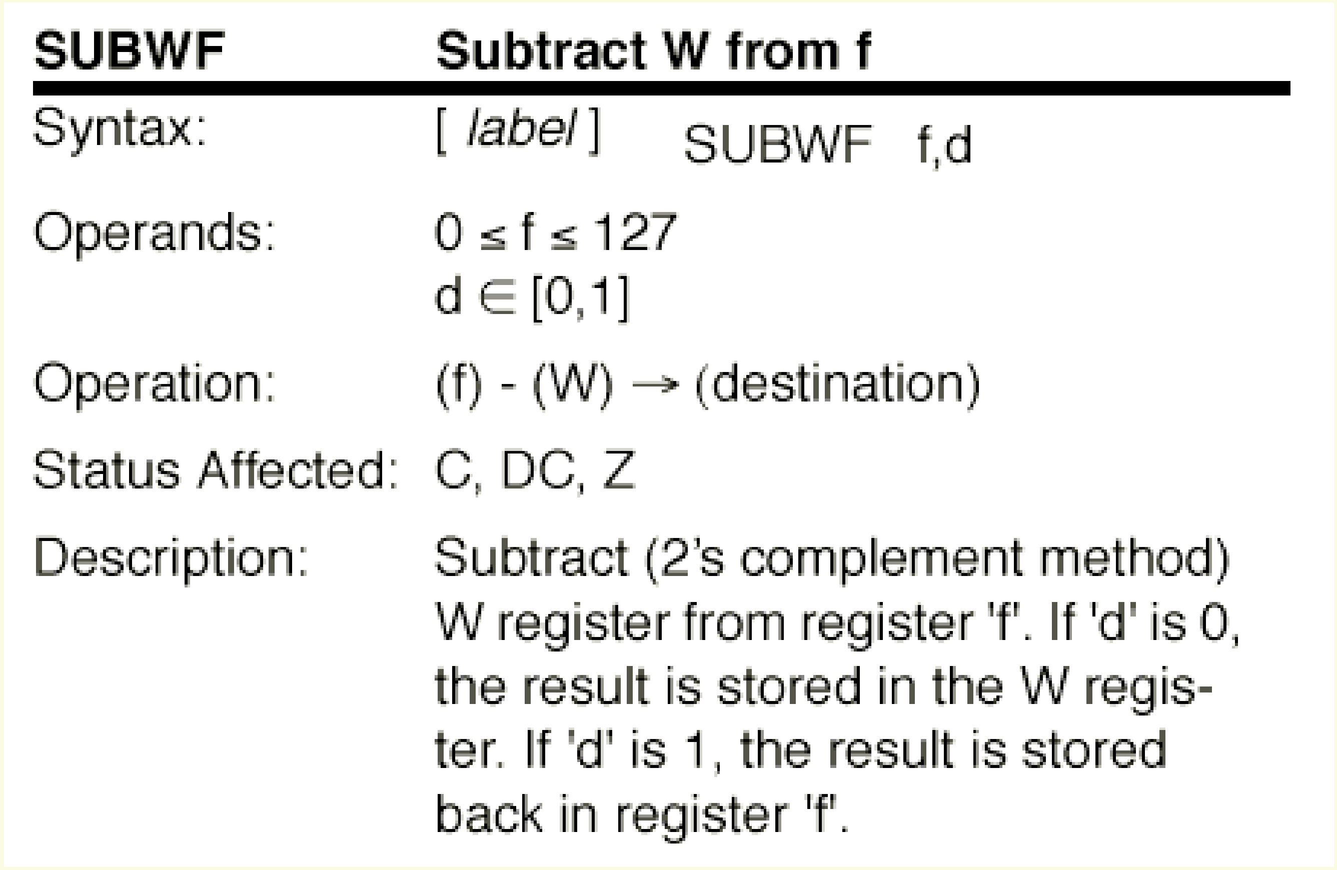
Syntax:	[*label*] SLEEP
Operands:	None
Operation:	00h → WDT, 0 → WDT prescaler, 1 → $\overline{\text{TO}}$, 0 → $\overline{\text{PD}}$
Status Affected:	$\overline{\text{TO}}$, $\overline{\text{PD}}$
Description:	The power-down status bit, $\overline{\text{PD}}$ is cleared. Time-out status bit, $\overline{\text{TO}}$ is set. Watchdog Timer and its prescaler are cleared. The processor is put into SLEEP mode with the oscillator stopped.

Instruction set



SUBLW	**Subtract W from Literal**
Syntax:	[*label*] SUBLW k
Operands:	0 ≤ k ≤ 255
Operation:	k - (W) → (W)
Status Affected:	C, DC, Z
Description:	The W register is subtracted (2’s complement method) from the eight bit literal 'k'. The result is placed in the W register.

Instruction set



SUBWF	**Subtract W from f**
Syntax:	[*label*] SUBWF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	(f) - (W) $\rightarrow$ (destination)
Status Affected:	C, DC, Z
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'.

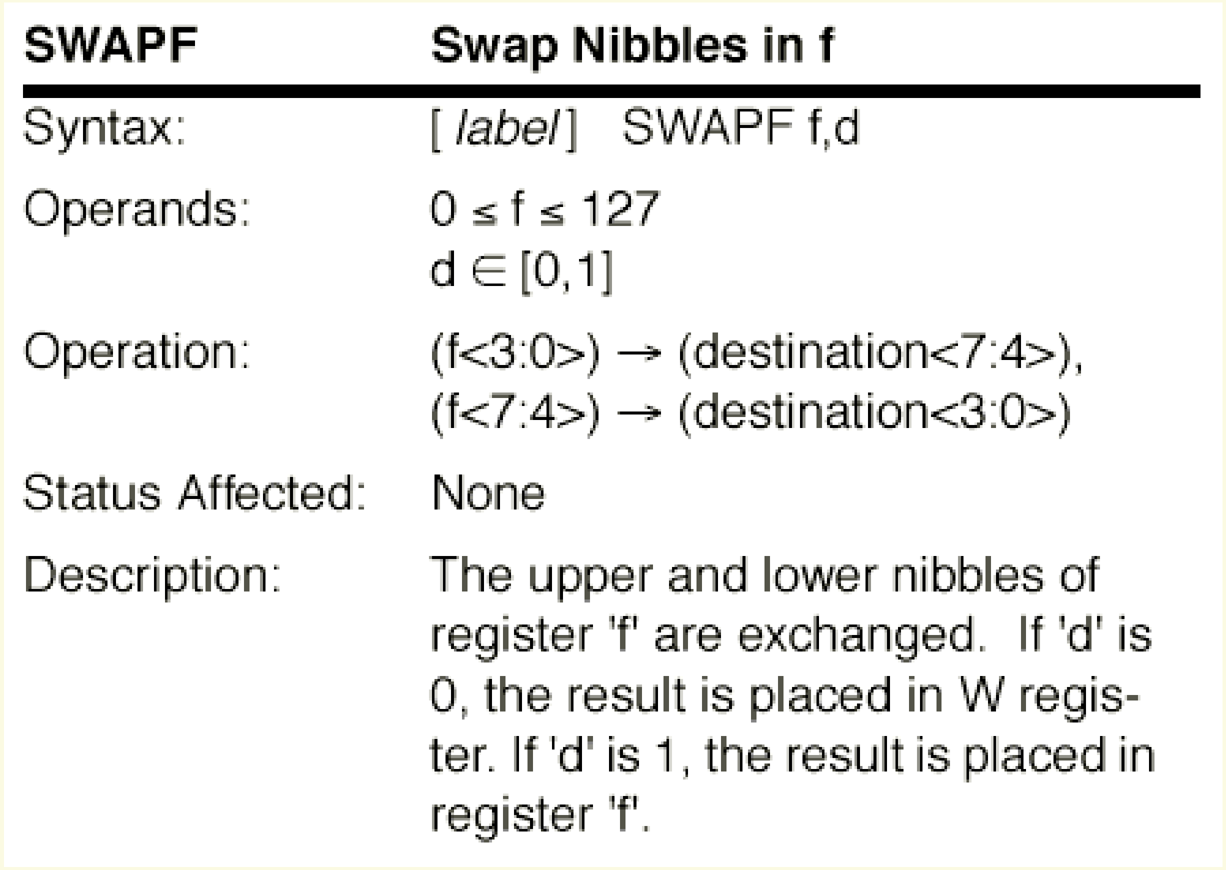
STATUS register: C, Carry bit/$\overline{\text{Borrow}}$

- ✔ Subtract Instruction
  - – PIC ALU performs 8-bit unsigned subtraction by forming the two's complement of the subtrahend
    - Examples

```
movlw  8     ;load W with 8

sublw   10   ;sub. W from 10


   0000 1010  ; 10
+ 1111 1000  ; 2'sComp. Of 8
1 0000 0010  ; 2
   C/Borrow bit (no borrow)

W = 2  and C bit =1
```

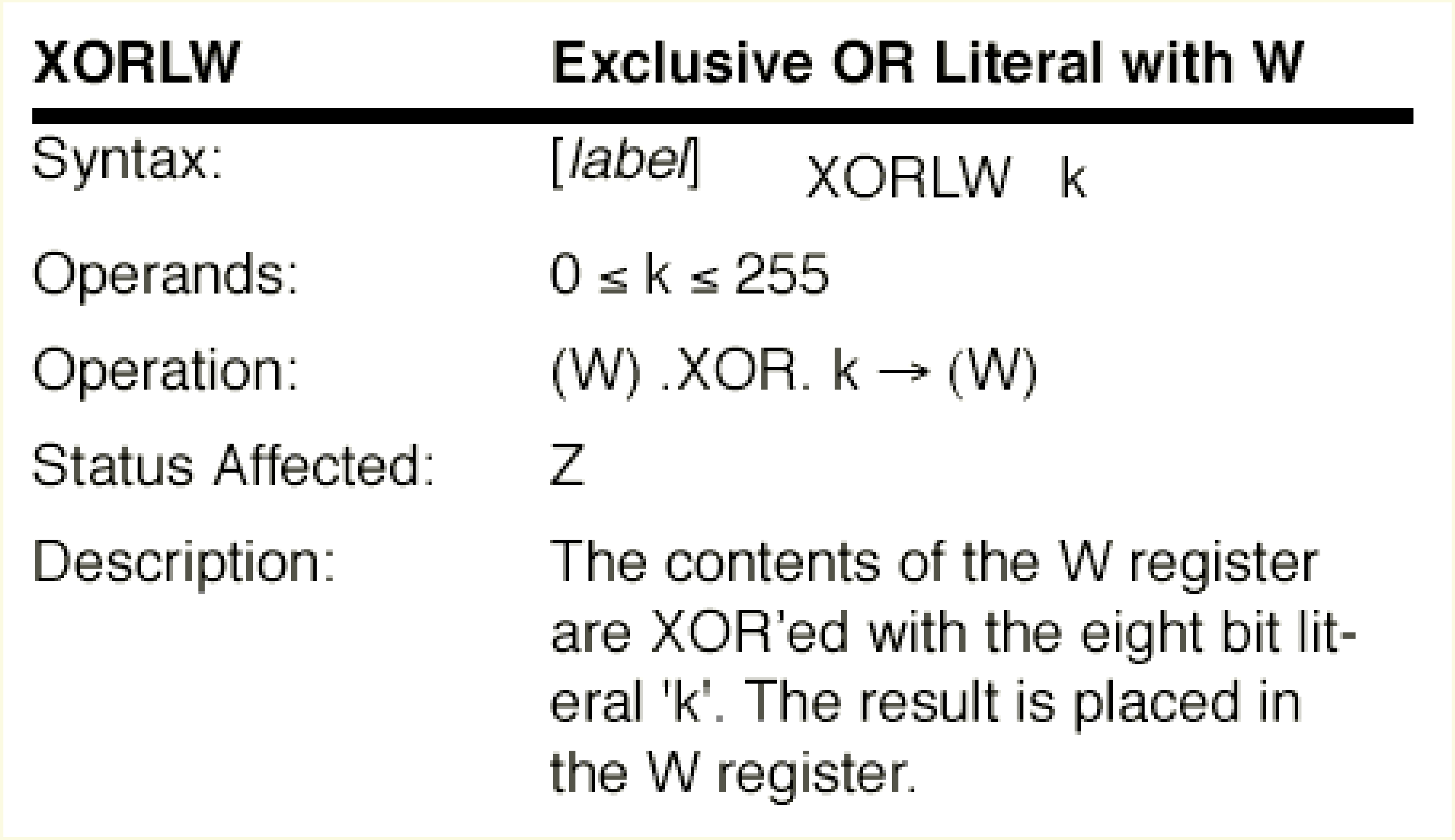
```
movlw  10    ;load W with 10

sublw   8      ;sub. W from 8


   0000 1000  ; 8
+ 1111 0110  ; 2'sComp. Of 10
0  1111 1110  ; 254!
   C/Borrow bit  (borrow)

W = 254  and C bit =0
```

Instruction set



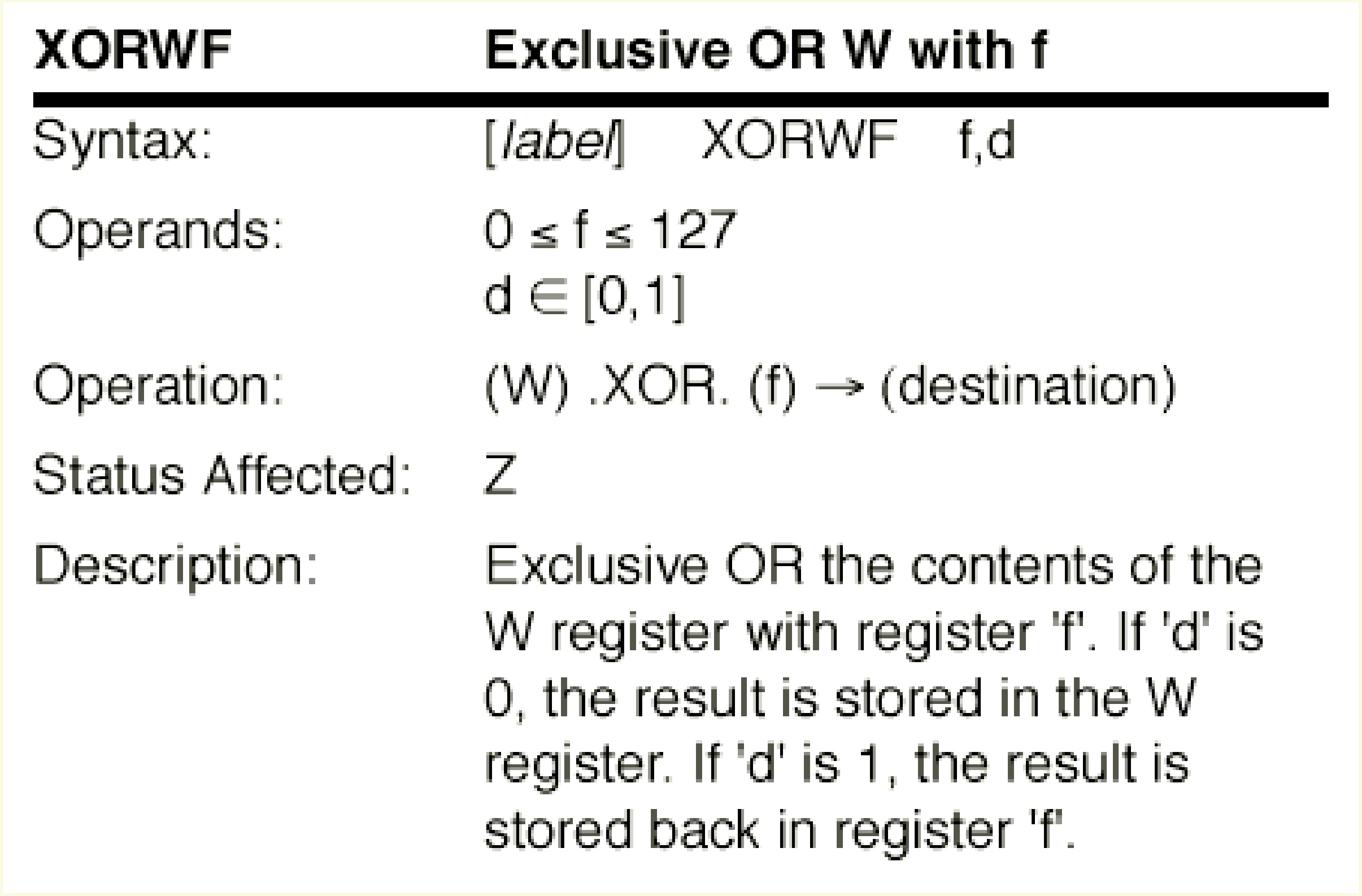
SWAPF	**Swap Nibbles in f**
Syntax:	[*label*] SWAPF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	$(f<3:0>) \rightarrow (destination<7:4>)$, $(f<7:4>) \rightarrow (destination<3:0>)$
Status Affected:	None
Description:	The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0,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

Instruction set



XORLW	**Exclusive OR Literal with W**
Syntax:	[*label*] XORLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .XOR. k → (W)
Status Affected:	Z
Description:	The contents of the W register are XOR'ed with the eight bit literal 'k'. The result is placed in the W register.

Instruction set



XORWF	**Exclusive OR W with f**
Syntax:	[*label*] XORWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) .XOR. (f) → (destination)
Status Affected:	Z
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'.