UNIVERSIDAD SIMÓN BOLÍVAR

DEPARTAMENTO DE ELECTRÓNICA Y CIRCUITOS

LABORATORIO DE MICROPROCESADORES EC-3074

INFORME - PRÁCTICA #1

Profesor

Mauricio Pérez

Estudiante

Giancarlo Torlone 20-10626

ÍNDICE

INTRODUCCIÓN

En la siguiente práctica de laboratorio se diseña un circuito con el microcontrolador PIC16F877A y dos LEDs, los cuales se encienden y se apagan de manera intercalada a una frecuencia de oscilación de 1 Hz. El código es en lenguaje Assembler. El retardo o delay se realizará con bucles anidados tomando en cuenta el ciclo de máquina de cada instrucción.

MARCO TEÓRICO

El PIC16F877A es un circuito integrado programable tipo FLASH reprogramable capaz de realizar y controlar tareas. El MCU cuenta con una RAM de 256 Bytes, frecuencia de trabajo de 20 MHz, empaquetado DIP-40 . Pertenece a la familia de microcontroladores PIC16.
El microcontrolador depende de una alimentación de al menos 5V y 0V en sus entradas de Vdd y Vss respectivamente para su operación, requiere de una señal de reloj que le indique la frecuencia de trabajo, está señal la introducimos a través de un oscilador de cristal de cuarzo, y una alimentación al pin MCLR, que es un pin de reset que activa al microcontrolador. El funcionamiento del microcontrolador está determinado por un programa almacenado en su memoria Flash ROM y puede programarse más de una vez para cambiar su estado y su comportamiento, lo que convierte al microcontrolador en una pieza esencial en el rápido desarrollo de aplicaciones electrónicas.
Algunas de sus aplicaciones son automatización y control de procesos, comunicaciones y red, electrónica de consumo, diseño embebido y desarrollo, multimedia, dispositivos portátiles, robótica, instrumentación o seguridad.

Algunas características del PIC16F877A

- 100.000 ciclos de borrado/escritura Enhanced Flash memoria del programa típica
- 1.000.000 de borrado/ciclo de escritura Datos EEPROM memoria típica
- Retención EEPROM de datos > 40 años
- Auto-reprogramable bajo control de software
- Programación serie en circuito(ICSP) a través de dos pines
- Programación serie de 5V in-circuit de un solo suministro
- Temporizador watchdog (WDT) con su propio RC en chip oscilador para un funcionamiento fiable
- Protección programable del código
- Ahorro de energía Modo de suspensión
- Opciones de oscilador seleccionables
- Depuración en circuito (ICD) a través de dos pines

Algunas instrucciones para los registros

ADDWF	Suma de W & F
ANDWF	Función AND de W & F
CLRF	Borrar un Registro
CLRW	Borra el registro de trabajo W
COMF	Complementa el Registro F
DECF	Decrementa F en 1
DECFSZ	Decrementa en 1 y salta si el resultado 0
INCF	Incrementa el registro F
INCFSZ	Incrementa en 1 y salta si el registro es 0
IORWF	Función OR de W & F
MOVF	Mover el registro F
RLF	Rota el registro F a la izquierda
RRF	Rota el registro F a la derecha
SUBWF	Resta F – W
SWAPF	Intercambio de F

XORWF	Función XOR de W & F
NOP	No operación
BCF	Borra un bit
BSF	Activa un bit
BTFSC	Verifica un bit y salta si es 0
BTFSS	Verifica un bit y salta si es 1
ANDLW	(W AND Literal)
CALL	Llamada a subrutina
CLRWTD	Borra el watchdog timer
GOTO	Salto incondicional
IORLW	(W OR Literal)
MOVLW	Carga un Valor al Registro W
RETURN	Regresa de una Subrutina
RETLW	Regresa de una Subrutina y carga el valor K en W
RETFIE	Regresa de la rutina de servicio
SLEEP	Entra en estado de reposo

XORLW	Realiza la función XOR entre W & K, el resultado se almacena en W
SUBLW	Resta L - W
MOVWF	Mover el valor del registro W al registro F

ANÁLISIS DE RESULTADOS

Oscilador

El oscilador del microcontrolador se configuró para que sea HS (High Speed) y para que trabaje a una frecuencia de 20 MHz.

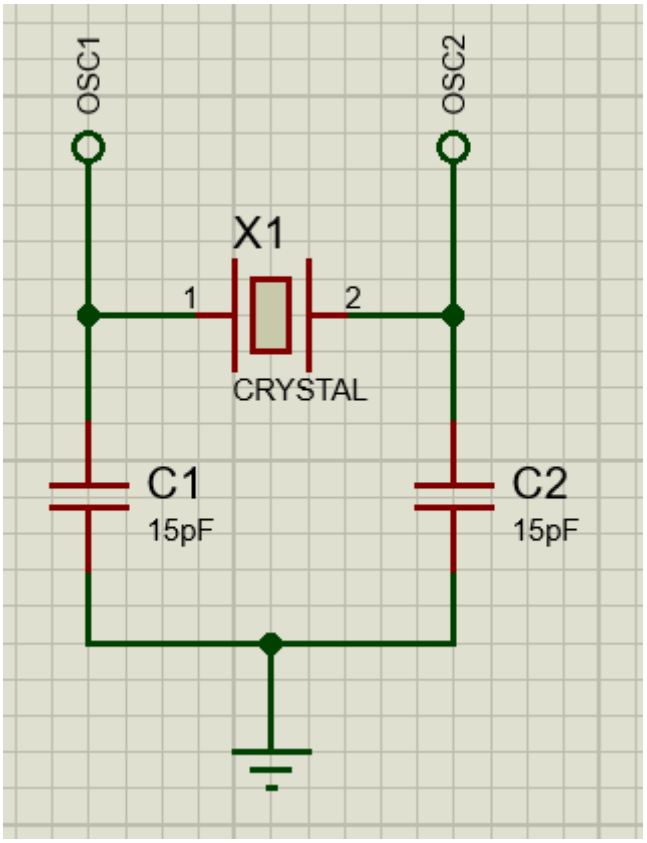


Figura 1. Conexiones para el oscilador

MCLR

Para el MCLR o reset se realizó la siguiente conexión en PULL UP.

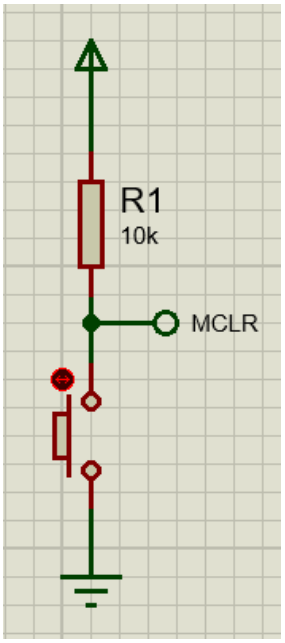


Figura 2. Conexión para el MCLR

Montaje Final

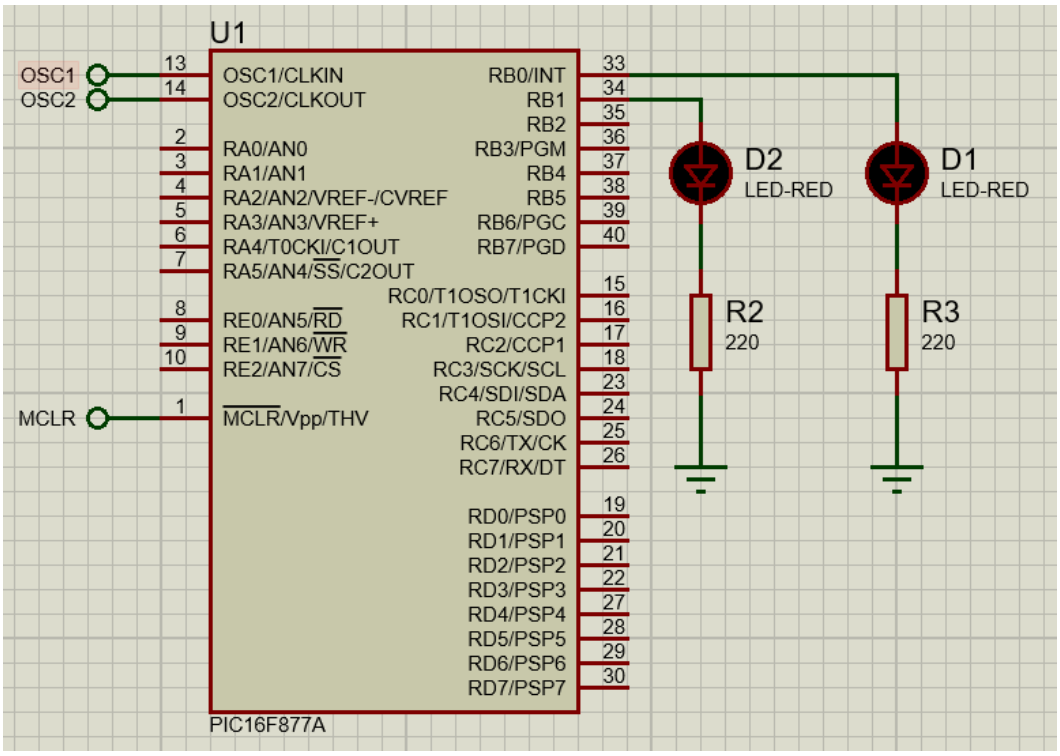


Figura 3. Montaje final con los dos LEDs

Selección de los pines

Los LEDs se conectan a los pines RB0 y RB1 del microcontrolador ya que el puerto B es un puerto bidireccional de 8 bit. De esta manera, se puede configurar para que se comporte como salida o entrada usando el registro TRISB. En este caso ambos pines fueron configurados como salida, para ello se deben colocar en “0”. Luego se utilizará el registro PORTB para encender o apagar los LEDs.

La instrucción BCF coloca un bit a Cero.
La instrucción BSF coloca un bit a Uno.

Código

En las primeras líneas de código se selecciona el PIC que se va a utilizar y su configuración:

```
#include "p16f877a.inc"
__CONFIG _FOSC_HS & _WDTE_OFF & _PWRTE_ON & _BOREN_OFF & _LVP_OFF
& _CPD_OFF & _WRT_OFF & _CP_OFF
LIST P=16F877A
ORG 0
```

Usando el datasheet del microcontrolador se puede observar que el registro para configurar los pines como salida es TRISB, el cual se encuentra en el Bank 1.

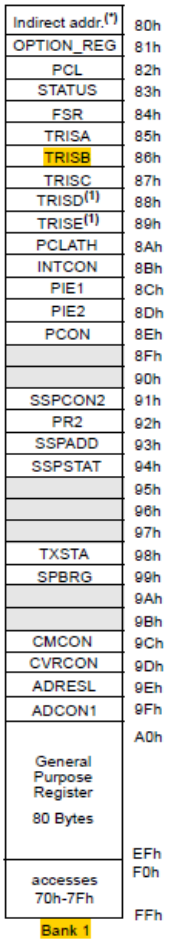


Figura 4. Ubicación del registro TRISB

Por ello primero debemos “movernos” al Bank 1.
Para poder realizar esto, se debe colocar el RP0 del registro STATUS en 1 utilizando BSF (Pone a **Uno** el **Bit** correspondiente del **Registro** especificado).

RP1:RP0: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h-1FFh)
10 = Bank 2 (100h-17Fh)
01 = Bank 1 (80h-FFh)
00 = Bank 0 (00h-7Fh)
Each bank is 128 bytes.

Una vez en el Bank 1, se puede configurar los pines como salida.

```
BSF STATUS, RP0  ; Ir al bank 1
BCF TRISB, 0  ; PIN RB0 como salida
BCF TRISB, 1  ; PIN RB1 como salida
```

Luego debemos dirigirnos al Bank 0, para poder controlar si el LED se debe encender o apagar usando el registro PORTB.

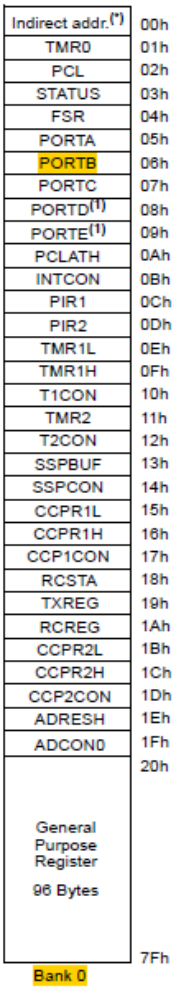


Figura 5. Ubicación del registro PORTB

Para esto, se hace de forma similar al Bank 1, solo que el Bank 0 debe usar el RP0 del registro STATUS en 0 utilizando BCF (Pone a **Cero** el **Bit** correspondiente del **Registro** especificado).

```
BCF STATUS, RP0 ; Ir al bank 0

BCF PORTB, 0 ; LED 1 inicialmente apagado
BCF PORTB, 1 ; LED 2 inicialmente apagado
```

Ahora procedemos a crear el Retardo de 1s (frecuencia 1 Hz).

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add Literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND Literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call Subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to Address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR Literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move Literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from Interrupt	2	00	0000	0000	1001		
RETLW	k	Return with Literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from Literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR Literal with W	1	11	1010	kkkk	kkkk	Z	

Figura 6. Tabla con el ciclo máquina de cada registro

Cálculo del tiempo que tarda cada ciclo máquina:

$$Ti = (4 \times \frac{1}{fosc}) \times cm$$

Ti = Tiempo de demora

fosc = frecuencia del oscilador

cm = ciclo máquina

$$Ti = (4 \times \frac{1}{20MHz}) \times 1$$

$$Ti = 0.2\,us$$

1 cm equivale a 0.2 us

Para poder generar un retardo de 1s se necesitan 5.000.000 ciclos máquina

$$cm = Ti \times \frac{fosc}{4}$$

$$cm = 1s \times \frac{20MHz}{4}$$

$$cm = 5.000.000$$

Sin embargo, el PIC sólo trabaja hasta 255 (8 bits). Por lo tanto se debe hacer uso de bucles para alcanzar el valor del ciclo máquina.

Por lo tanto, se necesitan 3 bucles: 255 x 255 x 255 = 16.581.375 (abarca los 5.000.000)

Pseudocódigo para el retardo de 1s:

Como se va a trabajar con variables que se irán actualizando, se deben declarar esas variables en registros de memoria dentro del microcontrolador. Esas variables se deben guardar a partir de la ubicación 0x20.

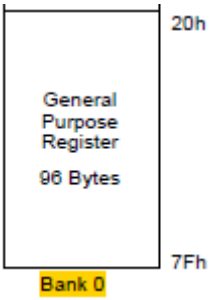


Figura 7. Registros de propósito general

Declaración de las variables:
num EQU 0x21
i EQU 0x22
Valor EQU 0x23

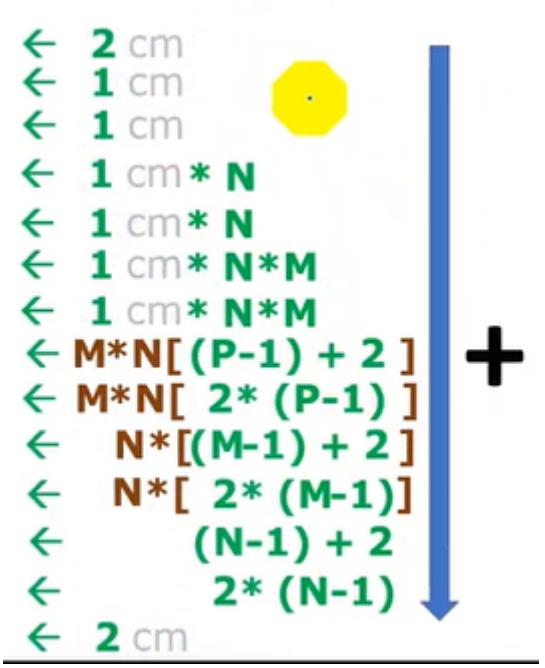
Retardo de 1s:

```
RETARDO1S
MOVLW d'X' ; ← Este es el valor que se debe calcular
MOVWF num
ARRIBA
MOVLW d'255'
MOVWF i
BUCLE1
MOVLW d'255'
MOVWF Valor
REPITE1
DECFSZ Valor,1
GOTO REPITE1
DECFSZ i,1
GOTO BUCLE1
DECFSZ num,1
GOTO ARRIBA
RETURN
```

Ciclo máquina de cada instrucción:

```
RETARDO1S ; 2cm                          num = N        i  = M        Valor = P
MOVLW d'N' ; 1cm --> valor de N
MOVWF num ; 1cm
ARRIBA
MOVLW d'255' ; 1cm*N --> valor de M
MOVWF i ; 1cm*N
BUCLE1
MOVLW d'255' ; 1cm*N*M --> valor de P
MOVWF Valor ; 1cm*N*M
REPITE1
DECFSZ Valor,1 ; --> (P-1)*M*N + 2M*N
GOTO REPITE1 ; --> (P-1)*M*N*2
DECFSZ i,1 ; --> (M-1)*N + 2N
GOTO BUCLE1 ; --> (M-1)*2*N
DECFSZ num,1 ; --> (N-1) + 2cm
GOTO ARRIBA ; --> (N-1)*2
RETURN ; 2cm
```

La cantidad de ciclo máquina será la suma de todo lo anterior.



cm = 2+1+1+N+N+NM+NM+[(P-1)MN+2MN]+[2(P-1)MN]+[(M-1)N+2N]+[(M-1)2N)]+(N-1)+2+2(N-1)+2

Despejado FINAL:

$$cm = 5+4N+4MN+3PMN$$

M=255
P=255
N=??

$$5.000.000 = 5 + 4N + 4(255)N + 3(255)(255)N$$

$$5.000.000 = 5 + 196099N$$

$$N = 25.49$$

$$N = 26$$

Con este valor de N = 26 estamos generando un retardo de 1.01s, bastante cercano a 1s

Código final del retardo de 1s:

```
RETARDO1S ; 2cm
MOVLW d'26' ; 1cm --> N
MOVWF num ; 1cm
ARRIBA
MOVLW d'255' ; 1cm*N --> M
```

```
MOVWF i ; 1cm*N
BUCLE1
MOVLW d'255' ; 1cm*N*M --> P
MOVWF Valor ; 1cm*N*M
REPITE1
DECFSZ Valor,1 ; --> (P-1)*M*N + 2M*N
GOTO REPITE1 ; --> (P-1)*M*N*2
DECFSZ i,1 ; --> (M-1)*N + 2N
GOTO BUCLE1 ; --> (M-1)*2*N
DECFSZ num,1 ; --> (N-1) + 2cm
GOTO ARRIBA ; --> (N-1)*2
RETURN ; 2cm
```

Al ya tener el retardo de 1s solo queda crear el programa para encender y apagar los LEDs

```
PROGRAMA
BSF PORTB, 0 ; encender LED 1
BCF PORTB, 1 ; apagar LED 2
CALL RETARDO1S ; llamar el retardo de 1s
BCF PORTB, 0 ; apagar LED 1
BSF PORTB, 1 ; encender LED 2
CALL RETARDO1S ; llamar el retardo de 1s
GOTO PROGRAMA
```

Finalmente terminamos el código con la instrucción END.

Código completo final:

```
#include "p16f877a.inc"
__CONFIG _FOSC_HS & _WDTE_OFF & _PWRTE_ON & _BOREN_OFF & _LVP_OFF
& _CPD_OFF & _WRT_OFF & _CP_OFF
LIST P=16F877A
ORG 0

num EQU 0x21
i EQU 0x22
Valor EQU 0x23

BSF STATUS, RP0 ; Ir al bank 1
BCF TRISB, 0 ; PIN 0 como salida
BCF TRISB, 1 ; PIN 1 como salida
BCF STATUS, RP0 ; Ir al bank 0

BCF PORTB, 0 ; LED 1 inicialmente apagado
BCF PORTB, 1 ; LED 2 inicialmente apagado

PROGRAMA
BSF PORTB, 0 ; encender LED 1
BCF PORTB, 1 ; apagar LED 2
CALL RETARDO1S ; llamar el retardo de 1s
BCF PORTB, 0 ; apagar LED 1
BSF PORTB, 1 ; encender LED 2
CALL RETARDO1S ; llamar el retardo de 1s
GOTO PROGRAMA

; retardo para 1HZ == 1s
RETARDO1S ; 2cm
MOVLW d'26' ; 1cm --> N
MOVWF num ; 1cm
ARRIBA
```

```
MOVLW d'255' ; 1cm*N --> M
MOVWF i ; 1cm*N
BUCLE1
MOVLW d'255' ; 1cm*N*M --> P
MOVWF Valor ; 1cm*N*M
REPITE1
DECFSZ Valor,1 ; --> (P-1)*M*N + 2M*N
GOTO REPITE1 ; --> (P-1)*M*N*2
DECFSZ i,1 ; --> (M-1)*N + 2N
GOTO BUCLE1 ; --> (M-1)*2*N
DECFSZ num,1 ; --> (N-1) + 2cm
GOTO ARRIBA ; --> (N-1)*2
RETURN ; 2cm

END
```