## 24.0 INSTRUCTION SET SUMMARY

PIC18F2420/2520/4420/4520 devices incorporate the standard set of 75 PIC18 core instructions, as well as an extended set of 8 new instructions, for the optimization of code that is recursive or that utilizes a software stack. The extended set is discussed later in this section.

### 24.1 Standard Instruction Set

The standard PIC18 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 four 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 PIC18 instruction set summary in Table 24-2 lists **byte-oriented**, **bit-oriented**, **literal** and **control** operations. Table 24-1 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 four double-word instructions. These instructions were made double-word to contain the required information in 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 24-1 shows the general formats that the instructions can have. All examples use the convention ‘nnh’ to represent a hexadecimal number.

The Instruction Set Summary, shown in Table 24-2, lists the standard instructions recognized by the Microchip Assembler (MPASM™).

**Section 24.1.1 “Standard Instruction Set”** provides a description of each instruction.

**TABLE 24-1: OPCODE FIELD DESCRIPTIONS**

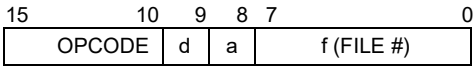
| Field | Description |
|---|---|
| a | RAM access bit<br>a = 0: RAM location in Access RAM (BSR register is ignored)<br>a = 1: RAM bank is specified by BSR register |
| bbb | Bit address within an 8-bit file register (0 to 7). |
| BSR | Bank Select Register. Used to select the current RAM bank. |
| C, DC, Z, OV, N | ALU Status bits: **C**arry, **D**igit **C**arry, **Z**ero, **Ov**erflow, **N**egative. |
| d | Destination select bit<br>d = 0: store result in WREG<br>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 (00h to FFh) or 2-bit FSR designator (0h to 3h). |
| $f_s$ | 12-bit Register file address (000h to FFFh). This is the source address. |
| $f_d$ | 12-bit Register file address (000h to FFFh). This is the destination address. |
| GIE | Global Interrupt Enable bit. |
| k | Literal field, constant data or label (may be either an 8-bit, 12-bit or a 20-bit value). |
| label | Label name. |
| mm<br><br>*<br>*+<br>*-<br>+* | The mode of the TBLPTR register for the table read and table write instructions.<br>Only used with table read and table write instructions:<br>No change to register (such as TBLPTR with table reads and writes)<br>Post-Increment register (such as TBLPTR with table reads and writes)<br>Post-Decrement register (such as TBLPTR with table reads and writes)<br>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. |
| 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. |
| $\overline{PD}$ | Power-down bit. |
| PRODH | Product of Multiply High Byte. |
| PRODL | Product of Multiply Low Byte. |
| s | Fast Call/Return mode select bit<br>s = 0: do not update into/from shadow registers<br>s = 1: certain registers loaded into/from shadow registers (Fast mode) |
| TBLPTR | 21-bit Table Pointer (points to a Program Memory location). |
| TABLAT | 8-bit Table Latch. |
| $\overline{TO}$ | Time-out bit. |
| TOS | Top-of-Stack. |
| u | Unused or unchanged. |
| WDT | Watchdog Timer. |
| 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. |
| $z_s$ | 7-bit offset value for indirect addressing of register files (source). |
| $z_d$ | 7-bit offset value for indirect addressing of register files (destination). |
| { } | Optional argument. |
| [text] | Indicates an indexed address. |
| (text) | The contents of text. |
| [expr]<n> | Specifies bit n of the register indicated by the pointer expr. |
| → | Assigned to. |
| < > | Register bit field. |
| ∈ | In the set of. |
| italics | User defined term (font is Courier). |

**FIGURE 24-1: GENERAL FORMAT FOR INSTRUCTIONS**

**Byte-oriented** file register operations

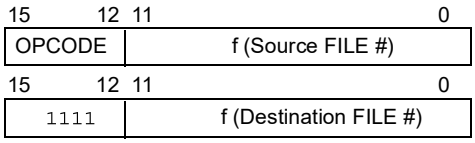
**Example Instruction**




```
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

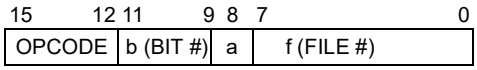
**Byte to Byte** move operations (2-word)




```
MOVFF MYREG1, MYREG2
```

f = 12-bit file register address

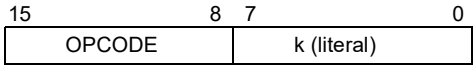
**Bit-oriented** file register operations




```
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

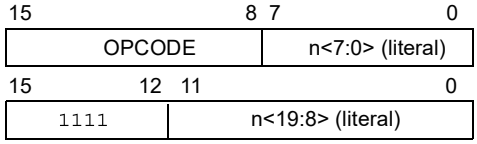



```
MOVLW 7Fh
```

k = 8-bit immediate value

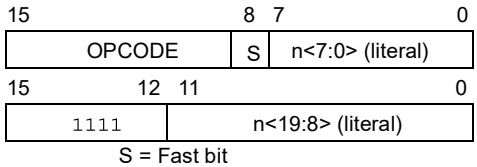
**Control** operations

**CALL, GOTO and Branch** operations



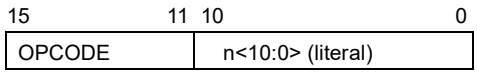

```
GOTO Label
```

n = 20-bit immediate value

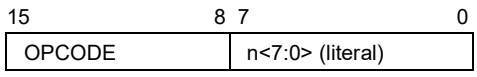



```
CALL MYFUNC
```

S = Fast bit




```
BRA MYFUNC
```




```
BC MYFUNC
```

**TABLE 24-2: PIC18FXXXX INSTRUCTION SET**

| Mnemonic, Operands | | Description | Cycles | 16-Bit Instruction Word | | | | Status Affected | Notes |
|---|---|---|---|---|---|---|---|---|---|
| | | | | MSb | | | LSb | | |
| **BYTE-ORIENTED 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<br>$f_d$ (destination) 2nd word | 2 | 1100<br>1111 | ffff<br>ffff | ffff<br>ffff | ffff<br>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 | 1, 2 |
| NEGF | f, a | Negate f | 1 | 0110 | 110a | ffff | ffff | C, DC, Z, OV, N | |
| RLCF | f, d, a | Rotate Left f through Carry | 1 | 0011 | 01da | ffff | ffff | C, Z, N | 1, 2 |
| RLNCF | f, d, a | Rotate Left f (No Carry) | 1 | 0100 | 01da | ffff | ffff | Z, N | |
| 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 | 1, 2 |
| SUBFWB | f, d, a | Subtract f from WREG with borrow | 1 | 0101 | 01da | ffff | ffff | C, DC, Z, OV, N | |
| SUBWF | f, d, a | Subtract WREG from f | 1 | 0101 | 11da | ffff | ffff | C, DC, Z, OV, N | 1, 2 |
| SUBWFB | f, d, a | Subtract WREG from f with borrow | 1 | 0101 | 10da | ffff | ffff | C, DC, Z, OV, N | |
| 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 | |

**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 two-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.

**TABLE 24-2: PIC18FXXXX INSTRUCTION SET (CONTINUED)**

| Mnemonic, Operands | | Description | Cycles | 16-Bit Instruction Word | | | | Status Affected | Notes |
|---|---|---|---|---|---|---|---|---|---|
| | | | | MSb | | | LSb | | |
| **BIT-ORIENTED 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 |
| **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 | 1 (2) | 1110 | 0001 | nnnn | nnnn | None | |
| BOV | n | Branch if Overflow | 1 (2) | 1110 | 0100 | nnnn | nnnn | None | |
| BRA | n | Branch Unconditionally | 2 | 1101 | 0nnn | nnnn | nnnn | None | |
| BZ | n | Branch if Zero | 1 (2) | 1110 | 0000 | nnnn | nnnn | None | |
| CALL | n, s | Call subroutine 1st 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 address 1st 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 two-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.

**TABLE 24-2: PIC18FXXXX INSTRUCTION SET (CONTINUED)**

| Mnemonic, Operands | | Description | Cycles | 16-Bit Instruction Word MSb | | | LSb | Status Affected | Notes |
|---|---|---|---|---|---|---|---|---|---|
| **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 FSR(f) 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 | 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 two-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.

### 24.1.1 STANDARD INSTRUCTION SET

**ADDLW** **ADD literal to W**

Syntax: ADDLW k

Operands: $0 \le k \le 255$

Operation: $(W) + k \rightarrow W$

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 15h

Before Instruction
W = 10h
After Instruction
W = 25h

**ADDWF** **ADD W to f**

Syntax: ADDWF f {,d {,a}}

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

Operation: $(W) + (f) \rightarrow$ 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 17h
REG = 0C2h
After Instruction
W = 0D9h
REG = 0C2h

**Note:** All PIC18 instructions may take an optional label argument preceding the instruction mnemonic for use in symbolic addressing. If a label is used, the instruction format then becomes: {label} instruction argument(s).

**ADDWFC** **ADD W and CARRY bit to f**

Syntax: ADDWFC f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 02h
W = 4Dh
After Instruction
CARRY bit = 0
REG = 02h
W = 50h

**ANDLW** **AND literal with W**

Syntax: ANDLW k

Operands: $0 \le k \le 255$

Operation: (W) .AND. k → W

Status Affected: N, Z

Encoding:

| | | | |
|---|---|---|---|
| 0000 | 1011 | kkkk | kkkk |

Description: The contents of W are AND'ed 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 05Fh

Before Instruction
W = A3h
After Instruction
W = 03h

**ANDWF** **AND W with f**

Syntax: ANDWF f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 17h
REG = C2h
After Instruction
W = 02h
REG = C2h

**BC** **Branch if Carry**

Syntax: BC n

Operands: $-128 \le n \le 127$

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: BCF f, b {,a}

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: $0 \rightarrow$ 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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = C7h
After Instruction
FLAG_REG = 47h

**BN** **Branch if Negative**

Syntax: BN n

Operands: $-128 \le n \le 127$

Operation: if NEGATIVE bit is '1'
$(PC) + 2 + 2n \rightarrow PC$

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: 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: 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: 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: 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: 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: 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', the Access Bank is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 0Ah
After Instruction
FLAG_REG = 8Ah

**BTFSC** **Bit Test File, Skip if Clear**

Syntax: BTFSC f, b {,a}

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh).
See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: BTFSS f, b {,a}

Operands: $0 \le f \le 255$
$0 \le b < 7$
$a \in [0,1]$

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh).
See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: BTG f, b {,a}

Operands: $0 \le f \le 255$
$0 \le b < 7$
$a \in [0,1]$

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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `BTG PORTC, 4, 0`

Before Instruction:
PORTC = 0111 0101 [75h]
After Instruction:
PORTC = 0110 0101 [65h]

**BOV** **Branch if Overflow**

Syntax: BOV n

Operands: $-128 \le n \le 127$

Operation: if OVERFLOW bit is '1'
$(PC) + 2 + 2n \rightarrow PC$

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 |

Example:
```
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: 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 |

Example: 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: 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 |

Example: 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: CLRF f {,a}

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

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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 5Ah
After Instruction
FLAG_REG = 00h

**CLRWDT** **Clear Watchdog Timer**

Syntax: 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 = 00h
WDT Postscaler = 0
$\overline{TO}$ = 1
$\overline{PD}$ = 1

**COMF** **Complement f**

Syntax: COMF f {,d {,a}}

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

Operation: $(\bar{f}) \rightarrow$ 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 13h
After Instruction
REG = 13h
W = ECh

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

Syntax: CPFSEQ f {,a}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: CPFSGT f {,a}

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

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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: CPFSLT f {,a}

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

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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (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: 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> + DC > 9] or [C = 1] then
(W<7:4>) + 6 + DC → W<7:4> ;
else
(W<7:4>) + DC → 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 = A5h
C = 0
DC = 0
After Instruction
W = 05h
C = 1
DC = 0

Example 2:

Before Instruction
W = CEh
C = 0
DC = 0
After Instruction
W = 34h
C = 1
DC = 0

**DECF** **Decrement f**

Syntax: 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 01h
Z = 0
After Instruction
CNT = 00h
Z = 1

**DECFSZ** **Decrement f, skip if 0**

Syntax: DECFSZ f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default). If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: DCFSNZ f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default). If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: GOTO k

Operands: $0 \le k \le 1048575$

Operation: $k \rightarrow PC<20:1>$

Status Affected: None

Encoding:

| | | | | |
|---|---|---|---|---|
| 1st word (k<7:0>) | 1110 | 1111 | $k_7kkk$ | $kkkk_0$ |
| 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: INCF f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = FFh
Z = 0
C = ?
DC = ?
After Instruction
CNT = 00h
Z = 1
C = 1
DC = 1

**INCFSZ** **Increment f, skip if 0**

Syntax: INCFSZ f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: INFSNZ f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: IORLW k

Operands: $0 \le k \le 255$

Operation: (W) .OR. k → W

Status Affected: N, Z

Encoding:

| | | | |
|---|---|---|---|
| 0000 | 1001 | kkkk | kkkk |

Description: The contents of W are ORed 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 35h

Before Instruction
W = 9Ah
After Instruction
W = BFh

**IORWF** **Inclusive OR W with f**

Syntax: IORWF f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 13h
W = 91h
After Instruction
RESULT = 13h
W = 93h

**LFSR** **Load FSR**

Syntax: 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, 3ABh`

After Instruction
FSR2H = 03h
FSR2L = ABh

**MOVF** **Move f**

Syntax: 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 22h
W = FFh
After Instruction
REG = 22h
W = 22h

**MOVFF** **Move f to f**

Syntax: 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$_s$ |
| 2nd word (destin.) | 1111 | 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.

Words: 2

Cycles: 2 (3)

Q Cycle Activity:

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

Example: MOVFF REG1, REG2

Before Instruction
REG1 = 33h
REG2 = 11h
After Instruction
REG1 = 33h
REG2 = 33h

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

Syntax: MOVLW k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow$ BSR

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 0000 | 0001 | kkkk | kkkk |

Description: The eight-bit literal 'k' is loaded into the Bank Select Register (BSR). The value of BSR<7:4> always remains '0', regardless of the value of $k_7$:$k_4$.

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 = 02h
After Instruction
BSR Register = 05h

**MOVLW** **Move literal to W**

Syntax: MOVLW k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow W$

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 5Ah`

After Instruction

W = 5Ah

**MOVWF** **Move W to f**

Syntax: MOVWF f {,a}

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

Operation: $(W) \rightarrow f$

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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 4Fh
REG = FFh

After Instruction

W = 4Fh
REG = 4Fh

**MULLW** **Multiply literal with W**

Syntax: MULLW k

Operands: $0 \le k \le 255$

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 the 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 0C4h

Before Instruction
W = E2h
PRODH = ?
PRODL = ?

After Instruction
W = E2h
PRODH = ADh
PRODL = 08h

**MULWF** **Multiply W with f**

Syntax: MULWF f {,a}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = C4h
REG = B5h
PRODH = ?
PRODL = ?

After Instruction
W = C4h
REG = B5h
PRODH = 8Ah
PRODL = 94h

**NEGF** **Negate f**

Syntax: 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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 [3Ah]
After Instruction
REG = 1100 0110 [C6h]

**NOP** **No Operation**

Syntax: 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: 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: 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 implementing a software stack by modifying TOS and then pushing 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 = 345Ah
PC = 0124h

After Instruction
PC = 0126h
TOS = 0126h
Stack (1 level down) = 345Ah

**RCALL** **Relative Call**

Syntax: 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: RESET

Operands: None

Operation: Reset all registers and flags that are affected by a MCLR Reset.

Status Affected: All

Encoding:

| | | | |
|---|---|---|---|
| 0000 | 0000 | 1111 | 1111 |

Description: This instruction provides a way to execute a 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: RETFIE {s}

Operands: $s \in [0,1]$

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<br>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: RETLW k

Operands: $0 \le k \le 255$

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 = 07h
After Instruction
W = value of kn

**RETURN** **Return from Subroutine**

Syntax: 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 Instruction:
PC = TOS

**RLCF** **Rotate Left f through Carry**

Syntax: RLCF f {,d {,a}}

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

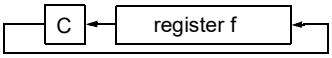
Operation: (f<n>) → dest<n + 1>,
(f<7>) → C,
(C) → dest<0>

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.




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: RLNCF f {,d {,a}}

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

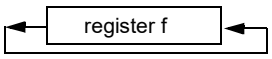
Operation: (f<n>) → dest<n + 1>,
(f<7>) → dest<0>

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.




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: RRCF f {,d {,a}}

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

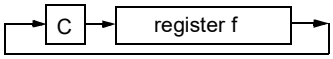
Operation: (f<n>) → dest<n – 1>,
(f<0>) → C,
(C) → dest<7>

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.




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: RRNCF f {,d {,a}}

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

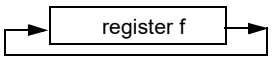
Operation: (f<n>) → dest<n – 1>,
(f<0>) → dest<7>

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).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.




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: SETF f {,a}

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

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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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 = 5Ah
After Instruction
REG = FFh

**SLEEP** **Enter Sleep mode**

Syntax: SLEEP

Operands: None

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

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

Encoding:

| | | | |
|---|---|---|---|
| 0000 | 0000 | 0000 | 0011 |

Description: The Power-down status bit ($\overline{PD}$) is cleared. The Time-out status bit ($\overline{TO}$) 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
$\overline{TO}$ = ?
$\overline{PD}$ = ?
After Instruction
$\overline{TO}$ = 1 †
$\overline{PD}$ = 0

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

**SUBFWB** **Subtract f from W with borrow**

Syntax: SUBFWB f {,d {,a}}

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

Operation: (W) – (f) – ($\overline{C}$) → 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: SUBLW k

Operands: $0 \le k \le 255$

Operation: $k - (W) \rightarrow W$

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   02h
```
Before Instruction
W = 01h
C = ?
After Instruction
W = 01h
C = 1 ; result is positive
Z = 0
N = 0

Example 2:
```
SUBLW   02h
```
Before Instruction
W = 02h
C = ?
After Instruction
W = 00h
C = 1 ; result is zero
Z = 1
N = 0

Example 3:
```
SUBLW   02h
```
Before Instruction
W = 03h
C = ?
After Instruction
W = FFh ; (2's complement)
C = 0 ; result is negative
Z = 0
N = 1

**SUBWF** **Subtract W from f**

Syntax: SUBWF f {,d {,a}}

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

Operation: $(f) - (W) \rightarrow$ 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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: SUBWFB f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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  =  19h   (0001 1001)
    W    =  0Dh   (0000 1101)
    C    =  1
After Instruction
    REG  =  0Ch   (0000 1011)
    W    =  0Dh   (0000 1101)
    C    =  1
    Z    =  0
    N    =  0     ; result is positive
```

Example 2:
```
SUBWFB  REG, 0, 0
Before Instruction
    REG  =  1Bh   (0001 1011)
    W    =  1Ah   (0001 1010)
    C    =  0
After Instruction
    REG  =  1Bh   (0001 1011)
    W    =  00h
    C    =  1
    Z    =  1     ; result is zero
    N    =  0
```

Example 3:
```
SUBWFB  REG, 1, 0
Before Instruction
    REG  =  03h   (0000 0011)
    W    =  0Eh   (0000 1101)
    C    =  1
After Instruction
    REG  =  F5h   (1111 0100)
                  ; [2's comp]
    W    =  0Eh   (0000 1101)
    C    =  0
    Z    =  0
    N    =  1     ; result is negative
```

**SWAPF** **Swap f**

Syntax: SWAPF f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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  =  53h
After Instruction
    REG  =  35h
```

**TBLRD Table Read**

Syntax: 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<br>nn=0 *<br>=1 *+<br>=2 *-<br>=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 (Continued)**

Example1:
```
TBLRD  *+ ;
```
Before Instruction
TABLAT = 55h
TBLPTR = 00A356h
MEMORY (00A356h) = 34h
After Instruction
TABLAT = 34h
TBLPTR = 00A357h

Example2:
```
TBLRD  +* ;
```
Before Instruction
TABLAT = AAh
TBLPTR = 01A357h
MEMORY (01A357h) = 12h
MEMORY (01A358h) = 34h
After Instruction
TABLAT = 34h
TBLPTR = 01A358h

**TBLWT** **Table Write**

Syntax: 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<br>nn=0 *<br>=1 *+<br>=2 *-<br>=3 +* |

Description: This instruction uses the 3 LSBs of TBLPTR to determine which of the 8 holding registers the TABLAT is written to. The holding registers are used to program the contents of Program Memory (P.M.). (Refer to **Section 6.0 "Flash Program Memory"** for additional details on programming Flash memory.) The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2-MByte 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 ) |

**TBLWT** **Table Write (Continued)**

Example1: TBLWT *+;

Before Instruction
TABLAT = 55h
TBLPTR = 00A356h
HOLDING REGISTER (00A356h) = FFh
After Instructions (table write completion)
TABLAT = 55h
TBLPTR = 00A357h
HOLDING REGISTER (00A356h) = 55h

Example 2: TBLWT +*;

Before Instruction
TABLAT = 34h
TBLPTR = 01389Ah
HOLDING REGISTER (01389Ah) = FFh
HOLDING REGISTER (01389Bh) = FFh
After Instruction (table write completion)
TABLAT = 34h
TBLPTR = 01389Bh
HOLDING REGISTER (01389Ah) = FFh
HOLDING REGISTER (01389Bh) = 34h

**TSTFSZ** **Test f, skip if 0**

Syntax: TSTFSZ f {,a}

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

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 is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

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    TSTFSZ  CNT, 1
NZERO   :
ZERO    :
```
Before Instruction
PC = Address (HERE)
After Instruction
If CNT = 00h,
PC = Address (ZERO)
If CNT ≠ 00h,
PC = Address (NZERO)

**XORLW** **Exclusive OR literal with W**

Syntax: XORLW k

Operands: $0 \le k \le 255$

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 |

Example:
```
XORLW   0AFh
```
Before Instruction
W = B5h
After Instruction
W = 1Ah

**XORWF** **Exclusive OR W with f**

Syntax: XORWF f {,d {,a}}

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

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 is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `XORWF REG, 1, 0`

Before Instruction
REG = AFh
W = B5h
After Instruction
REG = 1Ah
W = B5h

## 24.2 Extended Instruction Set

In addition to the standard 75 instructions of the PIC18 instruction set, PIC18F2420/2520/4420/4520 devices also provide an optional extension to the core CPU functionality. The added features include eight additional instructions that augment indirect and indexed addressing operations and the implementation of Indexed Literal Offset Addressing mode for many of the standard PIC18 instructions.

The additional features of the extended instruction set are disabled by default. To enable them, users must set the XINST configuration bit.

The instructions in the extended set can all be classified as literal operations, which either manipulate the File Select Registers, or use them for indexed addressing. Two of the instructions, ADDFSR and SUBFSR, each have an additional special instantiation for using FSR2. These versions (ADDULNK and SUBULNK) allow for automatic return after execution.

The extended instructions are specifically implemented to optimize re-entrant program code (that is, code that is recursive or that uses a software stack) written in high-level languages, particularly C. Among other things, they allow users working in high-level languages to perform certain operations on data structures more efficiently. These include:

- dynamic allocation and deallocation of software stack space when entering and leaving subroutines
- function pointer invocation
- software stack pointer manipulation
- manipulation of variables located in a software stack

A summary of the instructions in the extended instruction set is provided in Table 24-3. Detailed descriptions are provided in **Section 24.2.2 "Extended Instruction Set"**. The opcode field descriptions in Table 24-1 (page 268) apply to both the standard and extended PIC18 instruction sets.

| **Note:** | The instruction set extension and the Indexed Literal Offset Addressing mode were designed for optimizing applications written in C; the user may likely never use these instructions directly in assembler. The syntax for these commands is provided as a reference for users who may be reviewing code that has been generated by a compiler. |
|---|---|

### 24.2.1 EXTENDED INSTRUCTION SYNTAX

Most of the extended instructions use indexed arguments, using one of the File Select Registers and some offset to specify a source or destination register. When an argument for an instruction serves as part of indexed addressing, it is enclosed in square brackets ("[ ]"). This is done to indicate that the argument is used as an index or offset. MPASM™ Assembler will flag an error if it determines that an index or offset value is not bracketed.

When the extended instruction set is enabled, brackets are also used to indicate index arguments in byte-oriented and bit-oriented instructions. This is in addition to other changes in their syntax. For more details, see **Section 24.2.3.1 "Extended Instruction Syntax with Standard PIC18 Commands"**.

| **Note:** | In the past, square brackets have been used to denote optional arguments in the PIC18 and earlier instruction sets. In this text and going forward, optional arguments are denoted by braces ("{ }"). |
|---|---|

**TABLE 24-3: EXTENSIONS TO THE PIC18 INSTRUCTION SET**

| Mnemonic, Operands | | Description | Cycles | 16-Bit Instruction Word | | | | Status Affected |
|---|---|---|---|---|---|---|---|---|
| | | | | MSb | | | LSb | |
| ADDFSR | f, k | Add literal to FSR | 1 | 1110 | 1000 | ffkk | kkkk | None |
| ADDULNK | k | Add literal to FSR2 and return | 2 | 1110 | 1000 | 11kk | kkkk | None |
| CALLW | | Call subroutine using WREG | 2 | 0000 | 0000 | 0001 | 0100 | None |
| MOVSF | $z_s$, $f_d$ | Move $z_s$ (source) to 1st word | 2 | 1110 | 1011 | 0zzz | zzzz | None |
| | | $f_d$ (destination) 2nd word | | 1111 | ffff | ffff | ffff | |
| MOVSS | $z_s$, $z_d$ | Move $z_s$ (source) to 1st word | 2 | 1110 | 1011 | 1zzz | zzzz | None |
| | | $z_d$ (destination) 2nd word | | 1111 | xxxx | xzzz | zzzz | |
| PUSHL | k | Store literal at FSR2, decrement FSR2 | 1 | 1110 | 1010 | kkkk | kkkk | None |
| SUBFSR | f, k | Subtract literal from FSR | 1 | 1110 | 1001 | ffkk | kkkk | None |
| SUBULNK | k | Subtract literal from FSR2 and return | 2 | 1110 | 1001 | 11kk | kkkk | None |

### 24.2.2 EXTENDED INSTRUCTION SET

**ADDFSR** **Add Literal to FSR**

Syntax: ADDFSR f, k

Operands: $0 \le k \le 63$
$f \in [0, 1, 2]$

Operation: FSR(f) + k → FSR(f)

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 1110 | 1000 | ffkk | kkkk |

Description: The 6-bit literal 'k' is added to the contents of the FSR specified by 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read literal 'k' | Process Data | Write to FSR |

Example: ADDFSR 2, 23h

Before Instruction
FSR2 = 03FFh
After Instruction
FSR2 = 0422h

**ADDULNK** **Add Literal to FSR2 and Return**

Syntax: ADDULNK k

Operands: $0 \le k \le 63$

Operation: FSR2 + k → FSR2,
(TOS) → PC

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 1110 | 1000 | 11kk | kkkk |

Description: The 6-bit literal 'k' is added to the contents of FSR2. A RETURN is then executed by loading the PC with the TOS.
The instruction takes two cycles to execute; a NOP is performed during the second cycle.
This may be thought of as a special case of the ADDFSR instruction, where f = 3 (binary '11'); it operates only on FSR2.

Words: 1

Cycles: 2

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read literal 'k' | Process Data | Write to FSR |
| No Operation | No Operation | No Operation | No Operation |

Example: ADDULNK 23h

Before Instruction
FSR2 = 03FFh
PC = 0100h
After Instruction
FSR2 = 0422h
PC = (TOS)

**Note:** All PIC18 instructions may take an optional label argument preceding the instruction mnemonic for use in symbolic addressing. If a label is used, the instruction syntax then becomes: {label} instruction argument(s).

**CALLW** **Subroutine Call Using WREG**

Syntax: CALLW

Operands: None

Operation: (PC + 2) → TOS,
(W) → PCL,
(PCLATH) → PCH,
(PCLATU) → PCU

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 0000 | 0000 | 0001 | 0100 |

Description: First, the return address (PC + 2) is pushed onto the return stack. Next, the contents of W are written to PCL; the existing value is discarded. Then, the contents of PCLATH and PCLATU are latched into PCH and PCU, respectively. The second cycle is executed as a NOP instruction while the new next instruction is fetched. Unlike CALL, there is no option to update W, Status or BSR.

Words: 1

Cycles: 2

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read WREG | PUSH PC to stack | No operation |
| No operation | No operation | No operation | No operation |

Example: HERE CALLW

Before Instruction
PC = address (HERE)
PCLATH = 10h
PCLATU = 00h
W = 06h
After Instruction
PC = 001006h
TOS = address (HERE + 2)
PCLATH = 10h
PCLATU = 00h
W = 06h

**MOVSF** **Move Indexed to f**

Syntax: MOVSF [$z_s$], $f_d$

Operands: $0 \le z_s \le 127$
$0 \le f_d \le 4095$

Operation: ((FSR2) + $z_s$) → $f_d$

Status Affected: None

Encoding:

| | | | | |
|---|---|---|---|---|
| 1st word (source) | 1110 | 1011 | 0zzz | $zzzz_s$ |
| 2nd word (destin.) | 1111 | ffff | ffff | $ffff_d$ |

Description: The contents of the source register are moved to destination register '$f_d$'. The actual address of the source register is determined by adding the 7-bit literal offset '$z_s$' in the first word to the value of FSR2. The address of the destination register is specified by the 12-bit literal '$f_d$' in the second word. Both addresses can be anywhere in the 4096-byte data space (000h to FFFh).
The MOVSF instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register.
If the resultant source address points to an indirect addressing register, the value returned will be 00h.

Words: 2

Cycles: 2

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Determine source addr | Determine source addr | Read source reg |
| Decode | No operation<br>No dummy read | No operation | Write register 'f' (dest) |

Example: MOVSF [05h], REG2

Before Instruction
FSR2 = 80h
Contents of 85h = 33h
REG2 = 11h
After Instruction
FSR2 = 80h
Contents of 85h = 33h
REG2 = 33h

**MOVSS** **Move Indexed to Indexed**

Syntax: MOVSS [$z_s$], [$z_d$]

Operands: $0 \le z_s \le 127$
$0 \le z_d \le 127$

Operation: ((FSR2) + $z_s$) → ((FSR2) + $z_d$)

Status Affected: None

Encoding:

| | | | | |
|---|---|---|---|---|
| 1st word (source) | 1110 | 1011 | 1zzz | $zzzz_s$ |
| 2nd word (dest.) | 1111 | xxxx | xzzz | $zzzz_d$ |

Description: The contents of the source register are moved to the destination register. The addresses of the source and destination registers are determined by adding the 7-bit literal offsets '$z_s$' or '$z_d$', respectively, to the value of FSR2. Both registers can be located anywhere in the 4096-byte data memory space (000h to FFFh).
The MOVSS instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register.
If the resultant source address points to an indirect addressing register, the value returned will be 00h. If the resultant destination address points to an indirect addressing register, the instruction will execute as a NOP.

Words: 2

Cycles: 2

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Determine source addr | Determine source addr | Read source reg |
| Decode | Determine dest addr | Determine dest addr | Write to dest reg |

Example: MOVSS [05h], [06h]

Before Instruction
FSR2 = 80h
Contents of 85h = 33h
Contents of 86h = 11h
After Instruction
FSR2 = 80h
Contents of 85h = 33h
Contents of 86h = 33h

**PUSHL** **Store Literal at FSR2, Decrement FSR2**

Syntax: PUSHL k

Operands: $0 \le k \le 255$

Operation: k → (FSR2),
FSR2 – 1 → FSR2

Status Affected: None

Encoding:

| 1111 | 1010 | kkkk | kkkk |
|---|---|---|---|

Description: The 8-bit literal 'k' is written to the data memory address specified by FSR2. FSR2 is decremented by 1 after the operation. This instruction allows users to push values onto a software stack.

Words: 1

Cycles: 1

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read 'k' | Process data | Write to destination |

Example: PUSHL 08h

Before Instruction
FSR2H:FSR2L = 01ECh
Memory (01ECh) = 00h

After Instruction
FSR2H:FSR2L = 01EBh
Memory (01ECh) = 08h

**SUBFSR** **Subtract Literal from FSR**

Syntax: SUBFSR f, k

Operands: $0 \le k \le 63$

$f \in [0, 1, 2]$

Operation: FSR(f) – k → FSRf

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 1110 | 1001 | ffkk | kkkk |

Description: The 6-bit literal 'k' is subtracted from the contents of the FSR specified by 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: SUBFSR 2, 23h

Before Instruction
FSR2 = 03FFh

After Instruction
FSR2 = 03DCh

**SUBULNK** **Subtract Literal from FSR2 and Return**

Syntax: SUBULNK k

Operands: $0 \le k \le 63$

Operation: FSR2 – k → FSR2
(TOS) → PC

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 1110 | 1001 | 11kk | kkkk |

Description: The 6-bit literal 'k' is subtracted from the contents of the FSR2. A RETURN is then executed by loading the PC with the TOS. The instruction takes two cycles to execute; a NOP is performed during the second cycle.
This may be thought of as a special case of the SUBFSR instruction, where f = 3 (binary '11'); it operates only on FSR2.

Words: 1

Cycles: 2

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read register 'f' | Process Data | Write to destination |
| No Operation | No Operation | No Operation | No Operation |

Example: SUBULNK 23h

Before Instruction
FSR2 = 03FFh
PC = 0100h

After Instruction
FSR2 = 03DCh
PC = (TOS)

### 24.2.3 BYTE-ORIENTED AND BIT-ORIENTED INSTRUCTIONS IN INDEXED LITERAL OFFSET MODE

| **Note:** | Enabling the PIC18 instruction set extension may cause legacy applications to behave erratically or fail entirely. |
|---|---|

In addition to eight new commands in the extended set, enabling the extended instruction set also enables Indexed Literal Offset Addressing mode (**Section 5.5.1 "Indexed Addressing with Literal Offset"**). This has a significant impact on the way that many commands of the standard PIC18 instruction set are interpreted.

When the extended set is disabled, addresses embedded in opcodes are treated as literal memory locations: either as a location in the Access Bank ('a' = 0), or in a GPR bank designated by the BSR ('a' = 1). When the extended instruction set is enabled and 'a' = 0, however, a file register argument of 5Fh or less is interpreted as an offset from the pointer value in FSR2 and not as a literal address. For practical purposes, this means that all instructions that use the Access RAM bit as an argument – that is, all byte-oriented and bit-oriented instructions, or almost half of the core PIC18 instructions – may behave differently when the extended instruction set is enabled.

When the content of FSR2 is 00h, the boundaries of the Access RAM are essentially remapped to their original values. This may be useful in creating backward compatible code. If this technique is used, it may be necessary to save the value of FSR2 and restore it when moving back and forth between C and assembly routines in order to preserve the stack pointer. Users must also keep in mind the syntax requirements of the extended instruction set (see **Section 24.2.3.1 "Extended Instruction Syntax with Standard PIC18 Commands"**).

Although the Indexed Literal Offset Addressing mode can be very useful for dynamic stack and pointer manipulation, it can also be very annoying if a simple arithmetic operation is carried out on the wrong register. Users who are accustomed to the PIC18 programming must keep in mind that, when the extended instruction set is enabled, register addresses of 5Fh or less are used for Indexed Literal Offset Addressing.

Representative examples of typical byte-oriented and bit-oriented instructions in the Indexed Literal Offset Addressing mode are provided on the following page to show how execution is affected. The operand conditions shown in the examples are applicable to all instructions of these types.

#### 24.2.3.1 Extended Instruction Syntax with Standard PIC18 Commands

When the extended instruction set is enabled, the file register argument, 'f', in the standard byte-oriented and bit-oriented commands is replaced with the literal offset value, 'k'. As already noted, this occurs only when 'f' is less than or equal to 5Fh. When an offset value is used, it must be indicated by square brackets ("[ ]"). As with the extended instructions, the use of brackets indicates to the compiler that the value is to be interpreted as an index or an offset. Omitting the brackets, or using a value greater than 5Fh within brackets, will generate an error in the MPASM Assembler.

If the index argument is properly bracketed for Indexed Literal Offset Addressing, the Access RAM argument is never specified; it will automatically be assumed to be '0'. This is in contrast to standard operation (extended instruction set disabled) when 'a' is set on the basis of the target address. Declaring the Access RAM bit in this mode will also generate an error in the MPASM Assembler.

The destination argument, 'd', functions as before.

In the latest versions of the MPASM assembler, language support for the extended instruction set must be explicitly invoked. This is done with either the command line option, /y, or the PE directive in the source listing.

### 24.2.4 CONSIDERATIONS WHEN ENABLING THE EXTENDED INSTRUCTION SET

It is important to note that the extensions to the instruction set may not be beneficial to all users. In particular, users who are not writing code that uses a software stack may not benefit from using the extensions to the instruction set.

Additionally, the Indexed Literal Offset Addressing mode may create issues with legacy applications written to the PIC18 assembler. This is because instructions in the legacy code may attempt to address registers in the Access Bank below 5Fh. Since these addresses are interpreted as literal offsets to FSR2 when the instruction set extension is enabled, the application may read or write to the wrong data addresses.

When porting an application to the PIC18F2420/2520/4420/4520, it is very important to consider the type of code. A large, re-entrant application that is written in 'C' and would benefit from efficient compilation will do well when using the instruction set extensions. Legacy applications that heavily use the Access Bank will most likely not benefit from using the extended instruction set.

**ADDWF** **ADD W to Indexed (Indexed Literal Offset mode)**

Syntax: ADDWF [k] {,d}

Operands: $0 \le k \le 95$
$d \in [0,1]$

Operation: (W) + ((FSR2) + k) → dest

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

Encoding:

| | | | |
|---|---|---|---|
| 0010 | 01d0 | kkkk | kkkk |

Description: The contents of W are added to the contents of the register indicated by FSR2, offset by the value 'k'.
If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default).

Words: 1

Cycles: 1

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read 'k' | Process Data | Write to destination |

Example: ADDWF [OFST] , 0

Before Instruction
W = 17h
OFST = 2Ch
FSR2 = 0A00h
Contents of 0A2Ch = 20h
After Instruction
W = 37h
Contents of 0A2Ch = 20h

**BSF** **Bit Set Indexed (Indexed Literal Offset mode)**

Syntax: BSF [k], b

Operands: $0 \le f \le 95$
$0 \le b \le 7$

Operation: 1 → ((FSR2) + k)<b>

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 1000 | bbb0 | kkkk | kkkk |

Description: Bit 'b' of the register indicated by FSR2, offset by the value 'k', is set.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: BSF [FLAG_OFST], 7

Before Instruction
FLAG_OFST = 0Ah
FSR2 = 0A00h
Contents of 0A0Ah = 55h
After Instruction
Contents of 0A0Ah = D5h

**SETF** **Set Indexed (Indexed Literal Offset mode)**

Syntax: SETF [k]

Operands: $0 \le k \le 95$

Operation: FFh → ((FSR2) + k)

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 0110 | 1000 | kkkk | kkkk |

Description: The contents of the register indicated by FSR2, offset by 'k', are set to FFh.

Words: 1

Cycles: 1

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read 'k' | Process Data | Write register |

Example: SETF [OFST]

Before Instruction
OFST = 2Ch
FSR2 = 0A00h
Contents of 0A2Ch = 00h
After Instruction
Contents of 0A2Ch = FFh

### 24.2.5 SPECIAL CONSIDERATIONS WITH MICROCHIP MPLAB® IDE TOOLS

The latest versions of Microchip's software tools have been designed to fully support the extended instruction set of the PIC18F2420/2520/4420/4520 family of devices. This includes the MPLAB C18 C compiler, MPASM assembly language and MPLAB Integrated Development Environment (IDE).

When selecting a target device for software development, MPLAB IDE will automatically set default configuration bits for that device. The default setting for the XINST configuration bit is '0', disabling the extended instruction set and Indexed Literal Offset Addressing mode. For proper execution of applications developed to take advantage of the extended instruction set, XINST must be set during programming.

To develop software for the extended instruction set, the user must enable support for the instructions and the Indexed Addressing mode in their language tool(s). Depending on the environment being used, this may be done in several ways:

- A menu option, or dialog box within the environment, that allows the user to configure the language tool and its settings for the project
- A command line option
- A directive in the source code

These options vary between different compilers, assemblers and development environments. Users are encouraged to review the documentation accompanying their development systems for the appropriate information.