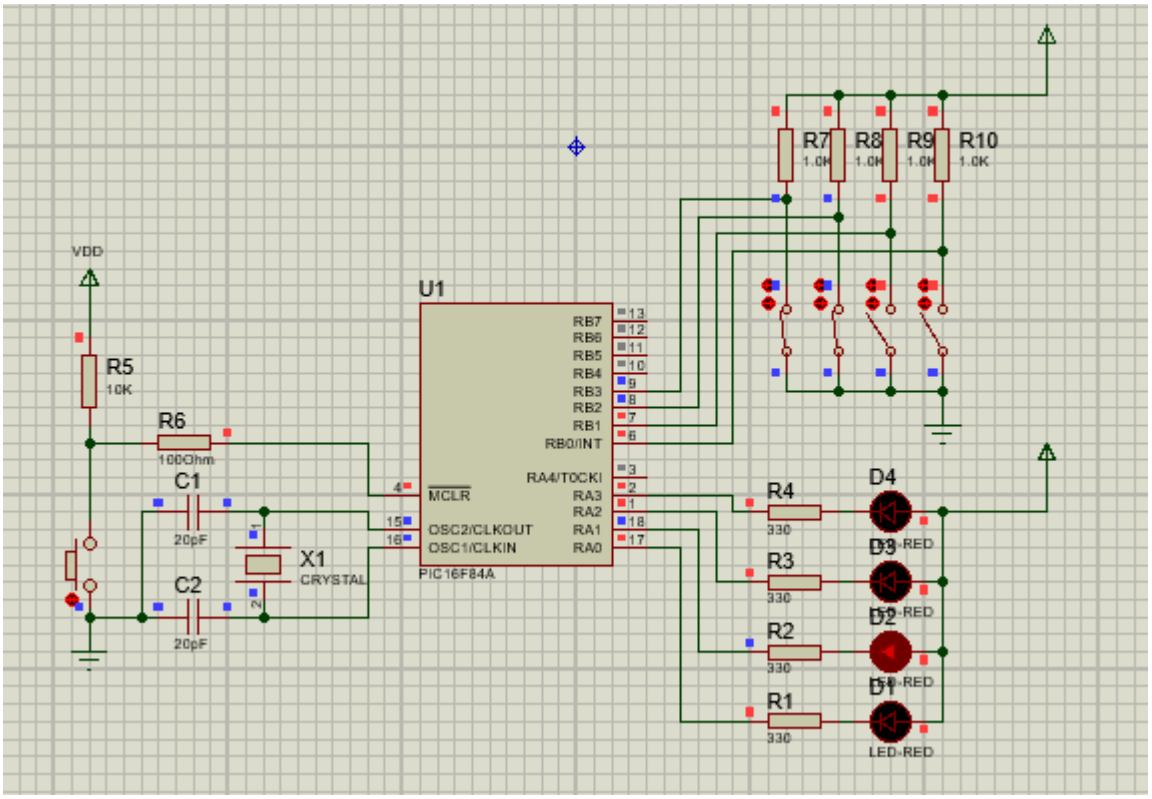
TP #6 -Practica con P16F84A-

Objetivos:

1. Familiarizarse con la programación en ensamblador en un microcontrolador.
2. Configurar y utilizar los registros especiales del PIC16F84A.
3. Implementar y probar un programa sencillo que controle LEDs con interruptores.
4. Comprender el funcionamiento básico de un microcontrolador.

EJEMPLO 1



```
46
47  ;=====================================================================
48  ; CODE SEGMENT
49  ;=====================================================================
50
51  org 5           ; El programa inicia en la dirección de memoria 5
52  inicio          ; Etiqueta para el comienzo del programa principal
53
54  bsf STATUS, 5   ; Banco 1 de registros (para configurar puertos)
55  movlw 0f0h      ; Configura Puerto A como salida (0=salida, 1=entrada)
56  movwf TRISA
57  movlw 0ffh      ; Configura Puerto B como entrada
58  movwf TRISB
59  bcf STATUS, 5   ; Banco 0 de registros (para trabajar con los puertos)
60
61  ciclo           ; Etiqueta para el bucle principal del programa
62
63  btfss PORTB, 0  ; ¿Está el Interruptor 1 (RB0) presionado (1)?
64  bcf PORTA, 0    ; Si no, apaga el LED 1 (RA0)
65  btfsc PORTB, 0  ; ¿Está el Interruptor 1 (RB0) sin presionar (0)?
66  bsf PORTA, 0    ; Si sí, enciende el LED 1 (RA0)
67
68  goto ciclo       ; Salta de nuevo al inicio del bucle para verificar continuamente
69
70  END             ; Fin del programa
71
```

1. Inicio del programa:

- `org 0x05`: Indica que el programa comenzará a cargarse en la memoria del microcontrolador a partir de la dirección 0x05. En el PIC16F84A, las primeras posiciones de memoria están reservadas para vectores de interrupción y configuraciones especiales.
- `inicio`: Es una etiqueta que marca el punto de inicio del programa principal.

2. Puertos:

- `bsf STATUS, 5`: Esta instrucción cambia al Banco 1 de registros del PIC16F84A. Los PIC tienen una arquitectura de bancos de memoria, y algunos registros solo son accesibles en bancos específicos. En este caso, necesitamos acceder a los registros `TRISA` y `TRISB`, que controlan la dirección (entrada/salida) de los pines de los puertos A y B, y estos registros están ubicados en el Banco 1.
- `movlw 0f0h`: Carga el valor hexadecimal 0f0h (11110000 en binario) en el registro de trabajo `W`. Este valor se utilizará para configurar los pines del Puerto A. En este caso, queremos que el pin RA0 sea una salida (para controlar el LED), y los demás pines del Puerto A sean entradas (no se usan en este ejemplo).
- `movwf TRISA`: Transfiere el valor del registro `W` (0f0h) al registro `TRISA`. Esto configura el pin RA0 como salida y los demás pines del Puerto A como entradas.
- `movlw 0ffh`: Carga el valor hexadecimal 0ffh (11111111 en binario) en el registro `W`. Este valor se utilizará para configurar los pines del Puerto B como entradas (para leer el estado del interruptor).
- `movwf TRISB`: Transfiere el valor del registro `W` (0ffh) al registro `TRISB`. Esto configura todos los pines del Puerto B como entradas.
- `bcf STATUS, 5`: Cambia de nuevo al Banco 0 de registros, donde se puede trabajar con los valores de los puertos (`PORTA` y `PORTB`).

3. Bucle Principal (ciclo):

- `btfss PORTB, 0`: Esta instrucción ("Bit Test File, Skip if Set") verifica si el bit 0 del registro `PORTB` (que corresponde al pin RB0, donde está conectado el interruptor) está en estado alto (1). Si el bit está en alto (interruptor presionado), la siguiente instrucción se salta.
- `bcf PORTA, 0`: Si el bit RB0 no está en alto (interruptor no presionado), esta instrucción ("Bit Clear File") pone a 0 el bit 0 del registro `PORTA` (que corresponde al pin RA0, donde está conectado el LED), apagando el LED.
- `btfsc PORTB, 0`: Esta instrucción ("Bit Test File, Skip if Clear") verifica si el bit 0 del registro `PORTB` está en estado bajo (0). Si el bit está en bajo (interruptor no presionado), la siguiente instrucción se salta.
- `bsf PORTA, 0`: Si el bit RB0 está en alto (interruptor presionado), esta instrucción ("Bit Set File") pone a 1 el bit 0 del registro `PORTA`, encendiendo el LED.
- `goto ciclo`: Esta instrucción crea un bucle infinito. El programa salta de nuevo a la etiqueta `ciclo` para repetir el proceso de verificación del interruptor y control del LED.

4. Fin del Programa:

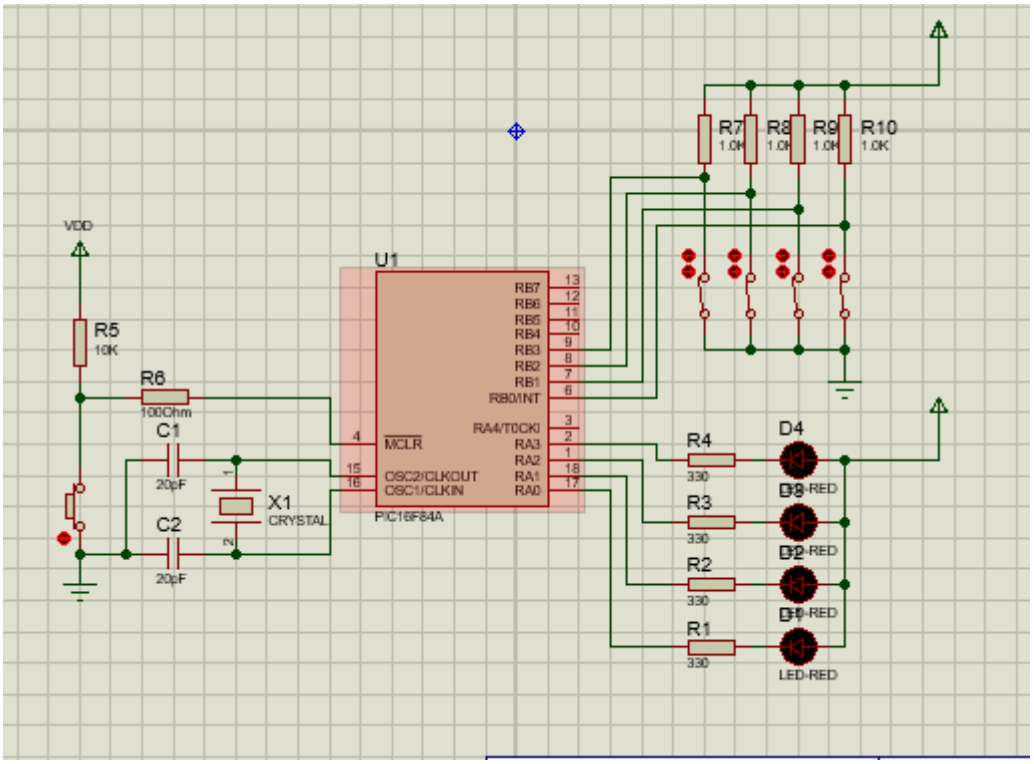
- `END`: Marca el final del programa en ensamblador.

Funcionamiento General:

Este programa crea un bucle infinito que hace lo siguiente:

1. **Lee el estado del interruptor:** Verifica si el pin RB0 (conectado al interruptor) está en alto (1) o bajo (0).
2. **Controla el LED:**
    - Si el interruptor está presionado (RB0 = 1), enciende el LED conectado a RA0.
    - Si el interruptor no está presionado (RB0 = 0), apaga el LED.
3. **Repite:** Vuelve al paso 1 para verificar nuevamente el estado del interruptor y actualizar el LED en consecuencia.

EJEMPLO 2




```
;***********************************************************************
; CODE SEGMENT
;***********************************************************************

    list p=16f84a  ;Directiva que indica el microcontrolador a usar
    include <p16f84a.inc>   ;Incluir archivo de cabecera

    org 0x05       ; El programa inicia en la dirección de memoria 5

inicio          ; Etiqueta para el comienzo del programa principal
    bsf STATUS,RP0 ; Cambio al banco 1
    movlw b'01110000'       ; Configura Puerto A como salida (0=salida, 1=entrada)
    movwf TRISA
    movlw b'11111111'      ; Configura Puerto B como entrada
    movwf TRISB
    bcf STATUS,RP0 ; Cambio al banco 0

ciclo           ; Etiqueta para el bucle principal del programa

    ; Verificar Interruptores 1 y 2 para LED1
    movf PORTB, W    ; Lee el valor del Puerto B y lo guarda en W
    andlw 0x03       ; Máscara para aislar los bits 0 y 1 (interruptores 1 y 2)
    xorlw 0x03       ; XOR para verificar si ambos bits están en 1
    btfsc STATUS,Z ; Si Z=1 (resultado XOR es 0), ambos interruptores presionados
    bsf PORTA, 0     ; Enciende LED1
    btfss STATUS,Z ; Si Z=0 (resultado XOR no es 0), al menos un interruptor no presionado
    bcf PORTA, 0     ; Apaga LED1

    ; Verificar Interruptores 3 y 4 para LED2
    movf PORTB, W    ; Lee el valor del Puerto B y lo guarda en W
    andlw 0x0C       ; Máscara para aislar los bits 2 y 3 (interruptores 3 y 4)
    xorlw 0x0C       ; XOR para verificar si ambos bits están en 1
    btfsc STATUS,Z ; Si Z=1 (resultado XOR es 0), ambos interruptores presionados
    bsf PORTA, 1     ; Enciende LED2
    btfss STATUS,Z ; Si Z=0 (resultado XOR no es 0), al menos un interruptor no presionado
    bcf PORTA, 1     ; Apaga LED2

goto ciclo       ; Salta de nuevo al inicio del bucle para verificar continuamente

END             ; Fin del programa
```

EXPLICACIÓN DEL CÓDIGO FUENTE PASO A PASO

1. **Inicio del programa:**

- `list p=16f84a`: Esta directiva le indica al ensamblador que estamos trabajando con el microcontrolador PIC16F84A. Esto es importante porque cada microcontrolador tiene un conjunto específico de registros y características.
- `include <p16f84a.inc>`: Esta directiva incluye un archivo de encabezado específico para el PIC16F84A. Este archivo define nombres simbólicos para los registros y bits especiales del microcontrolador, lo que hace que el código sea más legible y fácil de mantener. Por ejemplo, en lugar de usar números hexadecimales para los registros, podemos usar nombres como `TRISA`, `PORTB`, `STATUS`, etc.

2. Puertos:

- `org 0x05`: Indica que el programa se cargará en la memoria de programa a partir de la dirección 0x05.
- `inicio`: Etiqueta que marca el comienzo del programa principal.
- `bsf STATUS,RP0`: Cambia al banco 1 de registros. Los microcontroladores PIC tienen múltiples bancos de registros para acceder a todos sus registros internos. Algunos registros, como `TRISA` y `TRISB`, solo se pueden configurar en el banco 1.
- `movlw b'01110000'`: Carga el valor binario 01110000 en el registro de trabajo `W`. Este valor configura los pines RA0 y RA1 del Puerto A como salidas (para los LEDs) y los demás pines como entradas.
- `movwf TRISA`: Transfiere el valor de `W` al registro `TRISA`, que controla la dirección (entrada/salida) de los pines del Puerto A.
- `movlw b'11111111'`: Carga el valor binario 11111111 en `W`. Este valor configura todos los pines del Puerto B como entradas (para los interruptores).
- `movwf TRISB`: Transfiere el valor de `W` al registro `TRISB`.
- `bcf STATUS,RP0`: Cambia de nuevo al banco 0 de registros, donde se puede trabajar con los valores de los puertos (`PORTA` y `PORTB`).

3. Bucle Principal (ciclo):

- `movf PORTB, W`: Lee el estado actual de los interruptores del Puerto B y lo almacena en el registro `W`.
- `andlw 0x03`: Realiza una operación AND lógica entre el valor en `W` y el valor hexadecimal 0x03 (00000011 en binario). Esto enmascara todos los bits excepto los dos bits menos significativos, que corresponden a los interruptores 1 y 2.
- `xorlw 0x03`: Realiza una operación XOR lógica entre el valor en `W` (después de la máscara) y el valor 0x03. Si ambos interruptores 1 y 2 están presionados (1), el resultado de la operación XOR será 0, lo que establecerá la bandera Zero (Z) en el registro `STATUS`.
- `btfsc STATUS, Z`: Comprueba si la bandera Z está establecida (1). Si es así, significa que ambos interruptores 1 y 2 están presionados, por lo que salta a la siguiente instrucción.
- `bsf PORTA, 0`: Si la bandera Z está establecida, esta instrucción enciende el LED1 (conectado a RA0).
- `btfss STATUS, Z`: Comprueba si la bandera Z está borrada (0). Si es así, significa que al menos uno de los interruptores 1 o 2 no está presionado, por lo que salta a la siguiente instrucción.

- `bcf PORTA, 0`: Si la bandera Z está borrada, esta instrucción apaga el LED1.
  El código luego repite el mismo proceso para los interruptores 3 y 4 y el LED2 (conectado a RA1), utilizando la máscara 0x0C (00001100 en binario).
- `goto ciclo`: Esta instrucción crea un bucle infinito. El programa salta de nuevo a la etiqueta `ciclo` para repetir el proceso de lectura de los interruptores y control de los LEDs continuamente.

4. Fin del Programa:

- `END`: Marca el final del programa en ensamblador.