## I. REGISTROS EN LOS MICROCONTROLADORES PIC

Su función es almacenar los valores de datos, comandos, instrucciones o estados binarios que ordenan qué dato debe procesarse, como la forma en la que se debe hacer.

### 1.1 Memoria de datos:

Es la memoria que almacena los datos que se almacenan en un programa, los cuales varían continuamente, se trata de una memoria volátil RAM. Se divide en

**1.1.1 Registro de funciones especiales (SFR):** son los primeros registros, cada uno de ellos cumple un propósito especial en el control del microcontrolador.

**1.1.2. Registro de propósito general:** son registros de uso general que se pueden usar para guardar los datos temporales del programa que este ejecutándose, el primer registro de uso general es el (0.20h)

### 1.2 Registro de estado o status.

Ocupa la posición 03h del banco1 (0-83h) y es uno de los registros mas importantes y utilizados. Los bits de este registro indican el estado de la última operación aritmética o lógica realizada, la causa del reset y los bits de selección de banco para la memoria de datos. A los bist del registro de estado se les suele denominar flags o banderas.

| IRP | RP1 | RP0 | /TO | /PD | Z | DC | C |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

**IRP** : Registro de selección de Banco (usado para direccionamiento indirecto)

- **1** - Bancos 0 y 1 son activos (localidades de memoria 00h-FFh)
- **0** - Bancos 2 y 3 son activos (localidades de memoria 100h-1FFh)

**RP1, RP0** : Registro de selección de banco (usado para direccionamiento directo).

| RP1 | RP0 | BANCO ACTIVO |
|---|---|---|
| 0 | 0 | Banco 0 |
| 0 | 1 | Banco 1 |
| 1 | 0 | Banco 2 |
| 1 | 1 | Banco 3 |

**TO:** Time-out bit (bit de salida del temporizador perro guardián)

- **1** - Después de encender el microcontrolador, después de ejecutarse la instrucción **CLRWDT** que reinicia al WDT (temporizador perro guardián) o después de ejecutarse la instrucción SLEEP que pone al microcontrolador en el modo de bajo consumo.
- **0** - Después de acabarse el tiempo del WDT.

**PD:** Power-down bit (bit de apagado)

- **1**: Después de encender el microcontrolador, después de ejecutar la instrucción **CLRWDT** que reinicia al WDT.
- **0**: Después de ejecutarse la instrucción **SLEEP** que pone al microcontrolador en el modo de bajo consumo.

**Z:** Zero bit (bit cero)

- **1** : El resultado de una operación lógica o aritmética es 0.
- **0** : El resultado de una operación lógica o aritmética es distinto de 0.

**DC :** Digit carry/borrow **bit** (bit de acarreo/préstamo de dígito) cambia al sumar o al restar si ocurre un "desbordamiento" o un "préstamo" en el resultado.

- **1** : Hubo acarreo del cuarto bit de orden bajo (nibble bajo) en el resultado.
- **0** : No hubo acarreo del cuarto bit de orden bajo (nibble bajo) en el resultado.

**C :** Carry/Borrow bit (bit de acarreo/préstamo) cambia al sumar o al restar si ocurre un "desbordamiento" o un "préstamo" en el resultado, o sea si el resultado es mayor de 255 o menor de 0.

- **1** - Ocurrió acarreo en el bit más significativo (MSB) del resultado.
- **0** - No ocurrió acarreo en el bit más significativo (MSB) del resultado.

### 1.3 Registro de trabajo W (work)

Es el registro principal y participa en la mayoría de las instrucciones, se localiza dentro de la CPU del microcontrolador.

### 1.4 Registro relacionado con los puertos.

**PORTA**: el puerto A de entra/salida puede leerse o escribirse como si se tratace de un registro.

**TRISA:** es el registro de control para el puerto A, configura los pines como entrada o salida digital “0” corresponde a salida y “1” corresponde a entrada.

### 1.5 Conjunto de instrucciones de ensamblador.

Los microcontroladores PIC 16Fxx ejecutan solo 35 instrucciones diferentes cada instrucción tiene una longitud de palabra de 14 bits.

**1.5.1 Lenguaje ensamblador:** se aplica lenguaje de bajo nivel, es decir más cerca de la arquitectura interna del microcontrolador.

Como lenguaje de bajo nivel, las aplicaciones son muy limitadas (leer la hoja de datos), se tiene que hacer cosas muy bajas a partir de cosas básicas.

**1.5.2 Las instrucciones se agrupan en:**

- **Registros de operaciones orientados al manejo de archivos ( Byte)**

  ADDW f, d (agrega la dirección “d” en el registro W)
- **Registro de operaciones orientadas a bits.**

  BCF f,d ( poner a cero el bit del archivo)
- **Control de operaciones y literal.**

  ADDLW k (agrega un valor literal “k” al registro W)

La palabra *Litera*l significa (NÚMERO) como el número 9 o 16h. El número 16h es un número hexadecimal y en valores decimales esto representa el número (22).

TABLE 7-2: JUEGO DE INSTRUCCIONES DEL PIC16CXXX

| Mnemonic, Operands | | Description | Cycles | 14-Bit Opcode MSb | LSb | Status Affected | Notes |
|---|---|---|---|---|---|---|---|
| **BYTE-ORIENTED FILE REGISTER OPERATIONS** | | | | | | | |
| ADDWF | f, d | Add W and f | 1 | 00 0111 dfff ffff | | C,DC,Z | 1,2 |
| ANDWF | f, d | AND W with f | 1 | 00 0101 dfff ffff | | Z | 1,2 |
| CLRF | f | Clear f | 1 | 00 0001 1fff ffff | | Z | 2 |
| CLRW | - | Clear W | 1 | 00 0001 0xxx xxxx | | Z | |
| COMF | f, d | Complement f | 1 | 00 1001 dfff ffff | | Z | 1,2 |
| DECF | f, d | Decrement f | 1 | 00 0011 dfff ffff | | Z | 1,2 |
| DECFSZ | f, d | Decrement f, Skip if 0 | 1 (2) | 00 1011 dfff ffff | | | 1,2,3 |
| INCF | f, d | Increment f | 1 | 00 1010 dfff ffff | | Z | 1,2 |
| INCFSZ | f, d | Increment f, Skip if 0 | 1 (2) | 00 1111 dfff ffff | | | 1,2,3 |
| IORWF | f, d | Inclusive OR W with f | 1 | 00 0100 dfff ffff | | Z | 1,2 |
| MOVF | f, d | Move f | 1 | 00 1000 dfff ffff | | Z | 1,2 |
| MOVWF | f | Move W to f | 1 | 00 0000 1fff ffff | | | |
| NOP | - | No Operation | 1 | 00 0000 0xx0 0000 | | | |
| RLF | f, d | Rotate Left f through Carry | 1 | 00 1101 dfff ffff | | C | 1,2 |
| RRF | f, d | Rotate Right f through Carry | 1 | 00 1100 dfff ffff | | C | 1,2 |
| SUBWF | f, d | Subtract W from f | 1 | 00 0010 dfff ffff | | C,DC,Z | 1,2 |
| SWAPF | f, d | Swap nibbles in f | 1 | 00 1110 dfff ffff | | | 1,2 |
| XORWF | f, d | Exclusive OR W with f | 1 | 00 0110 dfff ffff | | Z | 1,2 |
| **BIT-ORIENTED FILE REGISTER OPERATIONS** | | | | | | | |
| BCF | f, b | Bit Clear f | 1 | 01 00bb bfff ffff | | | 1,2 |
| BSF | f, b | Bit Set f | 1 | 01 01bb bfff ffff | | | 1,2 |
| BTFSC | f, b | Bit Test f, Skip if Clear | 1 (2) | 01 10bb bfff ffff | | | 3 |
| BTFSS | f, b | Bit Test f, Skip if Set | 1 (2) | 01 11bb bfff ffff | | | 3 |
| **LITERAL AND CONTROL OPERATIONS** | | | | | | | |
| ADDLW | k | Add literal and W | 1 | 11 111x kkkk kkkk | | C,DC,Z | |
| ANDLW | k | AND literal with W | 1 | 11 1001 kkkk kkkk | | Z | |
| CALL | k | Call subroutine | 2 | 10 0kkk kkkk kkkk | | | |
| CLRWDT | - | Clear Watchdog Timer | 1 | 00 0000 0110 0100 | | $\overline{TO}$,$\overline{PD}$ | |
| GOTO | k | Go to address | 2 | 10 1kkk kkkk kkkk | | | |
| IORLW | k | Inclusive OR literal with W | 1 | 11 1000 kkkk kkkk | | Z | |
| MOVLW | k | Move literal to W | 1 | 11 00xx kkkk kkkk | | | |
| RETFIE | - | Return from interrupt | 2 | 00 0000 0000 1001 | | | |
| RETLW | k | Return with literal in W | 2 | 11 01xx kkkk kkkk | | | |
| RETURN | - | Return from Subroutine | 2 | 00 0000 0000 1000 | | | |
| SLEEP | - | Go into standby mode | 1 | 00 0000 0110 0011 | | $\overline{TO}$,$\overline{PD}$ | |
| SUBLW | k | Subtract W from literal | 1 | 11 110x kkkk kkkk | | C,DC,Z | |
| XORLW | k | Exclusive OR literal with W | 1 | 11 1010 kkkk kkkk | | Z | |

**Note 1:** When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

**2:** If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

**3:** If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

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

### 1.6 Los mnemotécnicos.

son instrucciones que se escriben en el programa.

La “f” representa una dirección

La “d” es el destino del resultado:

0: se guarda en el registro W

1: se guarda en el mismo registro

“b” es una constante y se usa para referirse a un bit dentro de un byte.

"K" representa una constante.

Cuando una instrucción termina con **W** o **f**, el destino del resultado será el registro W o el propio archivo f, se define con el designador '0' o '1' de la propia instrucción.

Ejemplo:

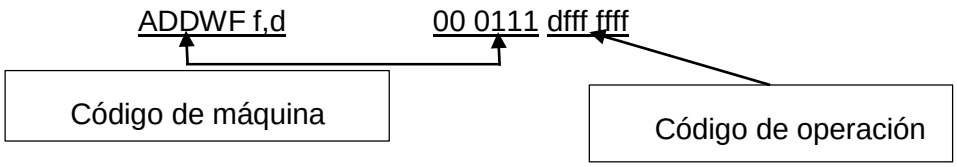
**ADDWF 1F,0** /(el resultado es almacenado en el registro de trabajo **W**.

**ADDWF 1F,1** / (el resultado es almacenado en el mismo registro (**F**).

### 1.6.1 Descripción de la palabra de 14 bits.

| Mnemonic, Operands | | Description | Cycles | 14-Bit Opcode MSb LSb | Status Affected | Notes |
|---|---|---|---|---|---|---|
| BYTE-ORIENTED FILE REGISTER OPERATIONS | | | | | | |
| ADDWF | f, d | Add W and f | 1 | 00 0111 dfff ffff | C,DC,Z | 1,2 |
| ANDWF | f, d | AND W with f | 1 | 00 0101 dfff ffff | Z | 1,2 |
| CLRF | f | Clear f | 1 | 00 0001 1fff ffff | Z | 2 |

En la primera columna de la tabla se encuentran los mnemónicos de las instrucciones, las cuales son palabras que facilitan recordar las instrucciones. La palabra se encuentra la columna cuatro.




Ejemplo: interpretar el siguiente cogido:

**00 0111 1000 1000**

00 0111 : instrucción de máquina (ADDWF)

1 : lugar en donde se guarda el resultado (0 en W y si es 1 en F)

**000 1000 :** dirección accesada ( Fh = 15)

*Interpretación: sumar el contenido del registro W al registro F y el resultado guardarlo en F.*

**00 0111 1000 1000= ADDWF Fh,1**

*Código en ensamblador*

### 1.7 Descripciones de las instrucciones

Las siguientes instrucciones son generales para todos los PIC's, además de las particularidades de cada instrucción, por orden alfabético que, presentan para cada micro tanto para el PIC16F84A como para el PIC12C508A

**ADDLW**: Agregar (sumar) el Literal al registro W (acumulador o registro de trabajo) resultado en W.

**ADDLW 00 al FF** Un número fijo (llamado literal) es sumado al registro W (registro de trabajo). El literal (número) puede estar comprendido entre el 00 y FF. En el registro STATUS se ven afectadas tres banderas (o flags) por la orden **ADDLW** (**Z**, **DC** y **C**), ver debajo.

- **C** Se pone a 1 si se produce un acarreo desde el bit de mayor peso (desbordamiento).
- **DC** Se pone a 1 si se genera un acarreo del bit 3 al bit 4.
- **Z** Se pone a 1 si el resultado de la operación es cero.

**ADDWF:** Suma aritmética del contenido del registro W con el contenido de un archivo (f).

**ADDWF 00 a 1F, 0**; El resultado es almacenado en el registro de trabajo W, debido al valor 0 en la instrucción.

**ADDWF 00 a 1F, 1** El resultado es almacenado en el mismo archivo, debido al valor 1 del designador en la instrucción.

Con la orden ADDWF, en el registro STATUS se ven afectados los bits: **C** (**C**arry), **Z** (Cero) y el **DC** (**D**ígito **C**arry).

Si el resultado de una instrucción ADD rebasa FF, la bandera **C** (Carry) es puesta a 1, si tiene cualquier otro valor es 0.

Si el resultado de una instrucción ADD es cero 0000 0000, la bandera **Z** (Cero) se pone a 1 y 0 si tiene cualquier otro valor.

Por ejemplo: Si agregamos 21h a 3Ch, el resultado es 5Dh, esto no afecta la bandera Carry, por lo que la bandera DC (dígito carry) será puesta a 1, pero si a 2Dh le agregamos 3Eh, el resultado es 6Bh, lo que desborda el contador (6B>FF) por lo que la bandera C (Carry) será puesta a 1.

**ANDLW:** Esto significa: producto lógico AND del Literal con el registro W.

**ANDLW 00 a FF**; El objetivo de la operación es, descubrir cuantos bits de L y W, en binarios están a 1. Si ambos bits son cero, el resultado es cero y la instrucción usada en este caso es XOR. Esta instrucción hace un AND lógico entre un número fijo (literal) y el contenido del registro W, el resultado es colocado en el registro W. Con

la orden ANDLW, en el registro STATUS se ven afectados los bits: C (Carry), Z (Cero) y el DC (Dígito Carry).

La operación AND puede decirse que se usa para enmascarar (separar o quitar) los bits que nos interesen. Por ejemplo:

**0F**h enmascarará (quita) los cuatro bits altos (nibble alto) y **F0**h quitará los cuatro bits bajos (nibble bajo). Veamos como se usa:

**Utilización del 0Fh: ANDLW 0x0F**

**Primer número**: 1001 0011 [Número en **W** anterior a la instrucción ANDLW] **Segundo número** (**0F**): 0000 1111 [Número **L** Máscara].

**Respuesta**

**AND**: 0000 0011 [Al aplicar la función **AND** entre ambos] (ANDed)

**Utilización del F0h: ANDLW 0xF0**

Primer número: 1001 0011 [Número en **W** anterior a la instrucción ANDLW] Segundo número (**F0**): 1111 0000 [Número **L** Máscara].**Respuesta**

**AND**: 1001 0000 [Al aplicar la función **AND** entre ambos]

**ANDWF** Esto significa: Operación AND producto lógico de **W** con el archivo **f**

**ANDWF 00 a 1F, 0** El resultado se almacena en el registro W, por el valor 0 en la instrucción.

**ANDWF 00 a 1F, 1** El resultado se almacena en el archivo f, por el valor en la instrucción.

Al registro W se le aplica el producto AND con un archivo f. Como dos archivos juntos no se pueden operar (AND), un archivo debe ponerse en W y se hace AND con el segundo archivo. Con la instrucción **ANDWF**, sólo se afecta la bandera **Z** (cero). Si la respuesta es cero, la bandera **Z** en el registro **STATUS** se pone a 1, si la respuesta es distinta de cero, la bandera se pone a 0.

**BCF:** Bit Clear File (poner a (0) o aclara el bit indicado (detrás de la coma) en el archivo **f** ).

**BCF 00 a 1F, bit** / se tiene 300 instrucciones para esta orden. Hay 79 archivos en el PIC16F84A, los 13 primeros archivos se llaman Registros de Función Especial (**SFR**'s), el resto (66) se llaman Archivos de Propósito General (**GPR**'s) del 0Ch a 4Fh. No afecta los bits de STATUS.

**BCF** se usa para limpiar (clear) un bit (pone a 0 el bit identificado en el archivo **f**).

**Ejemplo: BCF 06h,5**

Significa que el **bit 5** del archivo **06** debe ser puesto a (0) (aclarado), el resto de bits no se influyen. El archivo 6 contiene líneas de E/S comúnmente se llaman I/O del puerto.

**BSF**: Esto significa: Bit Set File (poner a 1 el bit indicado, en el archivo **f**).

**BSF 00 a 1F, bit** Hay casi 300 instrucciones para esta orden. Hay 79 archivos en el PIC16F84A, de las que los primeros 13 son los **SFR**'s y los siguientes 66 son los conocidos **GPR**'s.

**BSF** significa poner a 1 lógico el bit especifico en el archivo **f**. No afecta los bits de STATUS.

**Ejemplo: BSF 06h, 5**

Indica que el **bit 5** del archivo **6** será puesto a 1, El resultado es la inversa a la instrucción anterior, el 0 se sustituye por un 1.

**BTFSC** : Esto significa: Bit Test, Skip if Clear (Bit de Test, Salta si es «0»).

BTFSC 00 a 1F, bit / **BTFSC** significa, comprobar el bit identificado en el registro llamado y si es 0 saltar una instrucción (no ejecuta la instrucción que sigue). No afecta los bits de STATUS.

ejemplo: **BTFSC 06h,4**

Es la forma de comprobar si el bit 4 en el archivo 6 es «0», si es cero, salta la próxima instrucción (pasar sin ejecutar) y continuar con la posterior.

```
BTFSC 06h,4 // comprueba si el bit 4 es 0.
GOTO Etiqueta2 // si no es 0, salta hasta Etiqueta2.
CALL Dlay// si es 0, llama a subrutina Dlay.
```

**BTFSS:** Esto significa: Bit Test, Skip if Set (Bit de Test, Salta si es «1»).

BTFSS 00 a 1F, bit / BTFSS significa, comprobar el bit identificado en el registro llamado y salta la próxima instrucción si es 1. No afecta los bits de STATUS.

En **BTFSS 3h, 2** comprobamos el bit 2 del registro 3h y si dicho bit es 1, salta la próxima instrucción, si no, continua con la siguiente

**Ejemplo.**

**BTFSS 03h, 2** // comprueba si el bit 2 es 1.

**GOTO Etiqueta2** // si no, va a Etiqueta2.

**CALL Dlay** //si es 1, llama a subrutina Dlay para seguir. ; si es 1 viene a esta instrucción y sigue.

**CALL**: Esto significa: Llamada incondicional a subrutina.

En un programa, esto se escribe como: CALL ***Salida o CALL Tono1, etc.*** donde ***Salida o Tono1*** son etiquetas.

Un **RETURN** debería encontrarse al final de la subrutina CALL para que el micro vuelva a la siguiente instrucción después de la instrucción CALL que la llamó, de lo contrario se producirá un desborde de pila, con el consiguiente bloqueo del programa. No afecta los bits de STATUS.

Cada vez que se ejecuta una instrucción **CALL**, un valor de dirección es colocado (empujado) en la Pila (**Stack**), entonces el micro sabe donde volver después de ejecutada la subrutina, la Pila sólo puede tener 8 niveles de alto, entonces es necesario llevar cuidado para no quedarse sin Pila (**Stack**).

*Loop2: BTFSS 05,2 // ¿esta apretado el pulsador?*
*GOTO Loops2; No, salta a Loop2*
*MOVLW 01 /si*
*XORWF 06,1 // encender el LED*
*CALL Delay; llama a la rutina de retardo*
*GOTO Loop1; para ver Led encendido*
*Delay: DECFSZ 1Bh,1; subrutina anidada de retardo*
*Goto Delay // retardo exterior*
*DECFSZ 1Ch,1 //retardo interior*
*RETURN*

**CLRF:** Clear f (Limpia **f**) (poner a 0 los 8 bits del archivo **f**)

**CLRF 00 a 1F** El contenido de un archivo se pone a 0 (Clear) y el bit **Z** del registro **STATUS** es puesto a 1, esto es, los ocho bits se ponen a «0».

**CLRW:** clear **W** (limpiar el registro de trabajo – llamado acumulador en otros micros)

**CLRW** El registro W (de trabajo) es aclarado, todos los bits son puestos a 0. Cuando el **W** es puesto a 0, la bandera cero (flag **Z**) es puesta a 1, en otras palabras la bandera **Z** del registro **STATUS** es puesta a 1.

**CLRWDT**: aclarado (puesta a 0) del Temporizador Perro Guardián (El bit **WDT** = 0).

**CLRWDT/** La instrucción repone (resetea) el Temporizador Perro Guardián, esto también repone el preescaler del **WDT** y consume 2 ciclos máquina.

**COMF**: esto significa: Complemento del archivo f.

**COMF 00 a 1F, 0** El resultado estará en **W** por el valor 0 detrás de la coma.

**COMF 00 a 1F, 1** El resultado estará en **f**.
El contenido **f** es complementado (los 0's se cambian a 1's y los 1's a 0's).

**DECF:** Esto significa: Decremento del archivo f.

**DECF 00 a 1F, 0** El resultado estará en **W**. El contenido del archivo **f** es decrementado y puesto (**W**)

**DECF 00 a 1F, 1** Aquí, el resultado estará en **f**.
El contenido del archivo (**f)** es decrementado, significa que es deducido (tomado) 1 del archivo. Si el archivo es 0000 0000 esto da la vuelta a 1111 1111 (255) afectando a la bandera **Z**. Cuando el archivo es 0000 0001 y se ejecuta una instrucción DECF, el archivo pasa a 0000 0000 y la bandera **Z** (cero) del **STATUS** se pone a 1, en otro caso es 0.

**DECFSZ**: decrement f, Skip if Zero (Decrementa el archivo **f** y salta si es 0).

**DECFSZ 00 a 1F,0** El resultado estará en **W**.
**DECFSZ 00 a 1F, 1** El resultado estará en **f**.

Un empleo importante está en una subrutina de retardo. En esta rutina la instrucción **DECFSZ** crea un lazo en el que el micro es enviado a una instrucción por encima-del-programa y se ejecutará un juego de instrucciones una y otra vez, esta es una táctica de pérdida de tiempo para crear un retardo. No afecta al registro STATUS.

Cada vez que el micro llega a **DECFSZ**, el contenido del archivo es decrementado y si el archivo no es cero, se ejecutará la siguiente instrucción en el programa. La siguiente instrucción es normalmente GOTO y ésta envía de nuevo al micro por encima-del-programa.

ejemplo

```
ret:    DECFSZ Ch,0 // Decrementa Ch y si es 0, salta una línea.
GOTO ret            ; no es 0, ejecuta esta línea.
…                   ; si es 0, viene hasta aquí.
…                   ; sigue programa.
```

**GOTO**: ordena que la ejecución del programa continue en una línea en particular

**GOTO k** GOTO es la bifurcación incondicional en la que el micro es enviado a la dirección especificada. También se usa **$-n** o **$+n** donde n es el número de líneas que ha de, volver atrás o avanzar en el programa.

Ejemplo.

```
ret DECFSZ 0Ch,0 ; decrementa 0Ch, si es 0 salta 1 instrucción.

GOTO $-1         ; No, vuelve 1 línea atrás. No requiere etiqueta.

…                ; Si, sigue programa
```

**INCF**: Esto significa: Incrementar el archivo **f**.

**INCF 00 a 1F,0** El resultado del incremento estará en **W**.
**INCF 00 a 1F,1** El resultado estará en **f**.
El contenido del archivo '**f**' es incrementado, esto simplemente significa que se agrega 1 al archivo, si el archivo es 1111 1111 (255) esto da la vuelta a 0000 0000. Cuando el archivo es FFh y se ejecuta la instrucción **INCF**, el archivo pasa a 0000 0000 y la bandera **Z** (cero) es puesta a 1 en otro caso es 0.

**INCFSZ**: Esto significa: increment f and Skip if 0 (Incrementar el archivo f y salta si es 0).

**INCFSZ 00 a 1F,0** El resultado estará en **W**.
**INCFSZ 00 a 1F,1** El resultado estará en **f**.
Normalmente la función de decremento DECFSZ se usa para crear un retardo, pero también se puede usar un INCFSZ. No afecta al registro STATUS.

Esto trabaja así: Si el contenido de un archivo es incrementado y el resultado no es 0, entonces la siguiente instrucción es ejecutada con un GOTO una dirección anterior y ejecutará otro INCFSZ. Eventualmente el archivo será 1111 1111 y el próximo incremento lo devolverá a 0000 0000, el resultado será cero y la instrucción GOTO será ignorada, ejecutándose la siguiente instrucción.

**IORLW**: Esto significa: Inclusive **OR** Literal con W.

**IORLW 00 a FF/** El contenido del archivo W es sumado (lógico) con un número. El resultado es colocado en registro de trabajo W, el número literal puede ser desde 00 a FF. Afecta al bit **Z** del registro **STATUS**.

Esto es simplemente una operación **OR** (suma lógica) y el objeto de su realización es cambiar dos o más bits a (1), si un bit es ORed con 0, la respuesta no se altera, si el bit es ORed con 1, el resultado es 1.

Ejemplo: Si el registro W
se carga con 1111 0000 (es una máscara de 4 bits altos F0h) y
un número como 0100 1110 (4Eh) es **OR**ed con W,
el resultado es 1111 1110 (FEh).

**IORWF:** Esto significa: Inclusive **OR** con el archivo f

**IORWF 00 a 1F,0** El resultado estará en **W**.
**IORWF 00 a 1F,1** El resultado estará en **f**.

El contenido del registro **W** es ORed con el archivo **f**, esto simplemente es una operación «OR» y el objeto de su realización es cambiar dos o más bits a «1». Si un bit es ORed con 0, la respuesta no se altera, si el bit es ORed con un 1 el resultado es 1. Afecta al bit **Z** del registro STATUS.

Ejemplo: Si el registro W es cargado con 1111 0000 (F0h es una máscara de 4 bits altos) y un archivo con un número como 0100 1110 (4Rh) es ORed con W, el resultado es 1111 1110 (FEh).

**MOVF:** Esto significa: Mueve el contenido del archivo 00 a 1F dentro y fuera del archivo 0 al W.

**MOVF 00 a 1F,0** El contenido del archivo es movido al W. El resultado estará en **W**.
**MOVF 00 a 1F,1** El resultado estará en **f**.
Para esta instrucción MOVF 00 a 1F, 1 el contenido es movido fuera del archivo y devuelto a él otra vez, pero no entra en W. Esto es una prueba útil ya que la bandera **Z** (cero) del STATUS se ve afectada. Si el contenido es cero, la bandera **Z** es puesta a 1, si el contenido es 1, la bandera **Z** es 0.

**MOVF f,W** donde f es un registro entre 00 y FF. También puede usarse **MOVF f,0** que viene a ser lo mismo.

**MOVLW:** Mueve Literal a W. (guarda una constante en el registro de trabajo)

**MOVLW 00 a FF /**Un número **f** (Literal) es cargado en el registro **W**. El Literal puede ser 00 a FF. No afecta al registro STATUS.

**MOVWF:** Copia **W** al archivo llamado **f**. (guarda el contenido del registro de trabajo en una localidad de memoria)

**MOVWF 00 a 1F** Esta instrucción copia datos del registro **W** al archivo **f**. No afecta al registro STATUS.

**NOP:** Ninguna operación. Es decir, el micro no realiza ninguna operación, sólo consume el tiempo de 1 instrucción.

**OPTION**: Carga registro **OPTION**. El contenido del registro **W** es cargado en el registro **OPTION**.

**RETFIE** : Cuando hay una interrupción, RETURN con valor de lo alto de la Pila y lo deja en el PC.

**RETFIE** Carga el **PC** con el valor que se encuentra en la parte superior de la pila, asegurando así la vuelta de la interrupción. Pone a 1 el bit **GIE**, con el fin de autorizar o habilitar de nuevo que se tengan en cuenta las interrupciones. **TOS** → **PC**, 1 → **GIE**. No afecta al registro STATUS

**RETLW:** Esto significa: **RETURN** con Literal en **W**.

**RETLW 00 a FF** El registro **W** es cargado con el valor del literal, normalmente devuelve un dato procedente de una tabla. El Contador de Programa (PC) es cargado de la cima de la pila (Stack), que corresponde a la dirección de RETURN de programa. No afecta al registro STATUS.

**RETURN:** Retorno de Subrutina. Esta instrucción está al final de una rutina o subrutina. No afecta al registro STATUS.

**RLF:** Esto significa: Rotar el archivo **f** a Izquierda con a**c**arreo (Carry).

**RLF 00 a 1F,0** El resultado se almacena en **W**.

**RLF 00 a 1F,1** Resultado se almacena en **f**.

El contenido de un archivo, es rotado un bit a la izquierda por la bandera Carry (acarreo), esto requiere de 9 desplazamientos para recuperar el valor original del archivo. Afecta al bit **C** del **STATUS**.

El uso de: *RLF…..Reg, Destino……..; rota los bits de un registro un lugar la izquierda.*

Si Reg = b'00010110′

Después de la instrucción: Reg = b'0010110C' donde C es el valor del bit STATUS, C en el momento de ejecutarse la instrucción RLF.

Un grupo de 8 bits es registro, o sea: Registro = B7 B6 B5 B4 B3 B2 B1 B0

Al aplicar la instrucción *RLF…..Reg, F* ocurre que:

(STATUS,C <== B7) <== B6 B5 B4 B3 B2 B1 B0 (C <== STATUS,C)

Esto significa que, todos los bits de Reg son rotados (desplazados) una posición hacia la izquierda. El espacio generado a la derecha de Reg es decir, el bit0 (B0) del registro, es ocupado por el valor que tenía en ese momento el bit C del registro STATUS. A su vez, el Bit7 (B7) de Reg sale del Registro y se rellena con el bit C del registro STATUS.

Reg = b'00001100′ (Ch) y STATUS,C = 1

Aplicamos; *RLF Reg,F* Entonces: Reg = b'00011001′ (19h = .25) y STATUS,C = 0

En este caso, antes de la aplicación de RLF Reg valía 12 en decimal y después de aplicar la instrucción Reg vale 25 en decimal, por qué ocurre este error si hemos aplicado la misma instrucción al mismo registro Reg. Debemos considerar el motivo.

El motivo radica en que el bit C del registro STATUS, antes de ejecutar la instrucción RLF, valía 1, en el segundo caso y ocupó el bit0 del Reg al ejecutar la instrucción RLF. Por tanto, en este segundo caso, al hacer RLF Reg,F equivale a hacer Reg * 2 + 1.

Precauciones a tener en cuenta para evitar incurrir en este error. Para asegurarnos en una multiplicación por dos, siempre limpiaremos el bit C del STATUS antes de aplicar la instrucción RLF, asegurándonos así que el bit STATUS,C no «corrompa» la operación.

**II: PROGRAMACIÓN EN ENSAMBLADOR**.

En código máquina se emplea el siguiente formato de constantes.

| TIPO | SINTAXIS | EJEMPLO |
|---|---|---|
| DECIMAL | D´<cantidad>´<br>d´<cantidad>´<br>.<cantidad> | D´15´<br>d´15´<br>.15 |
| HEXADECIMAL | H´<cantidad>´<br>h´<cantidad>´<br>0x<cantidad><br><cantidad> | H´F´<br>h´F´<br>0xF<br>Fh |
| BINARIO | B´<cantidad>´<br>b´<cantidad>´ | B´00001000´<br>b´00001000´ |

## 2.1 Instrucciones en lenguaje ensamblador.

**MOVLW k** (guarda el valor de una constante en el registro W)

| **MOVLW 2Ah** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Sea el registro **W y** K con los datos | | | | | | | | |
| W | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| K | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| Después | | | | | | | | |
| | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| W | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

**MOVWF f** (mueve el contenido del registro W al registro (f)

| **MOVLW PORTA (05h)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Sea el registro PORT A y **W** con los datos | | | | | | | | |
| PORTA | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| W | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Después | | | | | | | | |
| | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| PORTA | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| W | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

**DECF f, d** (decrementa el contenido de un archivo (f=f-1))

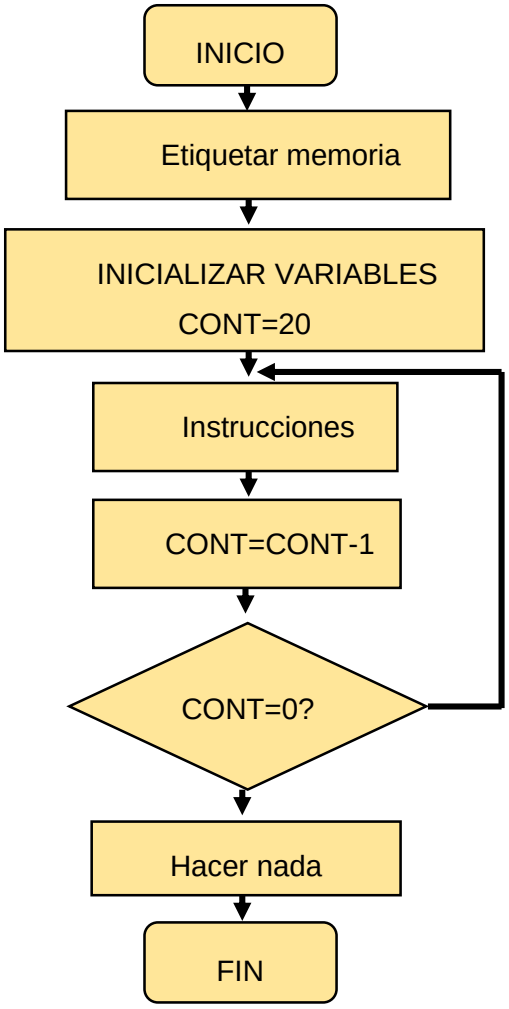
| **DECF PORTC,W** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Sea el registro PORT A y **W** con los datos | | | | | | | | |
| PORTC | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Después | | | | | | | | |
| | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| PORTC | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| W | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

**Ejercicio 1.**

Realizar un ciclo de 20 interacciones.

Solución.

Primero es importante realizar un diagrama de flujo.




```
PROCESSOR 16f887
INCLUDE P16F887.INC
CONT    EQU 20h         ; dirección de inicio de programa
        ORG 0h          ; dirección de inicio de memoria de programa
        MOVLW 14h       ; W=20
        MOVWF CONT      ; CONT=W=20
CICLO NOP               ; no hace nada (instrucciones)
        DECFSZ CONT, F  ; decrementa CONT y salta si es cero
        GOTO CICLO      ; ir a CICLO
        NOP
        END
```

```
//// encender un led
list p = 16f84A
include <p16f84A.inc>

        org 0 ; inicio en la posicion cero
        bsf STATUS, RP0; coloque en uno el BIT RPO=B =1
        CLRF PORTB ; puerto B colocado en cero( salida)
        bcf STATUS, RP0;regreso al banco cero
        BSF PORTB ,1; colocar en 1 el pin 1 del puerto B
END

/// encender y apagar un led.
list p = 16f84A
include <p16f84A.inc>
TIEMPO EQU 0x0C
org 0 ; inicio en la posicion cero
    bsf STATUS, RP0; coloque en uno el BIT RPO=B =1
    CLRF PORTB ; puerto B colocado en cro( salida)
    bcf STATUS, RP0;regreso al banco cero
INICIO   BCF PORTB,B'00000010'; loca en cero el pin 1 del puerto B
    CALL RETARDO
    BSF PORTB, B'00000010';coloca en 1 el pin 0 del puerto B
    CALL RETARDO
     GOTO INICIO
RETARDO
    MOVLW h'FF'; se asigna el valor de 255  W
    MOVFW TIEMPO; tiempo =255
    DEC    DECFSZ TIEMPO
    GOTO DEC
  RETURN
END
/// encender y apagar led con pulsador.

list P=PIC16F84A
#include<P16F84A.inc>
org 0
; configuracion de los puertos
  Bsf STATUS, RP0; vamos al banco 1
    Bsf TRISA, 1; enptrada
    Bcf TRISB,1; salida
```

```
      Bcf STATUS, RP0; regresamos al banco cero
; codigo
ENCENDIDO
      Btfss PORTA, 1; si se preciona el boton
      Goto ENCENDIDO
       Bsf PORTB, 1;
      Goto APAGADO
APAGADO
       Btfsc PORTA, 1; si hay un cero en el puerto a
       Goto APAGADO
       Bcf PORTB, 1
       Goto ENCENDIDO
END
```