J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

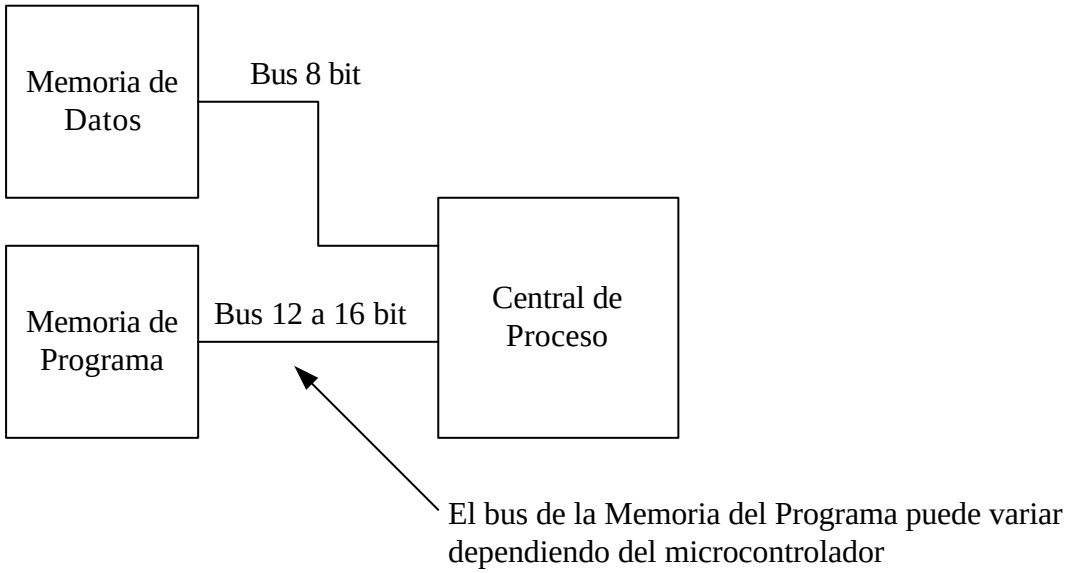
Microcontroladores PIC

Introducción

Los microcontroladores son procesadores que poseen memoria y dispositivo de entrada salida todo encapsulado en un mismo integrado, lo que permite su uso sin la necesidad de estar disponiendo de un bus para unir al microprocesador con memoria, PPI, etc.
Esto permite un uso relativamente sencillo y además, la construcción de una placa de baja complejidad y por lo tanto menor probabilidad de fallo.
Al estar todo integrado, un microcontrolador no se define solamente por su capacidad de direccionamiento o por su velocidad de proceso, es mas, en la mayoría de los microcontroladores no se dispone de direccionamiento externo, como así tampoco de ninguna señal de control de buses, simplemente tiene las salidas y entradas de sus dispositivos de E/S, junto con alguna línea de control como Reset y la entrada del oscilador.
Para definir un microcontrolador entonces, se tendrá en cuenta su memoria de programa interna, su memoria de dato interna, los dispositivos de E/S que posee, su velocidad de proceso y los dispositivos auxiliares como Timer programables o memoria no volátil por ejemplo.

Microcontroladores de Microchip

Los microcontroladores de Microchip, utiliza una estructura de buses tipo Harvard, esto significa que los buses de memoria y de programa internos se encuentran en forma separada, permitiendo por un lado aumentar la eficiencia del proceso del microcontrolador al recibir datos y comandos por dos vías separadas y por otro, hacer independiente la longitud de la palabra entre los dos buffer, esto permite tener por ejemplo, un bus de datos de 8 bits, y un bus de comandos que puede ir desde 12 bits hasta 16 bits según la familia.



Además posee arquitectura RISC, con una instrucción por ciclo y un set de instrucciones bastante reducido.

J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

En lo siguiente se tratará el uso de un microcontrolador en especial, el 16F84, unos de los mas pequeños que posee Microchip. Si bien es uno en particular, se puede extrapolar a los demás microcontroladores.
En primer lugar veamos como está dispuesta la memoria del microcontrolador.
Al ser una arquitectura Harvard, tenemos que pensar un poco diferente a lo que hacíamos con el 8088, en este caso, la memoria de programa no tiene NADA que ver con la memoria de variables, siendo todas las instrucciones de lectura/escritura de memoria dirigidas a la memoria de variables o de datos, no siendo accesible la memoria de programa para estas instrucciones.
Ejemplo: vamos a crear un programa que pone en cero una variable y luego la incrementa continuamente en un bucle infinito.

```
000F  018F          clrf   0xF
0010  0A8F  repetir incf   0xF
0011  2810          goto   repetir
```

En este caso, vemos que tenemos en la primer columna a la izquierda la dirección de memoria donde se encuentra el programa, (dirección 000Fh en hexadecimal). Luego tenemos en la segunda columna el código maquina de la instrucción con 2 bytes de longitud para representar los 14 bits de longitud de instrucción que posee este microcontrolador, ahora la dirección (primera columna) se incrementa de a una unidad por instrucción, esto significa, que cada instrucción (14 bits) ocupa una dirección de memoria. Lo que se encuentra a la derecha de la segunda columna representa el assembler de nuestro programa, tenemos las instrucciones con sus correspondientes parámetros y una etiqueta (repetir) para realizar el bucle.
La instrucción **clrf 0xF** borra una variable (una posición de memoria de datos). Esta variable será de 8 bits como estamos acostumbrados a usar, y en este caso estará almacenada en la posición 00Fh; el código maquina de la instrucción esta también guardado en una dirección de programa con el valor 000Fh, pero al ser memorias diferentes no existirá conflicto.
La instrucción **incf** incrementa el valor guardado en 00Fh y por último el **goto** que es la instrucción que realiza el salto incondicional al igual que el **jmp** en el 8088, realizará un salto a la etiqueta “repetir”.

Características

Las características principales de este microcontrolador son:

Memoria de programa	1Kb de memoria Flash
Memoria de datos	68 bytes
Memoria EEPROM	64 bytes
Oscilador	10 Mhz.
Salidas digitales de I/O	13 pines en dos puertos A de 8 bits y B de 5 bits .
Timer	programable de 8 bit con prescaler.
Velocidad	2,5 MIPS – 400 ns por instrucción *

*** El microcontrolador funciona a una velocidad de ¼ del oscilador de entrada y al ser RISC y procesar una instrucción por ciclo de reloj, podemos calcular fácilmente la cantidad de instrucciones que puede realizar por segundo (MIPS)**

Mapa

Los dispositivos de entrada salida y los auxiliares están mapeados dentro de la memoria de datos, como así también los registros del procesador.

J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

Por lo tanto, tendremos direcciones de memoria que en vez de estar referidas a una posición vacía para ser usada, estarán apuntando a algún registro del microcontrolador, es por eso que por ejemplo en el 16F84 tenemos las direcciónes 00h hasta la 0Bh y 80h hasta 8Bh de la memoria de datos ocupadas con registros que controlan los distintos periféricos del micro.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- ----	---- ----
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- ----	---- ----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

En esta tabla, extraída de la hoja de datos del 16F84, vemos los distintos registros del microcontrolador, con su nombre y su dirección de memoria.
Por ejemplo el registro 03h es el STATUS, allí se encuentran los flags del microcontrolador Carry, Zero, etc
Además de estos registros existe una zona de memoria para propósitos generales, que es la zona que se utiliza para el almacenamiento de nuestras variables. Incluyendo esta zona en el mapa la memoria queda estructurada de la siguiente forma:

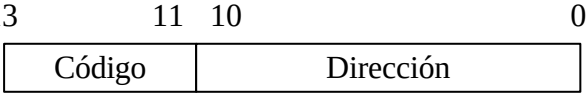
00h	**Dirección indir.**
01h	**TMR0**
02h	**PCL**
03h	**STATUS**
04h	**FSR**
05h	**PORTA**
06h	**PORTB**
07h	*Sin uso*
08h	**EEDATA**
09h	**EEADR**
0Ah	**PCLACH**
0Bh	**INTCON**
0Ch – 4Fh	68 bytes de memoria de propósitos generales

80h	**Dirección indir.**
81h	**OPTION**
82h	**PCL**
83h	**STATUS**
84h	**FSR**
85h	**TRISA**
86h	**TRISB**
87h	*Sin uso*
88h	**EECON1**
89h	**EECON2**
8Ah	**PCLACH**
8Bh	**INTCON**
8Ch – CFh	Imagen de la memoria 0Ch hasta 4Fh

Manejo de Memoria

Memoria de Programa

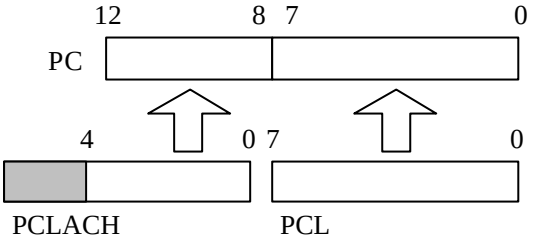
El principal inconveniente que poseen estos microcontroladores en lo que respecta a programación, es el echo de que al tener que encapsular instrucción mas parámetros en una cantidad fija de bits (14 bit para el 16F84) aparecen problemas en la longitud del parámetro que puede manejar.
Veamos un ejemplo:
La instrucción **goto** realiza un salto incondicional a una posición de memoria de programa, es necesario entonces que en los 14 bits encapsule la instrucción **goto** mas la dirección de salto.



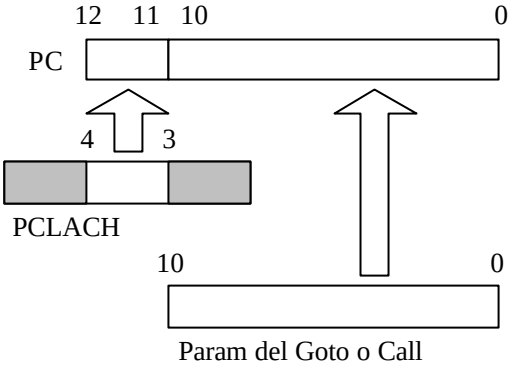
Vemos que el código de la instrucción es guardado en 3 bits, quedando 11 bits para guardar la dirección de salto, esto nos da un tamaño máximo de salto de $2^{11} = 2048 = 2Kb$, hay microcontroladores que poseen una memoria de programa mayor, a la que no se podría acceder. Esto fue solucionado utilizando dos bits de un registro especial (el bit 3 y 4 del PCLACH) en conjunto con los 11 bits disponibles para representar la dirección de salto, por lo tanto cuando el microcontrolador encuentra una instrucción **goto** salta a una posición de memoria de 13 bits formada con los dos bits del registro como bits mas significativos y los 11 bits restantes del parámetro de la instrucción.
Otra forma de realizar un salto es modificando directamente el contador de programa, que al igual que el resto de los periféricos esta mapeado en un registro de la memoria, entonces el contador estará formado por un registro (PCL) el cual contiene los 8 bits mas bajos de los

13bits del PC y para los 5 bits restantes tendremos el PCLACH usando para este caso los 5 bits menos significativos.
El caso particular de PCLACH es un registro LACH, esto significa que lo que se copia a PCLACH será guardado y no será bajado inmediatamente en el PC, si no que lo hará en el momento que se modifique el PCL o que se realice un salto **goto** o **call**., esta es la única forma de poder modificar el PC en una sola instrucción.

Actualización del PC modificando los registros de PCL y PCLACH



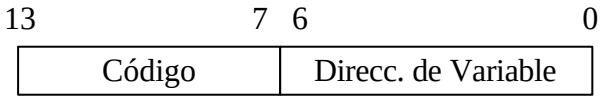
Actualización del PC mediante una instrucción GOTO o CALL



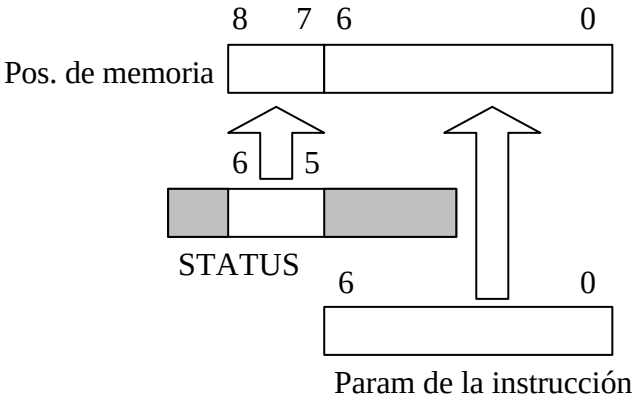
Para el caso particular de este microcontrolador que dispone de 1Kb de memoria de programa el uso del los bit 3 y 4 del PCLACH no son utilizados.

Memoria de Dato

Para el caso de la memoria de dato ocurre algo similar al caso de memoria de programa
Las instrucciones que poseen una dirección de variable como parámetro tienen la forma.



Para el caso entonces de la memoria de dato podemos acceder a las primeras 128 bytes de memoria. La mayoría de los microcontroladores superan esta cantidad entre registros y memorias de datos, es por eso que se dispone de mapas de memoria, los cuales pueden ser cambiados mediante los bits 6 y 5 del registro STATUS denominados RP0 y RP1. Para acceder entonces a una posición de memoria determinada, la formaremos de la siguiente forma:

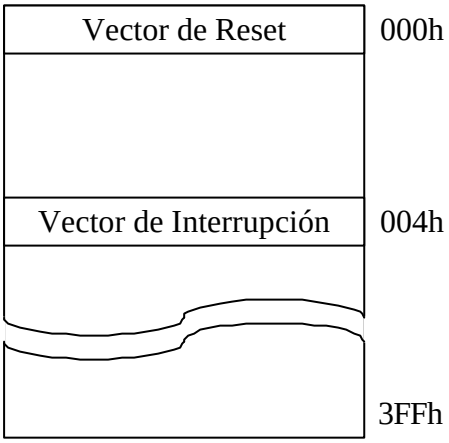


J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

Vectores

Poseen solo dos vectores el de comienzo de programa o vector de Reset y el de interrupción
El vector de comienzo, a diferencia de los vectores del 8088, cuando el microcontrolador arranque ejecutara directamente la instrucción grabada en esa posición, para el caso del vector de interrupción, el PC del microcontrolador saltará a esa posición de memoria por cada interrupción que se genere, por ejemplo por desborde del Timer, interrupción externa, interrupción de software, etc.

Memoria de Programa del 16F84



Manejo de Stack

El stack en estos microcontroladores es muy básico dispone de 8 posiciones de memoria de 13 bits.
No se permite el control del stack por parte del programador, simplemente guarda la posición donde está el PC tras una interrupción o un llamado a subrutina (**call**). Esto permite tener un máximo de 8 niveles de profundidad.
El problema de este stack tan primitivo, es que si deseamos guardar los valores de los registros importantes del micro para protegerlos de alguna modificación dentro de la rutina de interrupción o subrutina y luego recupéralos una vez que termine, deberemos implementarlo nosotros mediante código.
A continuación se muestra el código necesario para guardar el registro w y el STATUS en variables temporales y luego recuperarlos.

Salva los registros W y STATUS a variables temporales

```
1  movwf  W_Temp    ; copia el contenido del registro W a una variable
2  movf   STATUS,W  ; copia el contenido del reg. STATUS a W
3  clrf   STATUS    ; borra RP0 y RP1 para ir a banco 0
4  movwf  S_Temp    ; copia el STATUS guardado en W a una var.
```

Recupera el valor de W y STATUS de las variables temporales

```
5  movf   S_Temp,W  ; copia el contenido de la variable a W
6  movwf  STATUS    ; copia la variable guardada en W al reg STATUS
7  swapf  W_Temp    ; invierte los nibbles del contenido de la variable
8  swapf  W_Temp,W  ; vuelve a inv.los nibbles y guarda el resul. en w
```

Descripción: estas dos rutinas se ubicarán al comienzo y al final respectivamente de la rutina o interrupción que realicemos, las mismas son utilizadas principalmente en llamada a interrupciones, puesto que en este caso se hace necesario que cuando se vuelva al programa principal el estado del microcontrolador sea el mismo que tenia en el momento del pedido de interrupción, para el caso de llamadas a subrutinas puede ser o no necesaria y dependerá de cada situación, además en caso de modificar la interrupción algún otro registro deberá ser considerado su agregado también en la rutina antes mostradas, estos registros pueden ser por ejemplo el PCLACH, FSR, etc.
En primer lugar declaramos dos variables W_Temp y S_Temp, en el caso particular de W_Temp como no sabemos en que banco de memoria estaba el programa principal cuando se realice una interrupción deberemos realizar una imagen de esa variable en cada uno de los bancos de memoria que podamos llegar a utilizar, esto significa que si declaramos a W_Temp en la posición 0x21 deberá existir una posición vacía en la posición 0xA1, 0x121 y 0x1A1, en caso de que nuestro microcontrolador disponga de 4 bancos, de esta forma nos aseguramos que si el programa principal estaba por ejemplo en el banco 2, al encontrar la rutina un **movwf W_Temp**. que para nuestro caso seria **movwf 0x21**, guardará el registro en una posición vacía (la 0x1A1).
Luego de esto, guardamos el STATUS en el registro w y antes de copiarlo a la variable realizamos un **clrf STATUS,** para asegurar de esta forma que nos encontremos en el banco 0, luego de copiar el STATUS a la variable ya estamos en condiciones de comenzar con nuestro programa de atención a la interrupción.
Al terminar este programa comienza el bloque que recupera la información de las variables, en primer lugar copiamos la variable del STATUS (la S_Temp) al registro w, y luego la copiamos a el registro STATUS, en este punto debemos tener en cuenta que no podremos usar ninguna instrucción que modifique las banderas del STATUS puesto que al retornar al programa principal no se encontraría este registro de la misma forma en que salió, la única instrucción que se puede utilizar para copia una variable a el registro W sin tocar las banderas es la **swapf** la cual invierte los nibbles, por lo tanto colocamos primero un **swapf W_Temp,f** para que invierta una vez los nibbles y luego recién la **swapf W_Temp,w** que realiza la copia definitiva a el registro w.
En este punto podemos entonces volver al programa principal, con todos los registro en el estado con el que fue llamada la rutina.

Instrucciones

El set de instrucciones de estos microcontroladores es muy sencilla comparada con el 8086, dispones de solo 35 instrucciones divididas en tres categorías

- **Instrucciones orientadas a bytes**
- **Instrucciones orientadas a bit**
- **Instrucciones de control y literales**

En las siguiente tablas de describe cada una de las instrucciones de microcontroladores de la familia 16XX, todas estas instrucciones ocupan una sola dirección de memoria de 14 bits, y se ejecutan en 1 ciclo de reloj, excepto las instrucciones que involucren salto (condicional o no) las cuales tardan 2 ciclos de reloj

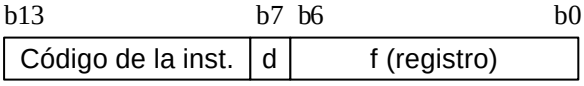
J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

Descripción de los campos

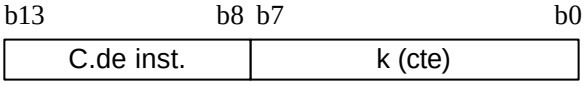
f	Indica una posición de memoria o registro puede ser 0 <= f <= 127, por lo tanto si deseamos acceder a los otros mapas deberemos sumar al código la modificación de RP0 y RP1
d	Define el destino de la operación, las instrucciones que tengan este parámetro podrán almacenar su resultado en w o en el registro indicado por f , los valores posibles para d son: ▪ 0 si el destino es w ▪ 1 si el destino es f esto valores están definido en el compilador de microchip, por lo tanto puede ser reemplazado por w o f respectivamente
b	Es utilizado en las instrucciones orientadas a bit y representa el nro de bit donde se realiza la operación, puede ser 0<= b <=7
k	Representa una cte de 8 bits excepto en los saltos y llamada a subrutina (CALL, GOTO) donde es de 11 bits

Codificación de la instrucción

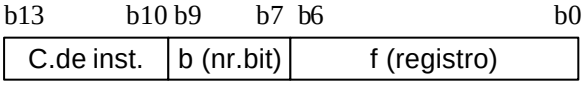
Instrucción Orientadas a bytes



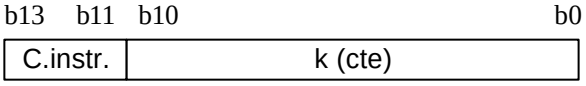
Instrucción Literal



Instrucción orientada a bit



Instrucción de Salto Incondicional (goto y call)



Instrucciones Orientadas a bytes

ADDWF f, d	Suma el W con el registro f	d=0,w W = W + f d=1,f f = W + f	C,DC,Z C=Carry
ANDWF f, d	And lógico del W con el registro f	d=0,w W = W and f d=1,f f = W and f	Z
CLRF f	Borra el registro f		Z
CLRW	Borra W		Z
COMF f, d	Complementa el registro f	d=0,w W = comp f d=1,f f = comp f	Z
DECF f, d	Decrementa en una unidad el registro f	d=0,w W = f –1 d=1,f f = f –1	Z
DECFSZ f, d	Decrementa en una unidad f, si el resultado es 0 saltea la próxima instrucción	d=0,w W = f –1 d=1,f f = f –1	
INCF f, d	Incrementa en una unidad el registro f	d=0,w W = f +1 d=1,f f = f +1	Z
INCFSZ f, d	Incrementa en una unidad f, si el resultado es 0 saltea la próxima instrucción	d=0,w W = f +1 d=1,f f = f +1	
IORWF f, d	Or inclusiva del W con el registro f	d=0,w W = W or f d=1,f f = W or f	Z
MOVF f, d	Mueve el registro f al destino (Con d=1 se utiliza solo para testear si es f es cero)	d=0,w W = f d=1,f f = f	Z
MOVWF f	Mueve de W a el registro f		
NOP	No realiza ninguna operación		
RLF f,d	Rota un bit hacia la izquierda el registro f a través del carry (al igual que RCL del 8088)	d=0,w W = rot izq f d=1,f f = rot izq f	C

J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

RRF f,d	Rota un bit hacia la derecha el registro f a través del carry (al igual que RCR del 8088)	d=0,w W = rot der f d=1,f f = rot der f	C
SUBWF f, d	Resta el W con el regis tro f (si k es < W se ocasiono un borrow y C = 1)	d=0,w W = f - W d=1,f f = f - W	C,DC,Z C = -borr
SWAPF f, d	Intercambia los nibbles de f Destino<7:4> = f<0:3> Destino<0:3> = f<7:4>	d=0,w W = swap f d=1,f f = swap f	
XORWF f, d	Or exclusiva del W con el registro f	d=0,w W = W xor f d=1,f f = W xor f	Z

Instrucciones Orientadas a bit

BCF f, b	Borra el bit nro b del registro f
BSF f, b	Activa bit nro b del registro f
BTFSC f, b	Realiza un test del bit b del registro f, si es 0 saltea la próxima instrucción
BTFSS f, b	Realiza un test del bit b del registro f, si es 0 saltea la próxima instrucción

Instrucciones Literales y de Control

ADDLW k	Suma la cte k a W y guarda el resultado en W	W = W + k	C,DC,Z C = Carry
ANDLW k	Realiza un And lógico de W con la cte k	W = W and k	Z
CALL k	Llama a una rutina ubicada en la posición k, guardando previamente el PC+1 en el Stack	PC+1 → Stack PC<10:0> = k PC<12:11>=PCLACH<4:3>	
CLRWDT	Borra el contador de watch dog	$\overline{TO} = 1$ $\overline{PD} = 1$ WDT = 0 WDT(presc) = 0	
GOTO k	Realiza un salto incondicional a k	PC<10:0> = k PC<12:11>=PCLACH<4:3>	
IORLW k	Realiza un or inclusivo de W con la cte k	W = W or k	Z
MOVLW k	Copia la cte k en W	W = k	
RETFIE	Retorna de una interrupción, la única diferencia con **return** es que pone la bandera de interrupción global en 1 (GIE=1)	GIE = 1 Stack → PC	
RETLW k	Retorna de una subrutina, y copia la cte k en el acumulador W	W = k Stack → PC	
RETURN	Retorna de una subrutina	Stack → PC	
SLEEP	Pone al microcontrolador en modo dormido, hasta que una interrupción lo despierte	$\overline{TO} = 1$ $\overline{PD} = 0$ WDT = 0 WDT(presc) = 0	
SUBLW k	Resta W a la cte k y guarda el resultado en W (si k es < W se ocasiono un borrow y C = 1)	W = k -W	C,DC,Z C = -Borr
XORLW	Realiza un or exclusivo de W con la cte k	W = W xor k	Z

Instrucciones de Salto condicional

Las instrucciones de Salto condicional realizan un test a un bit determinado de un registro determinado, contando con dos instrucciones una para realizar el test si el bit está en uno y otra

J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

por si esta en cero, ambas instrucciones solo realiza un salto fijo de una instrucción en caso de ser verdadera la condición.
Ejemplo: realicemos un programa que realice un test a la variable "NUMERO" incrementando en uno la variable "MENOR" si el valor es menor o igual a 128 o incremente en uno la variable "MAYOR" si el valor es mayor a 128.

```
           movf  NUMERO,w     ; copia el numero a realizar el test en w
           sublw d'128'       ; 128 – w C=0 si resultado es < 0 C=1 si > 0
           btfss STATUS,C     ; test de C si C=1 saltea la próxima instrucción
           goto  NUM_MAYOR    ; si C=0 ir a NUM_MAYOR

NUM_MENOR  incf  MENOR,f      ; si C=1 incrementar MENOR
           goto  CONTINUAR    ; e ir a CONTINUAR

NUM_MAYOR  incf  MAYOR,f; si salto a NUM_MAYOR incrementar MAYOR

CONTINUAR  nop
```

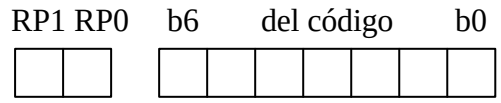
Este ejemplo utiliza la instrucción **btfss** que realiza el salto de una instrucción si el bit a realizar el test es 1.

Acceso indirecto a memoria

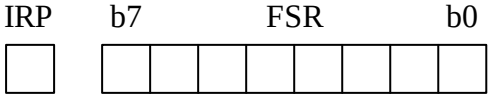
Memoria de Dato

Si bien no posee acceso indexado a memoria de variable, se a resuelto el problema utilizando dos registros para tal fin, tendremos entonces el registro FSR (04h para el 16F84) donde deberemos colocar la dirección de memoria a trabajar y el registro INDF (00h) el cual será una imagen de la posición de memoria que colocamos en el FSR.
De esta forma nos permite realizar vectores o tablas en las cuales podemos ingresar en FSR el comienzo del vector y luego incrementando este valor e ir accediendo a cada uno de sus elementos que se irán mostrando en el registro INDF, el registro INDF no es un registro físico, es una imagen del registro que señalamos en FSR.
Como vemos el FSR es un puntero de 8 bit, por lo tanto para mapear los 512 bytes que permite esta familia deberá hacer uso de un bit que se encuentra también en el registro STATUS el IRP

De esta forma se arma normalmente una dirección de registro tomada desde el código



De esta forma se arma un acceso indirecto



J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

ejemplo vamos a crear una rutina que borra el contenido de la memoria desde la posición 30h hasta 40h

```
      movlw  30
      movwf  FSR        ; copia en FSR el inicio del vector
OTRO1 clrf   INDF       ; borra el registro al que apunta FSR
      incf   FSR,f      ; incrementa FSR
      movlw  41
      subwf  FSR,w      ; le resta a FSR el 41
      btfss  STATUS,C   ; si el resultado es positivo ( C=1 ) llego al fin
      goto   OTRO1
```

Memoria de Programa

El acceso a memoria de programa no está permitido, puesto que no hay ninguna instrucción que lea estas posiciones de memoria, es por eso que si queremos almacenar alguna lista de valores constantes lo deberemos hacer mediante una técnica algo rebuscada.
En primer lugar usaremos una instrucción en particular la **retlw k**, esta instrucción debe ser utilizada dentro de una subrutina, puesto que realiza una función parecida al **return** copiando el ultimo valor ingresado al Stack en el PC, pero además devuelve en w el valor pasado como parámetro (k).
Para realizar entonces una tabla por ejemplo de valores ctes, tendremos que realizar una subrutina donde le pasemos por parámetro (por ejemplo en el w) la posición del elemento a leer, la rutina deberá modificar el PC para que salte al **retlw** correspondiente a la posición pasada en w y esta instrucción se encargará de pasa su valor a w y retorna al programa principal con el valor del elemento guardado en w.
Ejemplo vamos a crear una tabla que contenga la palabra “hola” para ser utilizada por ejemplo en un display LCD, o en cualquier elemento de salida.

```
TABLA_MEN  movwf TEMPTAB
           movlw HIGH TABLA_MEN ; cargo el PCLACH con el BMS
           movwf PCLATH         ; de la dirección actual
           movf  TEMPTAB,w      ; le sumo al PCL el valor pasado en w
           addwf PCL,f
           retlw 'H'
           retlw 'O'
           retlw 'L'
           retlw 'A'
           retlw 0
```

Se debe tener en cuenta, que PCL es de 8 bits por lo tanto la tabla tendrá un máximo de 255 elementos menos el valor del PCL en el momento de realizar la suma.
La forma de llamar a esta subrutina será la siguiente.

```
           Movlw 2            ; leer el segundo elemento
           call  TABLA_MEN    ; llamada a la subrutina de la tabla
           movwf LETRA        ; guarda el elemento 2 en la variable LETRA
```