PIC16F887 INSTRUCTION SET

In the sections below, the operands following the instructions are as follows:

f : the file register to operate on, e.g. PORTA or a GPR address
d : the destination for the result of the operation. This can be set to either 'f' or 'w', with 'f' being the default if no destination is given (note, it is best practice to always explicitly set the destination). If the destination is 'f', the result of the operation goes back to the file register that was being operated on. If the destination is 'w', the result goes to the W register. For example:

```
ADDWF  PORTA, f   ; PORTA = PORTA + W
ADDWF  PORTA, w   ; W = PORTA + W
```

k : a constant numeric value
L : a label or address
b : the bit of a file register to operate on, in the range 0-7.

The status bits affected by the instructions are any of:

Z : the operation resulted in a zero
C : the operation caused a carry operation (arithmetic overflow)
DC : the operation caused a digit/decimal carry

Note that it is not necessary to clear or set status bits before you check them, they will be set appropriately.

General byte manipulation

Copying Data

```
MOVF     f, d          ; Z affected
MOVWF    f
MOVLW    k
```

The move instructions are the most used of all of the instruction set and allow data to be moved to and from the file registers and the working register. The "move" here is actually a copy, the valued in the source register remains the same.

`MOVF` moves from a file register to either the same file register (which does have a use, despite not looking so), or to the working register. The assembler provides an alias `MOVFW f` which is the same as `MOVF f,W`.

`MOVWF` moves a value from the working register to a file register.

`MOVLW` sets the working register to a constant value defined in code.

```
MOVLW   d'9'        ; W = 9;
MOVWF   PORTA       ; PORTA = W
MOVF    PORTA,W     ; W = PORTA
MOVFW   PORTA       ; W = PORTA
```

Clearing registers

```
CLRF     f            ; Z
CLRW                  ; Z
```

These instructions set either a file register or the working register to zero.

```
    CLRF    PORTA       ; PORTA = 0;
    CLRW                ; W = 0;
```

Rotating Registers

```
RLF     f, d            ; C
RRF     f, d            ; C
```

RLF and RRF are "rotate left" and "rotate right" shift operations. These are arithmetic shift operations, so the new bit being shifted into the file register will always be zero. The STATUS flag C will contain the value of the bit which is shifted out.

```
    RLF     PORTA, W    ; W = PORTA << 1
    RRF     PORTA, F    ; PORTA = PORTA >> 1
```

Nibble Swapping

```
SWAPF   f, d
```

This instruction swaps the two nibbles of the file register, i.e. the top four bits are swapped with the bottom four bits.

```
    MOVLW   0x34        ; W = 0x34
    MOVWF   0x20        ; File reg at address 0x20 = W ( = 0x34)
    SWAPF   0x20        ; Reg@0x20 is now 0x43
```

Arithmetic

Addition

```
ADDWF   f, d            ; C, DC, Z
ADDLW   k               ; C, DC, Z
```

These instructions carry out addition, either W plus a file register, or W plus a constant. If the result is zero, the Z STATUS bit will be set. If the result overflows (>255) the C carry bit will be set. The digit/decimal carry bit requires a bit more explanation. This bit is set if there is an overflow from the lower four bits of the register:

```
    MOVLW   0x0F        ; W = 00001111 = 0x0F
    ADDLW   0x01        ; W = 00010000 = 0x10, DC will be set
```

You should see that the value has overflowed from one hex digit to the next.

Subtraction

```
SUBWF   f, d            ; C, DC, Z
SUBLW   k               ; C, DC, Z
```

These instructions carry out subtraction, either a file register minus W (d = f – W), or a constant minus W (d = k - W). Do not make the mistake of thinking this is W-x.

If the result is zero, Z will be set. If the result is negative, i.e. W was bigger than f or k, then the carry bit (in subtraction this would more correctly be called the borrow bit) will be **clear**. If the result is zero or positive then the carry bit will be set.

A similar result follows for the DC bit. If the subtraction of the two lower four bits is negative, DC will be clear. If zero or positive the DC bit will be set.

```
    MOVLW   0x20        ; W = 0x20
```

```
    SUBLW   0x21        ; W = 0x21 - 0x20 = 0x01, DC set, C set
```

Increment/Decrement

```
INCF    f, d            ; Z
DECF    f, d            ; Z
```

Increment (+1) and decrement (-1) instructions. Incrementing from 255 will give a result of 0 and cause the zero bit. Decrementing a value of 0 will give a result of 255.

Logical Operations

Logical And

```
ANDWF   f, d            ; Z
ANDLW   k               ; Z
```

```
    MOVLW   0xa6        ; W = 0xa6 = b'10100110'
    ANDLW   0x90        ; W = b'10100110' & b'10010000'
                        ;   = b'10000000' = 0x80
```

Logical Inclusive Or

```
IORWF   f, d            ; Z
IORLW   k               ; Z
```

```
    MOVLW   0xa6        ; W = 0xa6 = b'10100110'
    IORLW   0x90        ; W = b'10100110' | b'10010000'
                        ;   = b'10110110' = 0xb6
```

Logical Exclusive Or

```
XORWF   f, d            ; Z
XORLW   k               ; Z
```

```
    MOVLW   0xa6        ; W = 0xa6 = b'10100110'
    XORLW   0x90        ; W = b'10100110' ^ b'10010000'
                        ;   = b'00110110' = 0x36
```

Logical Complement (Not)

```
COMF    f, d            ; Z
```

Replaces 0->1, 1->0 in a file register.

```
    MOVLW   0xa6        ; W = 0xa6 = b'10100110'
    MOVWF   0x20        ; File register at address 0x20 = W
    COMF    0x20        ; Reg@0x20 = b'01011001' = 0x59
```

Bit manipulation

```
BCF     f, b
BSF     f, b
```

Bit clear or bit set on individual bits of a file register.

```
    BCF     PORTA, 7    ; PORTA bit 7 = 0 = 0xxxxxxx
    BSF     PORTA, 6    ; PORTA bit 6 = 1 = 01xxxxxx
```

Subroutines and Jumping

```
CALL     L
RETLW    k
RETURN
```

These instructions are used to implement subroutines/functions. `CALL` jumps to a new address in the program as defined by a label, and stores the address of the next instruction on stack memory. `RETURN` retrieves the most recent address from stack memory, jumps back to that location and the program resumes from there.

`RETLW` does the same job as `RETURN`, but allows you to place a literal value into W at the same time. Useful for implementing lookup tables or similar tasks.

```
GOTO     L
```

Goto makes an unconditional jump to a new location in the program defined by a label.

Conditional jump operators

```
INCFSZ   f, d
DECFSZ   f, d
```

Increment or decrement a file register, and if the result is zero skip the next line.

```
    MOVLW   0x03        ; For loop example
    MOVWF   0x20        ; Store loop variable in GPR 0x20.
loop:
    NOP                 ; Your loop code would go here
    DECFSZ  0x20        ; If the loop var is 0, skip the goto
    GOTO    loop
```

```
BTFSC    f, b
BTFSS    f, b
```

These instructions look at the value of a bit in a register, i.e. a "bit test on file register" and then skip the next instruction if the bit being checked was either clear (0) or set (1). Please be careful with these instructions, it is easy to get them the wrong way round.

Other

```
NOP
```

Do nothing for one instruction cycle.

```
SLEEP                   ; TO, PD
```

Put the microcontroller in a low power state. It can be woken up again using one of a few different ways – see the datasheet.

```
RETFIE
```

Return from interrupt. This must be called as the final instruction of your interrupt service routine.

```
CLRWDT                  ; TO, PD
```

Clear watchdog timer. See datasheet or notes on the watchdog.