EC2112 Sistemas Electrónicos Industriales II

Prof. Manuel Rivas

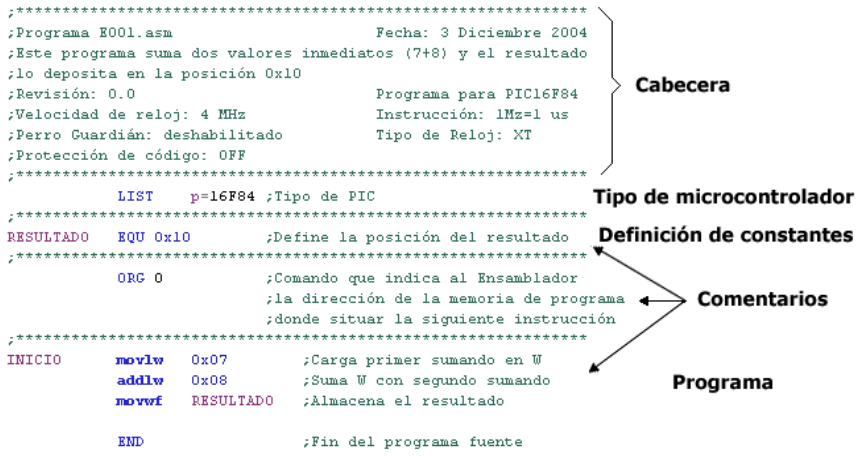
MICROCONTROLADORES (III)

Agenda

- Pautas generales de programación
- Operaciones con datos
- Operaciones aritméticas
- Operaciones de comparación
- Subrutinas
- Selección múltiple
- Configuración de puertos

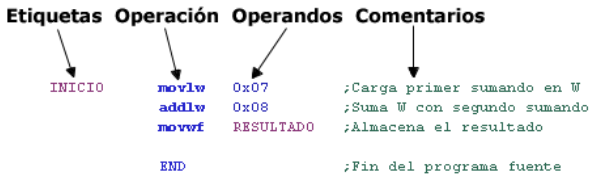
Pautas generales

- La estructura de un programa es similar a esta:



Pautas generales

- La organización de las instrucciones es así:



- 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

Pautas generales

- Columna 2: Operaciones. 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. Normalmente contendrá registros o literales con los que se operará

- Columna 4: Comentario. El campo del comentario o de etiquetas es opcional. 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

Pautas generales

- **Directiva EQU:** 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. Permite 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 nombres que asignamos mediante esta directiva se les denomina constantes, ya que el registro al que apuntan no variará durante el programa

```
DATO        EQU   22
PUERT0_A    EQU   5
INICIO      EQU   0
ACARREO     EQU   3
TIEMPO      EQU   5
BANCO_1     EQU   BSF STATUS,RP0
```

Pautas generales

- **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.

- 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 utiliza para: reset, programas de servicios de interrupción, programa principal, y subrutinas.

```
ORG     0x00     ;inicia el programa en la posición cero
```


Pautas generales

- **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. Esta línea debe colocarse al principio, y tiene la siguiente sintaxis:

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

- **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
```

Pautas generales

- **Directiva END:** 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:** 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.

```
#DEFINE CERO STATUS,2        ; 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.
```


Pautas generales

- **Directiva TITLE:** Es 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 crea 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.

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

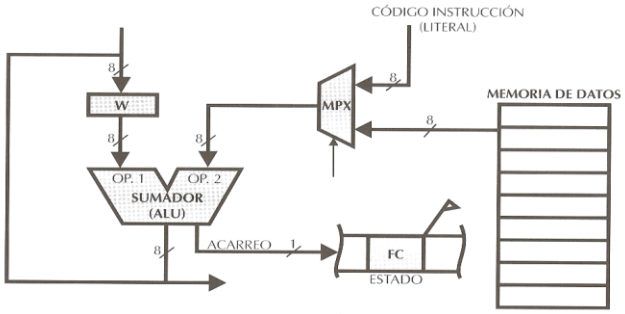
Operaciones con datos

- Programa que coloca un 0 en las posiciones de memoria 20 y 21 (en hexadecimal) usando un direccionamiento indirecto:

```
MOVLW 20h      ;carga en el registro w, una constante igual a 20h
MOVWF FSR      ;Copia el contenido de w a la posición de
               ;memoria f. En este caso, FSR representa una
               ;dirección de memoria. FSR es el registro usado
               ;para direccionamiento indirecto. En FSR se carga
               ;la dirección de memoria que se va a direccionar
               ;con INDF (otro registro)
CLRF INDF      ;coloca en 0 el registro en la posición de memoria
               ; f.
INCF FSR       ;incrementa en uno el valor almacenado en la
               ;posición de memoria f
CLRF INDF      ;coloca en 0 el registro en la posición de memoria
               ;f
```


Operaciones aritméticas

- El PIC puede sumar dos datos de 8 bits cada uno. Uno de los sumandos se puede alojar en el registro w y el otro sumando puede ser un literal o tomado de una posición de memoria. El resultado de la suma puede quedar en el registro w o ser guardado en una posición de memoria.



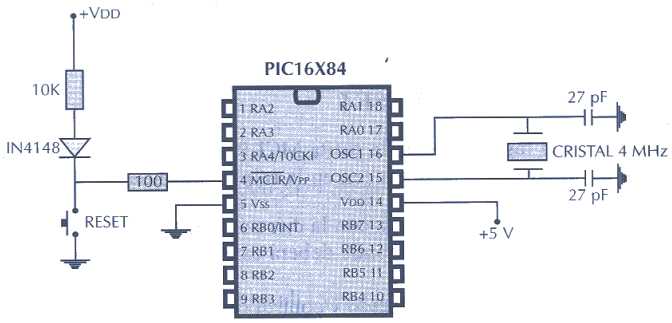
Ejemplo 1

- Programa que utiliza tres posiciones de la memorias de datos para sumar dos números y guardar el resultado.

```
LIST P=16F84              ;Indica el tipo de PIC usado
OPERANDO1 EQU 0x0c        ;Define la posición del operando 1
OPERANDO2 EQU 0x0d        ;Define la posición del operando 2
RESULTADO EQU 0x0e        ;Define la posición del resultado
ORG 0                     ;Define la posición de inicio
MOVLW  05                 ;Guarda un 5 en w
MOVWF  OPERANDO1          ;Guarda w en la posición de
                          ;operando 1
MOVLW  02                 ;Guarda un 2 en w
MOVWF  OPERANDO2          ;Guarda w en la posición de
                          ;operando 2
MOVFW  OPERANDO1          ;Guarda el operando 1 en w
ADDWF  OPERANDO2,0        ;Sumo w con operando 2 y guardo
                          ;en w
MOVWF  RESULTADO          ;Guardo w en la posición de
                          ;resultado
END                       ;Finaliza el programa
```


Ejemplo 1

- El circuito necesario para implementar esta aplicación es el siguiente:



Operaciones de comparación

- Las operaciones de comparación utilizan la instrucción de resta. La resta no es mas que sumar al minuendo el complemento a 2 del sustraendo

- **Igualdad:** Supongamos que estamos intentando determinar si un número es igual a 2:

```
MOVLW .2              ;coloco un 2 en el registro W
SUBWF N, W            ;al número N le resto lo que está en W
BTFSS STATUS, Z       ;verifico el estado de la bandera Z
GOTO NO_ES_IGUAL      ;si Z no es cero, salto al sitio
                      ;correspondiente
GOTO ES_IGUAL         ;si Z es cero, salto al sitio
                      ;correspondiente
```


Operaciones de comparación

- **Mayor que y menor que:** Supongamos que estamos intentando determinar si un número es mayor o menor de 2.

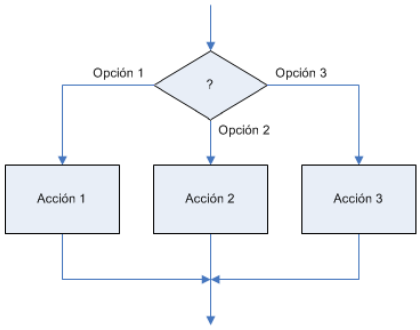
```
MOVLW .2              ;coloco un 2 en registro w
SUBWF N, W            ;resto número N con el contenido de w (2)
BTFSS STATUS, C       ;compruebo si se produjo un acarreo
GOTO MENOR            ;si N es menor que 2 salto a MENOR
GOTO MAYOR_IGUAL      ;si N es mayor o igual a 2 salto a
MAYOR_IGUAL
```

Subrutinas

- La instrucción **CALL** (llamada a subrutina) permite que la ejecución del programa continúe en la dirección donde se encuentra la subrutina a la que hace referencia. Es similar a **GOTO** pero coloca en la pila la dirección de la siguiente instrucción que se debe ejecutar después de terminar con la subrutina.
- La subrutina finaliza con la instrucción **RETURN** (retorno de la subrutina) que retoma la dirección guardada en la pila y la coloca en el contador de programa **PC** continuando el flujo de control con la instrucción que sigue a **CALL** .
- La pila tiene ocho niveles de memoria del tipo LIFO. Si se produce la llamada a una subrutina durante la ejecución de otra subrutina, la dirección de retorno de esta segunda es colocada en la cima de la pila sobre la dirección anterior. Esta segunda dirección es la primera en salir de la pila mediante la instrucción **RETURN** .
- Con la pila de ocho niveles, una subrutina puede llamar a otra y ésta, a su vez, llamar a otra hasta un máximo de ocho.

Selección múltiple

- Cuando se tiene que solucionar un diagrama de flujo como el de la siguiente figura, en el cual tenemos tres posibles respuestas a una pregunta, se plantean las soluciones aquí presentadas.



Selección múltiple

- **Solución:** comparamos uno por uno los valores de las diferentes opciones almacenadas en memoria RAM en una variable llamada OPCION

```
MOVLW OPCION1      ;
XORWF OPCION,0     ; verificación de OPCION respecto a W
BTFSC STATUS,Z     ; verificando la bandera Z
GOTO ACCION1       ;
MOVLW OPCION2      ;
XORWF OPCION,0     ; verificación de OPCION respecto a W
BTFSC STATUS,Z     ; verificando la bandera Z
GOTO ACCION2       ;
MOVLW OPCION3      ;
XORWF OPCION,0     ; verificación de OPCION respecto a W
BTFSC STATUS,Z     ; verificando la bandera Z
GOTO ACCION3       ;
```


Selección múltiple

```
ACCION1      .....      ;instrucciones de ACCION1
GOTO   UNION
ACCION2      .....      ;instrucciones de ACCION2
GOTO   UNION
ACCION3      .....      ;instrucciones de ACCION3
UNION        .....      ;continuación del programa
```

Instrucciones y puertos

- El PIC16F84 tiene 13 terminales que pueden ser configurados individualmente como entrada o como salida. Están divididos en dos puertos de 8 terminales y otro de 5, puerto B y puerto A, respectivamente.
- La dirección de cada bit está determinada por los bits de los registros **TRISA** y **TRISB** del banco de memoria 1. Un cero en un bit significa que es una salida, mientras que un uno significa que queda configurado como una entrada.
- NOTA: Los 4 bits más altos del puerto B pueden ser utilizados como interrupciones cuando son programados como entradas. El bit 0 del puerto B también puede ser usado como fuente de interrupción externa. El bit más alto del puerto A puede utilizarse también como entrada externa de reloj para el contador-temporizador

Configuración de los puertos

- Ejemplo de cómo configurar el puerto B alternando entradas y salidas:

```
BSF STATUS,RP0        ;activa el banco de memoria 1.
MOVLW 0xAA            ;w= 10101010
MOVWF TRISB           ;w es copiado en el registro
TRISB BCF STATUS,RP0  ;activa el banco de memoria 0.
```

- Ejemplo de cómo escribir en los puertos:

```
MOVLW 30h
MOVWF TRISB     o

MOVLW B'00110000'
MOVWF TRISB
```

Configuración de los puertos

- Al igual que hemos escrito en los puertos a través de dos métodos distintos, igualmente podemos leer de ellos usando ambas metodologías: la primera consiste en realizar el proceso inverso, colocando el valor del puerto a **w** y de ahí a donde se desee hacer uso de ese valor. La segunda se basa en las instrucciones que preguntan sobre el estado de un bit, esto es, **BTFSS** y **BTFSC** . Y dependiendo del bit, una realiza un salto si está a 0 y la otra si está a 1.

```
PRUEBA   BTFSS PORTA, 0 ;comprueba el bit 0 del puerto A
         GOTO PRUEBA ;si no cambia, vuelve a comprobarlo
         GOTO OTRO   ; si cambia, sale del bucle y va a otro lado
OTRO     .....       ;otras instrucciones
```

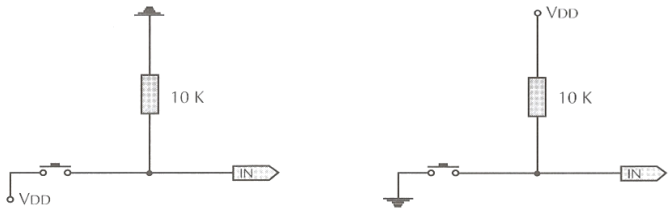

Configuración de los puertos

```
PRUEBA BTFSS PORTA, 0      ;comprueba el bit 0 del puerto A
GOTO PRUEBA                ;si no cambia, vuelve a comprobarlo
GOTO OTRO                  ;si cambia, sale del bucle y va a otro lado
OTRO .....          ;otras instrucciones
```

- Esto está bien en el caso de que lo apliquemos a entradas basadas en circuitería lógica, o que cambian de estado una vez cada mucho tiempo. Si quisiésemos aplicarlo, por ejemplo, a una entrada a la que tenemos conectado un pulsador, hemos de usar un circuito y un algoritmo anti-rebotes.

Configuración de los puertos

- Generalmente la entrada de información por los puertos del PIC se hace a través de pulsadores y resistencias de pull-up o resistencias de pull-down



- Estas resistencias permiten establecer niveles de tensión de entrada de 0 o V_{DD} voltios

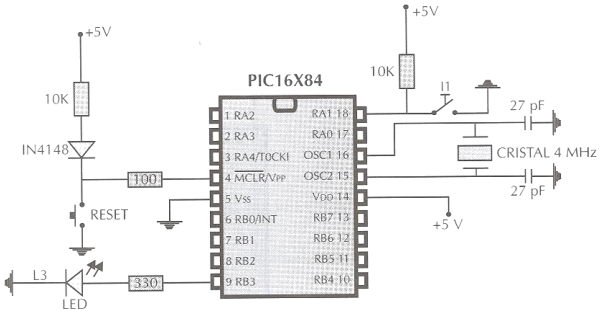
Ejemplo 2

- Programa permite poner a valor lógico uno la salida RB0 del microcontrolador cuando se detecta un valor lógico uno en la entrada RA0 mediante el uso de saltos incondicionales **GOTO**

```
SIGUE       BTFSS PORTA,0 ;lectura del pin RA0 del PORTA, si
                          ;encuentra un "1" salta una
                          ;instrucción, de lo contrario continua
            GOTO APAGA ;llamada a la subrutina APAGA
            GOTO ENCIENDE       ;salta a la subrutina ENCIENDE
APAGA       BCF PORTB,0   ;pone en "0" el pin RB0 del PORTB
            GOTO SIGUE    ;vuelve al testeo
ENCIENDE    BSF PORTB,0   ; pone en "1" el pin RB0 del PORTB
            GOTO SIGUE    ;todo de nuevo
```

Ejemplo 2

- El circuito necesario para implementar esta aplicación es el siguiente:



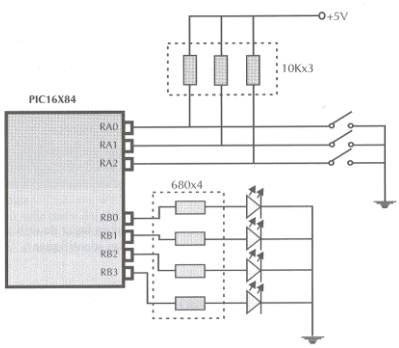
Ejemplo 3

- Se desea ingresar al PIC un número binario de 3 bits, sumarle 2 y mostrar el resultado de la operación en binario:

```
        LIST P=16F84
        ORG   0
        bsf   0x03,5
        movlw 0xff
        movwf 0x05
        movlw 0x00
        bcf   0x03,5
inicio  movf  0x05,0
        addlw 2
        movwf 0x06
        goto  inicio
        END
```

Ejemplo 3

- El circuito necesario para implementar esta aplicación es el siguiente:



Bibliografía

- Microcontroladores PIC: diseño practico de aplicaciones. José María Angulo. Editorial McGraw Hill
- Microchip 16F84 Datasheet