Jeu d'instructions

Directives d'assemblage

PIC16F877

MPASM Quick Reference Guide

This Quick Reference Guide gives all the instructions, directives, and command line options for the Microchip MPASM Assembler.

MPASM Directive Language Summary

Directive	Description	Syntax
CONSTANT	Declare Symbol Constant	constant <label> [= <expr>, ...,<label> [= <expr>]]
#DEFINE	Define Text Substitution	#define <name> [[(<arg>,...,<arg>)]<value>]
END	End Program Block	end
EQU	Define Assembly Constant	<label> equ <expr>
ERROR	Issue an Error Message	error "<text_string>"
ERROR-LEVEL	Set Messge Level	errorlevel 0|1|2|<+-><msg>
#INCLUDE	Include Source File	include <<include_file>> include "<include_file>"
LIST	Listing Options	list [<option>[,...,<option>]]
MESSG	User Defined Message	messg "<message_text>"
NOLIST	Turn off Listing Output	nolist
ORG	Set Program Origin	<label> org <expr>
PAGE	Insert Listing Page Eject	page
PROCESSOR	Set Processor Type	processor <processsor_type>
RADIX	Specify Default Radix	radix <default_radix>
SET	Assign Value to Variable	<label> set <expr>
SPACE	Insert Blank Listing Lines	space [<expr>]
SUBTITLE	Specify Program Subtitle	subtitl "<sub_text>"
TITLE	Specify Program Title	title "<title_text>"
#UNDEFINE	Delete a Substitution Label	#undefine <label>
VARIABLE	Declare Symbol Variable	variable <label> [= <expr>,...,]
ELSE	Begin Alternative Assembly to IF	else
ENDIF	End Conditional Assembly	endif
ENDW	End a While Loop	endw
IF	Begin Conditional ASM Code	if <expr>
IFDEF	Execute If Symbol Defined	ifdef <label>
IFNDEF	Execute If Symbol Not Defined	ifndef <label>
WHILE	Perform Loop While True	while <expr>

MPASM Directive Language Summary (Continued)

Directive	Description	Syntax
_ _BADRAM	Specify invalid RAM locations	_ _badram <expr>
CBLOCK	Define Block of Constants	cblock [<expr>]
_ _CONFIG	Set configuration bits	_ _config <expr> OR _ _config <addr>, <expr>
DA	Pack Strings in 14-bit Memory	[<label>] da <expr> [, <expr2>, ..., <exprn>]
DATA	Create Numeric/Text Data	data <expr>,[,<expr>,...,<expr>] data "<text_string>"[,"<text_string>",...]
DB	Declare Data of One Byte	db <expr>[,<expr>,...,<expr>]
DE	Declare EEPROM Data	de <expr>[,<expr>,...,<expr>]
DT	Define Table	dt <expr>[,<expr>,...,<expr>]
DW	Declare Data of One Word	dw <expr> [,<expr>,...,<expr>]
ENDC	End CBlock	endc
FILL	Specify Memory Fill Value	fill <expr>, <count>
_ _ IDLOCS	Set ID locations	_ _idlocs <expr>
_ _MAXRAM	Specify max RAM adr	_ _maxram <expr>
RES	Reserve Memory	res <mem_units>
ENDM	End a Macro Definition	endm
EXITM	Exit from a Macro	exitm
EXPAND	Expand Macro Listing	expand
LOCAL	Declare Local Macro Variable	local <label> [,<label>]
MACRO	Declare Macro Definition	<label> macro [<arg>,...,<arg>]
NOEXPAND	Turn off Macro Expansion	noexpand
BANKISEL	Select Bank for indirect	bankisel <label>
BANKSEL	Select RAM bank	banksel <label>
CODE	Executable code section	[<name>] code [<address>]
EXTERN	Declare external label	extern <label> [,<label>]
GLOBAL	Export defined label	extern <label> [.<label>]
IDATA	Initialized data section	[<name>] idata [<address>]
PAGESEL	Select ROM page	pagesel <label>
UDATA	Uninitialized data section	[<name>] udata [<address>]
UDATA_ACS	Access uninit data sect	[<name>] udata_acs [<address>]
UDATA_OVR	Overlay uninit data sect	[<name>] udata_ovr [<address>]
UDATA_SHR	Shared uninit data sect	[<name>] udata_shr [<address>]

MPASM Radix Types Supported

Radix	Syntax	Example
Decimal	D'<digits>' .<digits>	D'100' .100
Hexadecimal (default)	H'<hex_digits>' 0x<hex_digits>	H'9f' 0x9f
Octal	O'<octal_digits>'	O'777'
Binary	B'<binary_digits>'	B'00111001'
Character (ASCII)	'<character>' A'<Character>'	A'C' 'C'

MPLINK Command Line Options

Option	Description
/o filename	Specify output file 'filename'. Default is a.out.
/m filename	Create map file 'filename'.
/l pathlist	Add directories to library search path.
/k pathlist	Add directories to linker script search path.
/n length	Specify number of lines per listing page.
/h, /?	Display help screen.
/a hexformat	Specify format of hex output file.
/q	Quiet mode.
/d	Don't create an absolute listing file.

Key to 12, 14, and 16-bit PICmicro Family Instruction Sets

Field	Description
b	Bit address within an 8 bit file register
d	Destination select; d = 0 Store result in W (f0A). d = 1 Store result in file register f. Default is d = 1.
f	Register file address (0x00 to 0xFF)
k	Literal field, constant data or label
W	Working register (accumulator)
x	Don't care location
i	Table pointer control; i = 0 Do not change. i = 1 Increment after instruction execution.
p	Peripheral register file address (0x00 to 0x1f)
t	Table byte select; t = 0 Perform operation on lower byte. t = 1 Perform operation on upper byte.
PH:PL	Multiplication results registers

ASCII Character Set

		Most Significant Character							
	Hex			2	3	4	5	6	7
Least Significant Character	0			Space	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	–	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

MPLIB Usage Format

MPLIB is invoked with the following syntax:

```
mplib [/q] /{ctdrx} LIBRARY [MEMBER...]
```

options:

/c	create library;	creates a new LIBRARY with the listed MEMBER(s)
/t	list members;	prints a table showing the names of the members in the LIBRARY
/d	delete member;	deletes MEMBER(s) from the LIBRARY; if no MEMBER is specified the LIBRARY is not altered
/r	add/replace member;	if MEMBER(s) exist in the LIBRARY, then they are replaced, otherwise MEMBER is appended to the end of the LIBRARY
/x	extract member;	if MEMBER(s) exist in the LIBRARY, then they are extracted. If no MEMBER is specified, all members will be extracted
/q	quiet mode;	no output is displayed

MPLIB Usage Examples

Suppose a library named `dsp.lib` is to be created from three object modules named `fft.o`, `fir.o`, and `iir.o`. The following command line would produce the desired results:

```
mplib /c dsp.lib fft.o fir.o iir.o
```

To display the names of the object modules contained in a library file names `dsp.lib`, the following command line would be appropriate:

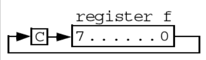
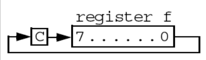
```
mplib /t dsp.lib
```

14-Bit Core Instruction Set

14-Bit Core Literal and Control Operations

Hex	Mnemonic	Description	Function
3Ekk	ADDLW k	Add literal to W	k + W → W
39kk	ANDLW k	AND literal and W	k .AND. W → W
2kkk	CALL k	Call subroutine	PC + 1 → TOS, k → PC
0064	CLRWDT T	Clear watchdog timer	0 → WDT (and Prescaler)
2kkk	GOTO k	Goto address (k is nine bits)	k → PC(9 bits)
38kk	IORLW k	Incl. OR literal and W	k .OR. W → W
30kk	MOVLW k	Move Literal to W	k → W
0062	OPTION	Load OPTION register	W → OPTION Register
0009	RETFIE	Return from Interrupt	TOS → PC, 1 → GIE
34kk	RETLW k	Return with literal in W	k → W, TOS → PC
0008	RETURN	Return from subroutine	TOS → PC
0063	SLEEP	Go into Standby Mode	0 → WDT, stop oscillator
3Ckk	SUBLW k	Subtract W from literal	k - W → W
006f	TRIS f	Tristate port f	W → I/O control reg f
3Akk	XORLW k	Exclusive OR literal and W	k .XOR. W → W

14-Bit Core Byte Oriented File Register Operations

Hex	Mnemonic	Description	Function
07ff	ADDWF f,d	Add W and f	W + f → d
05ff	ANDWF f,d	AND W and f	W .AND. f → d
018f	CLRF f	Clear f	0 → f
0100	CLRW	Clear W	0 → W
09ff	COMF f,d	Complement f	.NOT. f → d
03ff	DECF f,d	Decrement f	f - 1 → d
0Bff	DECFSZ f,d	Decrement f, skip if zero	f - 1 → d, skip if 0
0Aff	INCF f,d	Increment f	f + 1 → d
0Fff	INCFSZ f,d	Increment f, skip if zero	f + 1 → d, skip if 0
04ff	IORWF f,d	Inclusive OR W and f	W .OR. f → d
08ff	MOVF f,d	Move f	f → d
008f	MOVWF f	Move W to f	W → f
0000	NOP	No operation	
0Dff	RLF f,d	Rotate left f	
0Cff	RRF f,d	Rotate right f	
02ff	SUBWF f,d	Subtract W from f	f - W → d
0Eff	SWAPF f,d	Swap halves f	f(0:3) ↔ f(4:7) → d
06ff	XORWF f,d	Exclusive OR W and f	W .XOR. f → d
1bff	BCF f,b	Bit clear f	0 → f(b)
1bff	BSF f,b	Bit set f	1 → f(b)
1bff	BTFSC f,b	Bit test, skip if clear	skip if f(b) = 0
1bff	BTFSS f,b	Bit test, skip if set	skip if f(b) = 1

12-Bit/14-Bit Core Special Instruction Mnemonics

Mnemonic		Description	Equivalent Operation(s)		Status
ADDCF	f,d	Add Carry to File	BTFSC INCF	3,0 f,d	Z
ADDDCF	f,d	Add Digit Carry to File	BTFSC INCF	3,1 f,d	Z
B	k	Branch	GOTO	k	–
BC	k	Branch on Carry	BTFSC GOTO	3,0 k	–
BDC	k	Branch on Digit Carry	BTFSC GOTO	3,1 k	–
BNC	k	Branch on No Carry	BTFSS GOTO	3,0 k	–
BNDC	k	Branch on No Digit Carry	BTFSS GOTO	3,1 k	–
BNZ	k	Branch on No Zero	BTFSS GOTO	3,2 k	–
BZ	k	Branch on Zero	BTFSC GOTO	3,2 k	–
CLRC		Clear Carry	BCF	3,0	–
CLRDC		Clear Digit Carry	BCF	3,1	–
CLRZ		Clear Zero	BCF	3,2	–
LCALL	k	Long Call	BCF/BSF BCF/BSF CALL	0x0A,3 0x0A,4 k	–
LGOTO	k	Long GOTO	BCF/BSF BCF/BSF GOTO	0x0A,3 0x0A,4 k	–
MOVFW	f	Move File to W	MOVF	f,0	Z
NEGF	f,d	Negate File	COMF INCF	f,1 f,d	Z
SETC		Set Carry	BSF	3,0	–
SETDC		Set Digit Carry	BSF	3,1	–
SETZ		Set Zero	BSF	3,2	–
SKPC		Skip on Carry	BTFSS	3,0	–
SKPDC		Skip on Digit Carry	BTFSS	3,1	–
SKPNC		Skip on No Carry	BTFSC	3,0	–
SKPNDC		Skip on No Digit Carry	BTFSC	3,1	–
SKPNZ		Skip on Non Zero	BTFSC	3,2	–
SKPZ		Skip on Zero	BTFSS	3,2	–
SUBCF	f,d	Subtract Carry from File	BTFSC DECF	3,0 f,d	Z
SUBDCF	f,d	Subtract Digit Carry from File	BTFSC DECF	3,1 f,d	Z
TSTF	f	Test File	MOVF	f,1	Z

13.0 INSTRUCTION SET SUMMARY

Each PIC16F87X 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 PIC16F87X instruction set summary in Table 13-2 lists **byte-oriented**, **bit-oriented**, and **literal and control** operations. Table 13-1 shows the opcode field descriptions.

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

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

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

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

TABLE 13-1: OPCODE FIELD DESCRIPTIONS

Field	Description
`f`	Register file address (0x00 to 0x7F)
`W`	Working register (accumulator)
`b`	Bit address within an 8-bit file register
`k`	Literal field, constant data or label
`x`	Don't care location (= `0` or `1`). The assembler will generate code with x = `0`. It is the recommended form of use for compatibility with all Microchip software tools.
`d`	Destination select; d = `0`: store result in W, d = 1: store result in file register f. Default is d = `1`.
`PC`	Program Counter
`TO`	Time-out bit
`PD`	Power-down bit

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

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

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

Table 13-2 lists the instructions recognized by the MPASM™ assembler.

Figure 13-1 shows the general formats that the instructions can have.

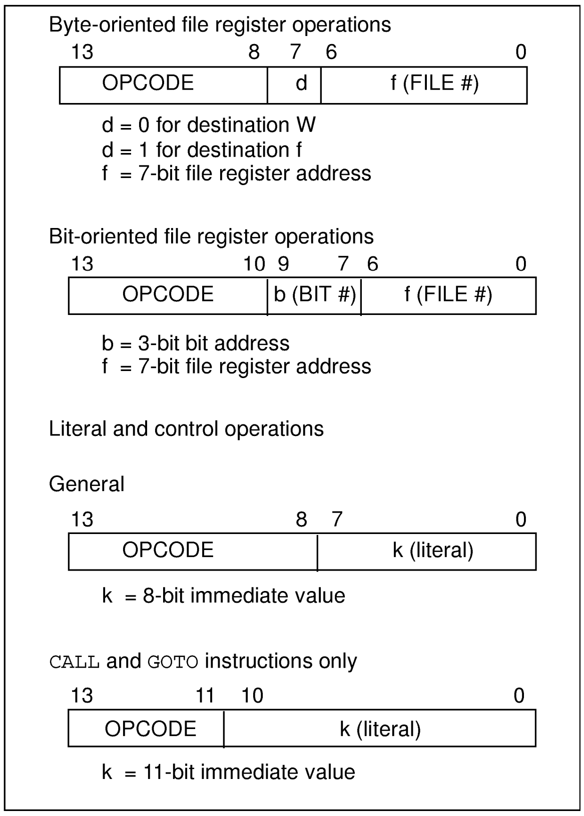
Note: To maintain upward compatibility with future PIC16F87X products, do not use the `OPTION` and `TRIS` instructions.

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

0xhh

where h signifies a hexadecimal digit.

FIGURE 13-1: GENERAL FORMAT FOR INSTRUCTIONS



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

A description of each instruction is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

TABLE 13-2: PIC16F87X INSTRUCTION SET

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

Note: Additional information on the mid-range instruction set is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

13.1 Instruction Descriptions

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

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'.

ANDLW	**AND Literal with W**
Syntax:	[*label*] ANDLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .AND. (k) $\rightarrow$ (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.

ANDWF	**AND W with f**
Syntax:	[*label*] ANDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) .AND. (f) $\rightarrow$ (destination)
Status Affected:	Z
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'.

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.

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.

BTFSS	**Bit Test f, Skip if Set**
Syntax:	[*label*] BTFSS f,b
Operands:	$0 \le f \le 127$ $0 \le b < 7$
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.

BTFSC	**Bit Test, Skip if Clear**
Syntax:	[*label*] BTFSC f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	skip if (f<b>) = 0
Status Affected:	None
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.

CALL	Call Subroutine
Syntax:	[*label*] CALL k
Operands:	$0 \leq k \leq 2047$
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.

CLRF	Clear f
Syntax:	[*label*] CLRF f
Operands:	$0 \leq f \leq 127$
Operation:	00h $\rightarrow$ (f) 1 $\rightarrow$ Z
Status Affected:	Z
Description:	The contents of register ’f’ are cleared and the Z bit is set.

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

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

COMF	Complement f
Syntax:	[*label*] COMF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
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’.

DECF	Decrement f
Syntax:	[*label*] DECF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	(f) - 1 $\rightarrow$ (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’.

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.

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
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.

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'.

INCFSZ	Increment f, Skip if 0
Syntax:	[*label*] INCFSZ 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 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.

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.

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'.

MOVF **Move f**

Syntax:	[*label*] MOVF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	(f) → (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.

MOVLW **Move Literal to W**

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

MOVWF **Move W to f**

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

NOP **No Operation**

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

RETFIE **Return from Interrupt**

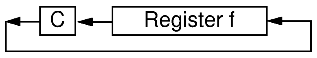
Syntax:	[*label*] RETFIE
Operands:	None
Operation:	TOS → PC, 1 → GIE
Status Affected:	None

RETLW **Return with Literal in W**

Syntax:	[*label*] RETLW k
Operands:	$0 \leq k \leq 255$
Operation:	k → (W); TOS → 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.

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
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'.

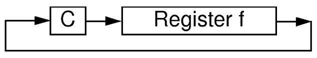


RETURN — Return from Subroutine

Syntax:	[*label*] RETURN
Operands:	None
Operation:	TOS $\rightarrow$ 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.

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'.



SLEEP

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

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

SUBWF — Subtract W from f

Syntax:	[*label*] SUBWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
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'.

SWAPF **Swap Nibbles in f**

Syntax:	[*label*] SWAPF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
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 the W register. If 'd' is 1, the result is placed in register 'f'.

XORLW **Exclusive OR Literal with W**

Syntax:	[*label*] XORLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .XOR. k $\rightarrow$ (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.

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) $\rightarrow$ (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'.