Universidad
Francisco de Vitoria
UFV Madrid

Arquitectura y Organización de

Sesión de Laboratorio 4

En esta sesión continuaremos avanzando en el uso de instrucciones de salto, en este caso para la implementación de bucles. También aprenderemos a utilizar subrutinas como método para estructurar y reutilizar el código. Introduciremos un método común para programar selecciones múltiples de tipo switch-case.

Objetivos

Los objetivos generales que se persiguen son:

- Profundizar en el uso de las instrucciones de salto del PIC16F886 para la implementación de bucles y de estructuras de selección tipo switch-case.
- Comprender la necesidad de utilizar subrutinas en el programa y aprender a crearlas y utilizarlas.
- Recordar los métodos de diseño algorítmico utilizando diagramas de flujo.

Resultados de aprendizaje

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

- Diseñar algoritmos con diagramas de flujo.
- Implementar algoritmos iterativos utilizando las instrucciones de salto ya conocidas.
- Crear nuevas subrutinas y utilizarlas adecuadamente dentro del programa.
- Crear estructuras de decisión múltiple tipo switch-case.

Teoría

Bucles

Una aplicación muy importante de los saltos condicionales son los bucles o lazos, que son fragmentos de programa que se repiten un número finito o infinito de veces.

Bucles infinitos

Es un salto incondicional a una posición anterior del programa, formando un lazo que se repite indefinidamente. Se programa utilizando únicamente la instrucción goto.

En nuestros programas, existirá un único bucle infinito, que es el que hace que se repita el programa infinitas veces una vez que se inicia su ejecución. Se implementa escribiendo como última instrucción un goto que lleve de nuevo a ejecutar la primera instrucción del programa (bueno, no siempre será la primera, dado que ciertas configuraciones iniciales no son necesario repetirlas todas las veces). En nuestro caso el bucle principal sigue el modelo "cyclic executive", que simula el funcionamiento de un sistema de tiempo real mediante multitarea cooperativa, es decir, realizando acciones una detrás de otra de forma muy rápida.

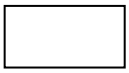
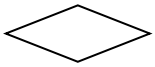
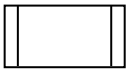
Bucles finitos

Son bucles que se repiten un número finito de veces, mientras se cumple cierta condición. Se programan utilizando la instrucción goto y las instrucciones de salto condicional btfss, btfsc, decfsz, incfsz. Con este tipo de bucles se implementan las instrucciones de alto nivel while, do-while y for.

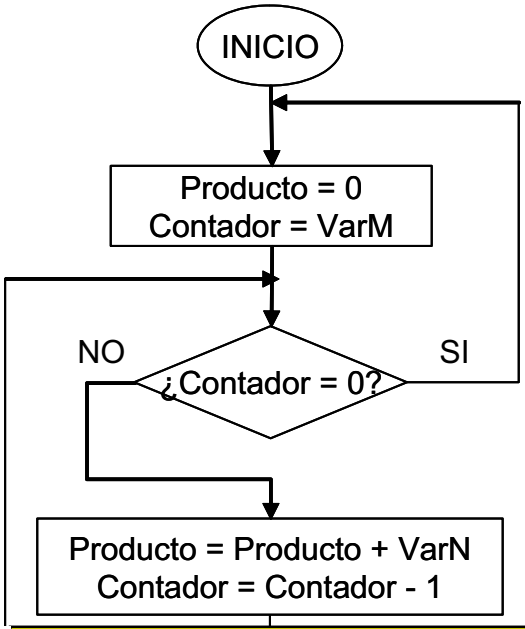
- **Bucles de tipo while**: son bucles que se repiten cero o más veces, dependiendo de si se cumple o no una determinada condición.
- **Bucles de tipo do-while**: son bucles que se repiten una o más veces, dependiendo de si se cumple o no una determinada condición.
- **Bucles de tipo for**: son bucles que se repiten un número determinado de veces, que viene dado por un contador.

Diagramas de flujo

Los símbolos más comunes utilizados en los grafos son:

	Representa el INICIO o FIN del algoritmo
	Representa una acción o conjunto de acciones básicas
	Representa una toma de decisión que da lugar a varios caminos alternativos de ejecución (varios flujos de control)
	Representa una llamada a una subrutina
	Representa una acción de entrada/salida de datos

EJEMPLO 1. A continuación se muestra el diagrama de flujo de un algoritmo que es capaz de realizar la multiplicación de dos números naturales mediante sumas. El algoritmo se basa en que para multiplicar N * M basta con sumar N tantas veces como indique M. Parte de dos datos, VarN y VarM, que están en memoria, y guarda el resultado también en memoria. En este caso no hay, por tanto, entrada ni salida de datos desde periféricos.



```
; Programa que calcula el producto de dos números naturales
; Entradas: VarN, VarM
; Salidas: Producto

            LIST p =16F886
            INCLUDE "P16F886.INC"
            goto    Inicio
            ORG     05h

VarN        EQU     h'020'
VarM        EQU     h'021'
Producto    EQU     h'022'
Contador    EQU     h'023'

Inicio      clrf    Producto  ; pone Producto a 0
            movf    VarM,W
            movwf   Contador  ; inicializa Contador a VarM
Bucle       movf    Contador  ; mueve Contador para ver si es cero
            btfsc   STATUS,Z; comprueba que no sea 0
Fin         goto    Inicio    ; fin del bucle y del programa, empezar
Cuerpo      movf    VarN,W
            Addwf   Producto  ; suma VarN a Producto
            decf    Contador  ; decrementa Contador
            goto    Bucle     ; repite el bucle
            END
```

Subrutinas

Las macros y subrutinas son dos mecanismos del lenguaje ensamblador que permiten agrupar una colección de instrucciones bajo un nombre común. Mediante el uso de macros y subrutinas se simplifica la labor de edición de código, dado que se evita tener que repetir varias veces un mismo fragmento, y se facilita la comprensión y depuración del programa.

Aunque en este sentido son parecidas, existe una diferencia fundamental entre ambas y reside en lo que ocurre con ellas durante el proceso de ensamblado y posterior ejecución del programa.

Creación y uso de subrutinas

Una subrutina queda fuera del flujo secuencial del programa principal, luego son necesarios mecanismos para poder llegar a ella (es decir, a la zona de memoria donde se encuentra almacenada) y una vez ejecutadas las instrucciones que la componen, volver al punto donde se quedó la ejecución del programa.

La acción de pasar del programa principal a la subrutina se denomina "***llamada a la subrutina***", y se realiza mediante la instrucción call, que lleva como argumento la dirección de comienzo de la subrutina, es decir, la dirección de memoria donde se encuentra almacenada la primera instrucción de dicha subrutina.

La acción de regresar al programa principal cuando la subrutina termina se denomina "***retorno de la subrutina***" y se realiza con la instrucción return. Esta instrucción, a diferencia de call, no lleva argumento, dado que la dirección a la que se retorna después de ejecutar una subrutina no es fija: depende de dónde se realizara la llamada a dicha subrutina. La instrucción que se ejecuta después de un return es la instrucción que aparece en el programa principal después de la llamada a la subrutina. La dirección de esta instrucción se conoce en el momento de ejecutar el return porque, al ejecutar el correspondiente call, se almacenó la dirección de retorno en la pila del sistema.

También puede realizarse el retorno de una subrutina mediante la instrucción retlw, que, además de volver al programa que la llamó, coloca el valor que se le pasa como argumento en el registro W.

Ejemplo 2. A continuación se muestra cómo crear una subrutina que multiplique dos números en memoria y cómo elaborar un programa que la utilice para calcular la multiplicación de tres datos: d = a*b*c.

```
; Subrutina que calcula el producto de dos números naturales
; Entradas: VarN, VarM
; Salidas: Producto
; Variables locales: Contador
; Nota: todas las etiquetas deben ser definidas en el programa principal,
; para evitar utilizar posiciones de memoria ya utilizadas para otros datos

Multiplicar:    clrf    Producto  ; pone Producto a 0
                movf    VarM,W
                movwf   Contador ; inicializa Contador a VarM
Bucle           movf    Contador ; mueve Contador para ver si es cero
                btfsc   STATUS,Z; comprueba que no sea 0
Fin             return           ; fin del bucle y del programa, empezar
```

```
; Entradas: VarA, VarB, VarC
; Salidas: VarD

            LIST p =16F886
            INCLUDE "P16F886.INC"
            goto    Inicio
            ORG     05h

; etiquetas de la subrutina
VarN        EQU     20h
VarM        EQU     21h
Producto    EQU     22h
Contador    EQU     23h

; etiquetas para los datos del programa principal
VarA        EQU     24h
VarB        EQU     25h
VarC        EQU     26h
VarD        EQU     27h


Inicio      movf    VarA,W
            movwf   VarN
            movf    VarB,W
            movwf   VarM
            call    Multiplicar     ; multiplica VarA*VarB
            movf    Producto,W
            movwf   VarN
            movf    VarC,W
            movwf   VarM
            call    Multiplicar     ; multiplica VarA*VarB*VarC
            movf    Producto,W
            movwf   VarD
            goto    Inicio

; aquí se incluiría el código de la subrutina antes creada, o bien puede
; estar en un archivo independiente, con extensión .inc, que se incluya
; aquí mediante la directiva INCLUDE
            END
```

En este programa se llama dos veces a la subrutina Multiplicar. La primera llamada está ubicada en la posición 09h, por lo que la instrucción return volverá a la instrucción en la posición 0Ah (está dirección será almacenada en la pila en el momento de ejecutar la instrucción call, para que esté disponible para el return). De esta manera, tras la ejecución de la subrutina, el programa principal continua adecuadamente con

Las subrutinas permanecen tras el proceso de ensamblado, y se ubican en memoria en el lugar y orden en que aparecen en el código fuente.

Subrutinas anidadas

Cuando una subrutina llama a otra subrutina se produce una situación conocida como anidamiento de subrutinas. Cada call sucesivo sin que intervenga un return crea un nivel de anidamiento adicional y, por tanto, una nueva entrada en la pila.

El nivel de anidamiento está limitado en cada microcontrolador, y en el caso del PIC16F886 es de 8: es decir, no pueden producirse más de 8 llamadas anidadas.

La pila (stack)

Es una zona de memoria que se encuentra separada tanto de la memoria de datos como de la de programa. Está formada por 8 registros y tiene una estructura de tipo LIFO (*Last In, First Out*). El acceso a esta memoria se realiza únicamente mediante las instrucciones call (introduce un nuevo valor en la cima de la pila) y return (extrae el valor en la cima de la pila). Esta es una particularidad de la arquitectura PIC16. En la mayoría de los procesadores y microcontroladores (incluidos los PIC18 y superiores), la pila se ubica en memoria y es accesible como cualquier otro dato.

La selección múltiple (switch – case)

Hasta ahora hemos aprendido a hacer estructuras del tipo if-then-else, lo que nos permitiría realizar una estructura tipo switch mediante anidación. Sin embargo esta aproximación es poco legible y muy susceptible de equívocos. Además es una solución bastante lenta en general y, a medida que una opción está más abajo en la lista de comprobaciones, tarda más en ejecutarse.

En el caso de que tengamos una selección múltiple sobre un rango de datos pequeño, lo que es habitual, podemos crear lo que se conoce como tabla de salto. Esté método, aplicado al PIC16, consiste en modificar el contador de programa añadiendo el valor de la selección. Justo a continuación de la instrucción addwf PCL, f se colocan una serie de instrucciones goto, ordenadas, a cada una de las opciones de selección múltiple.

Las tablas de salto se utilizan con frecuencia para optimizar el tiempo de respuesta de un sistema. Los compiladores suelen utilizarlas para las instrucciones switch cuando el rango es pequeño, ya que a medida que el rango crece, la tabla debe hacerse más grande con el consiguiente consumo de memoria de programa.

El siguiente fragmento de código es un ejemplo de este método

```
movf   opcion, w      ; carga el valor de la variable opcion en w
andwl  0x07           ; Opcionalmente forzamos el rango de opcion entre 0 y 7
addwf  PCL, f         ; Suma ese valor al contenido de PCL
goto   mirutinaop0    ; cuando opcion=0, PCL  apunta a esta instrucción
goto   mirutinaop1    ; cuando opcion=1, PCL apunta a esta instrucción
goto   mirutinaop2    ; cuando opcion=2, PCL apunta a esta instrucción
goto   mirutinaop3    ; cuando opcion=3, PCL apunta a esta instrucción
goto   mirutinaop4    ; cuando opcion=4, PCL apunta a esta instrucción
goto   mirutinaop5    ; cuando opcion=5, PCL apunta a esta instrucción
goto   mirutinaop6    ; cuando opcion=6, PCL apunta a esta instrucción
goto   mirutinaop7    ; cuando opcion=7, PCL apunta a esta instrucción
…                     ; Al limitar el valor de opcion con una máscara y andwf
…                     ;  nunca podremos pasar salirnos de las siete instrucciones
…                     ;  anteriores
```

Como el contador de programa es de 13 bits, los 5 bits superiores se cargan del registro PCLATH en la misma operación en la que se escriba el registro PCL. Hay que tener esto en cuenta si nuestra tabla de saltos va a cruzar una página de código de 256 bytes o si se encuentra más allá de los primeros 256 bytes de código. Para más información, revisa el apartado 2.3 del Data Sheet.

Esta técnica puede adaptarse a las necesidades particulares de cada programa, por ejemplo haciendo que la tabla de saltos sea más grande para que pueda contener subprogramas (de tamaño fijo) por cada opción.

Ejercicio de clase:

Conectando el puerto C a los leds de la placa, y utilizando el generador lógico como reloj, muestra la representación binaria de todos los posibles números en un byte (0-255) sobre los leds.

PASOS:

1) Configura la placa conectando 8 cables entre el puerto C y el array de leds.
2) Conecta la salida del generador lógico al puerto A0
3) Inicializa el código con las opciones HW del PIC y con la rutina para habilitar los puertos GPIO.
4) Configura en tu código el puerto C como salida y el puerto A0 como entrada

```
confports:    bsf STATUS, RP0  ; cambiar al banco donde están TRISC y TRISA
```

```
mostrarc:   movwf PORTC     ; portc está en el banco 0, no hace falta cambiar
            return
```

6) Haz una subrutina que espere el flanco de bajada del puerto A0, es decir, la transición entre el estado 1 y el estado 0.

```
esperafba0: btfss PORTA, 0 ; esperamos mientras que sea 0
            goto esperafba0
espera2:    btfsc PORTA, 0 ; esperamos mientras que sea 1
            goto espera2
```

7) Crea un cuerpo principal de programa que sea un bucle "For" en el cual se incluye la llamada (**call**) a mostrarc y otra a esperafba0.
   a. Asegúrate de que w contiene el valor del índice del "For" antes de llamar a "mostrarc"

Cuestiones y ejercicios

Diseño de algoritmos iterativos mediante diagramas de flujo

D1. Analiza el algoritmo del ejemplo 1. ¿Cuántas veces se ejecuta el bucle si VarN = 0 y VarM = 0? ¿Y si VarN = 0 y VarM = 10? Modifica el el diagrama de flujo para que si cualquiera de los dos valores es 0, devuelva 0 directamente como resultado, sin entrar en el bucle.

D2. Diseña, mediante un diagrama de flujo, un algoritmo que determine si un número es divisible entre otro, ambos positivos. En ese caso, devolverá un 1 como resultado. En caso contrario, un 0.

D3. Diseña, mediante un diagrama de flujo, un algoritmo que calcule el cociente y el resto que se obtiene al dividir un número natural entre otro. El algoritmo tendrá como resultado, además, un código de error, que valdrá 1 si se intenta dividir entre cero, y 0 en caso contrario.

Codificación de algoritmos iterativos

P1. Codifica como una subrutina en ensamblador del PIC16F886 el algoritmo diseñado en el ejercicio D2. Crea un programa principal que utilice esa subrutina y verifica su funcionamiento utilizando el simulador de MPLAB X IDE.

P2. Codifica como una subrutina en ensamblador del PIC16F886 el algoritmo diseñado

P3. Codifica una estructura de selección múltiple de nueve opciones y comprueba su funcionamiento en el simulador de MPLAB X IDE. Haz dos versiones:

- Con anidación de estructuras if-then-else
- Con tablas de salto