

1.- INTRODUCCIÓN AL PIC16F84

En esta sección se resumen brevemente las principales características y forma de uso de las herramientas de desarrollo que serán utilizadas durante las prácticas. El contenido se presenta en forma de tutorial, que ha de realizarse en el laboratorio.

El código

El repertorio de instrucciones tipo *RISC* del micro-controlador *16F84A* carece de instrucciones de comparación de dos variables. Debido a la utilidad de este tipo de instrucciones para implementar sentencias *if-else*, desarrollaremos a continuación un programa que ilustra la forma en que pueden compararse dos números.

;Para realizar la comparación restamos los dos número (A-B). Si al realizar la resta los dos números son iguales el resultado será cero, activándose el bit Z del registro de Estado. Si al realizar la resta (suma del complemento a 2 de B) se produce un bit de acarreo el resultado es positivo (A> B), Ej.:3-2 = 0011 + 1110=1 0001).Si no se produce acarreo el resultado es negativo(A<B) Ej.: 2-3 = 0010 + 1101 = 0 1111).

```
list          p=16F84A
include      P16F84A.INC
NumA        equ      0x0C      ;Variable del número A
NumB        equ      0x0D      ;Variable del número B
Mayor       equ      0x0E      ;Variable que almacenará el mayor de los números
org         0x00      ;VectordeReset
goto        Inicio      ;Salto incondicional al principio del programa
org         0x05      ;Vector de interrupción
Inicio      movf      NumB,W    ;NumB->W (Acumulador)
            subwf     NumA,W    ;A-W->W
            btfsc    STATUS,Z  ;Bit de cero del registro de Estado a 1 0
            goto     A_igual_B  ;Si
            btfsc    STATUS,C  ;Bit de acarreo del registro de Estado a 1
            goto     A_menor_B  ;Si
A_menor_B   movf      NumB,W    ;No, A es menor que B
            movwf    Mayor      Suma A más B
            goto     Stop
A_mayor_B   movf      NumA,W    ;No, A es menor que B
            movwf    Mayor      ;Suma A más B
            goto     Fin
A_igual_B   clrf      Mayor     ;Pone a 0 el resultado
Stop        nop
            end           ;Fin del programa fuente
```

Código 1

En primer lugar con un editor de textos cualquiera, (Notepad, Wordpad, edit, ...) crear un fichero de nombre *p0.asm* (la extensión indica que es código ensamblador) y copiar el código anterior.

Una vez editado y creado el archivo *p0.asm* debemos compilarlo. Después de la compilación se habrán creado los siguientes ficheros:

Archivo	Descripción
---------	-------------

P0.err	Contiene los errores de compilación
P0.lst	Direcciones de memoria dónde se han ubicado instrucc. y tabla símbolos
P0.hex	Archivo con el código máquina

Durante la compilación del programa anterior se producirán varios errores, para solucionarlos debemos consultar el archivo *P0.err*. En este caso los errores son:

```

Make: The target "D:\PIC\Pruebas\p0.o" is out of date.
Executing: "D:\Archivos de programa\Microchip\MPASM Suite\MPASMWin.exe" /q /p16F84A "p0.asm" /l"p0.lst" /e"p0.err"
Message[301] D:\ARCHIVOS DE PROGRAMA\MICROCHIP\MPASM SUITE\16F84.INC 37 : MESSAGE:
(Processor-header file mismatch. Verify selected processor.)
Error[113] D:\PIC\PRUEBAS\16F84\16F84.ASM 19 : Symbol not previously defined (A_menor_B)
Error[122] D:\PIC\PRUEBAS\16F84\16F84.ASM 20 : Illegal opcode (mof)
Error[113] D:\PIC\PRUEBAS\16F84\16F84.ASM 25 : Symbol not previously defined (Fin)
Halting build on first failure as requested.
BUILD FAILED: Fri Feb 03 16:29:26 2006

```

En función de la lógica del programa solucionar los dos errores anteriores.

Depuración

La fase de depuración es una de las tareas de mayor importancia en el desarrollo de un programa, porque permite verificar su correcto funcionamiento, así como la corrección de errores. En el laboratorio usaremos el depurador/simulador *MPLABSIM* que simula al PIC que tengamos configurado.

1. Consultar el contenido de todos los registros, incluidos los registros de estado, pila y *W*: *Wiew->4File Registers*

Modificación de la memoria. El programa de prueba debe comparar dos números, y almacenará en la posición de memoria *Mayor* el mayor de ellos, o cero si son iguales. Resulta por lo tanto conveniente que podamos consultar y modificar el contenido de estas posiciones de memoria de forma rápida. Antes de comenzar la ejecución del programa debemos inicializar los valores de *NumA* y *NumB* con dos números. Por ejemplo inicializar *NumA* a 2 y *NumB* a 3. Para modificar el contenido hacer doble clic en el campo asociado e introducir el valor.


2. Ejecución del Programa (Puntos de Ruptura). Antes de ejecutar el programa es necesario poner un punto de ruptura en la última instrucción del programa, para poder así comprobar el resultado de la ejecución. Los puntos de ruptura se colocan haciendo doble clic sobre el número de línea. En este primer ejemplo pondremos un punto de ruptura al final del programa.

```

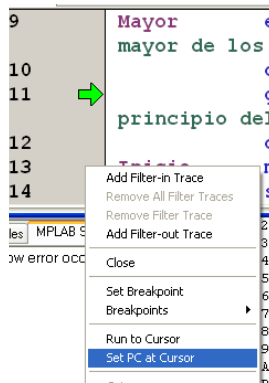
23      movwf    Mayor           ;Suma A más B
24      goto    Stop
25      clrf    Mayor
26      nop
27      end

```

The screenshot shows the assembly code editor with a red 'B' icon (breakpoint) on line 26. The code is as follows:

3. Para ejecutar el programa desde la dirección 0, pulsar 

4. Para carga en el contador de programa la dirección de memoria, y por tanto ejecutarlo desde esa instrucción se utiliza el botón derecho del cursor y la opción *Set PC at Cursor*.



Una vez finalizada la ejecución del programa podemos comprobar que el resultado almacenado en Mayor es 3. Si embargo, si ejecutamos el programa con $NumA=3$ y $NumB=2$ el resultado obtenido no es el correcto.

1. Depuración. Hemos encontrado un caso en el que nuestro programa no produce el resultado esperado ($NumA=3, NumB=2 \Rightarrow Mayor=2$). Para ver que sucede vamos a ejecutar el código paso a paso.
 - a. Punto de Ruptura en *Inicio*
 - b. Visualizar el registro de estado (*AD STATUS,B*)
 - c. Ejecutar el programa
 - d. Ejecución paso a paso

En función de lo observado en la ejecución paso a paso corregir el programa para que funciones adecuadamente.

2.1.-PUERTOS DE ENTRADA/SALIDA DEL PIC 16F84

El micro-controlador *PIC16F84* dispone de dos puertos de entrada/salida que denominaremos *PORTA* y *PORTB*. Ambos puertos se configuran por medio de los registros *TRISA* y *TRISB*, respectivamente. De forma que un 1 en un bit del registro *TRISx*, configura el correspondiente bit del registro *PORTx* como entrada. Consecuentemente, un 0 en un bit del registro *TRISx* configura correspondiente bit del registro *PORTx* como salida.

En el siguiente ejemplo se muestra como programar el puerto B del micro-controlador. Es importante recordar que los registros *PORTB* y *TRISB* se encuentran en dos páginas de memoria diferentes, por lo que es necesario fijar correctamente el bit *RP0* del registro de estado.

clrf	PORTB	;Inicializa los bits de salida
bsf	STATUS, RP0	;Selecciona la página 1
movlw	0x00	;Valor para configurar los puertos (todos como salida)
movwf	TRISB	;Fija RB<0:7> como salidas

Escribir un programa que lea la entrada del puerto A y lo refleje en el puerto B. El programa deberá dormirse cuando los interruptores de entrada tengan la configuración '11111'.

2.2.- SUBROUTINAS

La programación de subrutinas es muy sencilla en el micro-controlador 16F84, ya que dispone de una pila hardware. La estructura de una subrutina es la siguiente:

Subrutina1	instrucción	;primer instrucción de la subrutina, que se identifica
	...	;mediante la etiqueta
	...	;código de la subrutina
	return	;fin de la subrutina

Para llamar a una subrutina se utilizará la instrucción *call* de la siguiente forma:

call Subrutina1

El paso de parámetros a la subrutina se realizará mediante variables, accesibles tanto por el programa principal como por la subrutina.

Param1	equ	0x1C
	
Sub1	movf	Param1,W
		...
	return	
...		
Inicio	movlw	0xA2
	movwf	Param1
	call	Sub1
		...

2.3- MULTIPLICACIÓN DE NÚMEROS DE 8 BITS

El repertorio de instrucciones del micro-controlador 16F84 carece de Instrucciones de multiplicación. **Implementar una subrutina que multiplique dos números de 8 bits utilizando el algoritmo de sumas parciales.**

Inicialmente copiamos *NumB* en *DL*, para no modificar el operando original *NumB*. El algoritmo va realizando sumas parciales de *NumA* en un acumulador [*DH*]. En lugar de hacer desplazamientos hacia la izquierda (como multiplicamos nosotros), la parte menos significativa la va desplazando e introduciéndola en la variable *DL*. Estos desplazamiento se hacen utilizando el bit de Carry del registro de STATUS. El resultado de la multiplicación queda de la siguiente forma: el Byte más significativo en *DH*, y el menos significativo en *DL*.

```

Borrar DH
Copiar NumB en DL
for (i=0; i<8; i++)
  if (DL[0]== 1)          DH = DH + A;
  else                    borrar Carry;
  Desplazar DH hacia la derecha inyectando Carry;
  Desplazar DL hacia la derecha inyectando bit desplazado por la derecha del DH en el paso previo;
endfor
  
```

Instrucción	Descripción
clrf f	Borra el contenido de f
btfsc f,b	Comprueba el bit b de f y salta una instrucción si es cero
rrf f,d	Rota f a la derecha usando el bit de acarreo del registro STATUS
addwf f,d	Suma el contenido de W y f
decfsz f,d	Decrementa 1 y salta una instrucción si es cero.

Ejercicio 1: Diseñar un programa que multiplique dos números de 4 bits leídos desde el puerto A y muestre el resultado en el puerto B. De los cinco bits del puerto A, el más significativo indicará qué número se está introduciendo, y los otros cuatro bits indicarán el valor del número (debéis limpiar la parte alta, que se puede hacer con una AND con 0x0F).

- PORTA(4)=0=>PORTA(0..3)→NumA
- PORTA(4)=1=>PORTA(0..3)→NumB.

```

Configura puertos
Inicio      If PORTA(4)==0  PORTA(0..3)->NumA
            else          PORTA(0..3)->NumB
            Call MultiplicaA_B      ;DL(8b)=A(4b)*B(4b)
            DL->PORTB
            goto Inicio
  
```

3. EL TEMPORIZADOR TMR0

El micro-controlador PIC 16F84 dispone de un temporizador (*TMR0*), que está constituido por un contador ascendente de 8 bits conectado al reloj interno del procesador o a un reloj externo (mediante el bit 4 del puerto A). Además es posible aplicar a la señal de reloj del contador un divisor de frecuencias.

Para controlar el contador disponemos de los siguientes registros:

- Registro de Opciones (*OPTION_REG*): Los bits de este registro que afectan el funcionamiento del temporizador *TMR0* son:
 - *TOCS*: 1, si los pulsos de reloj del contador son introducidos por el bit 4 del puerto A (modo contador). 0, si se usa el reloj interno del micro controlador (modo temporizador). En esta práctica lo usaremos modo temporizador.
 - *PSA*: Determina si la división de frecuencias se aplica al *TMR0* (*PSA=0*) o al *WatchDog* (*PSA=1*)
 - *PS2,PS1,PS0*: Divisor de frecuencias de valor:

PS2	PS1	PS0	División TMR0
0	0	0	1:2
0	0	1	1:4
0	1	0	1:8
0	1	1	1:16
1	0	0	1:32
1	0	1	1:64
1	1	0	1:128
1	1	1	1:256

- Registro de Interrupciones (*INTCON*): El bit *T0IF* indica, cuando se pone a 1, que el contador se ha desbordado (Transición 0xFF => 0x00 paso por 0).

Para programar una determinada temporizador debemos seguir los siguientes pasos:

1. Configurar correctamente el temporizador modificando correspondientemente el registro *OPTION_REG*.
2. Borrar el bit de fin de cuenta del registro *INTCON*.
3. Cargar el contador con el valor adecuado. Debido a que el contador es ascendente conviene cargarlo con el complemento a dos del número de ciclos que queremos contar. Por ejemplo, si deseamos contar dos ciclos en lugar de cargarlo con 2, lo cargaríamos con -2 . Para calcular el complemento a 2 únicamente debemos restar el valor de los ciclos a 0.
4. Esperar el final de la cuenta comprobando el bit *T0IF* de *INTCON*.

El número de ciclos total que debemos cargar para producir un retardo determinado lo podemos calcular usando la expresión:

$$\text{Ciclos} = \text{Tiempo} \cdot 1\text{MHz/Escala}$$

El siguiente programa muestra como se generaría un retardo de 0.5 segundos. Para ello usaremos un divisor de frecuencias 1:256. El complemento a dos del número de ciclos, $0.5\text{sg} \cdot 10^6 / 256 = 1953 = 0x7A1$, se calcula restando a cero, esto se puede hacer 'off-line'.

Ret05	bsf	STATUS,RP0	;selección de la página 1
	movlw	b'00000111'	;inicializamos OPTION con una división
	movwf	OPTION_REG	;de 256
	bcf	INTCON,T0IF	;Borramos el bit de fin de cuenta
	bcf	STATUS,RP0	;selección de la página 0
	movlw	0x07A1	;Complemento a 2 del número de ciclos ;!!!!!!!!!!!!!!!!!!!!!!!!!!!!
	sublw	0x0000	;0x0000— 0x07A1 ;!!
	movwf	TMR0	;InicializamosTMR0
Bucle	btfs	INTCON,T0IF	;Comprobamos el final de la cuenta
	goto	Bucle	;Si no seguimos esperando

Diseñar una subrutina que genere un retardo de una centésima de segundo utilizando el Temporizador.

Retcs	...	;Utilizar el Temporizador para hacer 1cs
	return	

Diseñar una subrutina que genere un retardo de N centésimas de segundo. El número de centésimas se pasará a la subrutina mediante el registro W . Para ello construir un bucle que llame el número de veces necesario a la rutina base *Retcs*.

RetNcs	for i=1:N
	Call Retcs
	endfor
	return

Ejercicio 2: Diseñar un reloj que muestre una cuenta de segundos por el puerto B.

Configurar y borra PortB
N=100
Bucle Call RetNcs
PortB=PortB+1
Goto Bucle

4.-SISTEMA DE INTERRUPCIONES DEL PIC 16F84

El micro-controlador PIC 16F84 dispone de cuatro posibles fuentes de interrupción:

1. Activación del bit 0 de la puerta B (*RB0/INT*)
2. Desbordamiento del temporizador *TMRO*
3. Cambio de estado en alguna de las cuatro líneas más significativas del puerto B (*RB<7:4>*)
4. Finalización de una operación de escritura sobre la *EEPROM*

El registro de control de interrupciones (*INTCON*) permite discernir cuál ha sido la fuente exacta que ha provocado la Interrupción. Además este registro sirve para habilitar globalmente y localmente las interrupciones.

Tratamiento de una interrupción

Cuando en la ejecución de un programa se produce una petición de interrupción será tratada sólo si:

- la fuente en concreto que ha realizado la petición ha sido capacitada previamente
- las Interrupciones están habilitadas globalmente

Si la petición de Interrupción ha de ser tratada en el procesador se realizan los siguientes pasos:

1. El bit de capacitación global de interrupciones (*GIE*) se borra para evitar que se produzca una interrupción mientras se está tratando la interrupción en curso.
2. La dirección de retorno se guarda en la pila y el contador de programa se carga con la dirección 0x0004.
3. En la dirección 0x0004 deberá producirse un salto incondicional a la rutina de tratamiento de interrupción (*RTI*)
4. Si puede haber varias fuentes de interrupción en la *RTI* deberá comprobarse cada uno de los bits del registro *INTCON* para determinar cuál ha sido la fuente exacta de interrupción (Técnica de encuesta ó ‘polling’).
5. Antes de volver de la rutina de interrupción hay que borrar el bit asociado al periférico que ha producido la interrupción, para evitar entrar en un bucle infinito.
6. La rutina de tratamiento de interrupción ha de terminar con la instrucción *RETFIE* y no con la instrucción *RETURN*. La instrucción *RETFIE* recupera de la pila la dirección dónde había sido interrumpido el programa y la carga en el PC. Además vuelve a capacitar globalmente las interrupciones, por lo que no es necesario hacerlo manualmente.

Cuando sucede una interrupción no se guarda el contexto, por lo que es conveniente guardar el valor de los registros *W* y *STATUS*, para restaurarlos al final de la *RTI*.

RTI	movwf	W_T	;Guardar el Estado, ¡¡¡no podemos utilizar MOVF!!!
	swapt	STATUS,W	
	movwf	STATUS_T	


```

...                               ;Tratar la interrupción
swapf    STATUS_T,W ;Recuperar el Estado
movwf    STATUS
swapf    W_T,F
swapf    W_T,W
retfie

```

Cuando sea necesario producir una interrupción regularmente usando el temporizador *TMR0*, además de las consideraciones descritas en el apartado anterior hay que tener en cuenta que:

1. Hay que programar el temporizador y cargar un valor inicial para que produzca la primera interrupción, o disparar el Flag de interrupción la primera vez.
2. En la rutina de tratamiento de interrupción se debe volver a cargar el temporizador para que se produzca la siguiente cuenta y por tanto la siguiente interrupción.

Diseñar una subrutina que prepare un interrupción cada centésima de segundo utilizando el Temporizador.

```

Int1cs    ...                               ;Prepara Temporizador para hacer interrupción cada 1cs
return

```

Diseñar una rutina de tratamiento de interrupción genere gestione un retardo de *N* centésimas de segundo sin bloquear la ejecución del programa. Debe avisar al programa principal a través de una bandera *FlagNcs*, que será borrada por el programa principal.

```

RtiNcs    Call Int1cs
           contador=contador-1
           If contador=0
             contador=N
             FlagNcs=1
           EndIf
           refie

```

Ejercicio 3: Diseñar un reloj por interrupción que muestre una cuenta de segundos por el puerto B. Mientras hace la cuenta debe multiplicar PortB por PortA (este resultado no se mostrará)

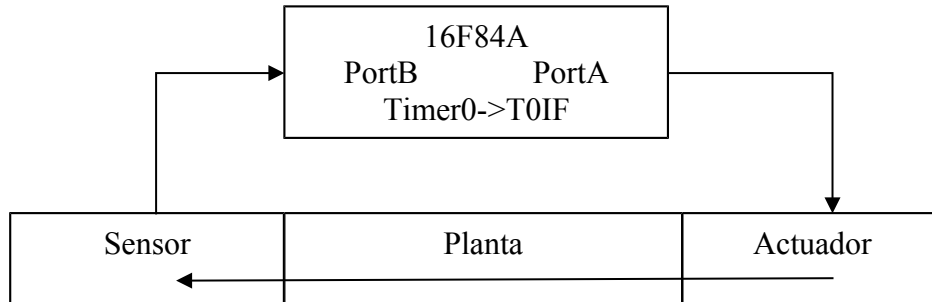
```

Configurar y borrar puertos
N=100; FlagNcs=0; contador=N;
Call Int1cs; %%%%%%%%%Por interrupción debe saltar RtiNcs
Bucle if FlagNcs
           PortB=PortB+1
           FlagNcs=0
       EndIf
       Multiplica(PortA,PortB)
Goto Bucle

```

5.-CONTROL PID CON EL PIC16F84A

Ejercicio 4: Realizar un programa que emule un Controlador Discreto PID con el PIC16F84A.



Utilizar una gestión de tiempos con interrupción por temporizador. Se supone que la referencia es un valor interno, *Referencia*. Para simplificar se supone que el actuador y el sensor manejan la información en complemento a 2.

- Cada 0.01 debe leer el sensor, *LeerPortB*.
- Cada 0.1s debe:
 - Calcular *MedidaFiltrada* como un promedio de las 8 últimas lecturas del sensor. Se puede realizar la media sumando dos elementos y un desplazamiento a la derecha y así sucesivamente hasta cerrar el árbol binario de medias, por eso he elegido 8 elementos.
 - Calcular el Control.
$$ErrorAnterior = Error$$
$$Error = Referencia - MedidaFiltrada$$
$$Integral = Integral + Error$$
$$Derivativa = Error - ErrorAnterior$$
$$Control = K_p * Error + K_i * Integral + K_d * Derivativa$$
 - Guardar telemetrías en la EEPROM.
- **Simular el sistema (en lazo abierto) inyectando estímulos en PortB que contengan ruido.**
- El PIC16F84A no es el adecuado para implementar un PID, pues además de su ALU tan limitada, no dispone ni de conversores analógico-digital ADC para manejar sensores analógicos, ni de generadores de modulación de ancho de pulsos (PWM) para manejar actuadores. Un microcontrolador muy básico que ya dispone de estos recursos es el PIC16F87. **Explicar que sería necesario modificar para adaptar este programa al PIC16F87, en caso de que tuviésemos que manejar sensores y actuadores analógicos (no es necesario reescribir el nuevo código).**