Universidad
Francisco de Vitoria
UFV Madrid

Arquitectura y Organización de Computadores

Sesiones de Laboratorio curso 2019-2020

Olga Peñalba Rodríguez y Daniel León

Sesión de Laboratorio 3

En esta sesión continuaremos avanzando en los aspectos básicos relacionados con la programación en ensamblador, introduciendo las instrucciones de salto, que te permitirán programar algoritmos con sentencias de selección condicional y bucles. Veremos también las instrucciones de rotación y de operación lógica.

Objetivos

Los objetivos generales que se persiguen son:

- Profundizar en el lenguaje del PIC16F886, incorporando a los programas instrucciones de control de flujo que implementen saltos condicionales del tipo if-then-else y bucles.
- Entender el concepto de rotación y sus usos.
- Conocer el uso de las operaciones lógicas.
- Comprender las ventajas del uso de etiquetas en el código, y las distintas formas de definirlas.

Resultados de aprendizaje

Tras la realización de esta sesión de laboratorio, deberás ser capaz de:

- Utilizar etiquetas para nombrar los diferentes datos que maneja el programa.
- Calcular el valor de las etiquetas que aparecen en un código.
- Calcular el valor de los bits de estado Z y C tras la ejecución de instrucciones.
- Utilizar las instrucciones btfss, btfsc y goto para implementar saltos condicionales.
- Utilizar rrf y rlf para rotar bits y operaciones lógicas para aplicar máscaras entre otros usos.

Teoría

Etiquetas

Las etiquetas en lenguaje ensamblador son un nombre que se asocia a un valor constante, que puede representar tanto un dato como una dirección de un dato o una dirección de una instrucción.

Las direcciones de memoria son números que pueden expresarse en cualquier base de numeración, aunque es habitual que sea en hexadecimal, representando un byte en dos caracteres.

Uso de etiquetas en lugar de direcciones

Dentro del código utilizamos direcciones como argumentos en:

- Las instrucciones **goto** y **call**: la dirección indica dónde se encuentra la dirección de la siguiente instrucción a ejecutar.
- En el resto de instrucciones que operan con datos en memoria (**addwf**, **subwf**, **movf**, etc.): en este caso las direcciones indican la posición de memoria donde se encuentra el dato para hacer la operación.

En ambos casos, puede utilizarse una etiqueta en lugar de la dirección, de manera que el código resultante es más fácil de comprender y depurar.

Dentro del código, utilizamos números como argumentos en las instrucciones que operan con literales. Estos números también pueden ser sustituidos por etiquetas debidamente definidas.

Definición de etiquetas

Para utilizar una etiqueta, ésta tiene que estar definida. Existen dos formas de definir etiquetas:

- Explícita: utilizamos la directiva **EQU**, que asocia el nombre de la etiqueta con el valor (dirección) que va a representar. Se utiliza normalmente para definir etiquetas asociadas a datos o direcciones de datos.
- Implícita: escribimos el nombre de la etiqueta delante del código de una instrucción. Se utiliza para definir etiquetas asociadas a direcciones de instrucciones.

En ambos casos, el nombre de la etiqueta debe aparecer en la primera columna de la línea correspondiente.

Existe una colección de etiquetas predefinidas en el fichero "P16F886.INC" que hacen referencia a diferentes elementos del procesador. Por ejemplo, cuando queramos utilizar el registro W como destino de una operación, podemos utilizar la etiqueta W en vez de 0, haciendo el código más legible.

Podemos incorporar estas etiquetas predefinidas en nuestros programas mediante la directiva INCLUDE, de la siguiente manera:

INCLUDE “P16F886.INC”

Flags de condición: Z, DC y C

Los flags Z, DC y C son bits que se almacenan en las posiciones menos significativas del registro STATUS y que indican determinadas propiedades del último resultado producido por una cierta instrucción.

- **Z**: indica si el resultado de la instrucción ha sido 0 (es el bit 2 del STATUS).
- **DC**: indica si se ha producido un acarreo al operar con los 4 bits menos significativos de los datos (es el bit 1 del STATUS)
- **C**: indica si se ha producido un acarreo en el resultado de la operación (es el bit 0 del STATUS).
    - Si se realiza una suma, hay acarreo cuando C=1.
    - Si se realiza una resta, hay acarreo cuando C= 0. Esto es debido a la forma en la que se hace la resta, sumando el primer operando con el complemento a dos del segundo.

No todas las instrucciones modifican dichos bits: algunas modifican los tres, como las sumas y restas, otras sólo el bit Z, como **movf**, y otras ninguno, como un **goto**.

Instrucciones de control de flujo en el lenguaje del PIC16F886

Las instrucciones de control de flujo (o saltos) presentes en el lenguaje se dividen en dos tipos: incondicionales y condicionales.

- ***Salto incondicional*** es aquel que se produce siempre, es decir, no depende de ninguna condición. Las instrucciones para salto condicional presentes en el PIC16F886 son **goto**, **call**, **retfie**, **retlw**. Las tres últimas se utilizan para llamadas y retornos de subrutinas. El **goto**, como ya vimos, realiza un salto a la dirección que se especifica como argumento.
- ***Salto condicional*** es aquel que se produce únicamente si se cumple una determinada condición. En el PIC16F876 existen 4 instrucciones que implementan saltos condicionales: **btfss**, **btfsc**, **incfsz**, **decfsz**. Para todas ellas, la dirección destino del salto está implícita, y es la de la instrucción que se encuentra dos posiciones más arriba en la memoria, es decir, si *dir* es la dirección de la instrucción de salto y la condición del salto se cumple, la siguiente instrucción ejecutada será *dir* + 2. De hecho, la penúltima letra del mnemotécnico de las instrucciones, "s" significa "skip", porque si se cumple la condición, se salta la siguiente instrucción.

Las instrucciones btfss y btfsc tienen la siguiente sintaxis y significado:

btfsc f,b	Comprueba el valor del bit b del dato f. Si b = '0' entonces salta.
btfss f,b	Comprueba el valor del bit b del dato f. Si b = '1' entonces salta.

Saltos en función de los resultados de comparar dos valores

Las instrucciones de control de flujo (o saltos) presentadas parecen sencillas de utilizar. Sin embargo, la mayoría de las decisiones que se toman en los programas no depende del valor de un bit, sino del resultado de comparar dos datos, como

If (a = b) then … else … If (a < b) then … else … If (a = 0) then … else …

Veremos en esta sección como programar esas situaciones comunes, combinando las instrucciones **btfsc**, **btfss**, **goto** y **subwf** (ésta última se utilizará para comparar, dado que no existe una instrucción específica de comparación).

Comparación mediante restas y valores de Z y C

La siguiente tabla refleja cómo interpretar la relación que hay entre dos datos en función de los resultados de una resta.

Resultado de A-B	Relación entre A y B	Valores de Z y C
Positivo	A > B	Z = 0, C = 1
Negativo	A < B	Z= 0, C = 0
Cero	A = B	Z = 1, C = 1

EJEMPLO 1. El siguiente fragmento de código muestra cómo implementar la siguiente instrucción de alto nivel

If (DatoA = DatoB) then *Iguales...* else *Distintos*...

En ensamblador del PIC resultaría:

```
                movf    DatoA,0         ;mueve el DatoA a W
                subwf   DatoB,0         ;se lo resta a DatoB y lo deja en W
                                        ;la resta actualiza el bit Z, poniéndolo a
                                        ;1 si el resultado es cero
                btfss   STATUS,2  ;comprobamos si es '1' el bit 2 (Z)
                goto    Distintos       ;se ejecuta esto si Z no es '1', luego
                                        ;los datos son distintos
Iguales     ...                     ;se ejecuta esto si Z es '1', luego los
                ...                     ;datos son iguales
                goto    Fin             ;hay que incluir un salto para no continuar
                                        ;realizando la parte de Distintos
Distintos       ...
                ...
Fin             ...
```

También podría hacerse utilizando btfsc, de la siguiente forma:

```
                movf    DatoA,0         ;mueve el DatoA a W
                subwf   DatoB,0         ;se lo resta a DatoB y lo deja en W
                                        ;la resta actualiza el bit Z, poniéndolo a
                                        ;1 si el resultado es cero
                btfsc   STATUS,2        ;comprobamos si es '0' el bit 2 (Z)
```

```
                goto    Iguales     ;se ejecuta esto si Z no es '0', luego
                                    ;los datos son iguales
Distintos       ...                 ;se ejecuta esto si Z es '0', luego los
                ...                 ;datos son distintos
                goto    Fin         ;hay que incluir un salto para no continuar
                                    ;realizando la parte de Iguales
Iguales         ...
Fin             ...
```

EJEMPLO 2. El siguiente fragmento de código muestra cómo implementar la siguiente instrucción de alto nivel

If (Dato = 0) then EsCero... else NoCero...

En ensamblador del PIC resultaría:

```
                movf    Dato,1      ;la instrucción movf actualiza Z
                btfss   STATUS,2    ;comprobamos si es '1' el bit 2 (Z)
                goto    NoCero      ;se ejecuta esto si Z no es '1', luego
                                    ;el dato no es cero
EsCero      ...                     ;se ejecuta esto si Z es '1', luego el
                                    ;dato es cero
                goto    Fin         ;hay que incluir un salto para no continuar
                                    ;realizando la parte de NoCero
NoCero          ...
Fin             ...
```

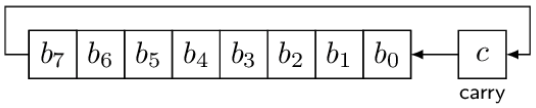
Obsérvese como para comparar con 0 no es necesario hacer la resta: una única instrucción, "movf Dato,1", consigue actualizar Z del mismo modo.

También se podría hacer utilizando btfsc, de la siguiente forma:

```
                movf    Dato,1      ;la instrucción movf actualiza Z
                btfsc   STATUS,2    ;comprobamos si es '0' el bit 2 (Z)
                goto    EsCero      ;se ejecuta esto si Z no es '0', luego
                                    ;el dato es igual a cero
NoCero          ...                 ;se ejecuta esto si Z es '0', luego el
                ...                 ;dato es distinto de cero
                goto    Fin         ;hay que incluir un salto para no continuar
                                    ;realizando la parte de EsCero
EsCero          ...
Fin             ...
```

Operaciones de rotación de registros

El repertorio de instrucciones incluye las operaciones **rlf** (rotate left through carry) y **rrf** (rotate right through carry). Estas instrucciones desplazan, a izquierda o derecha respectivamente, un registro, colocando el bit saliente en el indicador C de carry del registro **STATUS**. El bit entrante siempre es el que existía previamente en C.



Ejemplo de instrucción RLF

Algunos usos comunes de estas instrucciones son:

- Multiplicar por dos un valor de tipo byte. La instrucción rlf desplaza todo el contenido del registro un bit a la izquierda. Previamente hay que asegurarse que el Carry esté a 0. En el bit 0 se coloca un 0 (procedente del carry) y el bit 7 original queda en el indicador de carry del registro STATUS. De esta forma se puede saber si el número resultante es mayor de 255 y es posible hacer multiplicaciones con resultado de 16 bits.
- Dividir por dos un valor de tipo byte. La instrucción rrf desplaza el contenido del registro un bit a la derecha. En el bit 7 se coloca un 0 (previamente introducido en el carry) y el bit 0 original queda en el indicador de carry del registro STATUS. Aunque es necesario realizar algún paso más, divisiones de valores de 16bits también se realizan fácilmente gracias a esta instrucción.
- Conocer si un byte es par o impar, mediante una rotación a la derecha, si carry es 1, el valor es impar. Si es 0, el valor es par. Tras la rotación, carry contiene el valor original del bit 0 del registro.
- Saber si un dato es positivo o negativo. Mediante una rotación a la izquierda, el contenido de carry es el signo positivo (0) o negativo (1).
- Copiar un registro en otro en ocho iteraciones. Esto es especialmente útil si se quiere invertir el registro, alternando las instrucciones rlf y rrf sobre el registro fuente y destino.
- Calcular la paridad de un byte, mediante la combinación de 8 rotaciones y or exclusivos (xor).
- Transmitir el contenido de un registro mediante protocolos serie, como I2C, SPI o asíncrono, que se utilizan masivamente en electrónica. Con ayuda de las instrucciones de desplazamiento, podemos transmitir, bit a bit, el contenido de un registro.

Operaciones lógicas

Dentro de las 35 instrucciones de la arquitectura PIC16, podemos encontrar 6 instrucciones destinadas a la operación lógica, divididas en dos grupos:

- Operaciones sobre registros, con direccionamiento directo (o absoluto). En estas instrucciones, el primer operando direcciona un registro de memoria y el segundo indica el destino del resultado de la operación, que se guarda en el registro w si d=0 y en f si d=1. Todas las operaciones afectan al flag Z.
    - ANDWF f,d : Realiza un AND, bit a bit, entre el registro direccionado por f y el registro w.
    - IORWF f,d : Realiza un OR inclusivo, bit a bit, entre el registro direccionado por f y el registro w.
    - XORWF f,d : Realiza un OR exclusivo, bit a bit, entre el registro direccionado por f y el registro w.

- Operaciones sobre w con direccionamiento inmediato. El único argumento es un literal (número) que va a operarse con el contenido del registro de trabajo w. El resultado quedará almacenado en w y el flag Z se verá afectado.
    - ANDLW literal : W= W AND literal
    - IORLW literal : W= W OR literal
    - XORLW literal : W= W XOR literal

Además de todos los usos de dominio de las operaciones lógicas, se utilizan con asiduidad para aplicar máscaras sobre datos o para hacer cálculos de CRC o paridad. Como ejemplo, si se quieren poner a '0' los bits 3,5,6 y 7 de un dato, sin afectar al resto de bits, bastaría con hacer un AND de ese dato con la máscara b'00010111' (0x17). De la misma forma, si se desea poner esos bits a 1, se realizaría un OR con la máscara b'11101000', (0xE8), el complemento a 1 de la anterior.

Cuestiones y ejercicios

P1: Etiquetas

Escribe de nuevo el programa que suma dos datos en memoria, utilizando etiquetas para referirte a los datos y al resultado. Por ejemplo, puedes utilizar Var1 y Var2 para las direcciones que representan datos y Suma para el resultado.

P2: Etiquetas

Considera el siguiente código escrito en lenguaje ensamblador del PIC16F886.

```
                LIST        p=16F886
                INCLUDE     "P16F886.INC"
                ORG         0x00
                goto        Inicio
                ORG         0x10

VarA            EQU         h'030'
VarB            EQU         h'031'
VarC            EQU         h'032'

Inicio          movf        VarA,W
                subwf       VarB,W
                btfss       STATUS,C
                goto        Caso1
Caso2           movf        VarB,W
                movwf       VarC
                goto        Fin
Caso1           movf        VarA,W
                movwf       VarC
Fin             goto        Inicio
                END
```

¿Qué hace este programa? Verifícalo con MPLAB X IDE.

Reescribe el programa eliminando todas las etiquetas y comprueba con MPLAB IDE que funciona igual que el programa con etiquetas.

Nota: simplemente hay que sustituir cada uso de una etiqueta por el valor de dicha etiqueta. El programa sin etiquetas tendrá la primera columna totalmente en blanco.

E1: Flags de condición

Suponiendo que en la posición 30h de memoria hay almacenado un 0 y en la 31h un 5, ¿Qué valores tomarán los bits Z y C al ejecutarse la primera instrucción? ¿Y la segunda?.

E2: Saltos

Los cuatro siguientes fragmentos de código comparan los datos *DatoA* y *DatoB* y ejecutan el fragmento de código etiquetado con *Caso1* en función de la relación entre dichos datos: que sean iguales, que *DatoA* sea mayor, que *DatoA* sea menor, etc.

Analiza los cuatro fragmentos e indica en cada caso cuándo se ejecutarían las instrucciones etiquetadas con *Caso1* (en función de la relación entre *DatoA* y *DatoB*).

FRAGMENTO 1

```
                ...
Repetir         movf DatoB,W
                subwf DatoA,W
                btfsc STATUS,Z
                goto Caso1
                goto Repetir
Caso1   ...             ; Se ejecuta cuando.....................
```

FRAGMENTO 2

```
                ...
Repetir   movf DatoA,W
                subwf DatoB,W
                btfsc STATUS,C
                goto Caso1
                goto Repetir
Caso1   ...             ; Se ejecuta cuando.....................
```

FRAGMENTO 3

```
                ...
Repetir   movf DatoA,W
                subwf DatoB,W
                btfss STATUS,Z
                goto Caso1
                goto Repetir
Caso1   ...             ; Se ejecuta cuando.....................
```

FRAGMENTO 4

```
                ...
Repetir   movf DatoB,W
                subwf DatoA,W
                btfss STATUS,C
                goto Caso1
                goto Repetir
Caso1   ...             ; Se ejecuta cuando.....................
```

E3: Rotaciones

¿Cuántas instrucciones RRF tienes que realizar para dejar un registro en su estado original? ¿y RRF?

P4: Saltos

Escribe un programa que calcule el mayor y el menor de dos números dados. Los números estarán en las posiciones de memoria 20h y 21h; el mayor quedará almacenado en la 22h y el menor en la 23h.

Verifica su funcionamiento utilizando MPLAB X IDE.

P5: Saltos

Escribe un programa determine si un número dado es par. El resultado del programa será una variable llamada EsPar que valdrá 1 si el número es par o cero en caso contrario.

Verifica su funcionamiento utilizando MPLAB X IDE.

P6: Saltos, rotaciones, operaciones lógicas

Escribe un programa que mueva el valor de la posición 20h en 21h sin utilizar las instrucciones aritméticas ni de carga (mov*) para transferir los datos.

Existen varias posibilidades, algunas utilizan saltos y otras no. ¿Cuántas formas se te ocurren?

Verifica su funcionamiento utilizando MPLAB X IDE.