DISPOSITIVOS LÓGICOS MICROPROGRAMABLES	El lenguaje ensamblador del PIC16F84A	8.1

El lenguaje ensamblador del PIC16F84A

Índice de contenidos

DISPOSITIVOS LÓGICOS MICROPROGRAMABLES	El lenguaje ensamblador del PIC16F84A	8.2

Introducción

El juego de instrucciones de un microprocesdor o microcontrolador es el conjunto de entradas binarias que producen acciones definidas durante un ciclo de instrucción. Un juego de instrucciones es para el microcontrolador lo mismo que una tabla de verdad es para una puerta lógica, un registro de desplazamiento o un sumador. Por supuesto, las acciones que realiza un microcontrolador con cada instrucción, son más complejas que las que realizan los dispositivos y puertas antes mencionados.

Una instrucción es un patrón de dígitos binarios el cual debe estar a disposición del microcontrolador en el tiempo y forma que éste lo requiera.

Por ejemplo, cuando el procesador de un microcontrolador PIC16F84A recibe el patrón binario de 12 bits '0000 0100 0000' en el momento adecuado, significa: Clear (borrar o poner a cero) el registro W, y corresponde a la instrucción CLRW.

En instrucciones del PIC16F84A se muestra el juego de instrucciones completo del PIC16F84A

Para los PIC se han creado unas instrucciones y una estructura de programa que los hacen más sencillos y atractivos todavía..

Las instrucciones de los microcontrolador PIC cumplen con las siguientes condiciones:

- Juego de instrucciones reducido: Por ejemplo, solo existen 35 instrucciones en el PIC16F84A.
- Sencillas y rápidas: La mayoría se ejecuta en un ciclo de instrucción, y solo las de salto precisan 2 ciclos. El ciclo de instrucción consta de 4 ciclos de reloj principal. De esta manera un dispositivo con un cristal de cuarzo de 20 MHz realiza 5 millones de instrucciones por segundo.
- Ortogonalidad: La ubicación de los operandos que manejan es muy flexible. Cualquier objeto del procesador puede actuar como origen o como destino.
- Formato uniforme de las instrucciones: Todas las instrucciones tienen una longitud fija de bits. Esta característica significa un notable ahorro de la memoria de programa y una facilidad en la construcción de compiladores.
- Formato uniforme de los datos.

Un programa es una serie de instrucciones mediante las cuales un sistema basado en CPU realiza una tarea en particular y la forma mas simple de realizar un programa es mediante el lenguaje ensamblador (Ver Sistemas microprogramables, Lenguajes de programación para sistemas basados en CPU).

Podríamos decir que el lenguaje ensamblador "*es complejo por su sencillez*". Esto quiere decir que, a diferencia de los lenguajes de alto nivel, aquí no hay funciones que solucionen y simplifiquen algoritmos, si no que hay que implementar hasta los pasos más elementales.

Programa en ensamblador

Para introducir un programa en ensamblador en un sistema basado en CPU tenemos que traducirlo a hexadecimal o a binario. Para realizarlo a suvez se utiliza un programa de ordenador, llamado programa ensamblador. Éste sería un ejemplo de programación en ensamblador o mnemónicos:

```
        org     0
start   movlw   0Eh
        movwf   REG1
        movlw   100
        addwf   REG1,1
```

```
end
```

Desventajas del ensamblador

Existe una gran diferencia entre el juego de instrucciones de un sistema basado en CPU y las tareas que este debe realizar. Las instrucciones tienden a hacer cosas como: sumar contenidos de dos registros, desplazar el contenido de un acumulador un bit, o colocar un nuevo valor en el contador de programa.

Por otro lado, centrandonos en el caso de los microcontroladores, este deberá hacer cosas como: reaccionar cuando una entrada digital se activa, comprobar si un valor analógico se ha excedido de un cierto umbral, activar un relé en un momento determinado, mostrar resultados en un panel LCD, comunicarse vía serie con otros dispositivos, etc. El programador en lenguaje ensamblador debe "traducir" estas tareas a secuencias de simples instrucciones. Esto no suele ser fácil y consume tiempo de trabajo.

Otro inconveniente es la no portabilidad. Cada microprocesador o microcontrolador posee su propio juego de instrucciones en el y su propia arquitectura interna.

Un programa en ensamblador escrito para el PIC16F84A, no correrá en un 65C02, Z80, 8080, 8051, o cualquier otro sistema basado en CPU. Incluso dentro de los PIC hay diferencias entre las distintas gamas como número y tipo de instrucciones, recursos disponibles, dirección de registros o uso de la memoria.

Para solucionar estos problemas están los programas de alto nivel, como el lenguaje C o Basic.

Mnemónicos

La tarea principal del ensamblador es la traducción de los códigos de operación en mnemónico en sus equivalentes binarios.

El ensamblador realiza ésta tarea usando una tabla como si lo hiciésemos "a mano" pero además debe determinar cuantos operandos requiere la instrucción y de que tipo. Esto es un poco complejo; algunas instrucciones (como CLRW, SLEEP) no tienen operandos, otras (ADDLW 13, GOTO FIN) tienen una, mientras que otras (BSF STATUS,C o BTFSS PORTA,O) requieren dos.

Identificación de elementos

Flags o Banderas

Los Flags o banderas son marcadores, representados por bits dentro del registro de **STATUS**, los mas importantes son:

- **Z:** Flag de cero, se pone a 1 cuando una operación que le afecta da como resultado un 0.
- **C:** Flag de Carry, se pone a 1 cuando la operación que le afecta sobrepasa el nivel de representación del procesador, en nuestro caso es 8 bits, de esta manera si sumamos a **1111 1111 b** un **0000 0011 b** el resultado seria **0000 0010 b** y el **bit de Carry** pasaría a **1**.
- **DC:** Flag de carry del nibbles inferior, este se comporta igual que el **bit de Carry**, solo que el limite de representación son los 4 bits inferiores, de esta manera si tenemos **0000 1111 b** y sumamos **0000 0111 b**, el resultado será **0001 0110 b** y el **bit de DC** se pone a 1, el **bit de Carry** estará a 0 al no superarse los 8 bits y el **bit Z** a 0 al ser el número diferente de 0.

Registros

Un registro es un espacio en la memoria de datos del microcontrolador en el que podemos guardar información, existen también unos registros en los cuales podemos configurar el microcontrolador o saber el estado de este o algunos de sus periféricos.

Un registro está compuesto por 8 bits los cuales se representan dándoles un numero según su posición, de esta manera el bit menos significativo (LSB) se le da el número 0 y el más significativo (MSB) el 7.

BIT	7	6	5	4	3	2	1	0
REGISTRO	X	X	X	X	X	X	X	X

Donde X puede ser 1 ó 0.

A los bits del 0 al 3 se les denomina nibbles inferior, y del 4 al 7 se denominan nibbles superior.

La forma de representación de parte de los bits de un registro suele ser:

Registro<3:0>

lo que indica los bits del 3 al 0 del registro.

De esta forma, para identificar el BIT Z de STATUS se pondría:

STATUS<2>

Elementos de una instrucción

En el caso del PIC16F84A y los de los PIC de la gama media cada instrucción está formada por una palabra de 14 bits que utiliza un tipo de código denominado **OPCODE** (Código de Operación), que especifica el **mnemónico** de la operación y los operandos que correspondan, que son los datos con los que opera la instrucción.

Ejemplo, instrucción *CLRF f*:

CLRF				**CLRF**
Clear f				
Operación	00 h → **f** 1 → **Z**			
Sintaxis	[Etiqueta] **CLRF f**			
Operadores	0 < **f** < 127			
Ciclos	1			
OPCODE	00	0001	1fff	ffff
Descripción	Se borra el contenido del registro **f** y el flag **Z** se activa			

El OPCODE de *CLRF f* es en binario "0000011fffffff" donde "fffffff" se sustituiría por el registro que se quiera borrar. **f** es una de las abreviaturas que se utilizan para describir las instrucciones del PIC usados en el lenguaje ensamblador y que son:

- **f** Representa un registro cualquiera de la memora de datos.
- **w** Registro de trabajo (Working Register).
- **b** Dirección de un bit dentro de un registro de 8 bits (0-7).
- **l** ó **k** Literal o constante de 8 bits.
- **d** Bit de destino, 0 ó 1.
- **x** Los bits que estén representados por este tipo de dato no tienen ninguna función y su valor lo define el compilador.

A continuación se explican con más detalle:

f (file register)

Este carácter se usa para definir registros de cualquier tipo. Cualquier instrucción que contenga este campo, contendrá la dirección de un registro, no su contenido. Un registro puede variar entre las direcciones 00h y 7Fh.

En el caso de los registros especiales en vez de la dirección podemos poner directamente el nombre del registro que el ensamblador se encargará luego de traducir a las dirección real.

Ejemplo, instrucción *BSF f,b* , Pone a 1 el bit **b** del registro **f.**

En lugar de poner:

```
BSF     03,5
```

podemos poner:

```
BSF     STATUS,5
```

con lo se pone a 1 el bit 5 del registrro STATUS.

w (working register)

w da nombre al acumulador de los PICs, el cual lo vimos anteriormente cuando tratamos los registros. Este no es un registro situado en un banco de memoria, si no que es independiente. A diferencia que el anterior, cuando nos referimos a él, nos referimos al contenido. Su uso es muy sencillo, pues lo usaremos principalmente para pasar información de un registro a otro, o para contener la información entre dos o más instrucciones.

b (bit addres dentro de un registro)

Esta letra define la dirección de un bit dentro de un byte. En ciertas ocasiones en vez de modificar o acceder a bytes tendremos que modificar o acceder a bits. De esta manera podemos especificar a una instrucción que posición ocupa el bit sobre el cual recaerá la acción que esta ejecute. Al igual que en los registros especiales, podemos poner directamente el nombre de un bit dentro de un registro.

Ejemplo:

En lugar de:

```
BSF     STATUS,5
```

ponemos:

```
BSF     STATUS,RP0
```

l o k (literal)

Este valor será almacenado en la propia instrucción en tiempo de ensamblado, esto significa que son los valores que introducimos en las instrucciones para que trabaje con ellos (independientemente de los datos que podamos almacenar o contener en la EEPROM de datos). El valor que podemos introducir dentro de un literal está comprendido entre 0 y 255, ya que es el máximo que puede representar un byte.

d (destiny bit)

Donde encontremos esta letra, debemos especificar donde se almacenará el resultado de una instrucción, en **w** o en un registro. Puesto que esto no es un lenguaje de alto nivel, no podemos almacenar el resultado de una operación sobre una tercera variable o registro, así que este deberá ser almacenado en el registro origen (sobrescribiéndose), o en el acumulador. Esto se define a través de dos valores:

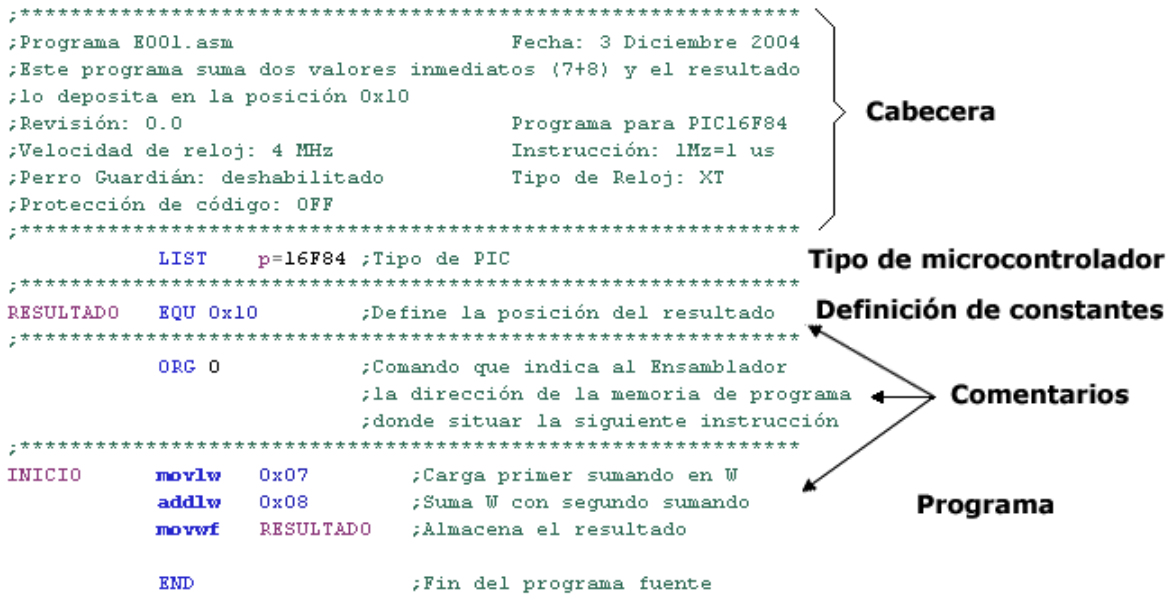
- 1: El resultado se almacenará en **f**.
- 0: El resultado se almacenará en **w**.

Estructura de un programa en ensamblador

Para hacer la tarea del programador más grata, se usan algunas convenciones. Cada uno puede adoptar las que más le agraden y ayuden para ser más productivo. En general, las convenciones son cualquier acción que facilita la revisión y comprensión de un programa, especialmente el que uno mismo ha escrito cuando tiene que revisarlo algunos meses después. Comentamos algunas convenciones que usaremos:

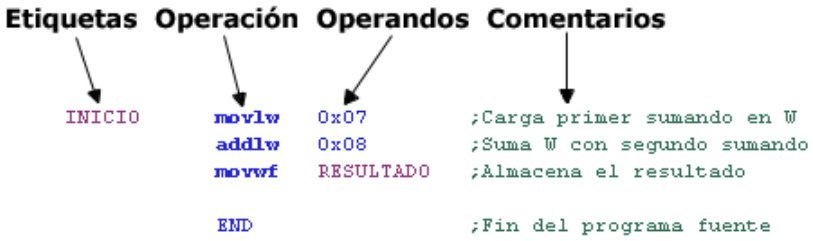
- Los ficheros de código fuente llevarán la extensión *.ASM
- Los ficheros de listado llevarán la extensión *.LST
- Los ficheros de código objeto llevarán la extensión *.OB]
- Los ficheros de errores llevarán la extensión *.ERR
- Los ficheros ejecutables en formato Intel Hex llevarán la extensión *.HEX
- Comentario descriptivo del programa (utilizar una cabecera estandarizada).
- Definir el microcontrolador que se usará (con las directivas LIST e INCLUDE).
- Introducir las opciones de compilación (que serán vistas más adelante) (opcional).
- Establecer las constantes que se usarán (con la directiva EQU).
- Reservar espacios de memoria (directiva RES) (si es necesario).
- Configurar los puertos.
- Desarrollar el programa con comentarios, en lo posible explicando cada línea de código..
- Los mnemónicos escritos en minúscula y las constantes y variables en mayúscula hacen que el código escrito sea más visible.
- Colocar las rutinas en el mismo sitio, todas contiguas.
- Dibujar diagramas de flujo o escribir seudocódigo.

Su estructura en un programa ejemplo muy simple:


;***
;Programa E001.asm Fecha: 3 Diciembre 2004
;Este programa suma dos valores inmediatos (7+8) y el resultado
;lo deposita en la posición 0x10
;Revisión: 0.0 Programa para PIC16F84
;Velocidad de reloj: 4 MHz Instrucción: 1Mz=1 us
;Perro Guardián: deshabilitado Tipo de Reloj: XT
;Protección de código: OFF
;***
LIST p=16F84 ;Tipo de PIC
;***
RESULTADO EQU 0x10 ;Define la posición del resultado
;***
ORG 0 ;Comando que indica al Ensamblador
;la dirección de la memoria de programa
;donde situar la siguiente instrucción
;***
INICIO movlw 0x07 ;Carga primer sumando en W
addlw 0x08 ;Suma W con segundo sumando
movwf RESULTADO ;Almacena el resultado
END ;Fin del programa fuente
Cabecera
Tipo de microcontrolador
Definición de constantes
Comentarios
Programa

DISPOSITIVOS LÓGICOS MICROPROGRAMABLES	El lenguaje ensamblador del PIC16F84A	8.9

Hemos visto la estructura general. Ahora veremos la posición de los elementos del código por 4 columnas:



- **Columna 1: *Etiquetas.*** Las etiquetas se rigen por las siguientes normas:
  - Debe situarse en la primera columna.
  - Debe contener únicamente caracteres alfanuméricos.
  - El máximo de caracteres es de 31.
- **Columna 2: *Operación.*** En esta columna se situarán las instrucciones. El campo del código de operación es el único que nunca puede estar vacío; éste siempre contiene una instrucción o una directiva del ensamblador.
- **Columna 3: *Operandos*** El campo de operandos o de dirección puede contener una dirección o un dato, o puede estar en blanco. Normanmente contendrá registros o literales con los que se operará (f, l o k , b y w).
- **Columna 4: *Comentario.*** El campo del comentario o de etiquetas es opcional. Aquí se situará cualquier comentario personalizado que deseemos. Estos son útiles para saber qué hace un programa sin tener que descifrar el código entero. El compilador (ensamblador) ignorará todo texto más allá del carácter punto y coma ";".

Los comentarios generalmente se sitúan en la cuarta columna para describir la acción de una línea de código, pero pueden situarse en cualquier parte de programa para describir cualquier otro evento, siempre que estén después del carácter ";" (semicolon en inglés).

Normalmente las columnas son separadas por una tabulación. El espacio mínimo entre dos columnas es de un carácter, que puede ser un espacio en vez de una tabulación.

Delimitadores (separación entre campos)

- Los campos van separados sólo con espacios y/o tabulaciones. No agregue nunca otros caracteres (comas, puntos, etc.)
- No utilice espacios extra, particularmente después de comas que separan operandos.(Ej: movlw 5, w)
- No use caracteres delimitadores (espacios y tabulaciones) en nombres o etiquetas.

Etiquetas (label)

Las etiquetas se sitúan a la izquierda de las instrucciones y sirven para agrupar fragmentos de código. Estos fragmentos pueden ser de dos tipos:

- El primer tipo no es un fragmento tal cual, si no que es un punto del programa al que podremos saltar de manera incondicional a través de la instrucción adecuada.
- El segundo tipo es denominado subrutina. Este empieza con una etiqueta y acaba con la instrucción RETURN o RETLW, que veremos más adelante.

Deberemos tener en cuenta:

- La etiqueta es el primer campo en una línea en lenguaje ensamblador y puede no existir.

- Si una etiqueta está presente, el ensamblador la define como el equivalente a la dirección del primer byte correspondiente a esa instrucción.
- Esta etiqueta puede volver a usarse en otro lugar pero como operando de una instrucción. El ensamblador reemplazará ésta etiqueta por el valor de cuando fue creada. Se usan frecuentemente en las instrucciones de salto.
- No puede existir más de una etiqueta en la primera columna o primer campo de instrucción.
- No pueden usarse como nombres de etiquetas a palabras ya reservadas por el ensamblador (ORG, EQU, etc.) o nombres de instrucciones (movlw, call, nop, etc.)

Ejemplo:

```
DATO    EQU    05h

INICIO  movlw  DATO
        goto   INICIO
```

La instrucción **goto INICIO** causa que la dirección de la instrucción con la etiqueta **INICIO (movlw)** se cargue en el **PC (Contador de Programa)**. Por lo tanto ésta instrucción será luego ejecutada.

No se permite el uso de números o caracteres no alfabéticos como primera letra de la etiqueta. Como regla práctica: usar siempre letras, y en mayúscula, al menos la primera.

Ejemplos:

```
TABLA2X2  Perrmitido
+PESO     NO permitido!
=>SALIDA  NO permitido!
-SALTO    NO permitido!
5ALFA     NO permitido!
Dato1     Permitido
Dato2     Permitido
Loop_A    Permitido
```

Operandos y direcciones

Los ensambladores permiten elegir con libertad el tipo de elemento a colocar en el campo de operando o dirección.

Sistemas de numeración

Los ensambladores aceptan números Hexadecimales, octales, binarios o decimal. Esta es la forma de representarlos:

```
Hexadecimal:
0A00h
$0A00

Binario:
%01001011
B'00100101'
01011010b

Octal:
@123
123Q

Decimal:
D'250'
.250
```

Ejemplo:

```
movlw .100
```

Significa: "mover el número literal 100 en decimal al registro de trabajo W"

Ya hemos indicado que MPLAB es el entorno de desarrollo de Microchip e incluye el ensamblador MPASM, para obtener información sobre la convención utilizada por este ver MPASM, el ensamblador de Microchip

Nombres

Los nombres pueden aparecer en el campo de operando; éstos son tratados como el dato que representan (Ver directiva EQU).

Códigos de caracteres

Algunos ensambladores permiten el uso de caracteres en ASCII. Por ejemplo:

```
      data  "hola 1,2,3"      ;cadena de caracteres
      data  'N'              ;carácter sencillo
CHAR  equ   't'
      movlw 'R'
```

Expresiones lógicas y aritméticas

Los ensambladores permiten conbinaciones de datos con operandos especiales, aritméticos o lógicos. Éstos operandos se llaman expresiones.

Por ejemplo:

```
REG1   EQU    05h
VALOR  EQU    20h

       movlw  VALOR+2
       addwf  REG1,1
       addwf  REG1+1,1
```

En estos casoo el compilador utilizará el resultado de sumar (VALOR+2) o (REG+1) como operando.

Directivas del ensamblador

Las instrucciones que podemos utilizar con un dispositivo son las que proporciona el fabricante para su producto y que forman parte del llamado "*repertorio de instrucciones*". Pero al utilizar un programa ensamblador podemos introducir además instrucciones o comando que proporciona el propio ensamblador. Estos comandos generalmente se utilizan para simplificar la tarea de programar, y reciben el nombre de directivas.

Por lo tanto las directivas no se traducen directamente a instrucciones del lenguaje máquina sino que asignan al programa ciertas áreas de memoria, definen símbolos, designan áreas de RAM para almacenamiento de datos temporales, colocan tablas o datos constantes en memoria y permiten referencias a otros programas.

Las directivas se utilizan como comandos escritos en el código fuente para realizar un control directo o ahorrar tiempo a la hora de ensamblar. El resultado de incorporar directivas se puede ver en el fichero *.LST, después de ensamblar el programa.

Para usar éstas directivas o pseudo-operandos, el programador las coloca en el campo del código de operación, y, si lo requiere la directiva, una dirección o dato en el campo de dirección.

Hay que aclarar que las instrucciones de los PIC's son únicas y que no hay nada mas, por ejemplo en el PIC16F84A son sólo 35 (ver instrucciones del PIC16F84A). Esto debe tenerse claro porque cuando se comienza con el ensamblador pueden confundirse un poco las propias instrucciones de los PIC's con las directivas propias del ensamblador.

A continuación se exponen las más relevantes.

Directiva EQU

El nombre viene de la palabra "equal", (igual)". La directiva EQU permite al programador "igualar" nombres personalizados a datos o direcciones. Los nombres utilizados se refieren generalmente a direcciones de dispositivos, datos numéricos, direcciones de comienzo, direcciones fijas, posiciones de bits, etc. Un nombre es más descriptivo que una simple dirección y la tarea de programar se hará mucho más sencilla. También podemos asignar un nombre a una instrucción que repitamos varias veces a lo largo de un algoritmo, de manera que sea mucho más sencilla la programación. A estos nombre que asignamos mediante esta directiva se les denomina constantes, ya que el registro al que apuntan no variará durante el programa

Ejemplos:

```
temp    equ    12
DATO    EQU    22
PORT_A  EQU    5
START   EQU    0
CARRY   EQU    3
TIEMPO  EQU    5
Bank_1  EQU    BSF STATUS,RP0
```

Estas líneas también pueden están incluidas en un archivo aparte al ASM (véase directiva INCLUDE).

No siempre es necesario que con esta directiva se igualen posiciones de memoria a las etiquetas, ya que podemos poner nombres a datos. Podemos definir una equivalencia con el nombre de otra equivalencia ya definida y rtealizar operaciones matemáticas. Por ejemplo, podemos calcular la frecuencia del ciclo máquina a partir de la frecuencia de reloj con la finalidad de emplearla para hacer otros cálculos de la manera que se describe a continuación:

```
PORT_B     EQU  PORT_A+1
PORT_C     EQU  PORT_A+2
FIN        EQU  START+100
FIN2       EQU  START+200
clockrate  EQU  .4000000     ;frecuencia del cristal
fclk       EQU  clockrate/4  ;frecuencia del reloj interno
```

El valor del operando debe estar ya definido anteriormente, sino el compilador entregará un error.

Además de esto, podemos igualar a las etiquetas cualquier otro tipo de valores que usemos, como, por ejemplo, el cero y el 1 en el bit de destino:

```
W       EQU     0
F       EQU     1
```

Con esto último, cuando usemos una instrucciónen donde debamos especificar donde se almacenará el resultado, en **w** o en un registro, en lugar de escribir :

- 1: para que el resultado se almacene en **f**.
- 0: para que el resultado se almacene en **w**.

Pondremos:

- F: para que el resultado se almacene en **f**.
- W: para que el resultado se almacene en **w**.

Generalmente esto último no será necesario realizarlo, siempre que incluyamos el fichero "INC" correspondiente al PIC con el que estemos trabajando (véase directiva INCLUDE).

Directiva ORG

Esta directiva dice al ensamblador a partir de que posición de memoria de programa se situarán las siguientes instrucciones. Rutinas de comienzo, subrutinas de interrupción y otros programas deben comenzar en locaciones de memoria fijados por la estructura del microcontrolador. Recordemos que el 16F84 sólo tiene 1024 posiciones de memoria flash para código.

La directiva ORG hace al compilador colocar el código que le sigue en una nueva dirección de memoria (la salida del compilador no solo coloca los códigos de operación sino también las direcciones de cada instrucción del programa). Usualmente se la utiliza para: reset, programas de servicios de interrupción, programa principal, subrutinas.

Ejemplos:

1) Inicia el programa en la posición cero:

```
ORG     0x00
```

2) Inicia el programa en la posición 0000h y luego pasa a la 0005h para no utilizar la posición del vector de interrupción (0004 h)

```
        ORG     0x00    ; El programa comienza en la dirección 0 y
        GOTO    inicio  ; salta a la dirección 5 para sobrepasar
        ORG     0x05    ; el vector de interrupción, situado en la posición 4
Inicio  xxx...
```

3) Inicia el programa en la posición 0000h y luego pasa a la 0005h para no utilizar la posición del vector de interrupción (0004 h). Si se produce una interrupción se pasa a la posición **interr**. Las subrutinas comienzan a partir de la dirección 0300h.

```
            ORG   00h         ;vector de reset
            goto  inicializa
            ORG   04h         ;vector de interrupción
            goto  interr
            ORG   05h
inicializa  movlw 08h         ;aquí comienza el programa
            .
            .
            ORG   300h        ;subrutinas
Subrutina1
            .
            .
            return
Subrutina2
            .
            .
            return
```

Directiva #INCLUDE

Esta directiva indica que archivos deberán tomarse en cuenta a la hora de compilar el código. Normalmente se usa para incluir el archivo de PIC que el ensamblador tiene entre sus archivos, con el cual el compilador será capaz de reconocer todos los registros especiales y sus bits. Su uso nos recordará al #include del lenguaje C. Esta línea debe colocarse al principio, y tiene la siguiente sintaxis:

```
#INCLUDE        ; Lista de etiquetas de microchip
```

En ciertas ocasiones gran cantidad errores son debidos a que el nombre del archivo puesto entre comillas no se escribe correctamente.

Si utilizamos MPLAB, un entorno de desarrollo que proporciona gratuitamente Microchip, dispondermos de los archivos con extension **.INC** para cada uno de los PIC desarrollados hasta la aparición de la versión de **MPLAB** que utilicemos. En estos archivos se definen todos los registros así como otros elementos de acuerdo al microcontrolador que estemos utilizando.

También podemos crear nuestros propios archivos "INC" con funciones, definiciones y subrutinas que utilicemos a menudo en nuestro código para evitar tener que copiarlas cada vez.

El archivo P16F84A.INC que viene con MPLAB contiene definiciones de registros, bits y bits de configuración. Los archivos INC pueden verse con cualquier editor de texto pero no se recomienda modificarlos, para no perder compatibilidad con programas desarrollados por otros.

Utilizar el **INC** del PIC que estamos utilizando en nuestro programa no es obligatorio, y podemos omitirlo, pero a cambio tendremos que definir los nombres de los registros que usemos o bien llamarlos por su posición de memoria.

Esto puede a la larga ser problemático de manera que se recomienda utilizar los archivos **INC** correspondientes al PIC que utilicemos porque además de facilitar la creación del programa al no tener que recordar las direcciones reales de los registros también se facilita el paso de un programa diseñado para un microcontrolador hacia otro distinto.

Si utilizamos las posiciones de memoria con la dirección real, podemos hacer incompatibles las operaciones entre registros. Por ejemplo, CLRF 0x05, borra el registro ubicado en esa direccion, que no es ni mas ni menos que el PORTA (Puerto A) en el PIC16F84A. Pues bien, si queremos

actualizarnos a otro microcontrolador pero resulta que en este el registro 0x05 tiene otra función nos será mucho mas dificil actualizar el programa. Ahora bien, si hubiésemos utilizado CLRF PORTA, y el .INC correspondiente al nuevo microcontrolador ya se ocupará el ensamblador de realizar las correspondencias.

Y por supuesto siempre será mas fácil recordar PORTA que no 0x05.

También permite incluir otros programas. Por ejemplo:

```
#INCLUDE "DISPLAY.ASM"
```

Esto le dice al compilador que incluya el código que se encuentra en el archivo DISPLAYY.ASM como si fuese parte del propio programa. Esto es muy util para reutilizar códigos realizados con anterioridad.

Directiva LIST

Este comando sirve para que el compilador tenga en cuenta sobre qué procesador se está trabajando. Este comando debe estar en todo proyecto, situado debajo del "include", con la siguiente sintaxis.

```
LIST P=PIC16F84A
```

Directiva END

Al igual que las dos anteriores, esta debe ir incluida una sola vez en todo el programa. En concreto, esta debe situarse al final, para indicar al ensamblador que el programa ha finalizado. Esta siempre debe estar presente, aunque el flujo de nuestro programa acabe en un bucle.

Directiva #DEFINE

#DEFINE es una directiva muy util. Define se usa para crear pequeñas macros. Con estas macros podremos poner nombres a pequeños fragmentos de código que nos facilitarán la realización y comprensión del algoritmo.

Por ejemplo, podremos poner nombres a bits.

```
#define CERO STATUS,2
```

Así, en vez de tener que llamar al bit por un numero y un registro, podremos usar directamente la palabra CERO.

```
#define  CINCO  5
```

Cada vez que se utilice la palabra **CINCO** será reemplazada en el momento de la compilación por el número 5.

Otro ejemplo muy práctico es el de poner nombre a un fragmento de código usado frecuentemente. Este fragmento de código, puede ser por ejemplo, el que conmuta entre los dos bancos.

```
BSF     OPTION,RP0
BCF     OPTION,RP0
```

Como cambiamos varias veces de banco a lo largo de un algoritmo, puede resultar más práctico ponerle un nombre.

```
#define BANCO1  BSF OPTION,RP0
#define BANCO0  BCF OPTION,RP0
```

De este modo bastará con poner BANCO1 o BANCO0 para conmutar entre los dos bancos de memoria de manera que cada vez que se utilice la palabra **BANCO1**, en realidad se estará utilizando la instrucción **BSF STATUS,RPO**

En el siguiente ejemplo:

```
#define    salida    PORTA,3
```

No tendremos necesidad de recordar cual era la patilla de salida, sino que solo lo mencionaremos como **salida**. Cada vez que aparezca la palabra **salida** en el código, ésta será interpretada como PORTA,1 que es una instrucción válida. Podemos ponerlo a cero con la instrucción.

```
  BCF salida
```

En vez de tener que poner.

```
  BCF PORTA,3
```

Una cosa a tener en cuenta es que con la directiva INCLUDE, podemos prescindir del carácter almohadilla (#), pero en el caso de la directiva DEFINE, no.

Esta directiva es muy util porque hace el código más fácil de leer y entender.

Directiva TITLE

Esta directiva no sirve de mucho, pero será útil para aquellos que quieran que el compilador tenga en cuenta el título que le ha puesto a su código. Tiene la siguiente sintaxis:

```
TITTLE "Nombre del código"
```

Este nombre aparecerá en los archivos .lst (listados) que cree el compilador.

Directivas IF...ELSE...ENDIF

Algunos ensambladores permiten incluir o excluir partes del programa dependiendo de condiciones que existan en el tiempo de compilación.

La forma típica es:

```
  IF CONDICION
      .
      .
```

```
    ELSE
        .
        .
    ENDIF
```

Ejemplo:

```
SINK  EQU  1 ; (cambiar por 0 en caso necesario)
      IF   SINK=1
           BCF PORTA,0
      ELSE
           BSF PORTA,0
      ENDIF
```

En este caso el valor de **SINK** hará que el compilador utilice distintas instrucciones de código.

Si la condición es verdadera en el tiempo de compilación, las instrucciones que están entre IF y ELSE se incluirán en el programa. Si la condición es falsa se incluirán en el programa las instrucciones entre ELSE y ENDIF.

Los usos típicos son:

- Para incluir o excluir variables extras
- Para incluir código de diagnóstico en condiciones de testeo (DEBUG).
- Para permitir datos de distintos tamaños.

Desgraciadamente, el ensamblado condicional, tiende a complicar la lectura del programa, por lo tanto, sólo debemos utilizarlo si es necesario.

Directiva MACRO

Esta directiva resulta muy potente y a diferencia de la directiva #define se pueden crear macros más extensas, lo que nos evitará tener que ejecutar reiteradamente fragmentos de código idénticos. Cuando una macro es invocada, esta es copiada por el ensamblador en el lugar de la invocación dentro del código fuente. La macro se declara con la directiva MACRO, y termina con la directiva ENDM.

Creación de una macro denominada **activar**:

```
activar macro
CLRF PORTA
BSF PORTB,2
endm
```

Hemos creado una macro llamada **activar** de manera que en nuestro código cada vez que pongamos la palabra **activar**, el ensamblador la reemplazará por **CLRF PORTA...** etc. hasta el final de la macro que termina con la directiva **ENDM** (fin macro).

Las macros permiten asignar un nombre a una secuencia de instrucciones de manera que son útiles cuando ocurren secuencias de instrucciones repetitivas. Luego se utiliza el nombre de la macro en el programa como si se usara la secuencia de instrucciones anterior.

Las macros no son lo mismo que las subrutinas. El código de las subrutinas aparece una sola vez en un programa y la ejecución del programa salta a la subrutina. En cambio, el ensamblador reemplaza cada aparición del nombre de la macro con la secuencia especificada de instrucciones. Por consiguiente la ejecución del programa no salta a la macro como una subrutina.

Ejemplo:

Archivo "MULX10.ASM"

```
MULX10  MACRO            ;comienzo de la macro
        MOVF   tiempo,W  ;guarda el tiempo en W
        RLF    tiempo    ;multiplica por 2
        RLF    tiempo    ;multiplica por 2
        RLF    tiempo    ;multiplica por 2
        ADDWF  tiempo    ;le suma una vez más
        ADDWF  tiempo    ;le suma una vez más
        ENDM             ;fin de la macro
```

Archivo "EJEMPLO1.ASM":

```
#INCLUDE   "MULX8.ASM"
tiempo     EQU  0Ch
resultado  EQU  0Dh

           MOVLW  20
           MOVWF  tiempo
           MULX10
           MOVWF  resultado
           END
```

Si ensamblamos "EJEMPLO1.ASM" notaremos que el listado final (EJEMPLO.LST) queda de la siguiente forma:

```
tiempo     EQU  0Ch
resultado  EQU  0Dh

           MOVLW  20
           MOVWF  tiempo
           MOVF   tiempo,W  ;guarda el tiempo en W
           RLF    tiempo    ;multiplica por 2
           RLF    tiempo    ;multiplica por 2
           RLF    tiempo    ;multiplica por 2
           ADDWF  tiempo    ;le suma una vez más
           ADDWF  tiempo    ;le suma una vez más
           MOVWF  resultado
END
```

Problemas con las MACROS

Con las macros se puede trabajar rápidamente, pero pueden resultaer poco eficientes. Veamos un error muy común al utilizar macros, en este caso se utiliza una macro denominada MOVFF:

```
MULX10   MACRO           ;comienzo de la macro
         MOVF   AUX1,W   ;Mueve contenido de un registro a otro
         MOVWF  AUX2     ;a través del acumulador
         ENDM            ;fin de la macro
```

Porción de código:

```
    MOVLW .1         ;TEMP=1
    MOVWF TEMP
    DECF  TEMP,F     ;Z se va a 1
    BTFSS STATUS,Z   ;salta si o si
    MOVFF AUX1,AUX2 ;Macro
    MOVWF PORTA
; ...
```

En la línea de la macro está el error porque los saltos (BTFSS) no pueden saltar macros. Las macros están compuestas por más de una instrucción, y el salto se producirá dentro de la misma.

El código anterior con la macro incrustada sería:

```
MOVLW .1        ;TEMP=1
MOVWF TEMP
DECF  TEMP,F    ;Z se va a 1
BTFSS STATUS,Z  ;salta si o si
MOVF   AUX1,W   ;líneas de anterior macro
MOVWF  AUX2     ;
MOVWF  PORTA
; ...
```

Otro tema importante, que se ilustra en este ejemplo, es que las macros pueden modificar registros (en este caso W) de forma que el programador podría no tener en cuenta.

En el ejemplo anterior, PORTA se debería cargar con 1, que aparentemente era el valor de W, pero la macro lo ha modificado, lo que resulta en otro error.

Ejemplos de macros

```
; ***********************************************************
; macros.asm ;
; "MACROS para 16F84" ;
; ***********************************************************

callz   macro  subbrutina
        btfsc  STATUS,Z
        call   subrutina
        endm

callnz  macro  subrutina
        btfss  STATUS,Z
        call   subrutina
        endm

movff   macro  f2,f1  ;(atención, se destruye W)
        movf   f1,w
        movwf  f2
        endm

movlf   macro  file,literal  ;(atención, se destruye W)
        movlw  literal
        movwf  file
        endm

;Atención, para usar estas macros ya debe estar activo el banco 1
CONF_PORTA  macro  dato
            movlw  dato
            movwf  TRISA
            endm

CONF_PORTB  macro  dato
            movlw  dato
            movwf  TRISB
            endm

;configurar Option Register:
CONF_OPTION macro  dato
            movlw  dato
            movwf  OPTION_REG
            endm

;configurar el registro de interrupciones:
CONF_INTCON macro  dato
            movlw  dato
            movwf  INTCON
```

```
            endm

SET_BANK_0  macro
            BCF     STATUS,RP0
            endm

SET_BANK_1  macro
            BSF     STATUS,RP0
            endm

;enable y disable all the mascarable interrupts (16F84):
EI          macro
            bsf     INTCON,GIE
            endm

DI          macro
            bcf     INTCON,GIE
            endm

#define  iEnable   EI
#define  iDisable  DI

;arrancar el timer:
RESET_TIMER macro
            bcf     INTCON,T0IF
            endm

; inicializar timer antes de hacer RESET_TIMER para que arranque.
INIT_TIMER  macro   dato
            movlw   dato
            movwf   TMR0
            endm

jmp         macro   salto
            goto    salto
            endm

ret         macro
            return
            endm

;Complemento a 1 de W:
comw        macro
            xorlw   0xff
            endm

;Instrucciones de salto tipo Z80

jz      macro  _salto   ;salta si zero
        btfsc  STATUS,Z
        goto   _salto
        endm

jnz     macro  _salto   ;salta si no zero
        btfss  STATUS,Z
        goto   _salto
        endm

jc      macro  _salto   ;salta si carry
        btfsc  STATUS,C
        goto   _salto
        endm

jnc     macro  _salto   ;salta si no carry
        btfss  STATUS,C
        goto   _salto
        endm
```

```
;*****************************************************
;      FIN
;*****************************************************
```

DISPOSITIVOS LÓGICOS MICROPROGRAMABLES	El lenguaje ensamblador del PIC16F84A	8.8