20.0 INSTRUCTION SET SUMMARY

The PIC18FXXX instruction set adds many enhancements to the previous PICmicro instruction sets, while maintaining an easy migration from these PICmicro instruction sets.

Most instructions are a single program memory word (16-bits), but there are three instructions that require two program memory locations.

Each single word instruction is a 16-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 instruction set is highly orthogonal and is grouped into four basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal** operations
- **Control** operations

The PIC18FXXX instruction set summary in Table 20-2 lists **byte-oriented**, **bit-oriented**, **literal** and **control** operations. Table 20-1 shows the opcode field descriptions.

Most **byte-oriented** instructions have three operands:

1. The file register (specified by 'f')
2. The destination of the result (specified by 'd')
3. The accessed memory (specified by 'a')

The file register designator 'f' specifies which file register is to be used by the instruction.

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

All **bit-oriented** instructions have three operands:

1. The file register (specified by 'f')
2. The bit in the file register (specified by 'b')
3. The accessed memory (specified by 'a')

The bit field designator 'b' selects the number of the bit affected by the operation, while the file register designator 'f' represents the number of the file in which the bit is located.

The **literal** instructions may use some of the following operands:

- A literal value to be loaded into a file register (specified by 'k')
- The desired FSR register to load the literal value into (specified by 'f')
- No operand required (specified by '—')

The **control** instructions may use some of the following operands:

- A program memory address (specified by 'n')
- The mode of the Call or Return instructions (specified by 's')
- The mode of the Table Read and Table Write instructions (specified by 'm')
- No operand required (specified by '—')

All instructions are a single word, except for three double-word instructions. These three instructions were made double-word instructions so that all the required information is available in these 32 bits. In the second word, the 4-MSbs are 1's. If this second word is executed as an instruction (by itself), it will execute as a `NOP`.

All single word instructions are executed in a single instruction cycle, unless a conditional test is true or the program counter is changed as a result of the instruction. In these cases, the execution takes two instruction cycles with the additional instruction cycle(s) executed as a `NOP`.

The double-word instructions execute in two instruction cycles.

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. Two-word branch instructions (if true) would take 3 μs.

Figure 20-1 shows the general formats that the instructions can have.

All examples use the format `'nnh'` to represent a hexadecimal number, where `'h'` signifies a hexadecimal digit.

The Instruction Set Summary, shown in Table 20-2, lists the instructions recognized by the Microchip Assembler (MPASM™).

Section 20.1 provides a description of each instruction.

TABLE 20-1: OPCODE FIELD DESCRIPTIONS

Field	Description
`a`	RAM access bit a = 0: RAM location in Access RAM (BSR register is ignored) a = 1: RAM bank is specified by BSR register
`bbb`	Bit address within an 8-bit file register (0 to 7)
`BSR`	Bank Select Register. Used to select the current RAM bank.

`d`	Destination select bit;
	d = 0: store result in WREG,
	d = 1: store result in file register f.

`dest`	Destination either the WREG register or the specified register file location
`f`	8-bit Register file address (0x00 to 0xFF)
`fs`	12-bit Register file address (0x000 to 0xFFF). This is the source address.
`fd`	12-bit Register file address (0x000 to 0xFFF). This is the destination address.
`k`	Literal field, constant data or label (may be either an 8-bit, 12-bit or a 20-bit value)
`label`	Label name
`mm`	The mode of the TBLPTR register for the Table Read and Table Write instructions. Only used with Table Read and Table Write instructions:
`*`	No Change to register (such as TBLPTR with Table reads and writes)
`*+`	Post-Increment register (such as TBLPTR with Table reads and writes)
`*-`	Post-Decrement register (such as TBLPTR with Table reads and writes)
`+*`	Pre-Increment register (such as TBLPTR with Table reads and writes)
`n`	The relative address (2's complement number) for relative branch instructions, or the direct address for Call/Branch and Return instructions
`PRODH`	Product of Multiply high byte
`PRODL`	Product of Multiply low byte
`s`	Fast Call/Return mode select bit. s = 0: do not update into/from shadow registers s = 1: certain registers loaded into/from shadow registers (Fast mode)
`u`	Unused or Unchanged
`WREG`	Working register (accumulator)
`x`	Don't care (0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
`TBLPTR`	21-bit Table Pointer (points to a Program Memory location)
`TABLAT`	8-bit Table Latch
`TOS`	Top-of-Stack
`PC`	Program Counter
`PCL`	Program Counter Low Byte
`PCH`	Program Counter High Byte
`PCLATH`	Program Counter High Byte Latch
`PCLATU`	Program Counter Upper Byte Latch
`GIE`	Global Interrupt Enable bit
`WDT`	Watchdog Timer
$\overline{\text{TO}}$	Time-out bit
$\overline{\text{PD}}$	Power-down bit
`C, DC, Z, OV, N`	ALU status bits Carry, Digit Carry, Zero, Overflow, Negative
`[ ]`	Optional
`( )`	Contents
→	Assigned to
`< >`	Register bit field
∈	In the set of
`italics`	User defined term (font is courier)

FIGURE 20-1: GENERAL FORMAT FOR INSTRUCTIONS

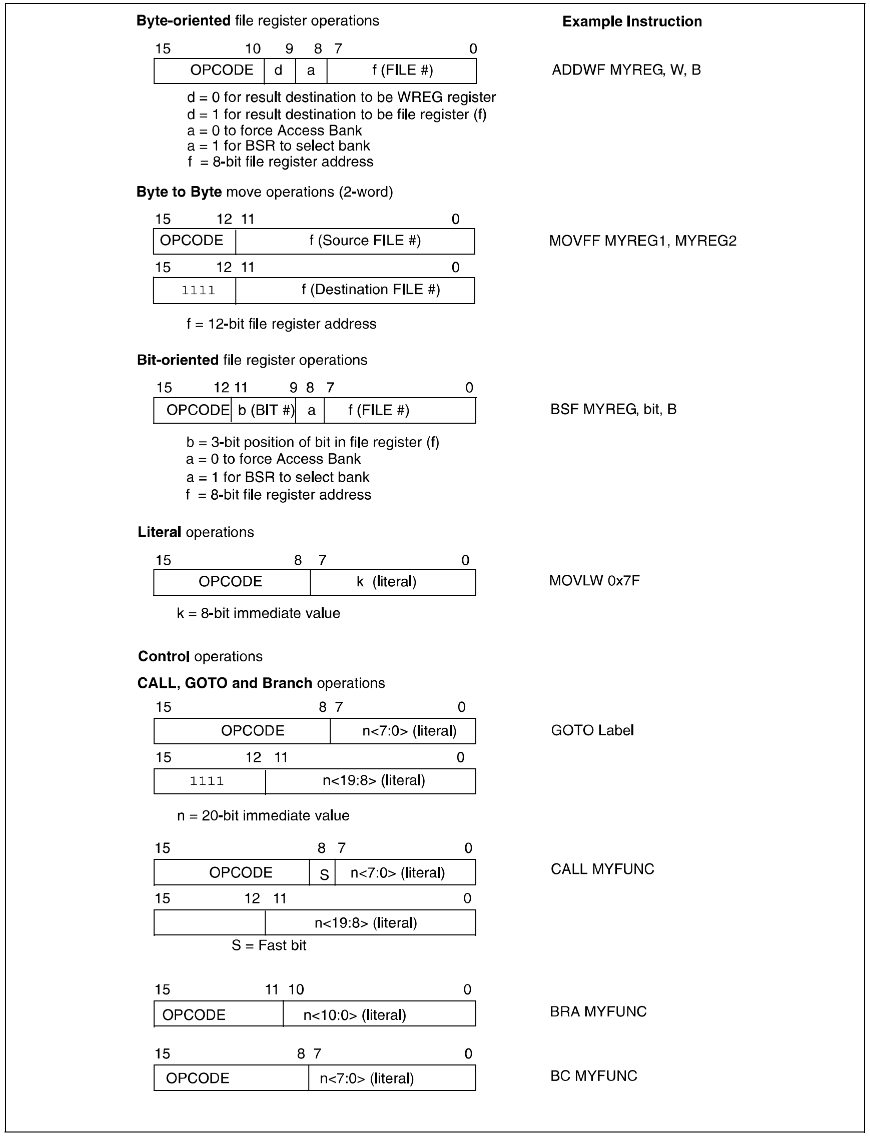

Byte-oriented file register operations
Example Instruction
15 10 9 8 7 0
OPCODE
d
a
f (FILE #)
ADDWF MYREG, W, B
d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address
Byte to Byte move operations (2-word)
15 12 11 0
OPCODE
f (Source FILE #)
MOVFF MYREG1, MYREG2
15 12 11 0
1111
f (Destination FILE #)
f = 12-bit file register address
Bit-oriented file register operations
15 12 11 9 8 7 0
OPCODE
b (BIT #)
a
f (FILE #)
BSF MYREG, bit, B
b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address
Literal operations
15 8 7 0
OPCODE
k (literal)
MOVLW 0x7F
k = 8-bit immediate value
Control operations
CALL, GOTO and Branch operations
15 8 7 0
OPCODE
n<7:0> (literal)
GOTO Label
15 12 11 0
1111
n<19:8> (literal)
n = 20-bit immediate value
15 8 7 0
OPCODE
S
n<7:0> (literal)
CALL MYFUNC
15 12 11 0
n<19:8> (literal)
S = Fast bit
15 11 10 0
OPCODE
n<10:0> (literal)
BRA MYFUNC
15 8 7 0
OPCODE
n<7:0> (literal)
BC MYFUNC

TABLE 20-2: PIC18FXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da0	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	0da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word f_d (destination) 2nd word	2	1100 1111	ffff ffff	ffff ffff	ffff ffff	None	
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	1, 2
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	1, 2
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	1, 2
SWAPF	f, d, a	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), 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.

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`.

4: Some instructions are 2-word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

TABLE 20-2: PIC18FXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	2	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	1 (2)	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to address1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device RESET	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), 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.

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`.

4: Some instructions are 2-word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

TABLE 20-2: PIC18FXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
LITERAL OPERATIONS									
ADDLW	k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move literal (12-bit) 2nd word	2	1110	1110	00ff	kkkk	None	
		to FSRx 1st word		1111	0000	kkkk	kkkk		
MOVLB	k	Move literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW	k	Move literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000	1101	kkkk	kkkk	None	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*		Table Read	2	0000	0000	0000	1000	None	
TBLRD*+		Table Read with post-increment		0000	0000	0000	1001	None	
TBLRD*-		Table Read with post-decrement		0000	0000	0000	1010	None	
TBLRD+*		Table Read with pre-increment		0000	0000	0000	1011	None	
TBLWT*		Table Write	2 (5)	0000	0000	0000	1100	None	
TBLWT*+		Table Write with post-increment		0000	0000	0000	1101	None	
TBLWT*-		Table Write with post-decrement		0000	0000	0000	1110	None	
TBLWT+*		Table Write with pre-increment		0000	0000	0000	1111	None	

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), 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.

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`.

4: Some instructions are 2-word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

20.1 Instruction Set

ADDLW	ADD literal to W
Syntax:	[*label*] ADDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) + k → W
Status Affected:	N, OV, C, DC, Z
Encoding:	`0000` `1111` `kkkk` `kkkk`
Description:	The contents of W are added to the 8-bit literal 'k' and the result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `ADDLW 0x15`

Before Instruction

W = 0x10

After Instruction

W = 0x25

ADDWF	ADD W to f
Syntax:	[*label*] ADDWF f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(W) + (f) → dest
Status Affected:	N, OV, C, DC, Z
Encoding:	`0010` `01da` `ffff` `ffff`
Description:	Add W to register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the BSR is used.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `ADDWF REG, 0, 0`

Before Instruction

W = 0x17
REG = 0xC2

After Instruction

W = 0xD9
REG = 0xC2

ADDWFC	ADD W and Carry bit to f
Syntax:	[*label*] ADDWFC f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(W) + (f) + (C) → dest
Status Affected:	N,OV, C, DC, Z
Encoding:	`0010` `00da` `ffff` `ffff`
Description:	Add W, the Carry Flag and data memory location 'f'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed in data memory location 'f'. If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the BSR will not be overridden.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `ADDWFC REG, 0, 1`

Before Instruction

Carry bit = 1
REG = 0x02
W = 0x4D

After Instruction

Carry bit = 0
REG = 0x02
W = 0x50

ANDLW	AND literal with W
Syntax:	[*label*] ANDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .AND. k → W
Status Affected:	N,Z
Encoding:	`0000` `1011` `kkkk` `kkkk`
Description:	The contents of W are ANDed with the 8-bit literal 'k'. The result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `ANDLW 0x5F`

Before Instruction

W = 0xA3

After Instruction

W = 0x03

ANDWF **AND W with f**

Syntax: [*label*] ANDWF f [,d [,a]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (W) .AND. (f) → dest

Status Affected: N,Z

Encoding:

0001	01da	ffff	ffff

Description: The contents of W are AND'ed with register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the BSR will not be overridden (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `ANDWF REG, 0, 0`

Before Instruction
W = 0x17
REG = 0xC2

After Instruction
W = 0x02
REG = 0xC2

BC **Branch if Carry**

Syntax: [*label*] BC n

Operands: -128 ≤ n ≤ 127

Operation: if carry bit is '1'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0010	nnnn	nnnn

Description: If the Carry bit is '1', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BC 5`

Before Instruction
PC = address (HERE)

After Instruction
If Carry = 1;
PC = address (HERE+12)
If Carry = 0;
PC = address (HERE+2)

BCF **Bit Clear f**

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

Operands: 0 ≤ f ≤ 255
0 ≤ b ≤ 7
a ∈ [0,1]

Operation: 0 → f<b>

Status Affected: None

Encoding:

1001	bbba	ffff	ffff

Description: Bit 'b' in register 'f' is cleared. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `BCF FLAG_REG, 7, 0`

Before Instruction
FLAG_REG = 0xC7

After Instruction
FLAG_REG = 0x47

BN **Branch if Negative**

Syntax: [*label*] BN n

Operands: -128 ≤ n ≤ 127

Operation: if negative bit is '1'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0110	nnnn	nnnn

Description: If the Negative bit is '1', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BN Jump`

Before Instruction
PC = address (HERE)

After Instruction
If Negative = 1;
PC = address (Jump)
If Negative = 0;
PC = address (HERE+2)

BNC — Branch if Not Carry

Syntax: [*label*] BNC n

Operands: -128 ≤ n ≤ 127

Operation: if carry bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0011	nnnn	nnnn

Description: If the Carry bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

```
Example:       HERE      BNC  Jump
  Before Instruction
      PC          =    address  (HERE)
  After Instruction
      If Carry    =    0;
            PC    =    address  (Jump)
      If Carry    =    1;
            PC    =    address  (HERE+2)
```

BNN — Branch if Not Negative

Syntax: [*label*] BNN n

Operands: -128 ≤ n ≤ 127

Operation: if negative bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0111	nnnn	nnnn

Description: If the Negative bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

```
Example:       HERE      BNN  Jump
  Before Instruction
      PC             =    address  (HERE)
  After Instruction
      If Negative    =    0;
               PC    =    address  (Jump)
      If Negative    =    1;
               PC    =    address  (HERE+2)
```

BNOV — Branch if Not Overflow

Syntax: [*label*] BNOV n

Operands: -128 ≤ n ≤ 127

Operation: if overflow bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0101	nnnn	nnnn

Description: If the Overflow bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

```
Example:       HERE      BNOV  Jump
  Before Instruction
      PC             =    address  (HERE)
  After Instruction
      If Overflow    =    0;
               PC    =    address  (Jump)
      If Overflow    =    1;
               PC    =    address  (HERE+2)
```

BNZ — Branch if Not Zero

Syntax: [*label*] BNZ n

Operands: -128 ≤ n ≤ 127

Operation: if zero bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0001	nnnn	nnnn

Description: If the Zero bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

```
Example:       HERE      BNZ  Jump
  Before Instruction
      PC         =    address  (HERE)
  After Instruction
      If Zero    =    0;
           PC    =    address  (Jump)
      If Zero    =    1;
           PC    =    address  (HERE+2)
```

BRA Unconditional Branch

Syntax: [*label*] BRA n

Operands: -1024 ≤ n ≤ 1023

Operation: (PC) + 2 + 2n → PC

Status Affected: None

Encoding:

`1101`	`0nnn`	`nnnn`	`nnnn`

Description: Add the 2's complement number '2n' to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is a two-cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

Example: `HERE BRA Jump`

Before Instruction
PC = address `(HERE)`

After Instruction
PC = address `(Jump)`

BSF Bit Set f

Syntax: [*label*] BSF f,b[,a]

Operands: 0 ≤ f ≤ 255
0 ≤ b ≤ 7
a ∈ [0,1]

Operation: 1 → f<b>

Status Affected: None

Encoding:

`1000`	`bbba`	`ffff`	`ffff`

Description: Bit 'b' in register 'f' is set. If ‘a’ is 0 Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `BSF FLAG_REG, 7, 1`

Before Instruction
FLAG_REG = 0x0A

After Instruction
FLAG_REG = 0x8A

BTFSC Bit Test File, Skip if Clear

Syntax: [*label*] BTFSC f,b[,a]

Operands: 0 ≤ f ≤ 255
0 ≤ b ≤ 7
a ∈ [0,1]

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

Status Affected: None

Encoding:

`1011`	`bbba`	`ffff`	`ffff`

Description: If bit 'b' in register ’f' is 0, then the next instruction is skipped.
If bit 'b' is 0, then the next instruction fetched during the current instruction execution is discarded, and a `NOP` is executed instead, making this a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE    BTFSC   FLAG, 1, 0
FALSE   :
TRUE    :
```

Before Instruction
PC = address `(HERE)`

After Instruction
If FLAG<1> = 0;
PC = address `(TRUE)`
If FLAG<1> = 1;
PC = address `(FALSE)`

BTFSS Bit Test File, Skip if Set

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

Operands: 0 ≤ f ≤ 255
0 ≤ b ≤ 7
a ∈ [0,1]

Operation: skip if (f<b>) = 1

Status Affected: None

Encoding:

`1010`	`bbba`	`ffff`	`ffff`

Description: If bit 'b' in register 'f' is 1, then the next instruction is skipped.
If bit 'b' is 1, then the next instruction fetched during the current instruction execution, is discarded and a `NOP` is executed instead, making this a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE    BTFSS   FLAG, 1, 0
FALSE   :
TRUE    :
```

Before Instruction
PC = address `(HERE)`

After Instruction
If FLAG<1> = 0;
PC = address `(FALSE)`
If FLAG<1> = 1;
PC = address `(TRUE)`

BTG — Bit Toggle f

Syntax:	[*label*] BTG f,b[,a]
Operands:	0 ≤ f ≤ 255 0 ≤ b ≤ 7 a ∈ [0,1]
Operation:	$(\overline{f<b>}) \rightarrow f<b>$
Status Affected:	None
Encoding:	`0111` `bbba` `ffff` `ffff`
Description:	Bit 'b' in data memory location 'f' is inverted. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

<u>Example</u>: `BTG PORTC, 4, 0`

Before Instruction:
PORTC = `0111 0101` [0x75]

After Instruction:
PORTC = `0110 0101` [0x65]

BOV — Branch if Overflow

Syntax:	[*label*] BOV n
Operands:	-128 ≤ n ≤ 127
Operation:	if overflow bit is '1' (PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	`1110` `0100` `nnnn` `nnnn`
Description:	If the Overflow bit is '1', then the program will branch. The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.
Words:	1
Cycles:	1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

<u>Example</u>: `HERE BOV Jump`

Before Instruction
PC = address `(HERE)`

After Instruction
If Overflow = 1;
PC = address `(Jump)`
If Overflow = 0;
PC = address `(HERE+2)`

BZ — Branch if Zero

Syntax:	[*label*] BZ n
Operands:	-128 ≤ n ≤ 127
Operation:	if Zero bit is '1' (PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	`1110` `0000` `nnnn` `nnnn`
Description:	If the Zero bit is '1', then the program will branch. The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.
Words:	1
Cycles:	1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

<u>Example</u>: `HERE BZ Jump`

Before Instruction
PC = address `(HERE)`

After Instruction
If Zero = 1;
PC = address `(Jump)`
If Zero = 0;
PC = address `(HERE+2)`

CALL — Subroutine Call

Syntax:	[*label*] CALL k [,s]
Operands:	0 ≤ k ≤ 1048575 s ∈ [0,1]
Operation:	(PC) + 4 → TOS, k → PC<20:1>, if s = 1 (W) → WS, (STATUS) → STATUSS, (BSR) → BSRS
Status Affected:	None

Encoding:

1st word (k<7:0>)	`1110`	`110s`	k_7`kkk`	`kkkk`$_0$
2nd word(k<19:8>)	`1111`	k_{19}`kkk`	`kkkk`	`kkkk`$_8$

Description:	Subroutine call of entire 2 Mbyte memory range. First, return address (PC+ 4) is pushed onto the return stack. If 's' = 1, the W, STATUS and BSR registers are also pushed into their respective shadow registers, WS, STATUSS and BSRS. If 's' = 0, no update occurs (default). Then, the 20-bit value 'k' is loaded into PC<20:1>. `CALL` is a two-cycle instruction.
Words:	2
Cycles:	2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	Push PC to stack	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

<u>Example</u>: `HERE CALL THERE,1`

Before Instruction
PC = address `(HERE)`

After Instruction
PC = address `(THERE)`
TOS = address `(HERE + 4)`
WS = W
BSRS = BSR
STATUSS= STATUS

CLRF — Clear f

Syntax: [*label*] CLRF f [,a]

Operands: 0 ≤ f ≤ 255
a ∈ [0,1]

Operation: 000h → f
1 → Z

Status Affected: Z

Encoding:

0110	101a	ffff	ffff

Description: Clears the contents of the specified register. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `CLRF FLAG_REG,1`

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

CLRWDT — Clear Watchdog Timer

Syntax: [*label*] CLRWDT

Operands: None

Operation: 000h → WDT,
000h → WDT postscaler,
1 → $\overline{TO}$,
1 → $\overline{PD}$

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

Encoding:

0000	0000	0000	0100

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

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process Data	No operation

Example: `CLRWDT`

Before Instruction
WDT Counter = ?
After Instruction
WDT Counter = 0x00
WDT Postscaler = 0
$\overline{TO}$ = 1
$\overline{PD}$ = 1

COMF — Complement f

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

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: $(\overline{f})$ → dest

Status Affected: N, Z

Encoding:

0001	11da	ffff	ffff

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' (default). If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `COMF REG, 0, 0`

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

CPFSEQ — Compare f with W, skip if f = W

Syntax: [*label*] CPFSEQ f [,a]

Operands: 0 ≤ f ≤ 255
a ∈ [0,1]

Operation: (f) – (W),
skip if (f) = (W)
(unsigned comparison)

Status Affected: None

Encoding:

0110	001a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of W by performing an unsigned subtraction.
If 'f' = W, then the fetched instruction is discarded and a `NOP` is executed instead, making this a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE     CPFSEQ REG, 0
NEQUAL   :
EQUAL    :
```

Before Instruction
PC Address = `HERE`
W = ?
REG = ?
After Instruction
If REG = W;
PC = Address `(EQUAL)`
If REG ≠ W;
PC = Address `(NEQUAL)`

CPFSGT — Compare f with W, skip if f > W

Syntax: [*label*] CPFSGT f [,a]

Operands: 0 ≤ f ≤ 255
a ∈ [0,1]

Operation: (f) – (W),
skip if (f) > (W)
(unsigned comparison)

Status Affected: None

Encoding:

0110	010a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of the W by performing an unsigned subtraction.
If the contents of 'f' are greater than the contents of WREG, then the fetched instruction is discarded and a NOP is executed instead, making this a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE        CPFSGT REG, 0
NGREATER    :
GREATER     :
```

Before Instruction
PC = Address (HERE)
W = ?

After Instruction
If REG > W;
PC = Address (GREATER)
If REG ≤ W;
PC = Address (NGREATER)

CPFSLT — Compare f with W, skip if f < W

Syntax: [*label*] CPFSLT f [,a]

Operands: 0 ≤ f ≤ 255
a ∈ [0,1]

Operation: (f) – (W),
skip if (f) < (W)
(unsigned comparison)

Status Affected: None

Encoding:

0110	000a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of W by performing an unsigned subtraction.
If the contents of 'f' are less than the contents of W, then the fetched instruction is discarded and a NOP is executed instead, making this a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected. If ’a’ is 1, the BSR will not be overridden (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE     CPFSLT REG, 1
NLESS    :
LESS     :
```

Before Instruction
PC = Address (HERE)
W = ?

After Instruction
If REG < W;
PC = Address (LESS)
If REG ≥ W;
PC = Address (NLESS)

DAW — Decimal Adjust W Register

Syntax: [*label*] DAW

Operands: None

Operation: If [W<3:0> >9] or [DC = 1] then
(W<3:0>) + 6 → W<3:0>;
else
(W<3:0>) → W<3:0>;

If [W<7:4> >9] or [C = 1] then
(W<7:4>) + 6 → W<7:4>;
else
(W<7:4>) → W<7:4>;

Status Affected: C

Encoding:

0000	0000	0000	0111

Description: DAW adjusts the eight-bit value in W, resulting from the earlier addition of two variables (each in packed BCD format) and produces a correct packed BCD result.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register W	Process Data	Write W

Example1: DAW

Before Instruction
W = 0xA5
C = 0
DC = 0

After Instruction
W = 0x05
C = 1
DC = 0

Example 2:

Before Instruction
W = 0xCE
C = 0
DC = 0

After Instruction
W = 0x34
C = 1
DC = 0

DECF — Decrement f

Syntax: [*label*] DECF f [,d [,a]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) – 1 → dest

Status Affected: C, DC, N, OV, Z

Encoding:

0000	01da	ffff	ffff

Description: Decrement register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If ’a’ is 0, the Access Bank will be selected, overriding the BSR value. If ’a’ = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: DECF CNT, 1, 0

Before Instruction
CNT = 0x01
Z = 0

After Instruction
CNT = 0x00
Z = 1

DECFSZ — Decrement f, skip if 0

Syntax: [*label*] DECFSZ f [,d [,a]]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) – 1 → dest,
skip if result = 0

Status Affected: None

Encoding:

0010	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If the result is 0, the next instruction, which is already fetched, is discarded, and a `NOP` is executed instead, making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE      DECFSZ   CNT, 1, 1
          GOTO     LOOP
CONTINUE
```

Before Instruction

```
PC      =   Address (HERE)
```

After Instruction

```
CNT     =   CNT - 1
If CNT  =   0;
    PC  =   Address (CONTINUE)
If CNT  ≠   0;
    PC  =   Address (HERE+2)
```

DCFSNZ — Decrement f, skip if not 0

Syntax: [*label*] DCFSNZ f [,d [,a]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) – 1 → dest,
skip if result ≠ 0

Status Affected: None

Encoding:

0100	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If the result is not 0, the next instruction, which is already fetched, is discarded, and a `NOP` is executed instead, making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE    DCFSNZ  TEMP, 1, 0
ZERO    :
NZERO   :
```

Before Instruction

```
TEMP          =   ?
```

After Instruction

```
TEMP          =   TEMP - 1,
If TEMP       =   0;
     PC       =   Address (ZERO)
If TEMP       ≠   0;
     PC       =   Address (NZERO)
```

GOTO — Unconditional Branch

Syntax: [*label*] GOTO k

Operands: 0 ≤ k ≤ 1048575

Operation: k → PC<20:1>

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	1111	$k_7$kkk	kkkk$_0$
2nd word(k<19:8>)	1111	k_{19}kkk	kkkk	kkkk$_8$

Description: `GOTO` allows an unconditional branch anywhere within entire 2 Mbyte memory range. The 20-bit value 'k' is loaded into PC<20:1>. `GOTO` is always a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	No operation	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `GOTO THERE`

After Instruction

```
PC  =   Address (THERE)
```

INCF — Increment f

Syntax: [*label*] INCF f [,d [,a]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) + 1 → dest

Status Affected: C, DC, N, OV, Z

Encoding:

0010	10da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `INCF CNT, 1, 0`

Before Instruction

```
CNT   =   0xFF
Z     =   0
C     =   ?
DC    =   ?
```

After Instruction

```
CNT   =   0x00
Z     =   1
C     =   1
DC    =   1
```

INCFSZ Increment f, skip if 0

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

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) + 1 → dest,
skip if result = 0

Status Affected: None

Encoding:

0011	11da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f'. (default) If the result is 0, the next instruction, which is already fetched, is discarded, and a `NOP` is executed instead, making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE     INCFSZ   CNT, 1, 0
NZERO    :
ZERO     :
```

Before Instruction
PC = Address (HERE)

After Instruction
CNT = CNT + 1
If CNT = 0;
PC = Address (ZERO)
If CNT ≠ 0;
PC = Address (NZERO)

INFSNZ Increment f, skip if not 0

Syntax: [*label*] INFSNZ f [,d [,a]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) + 1 → dest,
skip if result ≠ 0

Status Affected: None

Encoding:

0100	10da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If the result is not 0, the next instruction, which is already fetched, is discarded, and a `NOP` is executed instead, making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

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

If skip and followed by 2-word instruction:

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

Example:

```
HERE     INFSNZ   REG, 1, 0
ZERO
NZERO
```

Before Instruction
PC = Address (HERE)

After Instruction
REG = REG + 1
If REG ≠ 0;
PC = Address (NZERO)
If REG = 0;
PC = Address (ZERO)

IORLW Inclusive OR literal with W

Syntax: [*label*] IORLW k

Operands: 0 ≤ k ≤ 255

Operation: (W) .OR. k → W

Status Affected: N, Z

Encoding:

0000	1001	kkkk	kkkk

Description: The contents of W are OR'ed with the eight-bit literal 'k'. The result is placed in W.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `IORLW 0x35`

Before Instruction
W = 0x9A

After Instruction
W = 0xBF

IORWF Inclusive OR W with f

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

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (W) .OR. (f) → dest

Status Affected: N, Z

Encoding:

0001	00da	ffff	ffff

Description: Inclusive OR W with register 'f'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `IORWF RESULT, 0, 1`

Before Instruction
RESULT = 0x13
W = 0x91

After Instruction
RESULT = 0x13
W = 0x93

LFSR — **Load FSR**

Syntax: [*label*] LFSR f,k

Operands: $0 \le f \le 2$
$0 \le k \le 4095$

Operation: $k \rightarrow$ FSRf

Status Affected: None

Encoding:

`1110`
`1111`
	`1110`
	`0000`
		`00ff`
		`k`$_7$`kkk`
			`k`$_{11}$`kkk`
			`kkkk`

Description: The 12-bit literal 'k' is loaded into the file select register pointed to by 'f'.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k' MSB	Process Data	Write literal 'k' MSB to FSRfH
Decode	Read literal 'k' LSB	Process Data	Write literal 'k' to FSRfL

Example: `LFSR 2, 0x3AB`

After Instruction

FSR2H	=	0x03
FSR2L	=	0xAB

MOVF — **Move f**

Syntax: [*label*] MOVF f [,d [,a]

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: $f \rightarrow$ dest

Status Affected: N, Z

Encoding:

`0101`	`00da`	`ffff`	`ffff`

Description: The contents of register 'f' are moved to a destination dependent upon the status of 'd'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). Location 'f' can be anywhere in the 256 byte bank. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write W

Example: `MOVF REG, 0, 0`

Before Instruction

REG	=	0x22
W	=	0xFF

After Instruction

REG	=	0x22
W	=	0x22

MOVFF — **Move f to f**

Syntax: [*label*] MOVFF f_s,f_d

Operands: $0 \le f_s \le 4095$
$0 \le f_d \le 4095$

Operation: $(f_s) \rightarrow f_d$

Status Affected: None

Encoding:

1st word (source)	`1100`	`ffff`	`ffff`	`ffff`$_s$
2nd word (destin.)	`1111`	`ffff`	`ffff`	`ffff`$_d$

Description: The contents of source register 'f_s' are moved to destination register 'f_d'. Location of source 'f_s' can be anywhere in the 4096 byte data space (000h to FFFh), and location of destination 'f_d' can also be anywhere from 000h to FFFh.
Either source or destination can be W (a useful special situation).
`MOVFF` is particularly useful for transferring a data memory location to a peripheral register (such as the transmit buffer or an I/O port).

The `MOVFF` instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register.

Note: The `MOVFF` instruction should not be used to modify interrupt settings while any interrupt is enabled. See Section 8.0 for more information.

Words: 2

Cycles: 2 (3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f' (src)	Process Data	No operation
Decode	No operation No dummy read	No operation	Write register 'f' (dest)

Example: `MOVFF REG1, REG2`

Before Instruction

REG1	=	0x33
REG2	=	0x11

After Instruction

REG1	=	0x33,
REG2	=	0x33

MOVLB — **Move literal to low nibble in BSR**

Syntax: [*label*] MOVLB k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow$ BSR

Status Affected: None

Encoding:

`0000`	`0001`	`kkkk`	`kkkk`

Description: The 8-bit literal 'k' is loaded into the Bank Select Register (BSR).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write literal 'k' to BSR

Example: `MOVLB 5`

Before Instruction

BSR register	=	0x02

After Instruction

BSR register	=	0x05

MOVLW **Move literal to W**

Syntax: [*label*] MOVLW k

Operands: 0 ≤ k ≤ 255

Operation: k → W

Status Affected: None

Encoding:

0000	1110	kkkk	kkkk

Description: The eight-bit literal 'k' is loaded into W.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `MOVLW 0x5A`

After Instruction

W = 0x5A

MOVWF **Move W to f**

Syntax: [*label*] MOVWF f [,a]

Operands: 0 ≤ f ≤ 255
a ∈ [0,1]

Operation: (W) → f

Status Affected: None

Encoding:

0110	111a	ffff	ffff

Description: Move data from W to register 'f'. Location 'f' can be anywhere in the 256 byte bank. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `MOVWF REG, 0`

Before Instruction

W = 0x4F
REG = 0xFF

After Instruction

W = 0x4F
REG = 0x4F

MULLW **Multiply Literal with W**

Syntax: [*label*] MULLW k

Operands: 0 ≤ k ≤ 255

Operation: (W) x k → PRODH:PRODL

Status Affected: None

Encoding:

0000	1101	kkkk	kkkk

Description: An unsigned multiplication is carried out between the contents of W and the 8-bit literal 'k'. The 16-bit result is placed in PRODH:PRODL register pair. PRODH contains the high byte.
W is unchanged.
None of the status flags are affected.
Note that neither overflow nor carry is possible in this operation. A zero result is possible but not detected.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write registers PRODH: PRODL

Example: `MULLW 0xC4`

Before Instruction

W = 0xE2
PRODH = ?
PRODL = ?

After Instruction

W = 0xE2
PRODH = 0xAD
PRODL = 0x08

MULWF **Multiply W with f**

Syntax: [*label*] MULWF f [,a]

Operands: 0 ≤ f ≤ 255
a ∈ [0,1]

Operation: (W) x (f) → PRODH:PRODL

Status Affected: None

Encoding:

0000	001a	ffff	ffff

Description: An unsigned multiplication is carried out between the contents of W and the register file location 'f'. The 16-bit result is stored in the PRODH:PRODL register pair. PRODH contains the high byte.
Both W and 'f' are unchanged.
None of the status flags are affected.
Note that neither overflow nor carry is possible in this operation. A zero result is possible but not detected. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write registers PRODH: PRODL

Example: `MULWF REG, 1`

Before Instruction

W = 0xC4
REG = 0xB5
PRODH = ?
PRODL = ?

After Instruction

W = 0xC4
REG = 0xB5
PRODH = 0x8A
PRODL = 0x94

NEGF Negate f

Syntax: [*label*] NEGF f [,a]

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: $(\overline{f}) + 1 \rightarrow f$

Status Affected: N, OV, C, DC, Z

Encoding:

0110	110a	ffff	ffff

Description: Location 'f' is negated using two's complement. The result is placed in the data memory location 'f'. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `NEGF REG, 1`

Before Instruction

REG = `0011 1010` [0x3A]

After Instruction

REG = `1100 0110` [0xC6]

NOP No Operation

Syntax: [*label*] NOP

Operands: None

Operation: No operation

Status Affected: None

Encoding:

0000 1111	0000 xxxx	0000 xxxx	0000 xxxx

Description: No operation.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

Example:

None.

POP Pop Top of Return Stack

Syntax: [*label*] POP

Operands: None

Operation: (TOS) → bit bucket

Status Affected: None

Encoding:

0000	0000	0000	0110

Description: The TOS value is pulled off the return stack and is discarded. The TOS value then becomes the previous value that was pushed onto the return stack.
This instruction is provided to enable the user to properly manage the return stack to incorporate a software stack.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	POP TOS value	No operation

Example:

```
POP
GOTO    NEW
```

Before Instruction

TOS	=	0031A2h
Stack (1 level down)	=	014332h

After Instruction

TOS	=	014332h
PC	=	NEW

PUSH Push Top of Return Stack

Syntax: [*label*] PUSH

Operands: None

Operation: (PC+2) → TOS

Status Affected: None

Encoding:

0000	0000	0000	0101

Description: The PC+2 is pushed onto the top of the return stack. The previous TOS value is pushed down on the stack. This instruction allows to implement a software stack by modifying TOS, and then push it onto the return stack.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	PUSH PC+2 onto return stack	No operation	No operation

Example: `PUSH`

Before Instruction

TOS	=	00345Ah
PC	=	000124h

After Instruction

PC	=	000126h
TOS	=	000126h
Stack (1 level down)	=	00345Ah

RCALL — Relative Call

Syntax: [*label*] RCALL n

Operands: -1024 ≤ n ≤ 1023

Operation: (PC) + 2 → TOS,
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1101	1nnn	nnnn	nnnn

Description: Subroutine call with a jump up to 1K from the current location. First, return address (PC+2) is pushed onto the stack. Then, add the 2's complement number '2n' to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is a two-cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'n' Push PC to stack	Process Data	Write to PC
No operation	No operation	No operation	No operation

Example: `HERE     RCALL Jump`

Before Instruction
PC = Address `(HERE)`

After Instruction
PC = Address `(Jump)`
TOS = Address `(HERE+2)`

RESET — Reset

Syntax: [*label*] RESET

Operands: None

Operation: Reset all registers and flags that are affected by a $\overline{\text{MCLR}}$ Reset.

Status Affected: All

Encoding:

0000	0000	1111	1111

Description: This instruction provides a way to execute a $\overline{\text{MCLR}}$ Reset in software.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Start reset	No operation	No operation

Example: `RESET`

After Instruction
Registers = Reset Value
Flags* = Reset Value

RETFIE — Return from Interrupt

Syntax: [*label*] RETFIE [s]

Operands: s ∈ [0,1]

Operation: (TOS) → PC,
1 → GIE/GIEH or PEIE/GIEL,
if s = 1
(WS) → W,
(STATUSS) → STATUS,
(BSRS) → BSR,
PCLATU, PCLATH are unchanged.

Status Affected: GIE/GIEH, PEIE/GIEL.

Encoding:

0000	0000	0001	000s

Description: Return from Interrupt. Stack is popped and Top-of-Stack (TOS) is loaded into the PC. Interrupts are enabled by setting either the high or low priority global interrupt enable bit. If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, W, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	pop PC from stack Set GIEH or GIEL
No operation	No operation	No operation	No operation

Example: `RETFIE  1`

After Interrupt

PC	=	TOS
W	=	WS
BSR	=	BSRS
STATUS	=	STATUSS
GIE/GIEH, PEIE/GIEL	=	1

RETLW — Return Literal to W

Syntax: [*label*] RETLW k

Operands: 0 ≤ k ≤ 255

Operation: k → W,
(TOS) → PC,
PCLATU, PCLATH are unchanged

Status Affected: None

Encoding:

0000	1100	kkkk	kkkk

Description: W is loaded with the eight-bit literal 'k'. The program counter is loaded from the top of the stack (the return address). The high address latch (PCLATH) remains unchanged.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	pop PC from stack, Write to W
No operation	No operation	No operation	No operation

Example:

```
    CALL TABLE ; W contains table
               ; offset value
               ; W now has
               ; table value
    :
TABLE
    ADDWF PCL  ; W = offset
    RETLW k0   ; Begin table
    RETLW k1   ;
    :
    :
    RETLW kn   ; End of table
```

Before Instruction
W = 0x07

After Instruction
W = value of kn

RETURN **Return from Subroutine**

Syntax: [*label*] RETURN [s]

Operands: s ∈ [0,1]

Operation: (TOS) → PC,
if s = 1
(WS) → W,
(STATUSS) → STATUS,
(BSRS) → BSR,
PCLATU, PCLATH are unchanged

Status Affected: None

Encoding:

`0000`	`0000`	`0001`	`001s`

Description: Return from subroutine. The stack is popped and the top of the stack (TOS) is loaded into the program counter. If 's'= 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, W, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process Data	pop PC from stack
No operation	No operation	No operation	No operation

Example: `RETURN`

After Interrupt
PC = TOS

RLCF **Rotate Left f through Carry**

Syntax: [*label*] RLCF f [,d [,a]

Operands: $0 \le f \le 255$
d ∈ [0,1]
a ∈ [0,1]

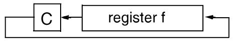
Operation: (f<n>) → dest<n+1>,
(f<7>) → C,
(C) → dest<0>

Status Affected: C, N, Z

Encoding:

`0011`	`01da`	`ffff`	`ffff`

Description: The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).



Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `RLCF REG, 0, 0`

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

RLNCF **Rotate Left f (no carry)**

Syntax: [*label*] RLNCF f [,d [,a]

Operands: $0 \le f \le 255$
d ∈ [0,1]
a ∈ [0,1]

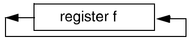
Operation: (f<n>) → dest<n+1>,
(f<7>) → dest<0>

Status Affected: N, Z

Encoding:

`0100`	`01da`	`ffff`	`ffff`

Description: The contents of register 'f' are rotated one bit to the left. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).



Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `RLNCF REG, 1, 0`

Before Instruction
REG = `1010 1011`
After Instruction
REG = `0101 0111`

RRCF **Rotate Right f through Carry**

Syntax: [*label*] RRCF f [,d [,a]

Operands: $0 \le f \le 255$
d ∈ [0,1]
a ∈ [0,1]

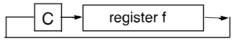
Operation: (f<n>) → dest<n-1>,
(f<0>) → C,
(C) → dest<7>

Status Affected: C, N, Z

Encoding:

`0011`	`00da`	`ffff`	`ffff`

Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).



Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `RRCF REG, 0, 0`

Before Instruction
REG = `1110 0110`
C = 0
After Instruction
REG = `1110 0110`
W = `0111 0011`
C = 0

RRNCF **Rotate Right f (no carry)**

Syntax: [*label*] RRNCF f [,d [,a]

Operands: $0 \le f \le 255$
d ∈ [0,1]
a ∈ [0,1]

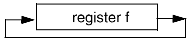
Operation: (f<n>) → dest<n-1>,
(f<0>) → dest<7>

Status Affected: N, Z

Encoding:

0100	00da	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the right. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).



Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: `RRNCF REG, 1, 0`

Before Instruction
REG = `1101 0111`
After Instruction
REG = `1110 1011`

Example 2: `RRNCF REG, 0, 0`

Before Instruction
W = ?
REG = `1101 0111`
After Instruction
W = `1110 1011`
REG = `1101 0111`

SETF **Set f**

Syntax: [*label*] SETF f [,a]

Operands: $0 \le f \le 255$
a ∈ [0,1]

Operation: FFh → f

Status Affected: None

Encoding:

0110	100a	ffff	ffff

Description: The contents of the specified register are set to FFh. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `SETF REG,1`

Before Instruction
REG = 0x5A
After Instruction
REG = 0xFF

SLEEP **Enter SLEEP mode**

Syntax: [*label*] SLEEP

Operands: None

Operation: 00h → WDT,
0 → $\overline{\text{WDT}}$ postscaler,
1 → $\overline{\text{TO}}$,
0 → $\overline{\text{PD}}$

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

Encoding:

0000	0000	0000	0011

Description: The power-down status bit ($\overline{\text{PD}}$) is cleared. The time-out status bit ($\overline{\text{TO}}$) is set. Watchdog Timer and its postscaler are cleared.
The processor is put into SLEEP mode with the oscillator stopped.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `SLEEP`

Before Instruction
$\overline{\text{TO}}$ = ?
$\overline{\text{PD}}$ = ?
After Instruction
$\overline{\text{TO}}$ = 1 †
$\overline{\text{PD}}$ = 0

† If WDT causes wake-up, this bit is cleared.

SUBFWB **Subtract f from W with borrow**

Syntax: [*label*] SUBFWB f [,d [,a]

Operands: $0 \le f \le 255$
d ∈ [0,1]
a ∈ [0,1]

Operation: (W) – (f) – ($\overline{\text{C}}$) → dest

Status Affected: N, OV, C, DC, Z

Encoding:

0101	01da	ffff	ffff

Description: Subtract register 'f' and carry flag (borrow) from W (2's complement method). If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: `SUBFWB REG, 1, 0`

Before Instruction
REG = 3
W = 2
C = 1
After Instruction
REG = FF
W = 2
C = 0
Z = 0
N = 1 ; result is negative

Example 2: `SUBFWB REG, 0, 0`

Before Instruction
REG = 2
W = 5
C = 1
After Instruction
REG = 2
W = 3
C = 1
Z = 0
N = 0 ; result is positive

Example 3: `SUBFWB REG, 1, 0`

Before Instruction
REG = 1
W = 2
C = 0
After Instruction
REG = 0
W = 2
C = 1
Z = 1 ; result is zero
N = 0

SUBLW Subtract W from literal

Syntax: [*label*] SUBLW k

Operands: 0 ≤ k ≤ 255

Operation: k – (W) → W

Status Affected: N, OV, C, DC, Z

Encoding:

`0000`	`1000`	`kkkk`	`kkkk`

Description: W is subtracted from the eight-bit literal 'k'. The result is placed in W.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1: `SUBLW 0x02`

```
Before Instruction
    W    =    1
    C    =    ?
After Instruction
    W    =    1
    C    =    1    ; result is positive
    Z    =    0
    N    =    0
```

Example 2: `SUBLW 0x02`

```
Before Instruction
    W    =    2
    C    =    ?
After Instruction
    W    =    0
    C    =    1    ; result is zero
    Z    =    1
    N    =    0
```

Example 3: `SUBLW 0x02`

```
Before Instruction
    W    =    3
    C    =    ?
After Instruction
    W    =    FF   ; (2's complement)
    C    =    0    ; result is negative
    Z    =    0
    N    =    1
```

SUBWF Subtract W from f

Syntax: [*label*] SUBWF f [,d [,a]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) – (W) → dest

Status Affected: N, OV, C, DC, Z

Encoding:

`0101`	`11da`	`ffff`	`ffff`

Description: Subtract W from register 'f' (2's complement method). If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: `SUBWF REG, 1, 0`

```
Before Instruction
    REG  =    3
    W    =    2
    C    =    ?
After Instruction
    REG  =    1
    W    =    2
    C    =    1    ; result is positive
    Z    =    0
    N    =    0
```

Example 2: `SUBWF REG, 0, 0`

```
Before Instruction
    REG  =    2
    W    =    2
    C    =    ?
After Instruction
    REG  =    2
    W    =    0
    C    =    1    ; result is zero
    Z    =    1
    N    =    0
```

Example 3: `SUBWF REG, 1, 0`

```
Before Instruction
    REG  =    1
    W    =    2
    C    =    ?
After Instruction
    REG  =    FFh  ;(2's complement)
    W    =    2
    C    =    0    ; result is negative
    Z    =    0
    N    =    1
```

SUBWFB Subtract W from f with Borrow

Syntax: [*label*] SUBWFB f [,d [,a]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) – (W) – ($\overline{C}$) → dest

Status Affected: N, OV, C, DC, Z

Encoding:

`0101`	`10da`	`ffff`	`ffff`

Description: Subtract W and the carry flag (borrow) from register 'f' (2's complement method). If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: `SUBWFB REG, 1, 0`

```
Before Instruction
    REG  =    0x19     (0001 1001)
    W    =    0x0D     (0000 1101)
    C    =    1
After Instruction
    REG  =    0x0C     (0000 1011)
    W    =    0x0D     (0000 1101)
    C    =    1
    Z    =    0
    N    =    0        ; result is positive
```

Example 2: `SUBWFB REG, 0, 0`

```
Before Instruction
    REG  =    0x1B     (0001 1011)
    W    =    0x1A     (0001 1010)
    C    =    0
After Instruction
    REG  =    0x1B     (0001 1011)
    W    =    0x00
    C    =    1
    Z    =    1        ; result is zero
    N    =    0
```

Example 3: `SUBWFB REG, 1, 0`

```
Before Instruction
    REG  =    0x03     (0000 0011)
    W    =    0x0E     (0000 1101)
    C    =    1
After Instruction
    REG  =    0xF5     (1111 0100)
                       ; [2's comp]
    W    =    0x0E     (0000 1101)
    C    =    0
    Z    =    0
    N    =    1        ; result is negative
```

SWAPF Swap f

Syntax: [*label*] SWAPF f [,d [,a]

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

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

Status Affected: None

Encoding:

`0011`	`10da`	`ffff`	`ffff`

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `SWAPF REG, 1, 0`

```
Before Instruction
    REG  =    0x53
After Instruction
    REG  =    0x35
```

TBLRD Table Read

Syntax: [*label*] TBLRD (*; *+; *-; +*)

Operands: None

Operation:
if TBLRD *,
(Prog Mem (TBLPTR)) → TABLAT;
TBLPTR - No Change;
if TBLRD *+,
(Prog Mem (TBLPTR)) → TABLAT;
(TBLPTR) +1 → TBLPTR;
if TBLRD *-,
(Prog Mem (TBLPTR)) → TABLAT;
(TBLPTR) -1 → TBLPTR;
if TBLRD +*,
(TBLPTR) +1 → TBLPTR;
(Prog Mem (TBLPTR)) → TABLAT;

Status Affected: None

Encoding:

`0000`	`0000`	`0000`	`10nn`
			`nn=0 *`
			`=1 *+`
			`=2 *-`
			`=3 +*`

Description: This instruction is used to read the contents of Program Memory (P.M.). To address the program memory, a pointer called Table Pointer (TBLPTR) is used. The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 Mbyte address range.

TBLPTR[0] = 0: Least Significant Byte of Program Memory Word

TBLPTR[0] = 1: Most Significant Byte of Program Memory Word

The `TBLRD` instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation
No operation	No operation (Read Program Memory)	No operation	No operation (Write TABLAT)

TBLRD Table Read (cont'd)

Example1: `TBLRD *+ ;`

Before Instruction

TABLAT	=	0x55
TBLPTR	=	0x00A356
MEMORY(0x00A356)	=	0x34

After Instruction

TABLAT	=	0x34
TBLPTR	=	0x00A357

Example2: `TBLRD +* ;`

Before Instruction

TABLAT	=	0xAA
TBLPTR	=	0x01A357
MEMORY(0x01A357)	=	0x12
MEMORY(0x01A358)	=	0x34

After Instruction

TABLAT	=	0x34
TBLPTR	=	0x01A358

TBLWT Table Write

Syntax: [*label*] TBLWT (*; *+; *-; +*)

Operands: None

Operation:
if TBLWT*,
(TABLAT) → Holding Register;
TBLPTR - No Change;
if TBLWT*+,
(TABLAT) → Holding Register;
(TBLPTR) +1 → TBLPTR;
if TBLWT*-,
(TABLAT) → Holding Register;
(TBLPTR) -1 → TBLPTR;
if TBLWT+*,
(TBLPTR) +1 → TBLPTR;
(TABLAT) → Holding Register;

Status Affected: None

Encoding:

`0000`	`0000`	`0000`	`11nn`
			`nn=0 *`
			`=1 *+`
			`=2 *-`
			`=3 +*`

Description: This instruction uses the 3 LSbs of the TBLPTR to determine which of the 8 holding registers the TABLAT data is written to. The 8 holding registers are used to program the contents of Program Memory (P.M.). See Section 5.0 for information on writing to FLASH memory.
The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 MBtye address range. The LSb of the TBLPTR selects which byte of the program memory location to access.

TBLPTR[0] = 0: Least Significant Byte of Program Memory Word

TBLPTR[0] = 1: Most Significant Byte of Program Memory Word

The `TBLWT` instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation
No operation	No operation (Read TABLAT)	No operation	No operation (Write to Holding Register or Memory)

TBLWT Table Write (Continued)

Example1: `TBLWT *+;`

Before Instruction

TABLAT	=	0x55
TBLPTR	=	0x00A356
HOLDING REGISTER (0x00A356)	=	0xFF

After Instructions (table write completion)

TABLAT	=	0x55
TBLPTR	=	0x00A357
HOLDING REGISTER (0x00A356)	=	0x55

Example 2: `TBLWT +*;`

Before Instruction

TABLAT	=	0x34
TBLPTR	=	0x01389A
HOLDING REGISTER (0x01389A)	=	0xFF
HOLDING REGISTER (0x01389B)	=	0xFF

After Instruction (table write completion)

TABLAT	=	0x34
TBLPTR	=	0x01389B
HOLDING REGISTER (0x01389A)	=	0xFF
HOLDING REGISTER (0x01389B)	=	0x34

TSTFSZ — Test f, skip if 0

Syntax: [*label*] TSTFSZ f [,a]

Operands: 0 ≤ f ≤ 255
a ∈ [0,1]

Operation: skip if f = 0

Status Affected: None

Encoding:

`0110`	`011a`	`ffff`	`ffff`

Description: If 'f' = 0, the next instruction, fetched during the current instruction execution, is discarded and a `NOP` is executed, making this a two-cycle instruction. If ’a’ is 0, the Access Bank will be selected, overriding the BSR value. If ’a’ is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

If skip and followed by 2-word instruction:

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

<u>Example</u>:

```
HERE    TSTFSZ  CNT, 1
NZERO   :
ZERO    :
```

Before Instruction
PC = Address (HERE)

After Instruction

```
If CNT   =   0x00,
PC       =   Address (ZERO)
If CNT   ≠   0x00,
PC       =   Address (NZERO)
```

XORLW — Exclusive OR literal with W

Syntax: [*label*] XORLW k

Operands: 0 ≤ k ≤ 255

Operation: (W) .XOR. k → W

Status Affected: N, Z

Encoding:

`0000`	`1010`	`kkkk`	`kkkk`

Description: The contents of W are XORed with the 8-bit literal 'k'. The result is placed in W.

Words: 1

Cycles: 1

Q Cycle Activity:

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

<u>Example</u>: `XORLW 0xAF`

Before Instruction
W = 0xB5

After Instruction
W = 0x1A

XORWF — Exclusive OR W with f

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

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (W) .XOR. (f) → dest

Status Affected: N, Z

Encoding:

`0001`	`10da`	`ffff`	`ffff`

Description: Exclusive OR the contents of W with register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in the register 'f' (default). If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ’a’ is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

<u>Example</u>: `XORWF REG, 1, 0`

Before Instruction
REG = 0xAF
W = 0xB5

After Instruction
REG = 0x1A
W = 0xB5