UNIVERSIDAD NACIONAL AUTONOMA DE NICARAGUA.

UNAN-MANAGUA.

RECINTO UNIVERSITARIO RUBEN DARIO RURD.

FACULTAD DE CIENCIAS E INGENIERIA.

DEPARTAMENTO DE TECNOLOGIA

INGENIERIA ELECTRONICA.

DISEÑO DE UNA INTERFAZ ENTRENADORA DEL MICROCONTROLADOR PIC16F84A PARA LOS ESTUDIANTES DE INGENIERIA ELECTRONICA DE LA UNAN-MANAGUA, PARA FINES DIDACTICOS.

INTEGRANTES:

Br. OMAR ENRIQUE ALEMAN LOPEZ

Br. GONZALO MALEAÑOS CENTENO

TUTOR

MSc. ALVARO SEGOVIA

MANAGUA AGOSTO 2014

DEDICATORIA:

Omar.

- A esas personas que se esforzaron por ver cumplidos mis sueños apoyándome cuando las circunstancias eran difíciles. Con todo mi cariño este proyecto se los dedico a ustedes:

Papá Oscar.
Mamá Mercedes.
Esposa Noelia.
Y nuestro pequeño hijo
Emmanuel.

Gonzalo.

- Con todo mi cariño y mi amor para las personas que hicieron todo en la vida para que yo pudiera lograr mis sueños, por motivarme y darme la mano cuando sentía que el camino se terminaba, a ustedes por siempre mi corazón.

Papá Rene.
Mamá Luzgarda.

AGRADECIMIENTO:

- A nuestro Dios Jesucristo, por habernos otorgado la fuerza de voluntad necesaria para seguir adelante en nuestro proyecto.
- A nuestras familias, por el apoyo incondicional brindado, así como su voz de aliento y apoyo en momentos de dificultad.
- A nuestro tutor y profesores de todos estos años que con su guía y enseñanza nos motivaron a seguir aprendiendo y a mejorar como personas y estudiantes.

RESUMEN.

La interfaz entrenadora del PIC16F84A ha sido diseñada con fines didácticos tanto teóricos como prácticos. Se comprendió que era necesaria una herramienta para introducir al estudiante en el mundo de los Microprocesadores y periféricos. Se escogió el PIC16F84A por ser un PIC fácil de adquirir en el mercado y su versatilidad para implementarlo con diferentes tipos de dispositivos. En este trabajo contiene cinco módulos que son: LCD, teclado hexadecimal, memoria EEPROM I2C, visualización de Puertos y display de siete segmentos.

Para diseñar la interfaz se utilizó programas como ISIS-PROTEUS y el programa ARES estos programas se utilizaron para probar de manera virtual nuestro diseño y su montaje final. Ya teniendo su diseño final se procedió a utilizar la técnica de termo transferencia o técnica del planchado para su montaje práctico en baquelita, los mismos procedimientos se hizo con el grabador JDM el circuito diseñado para grabar físicamente los programas en el PIC utilizando el software IC-PROG para cargar el programa en el microcontrolador.

Para compilar dichos programas de los cinco módulos antes expuestos se utilizó el entorno MPLAB IDE que es un software que se ejecuta bajo Windows haciendo uso del lenguaje ensamblador que es el lenguaje recomendado por el fabricante en PIC de bajo y mediano recursos. El MPLAP edita el archivo fuente y genera los archivos necesarios como es el .ASM y el .HEX que son los más importantes en nuestro caso el archivo fuente y el archivo ejecutable respectivamente y este último es el que se utiliza para grabar físicamente en el PIC.

Terminado el diseño de forma real se siguió con el paso final, en la validación de nuestro trabajo con una serie de ejercicios prácticos designados para cada uno de los módulos. Con el propósito de ser lo más claro y conciso posible para que el estudiante no sólo se introduzca en la programación de microcontroladores, sino que monte sus propios proyectos y sea capaz de afrontar en un lenguaje de más alto nivel y un microcontrolador con mayores recursos.

Contenido

INTRUDUCCION.

Una manera muy eficaz de incentivar el estudio de los microcontroladores en los estudiantes de ingeniería electrónica son las interfaces entrenadoras. Estas interfaces tienen como propósito familiarizar al estudiante con el comportamiento de los módulos internos de estos dispositivos para los cuales fueron creados. Nuestro trabajo consiste en el diseño y montaje de una interfaz entrenadora para el microcontrolador PIC16f84A el cual contenga los circuitos básicos pero necesarios para que el alumno pueda iniciar el estudio de este dispositivo y prepararse para afrontar el estudio de microcontroladores más estructurados.

Los circuitos de esta unidad entrenadora han sido diseñados con el software ISIS PROTEUS ya que este permite la simulación virtual de los circuitos con el PIC16F84A, en total se diseñaron cinco circuitos el cual nombramos como: circuito Visualizador de estado, circuito Display LCD, circuito Display 7 segmentos, circuito Teclado hexadecimal y circuito BUSI2C.

Los códigos fuentes de los programas se editaron y compilaron con el software MPLAB IDE de Microchip y el lenguaje de programación empleado es el Ensamblador.

Para la implementación y prueba de la interfaz entrenadora se desarrolló una práctica demostrativa con el docente encargado de la clase microprocesadores. Haciendo un cuestionario para evaluar la efectividad de la interfaz entrenadora.

El presente trabajo lo hemos divido en tres capítulos. El primer capítulo contiene una pequeña encuesta en la cual se comprueba la necesidad de una interfaz entrenadora real para los dispositivos microcontroladores y electrónica digital. El segundo capítulo contiene los diseños de los circuitos, se expone la teoría necesaria con sus respectivos cálculos y conexiones. El tercer capítulo contiene el diseño de los programas con sus correspondientes diagramas de bloques y una breve descripción de las librerías también contiene el método empleado para montar la tarjeta, los pasos necesarios para configurar y usar los software MPLAB IDE, ICPROG y los pasos para usar correctamente la interfaz entrenadora.

Además contiene las prácticas de evaluación. Los resultados del cuestionario de evaluación se han incluido en la sección de anexos

JUSTIFICACION:

Dentro de las actuales necesidades de la carrera de Ingeniería Electrónica de la UNAN Managua se encuentra el fortalecimiento del laboratorio en especial los de electrónica digital y procesadores. En la actualidad estas asignaturas cuentan con una interfaz entrenadora para el microcontrolador 8051 de INTEL. Este dispositivo pertenece a la gama media alta de microcontroladores y cuenta con una gran cantidad de módulos y recursos internos. (USART, CCP, SPI, EEPROM, I2C, Interrupciones, etc.). La interfaz entrenadora posee todos los circuitos necesarios para probar uno a uno los módulos de este microcontrolador.

Aunque esta unidad entrenadora es extraordinaria muchas veces los estudiantes se ven frustrados al no poder darle el uso correspondiente por desconocer la mayoría de temas correspondiente a los módulos debido a que apenas están iniciando el estudio de estos dispositivos microcontroladores. Por esto es necesario iniciar el estudio de estos dispositivos con un microcontrolador como el PIC16F84A que contiene pocos módulos y recursos y contar con su respectiva interfaz entrenadora. Esto fortalecería el laboratorio y por consiguiente el aprendizaje de los estudiantes de esta asignatura.

Otra de los inconvenientes que se presenta al estudiar estos dispositivos es el lenguaje de programación. Como es sabido entre más módulos y recursos tenga un microcontrolador es necesario contar con un lenguaje de alto nivel, como es C por ejemplo. Sin embargo en estos tipos de lenguaje la programación no requiere el conocimiento del conjunto de instrucciones o características del hardware del microcontrolador utilizado. Para fines didácticos esto no es conveniente porque entre más se conozca el hardware interno del dispositivo mejor es el aprendizaje y dominio de los módulos internos de los microcontroladores.

Por la antes expuesto es necesario iniciar programando estos dispositivos en un lenguaje de bajo nivel como el Ensamblador ya que este lenguaje requiere del conocimiento del hardware del microcontrolador a utilizar. El PIC16F84A por ser un microcontrolador básico es ideal para ser programado en este lenguaje y con la interfaz entrenadora el estudiante podrá ir viendo paso a paso el desarrollo de sus programas.

OBJETIVO GENERAL:

Diseñar una interfaz entrenadora del microcontrolador PIC16F84A para los estudiantes de ingeniería electrónica de la UNAN-MANAGUA para fines didácticos.

OBJETIVOS ESPECÍFICOS:

1. Investigar la existencia de herramientas didácticas para la enseñanza del microcontrolador PIC16F84A en la carrera de ingeniería electrónica.
2. Diseñar cada uno de los circuitos que componen la interfaz entrenadora del microcontrolador PIC16F84A con sus respectivos programas.
3. Implementar la interfaz entrenadora en la asignatura de Microprocesadores.
4. Realizar prácticas demostrativas que validen el funcionamiento y facilidad de uso de la interfaz entrenadora.

DESARROLLO

CAPITULO 1. ENCUESTA.

1.1 Elaboración de una encuesta que determine la necesidad de una herramienta didáctica real.

Para verificar si existe una herramienta didáctica (interfaz entrenadora real) en los laboratorios de electrónica del UNAN-Managua. Se hizo una pequeña encuesta siendo el universo los estudiantes de la carrera ingeniería electrónica y tomamos como muestra los alumnos egresados entre los años 2007 al 2014 que actualmente están llevando el seminario de graduación. Ya que al haber concluido con su pensum correspondiente ellos pueden dar fe según se experiencia como estudiantes si es necesario una herramienta didáctica real para los estudiantes que actualmente están cursando esta carrera.

1.2 Conclusiones generales sobre la encuesta:

La encuesta se puede ver en la sección de anexos A, de ella se puede ver que los dispositivos microcontroladores son muy conocidos por los estudiantes. Siendo las herramientas virtuales las más utilizadas. También se puede comprobar que el lenguaje más utilizado por los alumnos es el lenguaje C aunque el lenguaje ensamblador también es muy conocido y utilizado. Otro de los aspectos interesantes es que los microcontroladores más utilizados y conocidos por los estudiantes son los PIC de Microchip technology por encima del 8051 de Intel y Atmel. Se puede ver que una interfaz real para microcontroladores es muy recomendada. Es importante notar que aunque las herramientas virtuales proporcionan un gran dominio en el tema, los estudiantes en un cien por ciento recomiendan una herramienta real. Por lo que esta sería una complemento excelente para este tipo de estudio además que fortalecería los estudios de electrónica digital.

CAPITULO 2. DISEÑO Y PROGRAMACION DE LA INERFAZ ENTRENADORA

2.1 Diseño General.

En la figura 1 se muestra el diseño de la interfaz entrenadora donde se destacan cinco circuitos o módulos que componen esta unidad más el PIC16F84A y sus componentes.

El módulo 1 está constituido por 8 resistencias de 330 ohms, 8 diodos led rojos y 8 jumperes. Este circuito será utilizado para visualizar el estado del puerto B del PIC16F84A. Cuando el pin del puerto este en estado alto el LED estará encendido y cuando el pin del puerto este en estado bajo el LED estará apagado. Los jumperes sirven para deshabilitar los LED de forma individual. Este circuito se denomina circuito Visualizador de Estado.

El módulo 2 está constituido por dos display de 7 segmentos, dos transistores 2N3904, un jumper y las ocho resistencias de 330 ohm que comparte con el circuito Visualizador de Estado del módulo 1. El jumper sirve para deshabilitar el circuito cuando no se valla utilizar. El circuito sirve para visualizar números que van del 0 al 99. Este circuito se denomina Display 7 Segmentos.

El módulo 3 está formado por un display LCD, la resistencia de iluminación RLCD1 de 50 ohm, una resistencia variable RV1 de 10 k ohm que controla el contraste de los caracteres y un jumper (ENABLE) que sirve para habilitar el módulo. Este circuito sirve para visualizar mensajes o números y se llama circuito Display LCD.

El módulo 4 está formado por 16 switch en un arreglo matricial de 4 filas por cuatro columnas y cuatro resistencias de 330 ohm que son RF1 a RF4. Se utiliza para ingresar datos numéricos al microcontrolador. Este circuito se denomina Teclado Hexadecimal.

El módulo 5 está constituido por la eeprom 24C04 I2C y sus dos resistencias de pull up de 4.7 kohm. A este circuito se llama circuito Bus I2C.

El resto de componentes que constituyen la unidad entrenadora son dos switch que son SRA4 Y SRA3 con sus resistencias de pull up que son R1SW Y R2SW de 10 kohm y dos jumperes (JRA3 Y JRA4)que sirven para deshabilitar estos switch que pueden ser utilizados por cualquier módulo. El PIC16F84A que contiene su cristal de 4 MHz con sus capacitores y el switch de reset con sus resistencias.

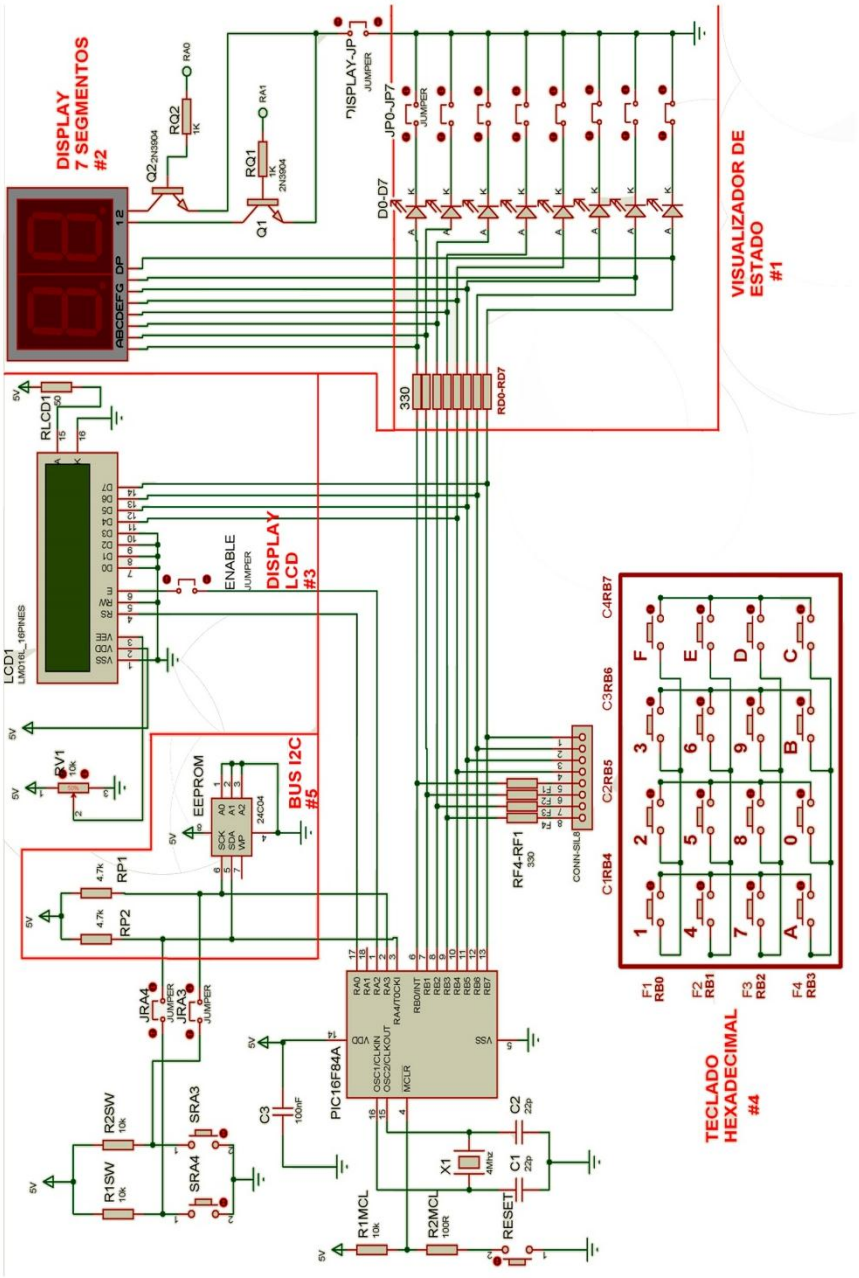


Figura 1. Circuito de la interfaz entrenadora.

2.2 Diseño del circuito para visualizar estado de puerto B del PIC16F84A.

Según la hoja de datos del PIC16F84A la corriente en modo fuente por cada pin del puerto B es de 25 mA sin embargo los ocho pines no pueden exceder los 100 mA (PIC16F84A, 2001, pág. 49). Se escogió un diodo led rojo porque estos se pueden activar con 10 mA y 1.8 v (Palacios, 2009, pág. 10). En caso donde los ocho pines del puerto estén en alto tendríamos 80 mA en total. Esto es 20% menos del máximo permitido.

Basados en la Figura 2 se hizo el siguiente cálculo para la resistencia limitadora.

Rd= (5v-1.8v)/10mA= 320 ohm

Los 5 voltios son los que provienen del puerto B (RB0-RB7) del pic.

Se escogió una resistencia de 330 ohm por ser un valor más estándar.

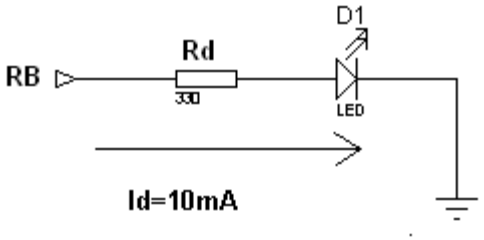


Figura .2 para cálculo resistencia limitadora

Al circuito completo se le añaden jumper individuales a los led permitiendo habilitarlos o deshabilitarlos de forma individual. Esto es importante por ejemplo cuando se trabaja con lcd y teclado y se quiere visualizar una acción por los puertos que no están ocupados así solo se habilitan los led que se quieran visualizar (Rb0 a Rb3). Además se escogió la configuración cátodo común por tanto los led se encenderán cuando el estado del puerto este en alto. Para futuras referencias a este circuito le llamamos “Visualizador de estado”. En la figura 3 se muestra el circuito conectado al microcontrolador pic16f84a

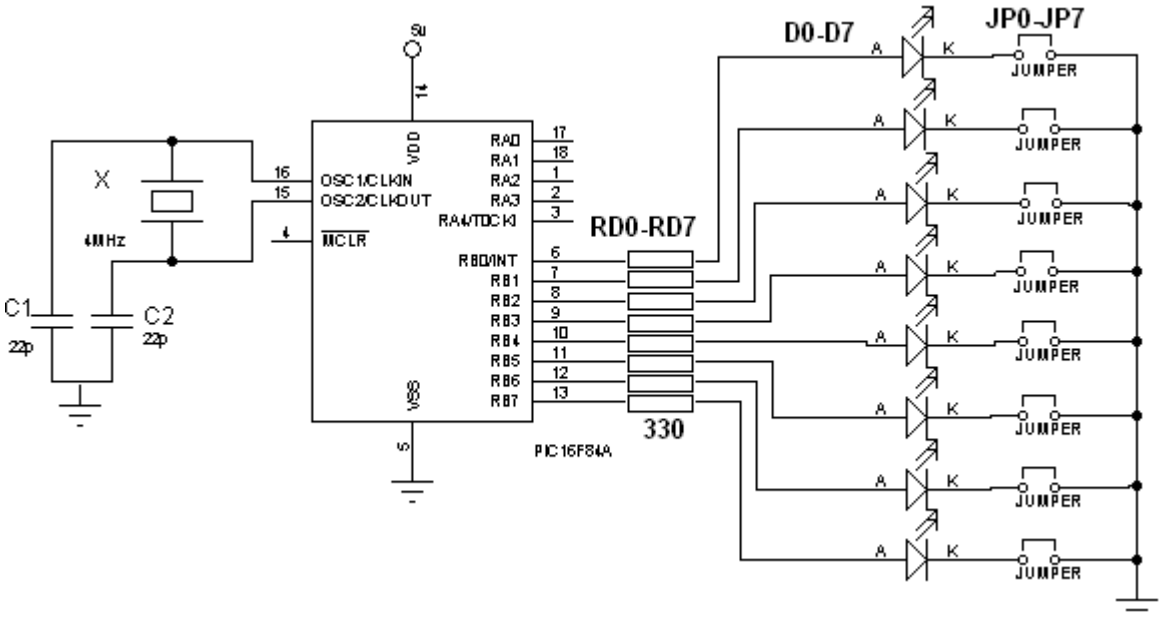


Figura 3. Circuito "Visualizador de estado" del puerto B conectado al Pic16f84a.

2.3 Diseño del circuito para visualizar dos dígitos con dos display de 7 segmentos.

La técnica que se utilizamos para visualizar los dos display de 7 segmentos es la multiplexación, este método nos permite conectar los dieciséis pines de los display a los ocho pines del puerto b del microcontrolador más dos pines de control, en total utilizaremos diez pines.

La multiplexación es un método basado en la persistencia del ojo humano. El ojo humano no es capaz de seguir el parpadeo de una fuente de luz que destelle a más de veinte veces o más por segundo, en el caso de led o lámparas hay que cuidar que la frecuencia de parpadeo sea mayor a 25 hz. Esto hay que tenerlo en cuenta a la hora de diseñar los programas.

La figura 4. Muestra el circuito utilizado para realizar los cálculos.

La razón para usar transistores es porque en los pines de control de encendido (RA0 y RA1) estarían retornado alrededor de 80mA, al estar encendidos todos los led por ejemplo el número 8, y el máximo valor en modo sumidero para un pin del puerto es de 25mA (PIC16F84A, 2001, pág. 49). Como en el diseño de los diodos led ya habíamos calculado la resistencia limitadora (330 ohms); utilizaremos estas mismas resistencias y solo calculamos las resistencias para la base de los transistores

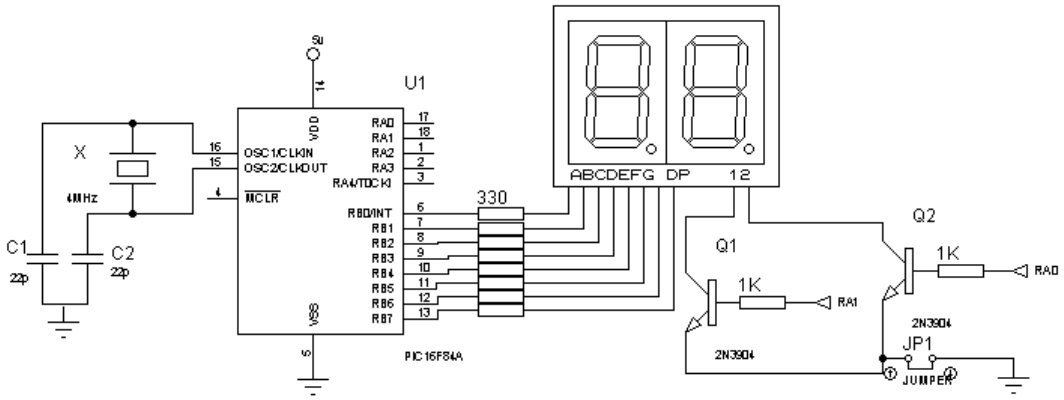


Figura 4. Circuito "Display 7 segmentos" conectado al Pic16f84a.

Los datos del transistor 2n3904 son hfe=200 (100 a 300), máxima corriente en colector 200mA (suficiente para este caso ya el máximo seria aproximadamente 80mA) (2n3904, 2011, págs. 1,2.).

Las fórmulas para los cálculos en un transistor en modo saturación son:

$Ib = (Ic/hfe)xF$

Donde Ib es corriente de base.

Ic corriente máxima colector-80mA.

F factor de sobreexcitación, se escogió x10.

$Rb = (V - Vbe)/Ib$

Rb resistencia de base,

V voltaje del puerto RA0 y RA1, 5v

Vbe voltaje base emisor 0.7v.

Ib corriente de base (Robert Boylestad, 2009, págs. 206,207,208).

Introduciendo valores tenemos:

Ib= (80mA/200) x10 =4mA

Rb= (5v-0.7v)/4mA = 1k ohms aproximadamente.

Al diseño final de este circuito se unen los dos emisores de los transistores y se conectan a tierra por medio de un jumper permitiendo al estudiante deshabilitarlos

y poder usar los puertos para otras funciones como se muestra en la figura 4. También se conectan Q2 a RA0 y Q1 a RA1 para facilitar el desplazamiento a la hora de multiplexar. A este circuito le llamamos “Display 7 segmentos”.

2.4 Diseño del circuito LCD para visualización de caracteres alfanuméricos.

El display que se escogió es el LM016L, este dispositivo permite visualizar caracteres alfanuméricos en dos líneas por dieciséis caracteres cada una. El tamaño de los caracteres puede ser 5x7 o 5x10 puntos siendo el primero el más utilizado. Una característica muy importante de este display es que puede ser gobernado con conexión a cuatro bits u ocho bits, en este trabajo se escoge la configuración de bus cuatro bits y tamaño de caracter 5x7 puntos (LM016L, 1998, págs. 1,2).

También cuenta con tres líneas de control que son:

Bit Enable (E): este bit habilita o deshabilita el lcd. Con E=0 deshabilitado E=1 habilitado. Este bit se conecta a RA2 por medio de un jumper para que el alumno pueda desconectarlo del circuito si no lo va utilizar.

Bit R/S: este bit escoge el modo de funcionamiento. R/S=0 activa el modo comando, E/S=1 activa el modo carácter. Este bit se conecta a RA0.

Bit R/W: este bit habilita la lectura escritura en el display, si R/W=0 activa el modo escritura, si R/W=1 activa el modo lectura. El modo lectura sirve principalmente para leer el “busy flag” que está ubicado en el bit 7 del bus de datos. Este “busy flag” indica si el procesador interno del display está ocupado. Si el bit 7 del bus de datos vale “1” indica que la pantalla LCD está ocupada realizando operaciones internas y no puede aceptar nuevas instrucciones ni datos. Hay que esperar que el “busy flag” valga “0” para enviarle la siguiente instrucción o carácter.

En nuestro caso se conecta este bit a tierra (R/W=0 modo escritura) sin lectura del bit “busy flag”. En vez de eso se utilizan retardos después de cada instrucción. Según el fabricante el microprocesador interno del LCD tarda 1.64 ms en la operación modo comando y 64 us en la operación modo carácter, los retardos deben ser mayores a estos tiempos y hay que tenerlos en cuenta a la hora de diseñar los programas para este módulo.

El bus de datos del LCD se conecta de la siguiente manera:

D0-D3 se conectan a tierra.

D4-D7 se conectan a los pines RB4-RB7 de PIC16F84A respectivamente.

Este circuito permite el número mínimo de pines (6 líneas) del microcontrolador para el control del LCD. En la figura 5. Se muestra el diseño del circuito denominado “Display LCD” conectado al microcontrolador.

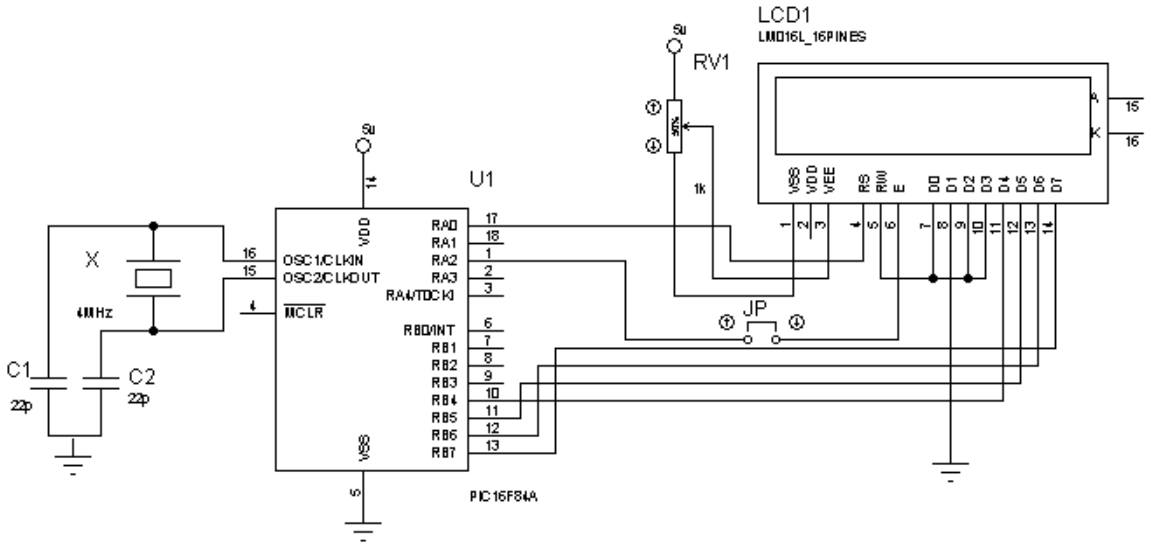


Figura 5. Circuito “Display LCD” conectado al PIC16F84A.

2.5. Diseño de circuito teclado hexadecimal

El teclado que se escogió para este proyecto es un teclado matricial hexadecimal, que básicamente es un arreglo de switch dispuestos en cuatro filas por cuatro columnas. La figura 6 muestra el diagrama para el diseño del teclado.

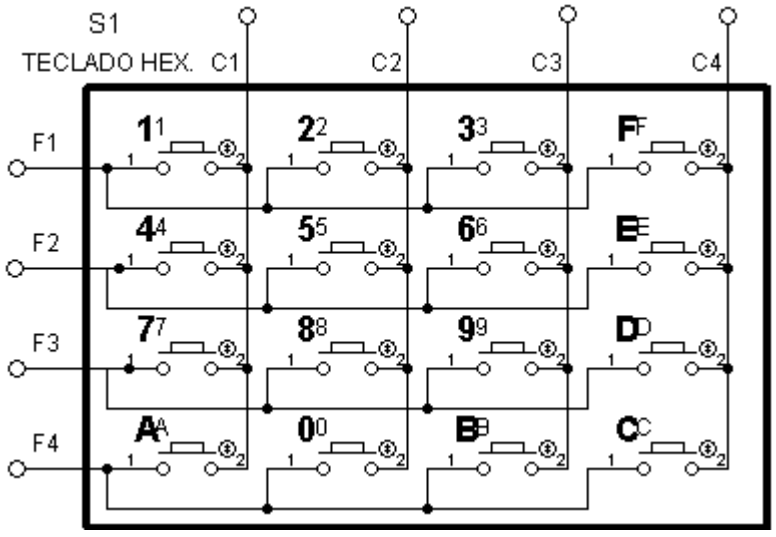


Figura 6. Teclado matricial hexadecimal con arreglo 4x4

Las filas del teclado se conectan a la parte baja del puerto B configuradas como salida por medio de resistencias de 330 ohms, estas resistencias se colocan para evitar cortocircuitos en el caso de que estas líneas se conecten a otro módulo por ejemplo el LCD. Ver figura 7.

Las columnas del teclado se conectan a la parte alta del puerto B del PIC16f84A configuradas como entrada, por esto al momento de diseñar los programas para este módulo se deben activar las resistencias de Pull-Up internas del microcontrolador para evitar que queden flotantes.

Para detectar si esta una tecla presionada se aplica a las filas un nivel bajo y tres altos. Si se presiona una tecla en la fila por la que se aplica el nivel bajo, ese nivel bajo aparecerá en la columna correspondiente con la que se ha hecho el contacto. A este circuito le llamamos “Teclado hexadecimal”.

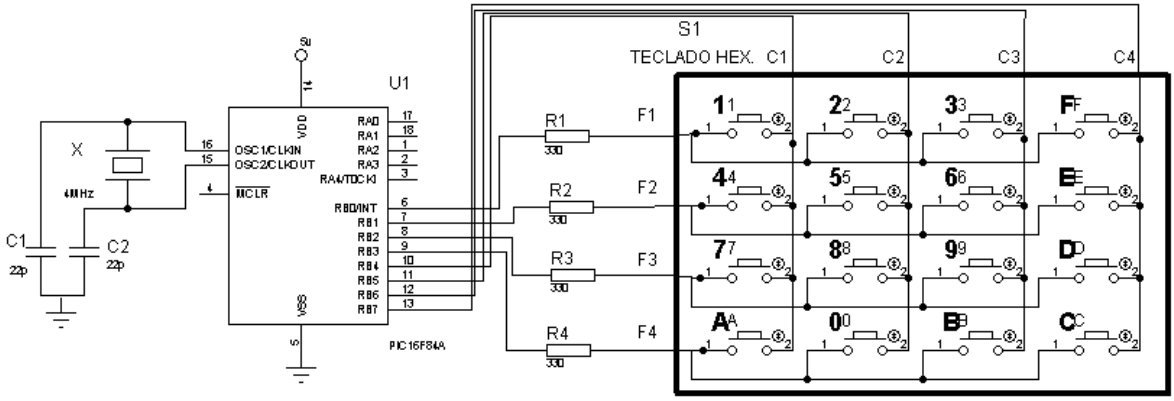


Figura 7. Circuito “Teclado hexadecimal” conectado al pic16f84a

2.6. Diseño de circuito Bus I2C.

El I2C (Inter-Integrate circuit) es un bus serie desarrollado por la empresa Philips Semiconductors y que es ampliamente utilizado en la industria electrónica.

El I2C es un bus, formado por dos hilos, que pueden conectar varios dispositivos por medio de un hardware muy simple, como se muestra en la figura 8. Por esos dos hilos se produce una comunicación serie, bit a bit. Se transmiten dos señales una por cada línea.

- **SCL,** (Serial Clock). Es la señal de reloj que se utiliza para la sincronización de los datos.
- **SDA,** (Serial Data). Es la línea para la transferencia de los datos.

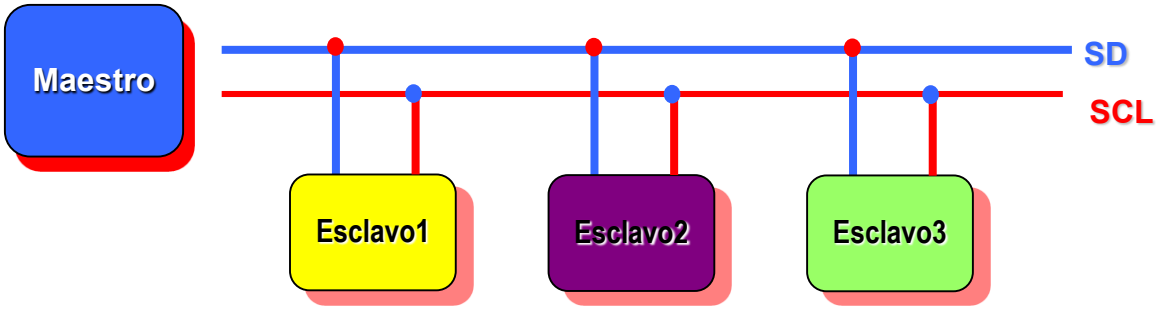


Figura 8. Estructura de un l bus I2C.

Los dispositivos conectados al bus I2C están regidos por un protocolo de tipo maestro/esclavo. Las funciones del maestro y el esclavo se diferencian en:

- El circuito maestro inicia y termina la transferencia de información, además controla la señal de reloj. Generalmente es un microcontrolador (el PIC16f84A en este trabajo).
- El esclavo es el circuito direccionado por el maestro. (EEPROM 24C04 para este caso).

La línea SDA es bidireccional, es decir, tanto el maestro como los esclavos pueden actuar como transmisores o receptores de datos dependiendo de la función de los dispositivos. Por ejemplo un display es solo un receptor de datos pero una memoria recibe y transmite datos

La generación de señal de reloj SCL siempre es responsabilidad del maestro.

Las velocidades de transmisión pueden ser:

- Estándar 100 kps. Esta velocidad es la que se adopta en este trabajo.
- Fast mode 400 kbps.
- Higthspeed mode 3.4 Mbps.

Cada dispositivo conectado al bus I2C posee una dirección única que la diferencia del resto de los circuitos conectados, generalmente estos dispositivos llevan de dos a tres pines para poder modificar su dirección.

Para operar un esclavo sobre el Bus I2C solo son necesarios seis simples códigos, suficientes para enviar o recibir información.

- Un bit de Inicio
- 7-bit o 10-bit de direccionamiento
- Un bit R/W que define si el esclavo es transmisor o receptor
- Un bit de reconocimiento
- Mensaje dividido en bytes
- Un bit de Stop

El dispositivo que se escogió para este proyecto es la memoria eeprom 24C04 de la familia 24Cxx, esta memoria tiene una capacidad de almacenamiento de 4Kbit o 512 byte. Dividida en 2 páginas por 256 posiciones (0-FF) (24C04, 2004, pág. 1). Dependiendo de la necesidad de almacenamiento se puede escoger otra memoria de más capacidad como la 24C08 (1024 byte) o la 24C16 (2048 byte). El diseño para este circuito se muestra en la figura 9. Se escoge el pin RA4 como la línea SDA del bus porque este pin tiene la característica de “Trigger Schmitt” que proporciona buena inmunidad al ruido cuando está configurado como entrada. El pin RA3 se escoge como SCL aunque es posible escoger otro pin para este propósito este se eligió por fines prácticos al estar junto al pin RA4. A este circuito se le nombró “Bus I2C”.

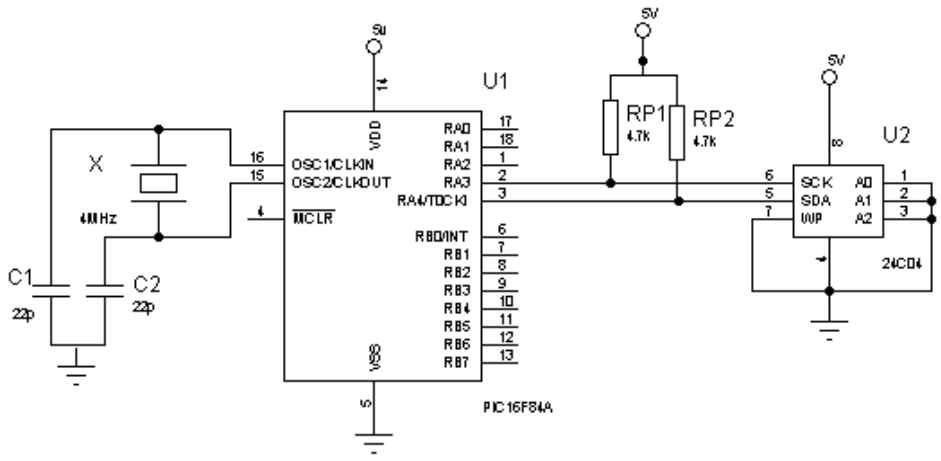


Figura 9. Circuito “Bus I2C” conectado al PIC16f84A.

2.7. Diseño de los programas para los circuitos de la interface entrenadora.

Al programar un microcontrolador PIC16F84A necesitamos conocer las instrucciones para generar el código fuente que será compilado en el software MPLAP IDE de Microchip y posteriormente grabarlo para implementarlo en el circuito correspondiente. El lenguaje de programación a utilizar es el "Ensamblador". El listado de instrucciones de Microchip (el fabricante del microcontrolador) se muestra en la tabla 1.

Un código fuente es un fichero ASCII con extensión .ASM que está formado por cuatro columnas para identificar las diferentes funciones.

- **Etiquetas:** sirven para nombrar diferentes partes del programa. Primera columna
- **Instrucciones:** son las operaciones que se pasan al microcontrolador para que este las ejecute. Segunda columna.
- **Datos:** son los operandos para las instrucciones y pueden estar orientadas a bit, registros, etiquetas o constantes (literales). Tercera columna
- **Comentarios:** son comentarios que se usan para dar mejor comprensión al programa y siempre van después de un punto y coma ";". Cuarta columna.

Una **librería** es un conjunto de subprogramas (subrutinas) diseñados para una tarea específica y pueden ser incluidos mediante la directiva INCLUDE en cualquier parte del programa principal.

Para la edición de los diferentes programas se tomó como principal referencia el libro "MICROCONTROLADOR PIC16F84. Desarrollo de proyectos". Palacios, Remiro, López. 2005. Las librerías incluidas en los diferentes programas son tomadas de este libro a acepción de la librería para la memoria 24C04 que no está en esta obra.

Estas librerías son incluidas en la sección de anexos y están debidamente comentadas para una mejor comprensión.

Tabla 1. Set de instrucciones para el PIC16f84A (PIC16F84A, 2001, pág. 36)

PIC16F84A

PIC16CXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
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 lfff 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 lfff 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	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

Note: Additional information on the mid-range instruction set is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

2.8 Programa 1 para el módulo “Circuito visualizador” de la figura 3.

El objetivo de este programa es: configurar el puerto B como salida para encender y apagar los led cada cierto tiempo poniendo los puertos en estado alto (5v) y bajo (0v) respectivamente. Ver diagrama flujo figura 10. En este programa se usa la librería RETARDOS.INC. Esta librería contiene ciclos que producen retardos que son indispensables en nuestros programas y van desde 2 uS hasta 20 ms. El código fuente está en la sección de anexos C como Visualizador_1.asm

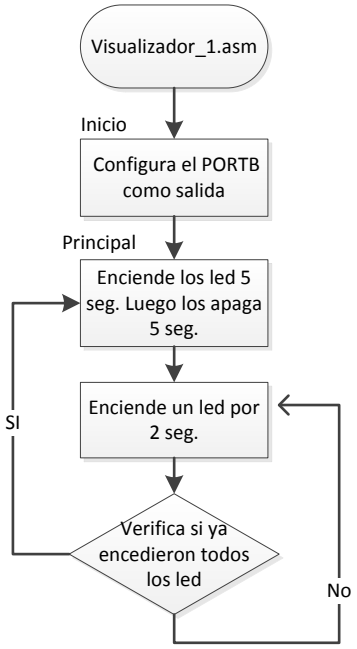


Figura 10. Diagrama de flujo programa Visualizador_1

Inicio: Esta subrutina configura el puerto B como salida.
Principal: Inicialmente el programa enciende todos los led y los mantiene encendido por 5 segundos, luego los apaga luego comienza a encenderlos uno a la vez por dos segundos empezando del RB0 hasta el RB7 para luego apagarse y volver a iniciar el programa desde la subrutina principal.

2.9 Programa 2 para el circuito “Visualizador de estado” figura 3.

En este programa se hace uso de los pulsadores que están conectados al puerto A (SRA4 Y SRA3) Figura 1. También se emplea la técnica de retardos anti rebotes.

Los rebotes son una característica que presentan todo los interruptores mecánicos, producen oscilaciones que pueden ser detectadas por el microcontrolador como una serie pulsos cuando en realidad es el mismo pulso que no se ha estabilizado. Figura 11. Hay muchas maneras de resolver este

inconveniente pero la más usada es generar retardos (un poco más de 20 ms) después de cada pulsación y luego volver a censar el mismo switch.

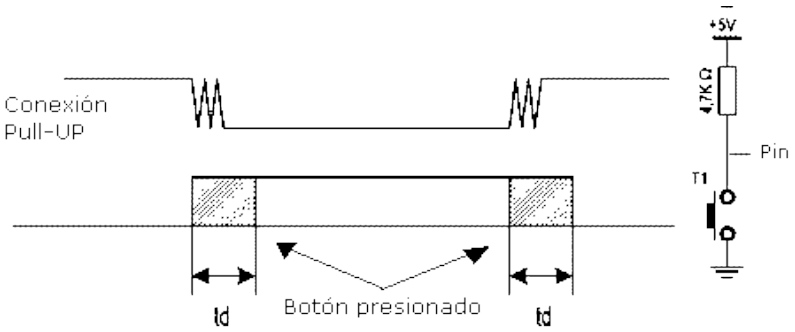


Figura 11. Rebotes producidos en interruptores.

El programa consiste en usar un switch (SRA4) para incrementar el valor del puerto B que es inicializado en cero y otro switch (SRA3) para complementar el valor del puerto B. los resultados se verán inmediatamente en los diodos led. Ver figura 12. Su código fuente se encuentra en anexos C como Visualizador_2.asm

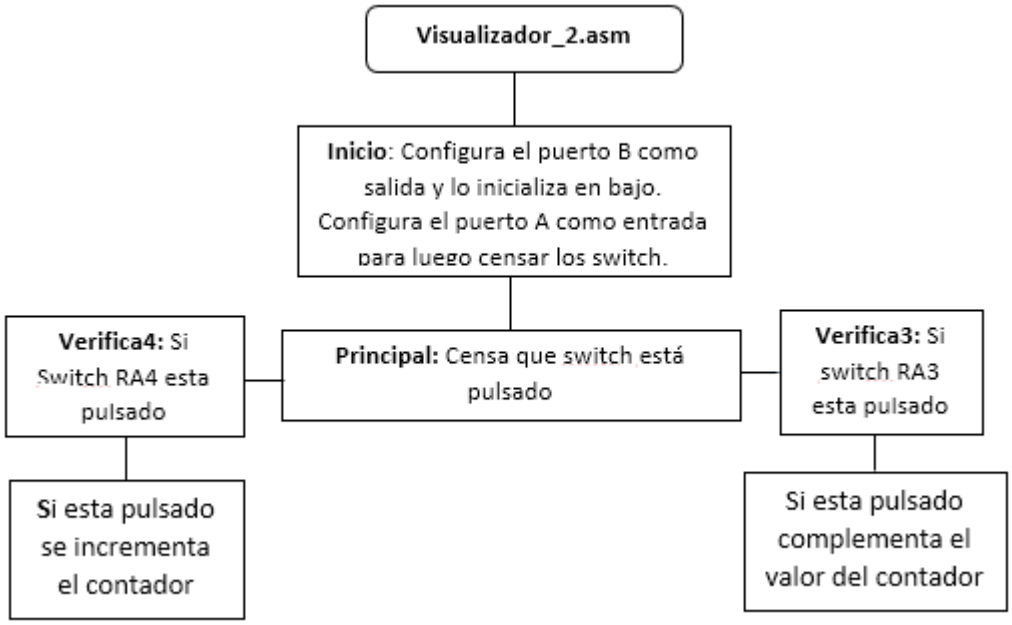


Figura 12. Diagrama de flujo para programa Visualizador_2.asm

Inicio: Esta subrutina inicializa los puertos A y B. Configura el puerto B como Salida y lo inicializa en bajo. El puerto A lo configura como entrada. También inicializa el registro llamado contador en cero.

Principal: La subrutina principal está dividida en dos partes ***Verifica4*** que es la que se encarga de verificar si el switch RA4 esta pulsado y a la vez descarta cualquier rebote. Si esta pulsado incremente el registro contador y lo visualiza en puerto B. La otra subrutina ***Verifica3*** se encarga de censar el switch RA3, si está

pulsado primero descarta cualquier rebote y luego complementa el valor del registro contador y lo visualiza en el puerto B.

Este proceso se repite constantemente.

2.10 Programa 1 para circuito "Display 7 Segmentos". Figura 4.

El propósito de este programa es visualizar un contador de 0 a 99 por los display de 7 segmentos. El programa incrementa automáticamente el conteo; el tiempo de encendido por cada display es aproximadamente de 5 mili segundos y el tiempo de apagado es de aproximadamente de 6 micro segundos para una frecuencia de multiplexación aproximada de 200 hz. En el programa se crea un bucle de aproximadamente 1 segundo por cada incremento. El código fuente se encuentra en la sección de anexos C como 7segmentos_1.asm. La figura 13. Muestra el diagrama de flujo para este programa.

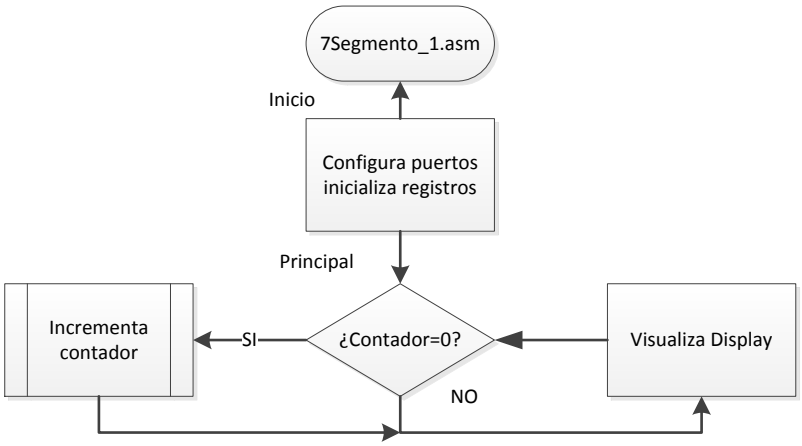


Figura 13. Programa 7Segmento_1

Inicio: La subrutina inicio configura dos registros que almacenan los valores para las unidades y las decenas llamados UNIDADES y DECENAS y a la vez los inicializa en cero. También configura el puerto B como salida para mostrar los valores en el display de 7 segmentos y el puerto A como salida los pines RA0 y RA1 que son los que activan y desactivan los transistores 2N3904 y lograr la multiplexación.

Principal: Esta subrutina hace un incremento del registro UNIDADES y verifica si se desborda con el valor de 9 para luego reiniciarlo en cero pero a la vez incrementa el valor del registro Decenas y también verifica si este se desborda en 9 para luego reiniciarlo en cero y volver a empezar el conteo. También está subrutina se encarga de visualizar el valor de las unidades y de las decenas en el display de 7 segmentos. El incremento los hace de forma automática.

2.11 Programa 2 para circuito “Display 7 Segmentos”. Figura 4.

Este programa es un poco parecido al anterior la diferencia es que se incrementa por medio del switch SRA4. El código fuente se encuentra en la sección de anexos C como 7Segmento_2.asm y la figura 14 muestra su respectivo diagrama de flujo.

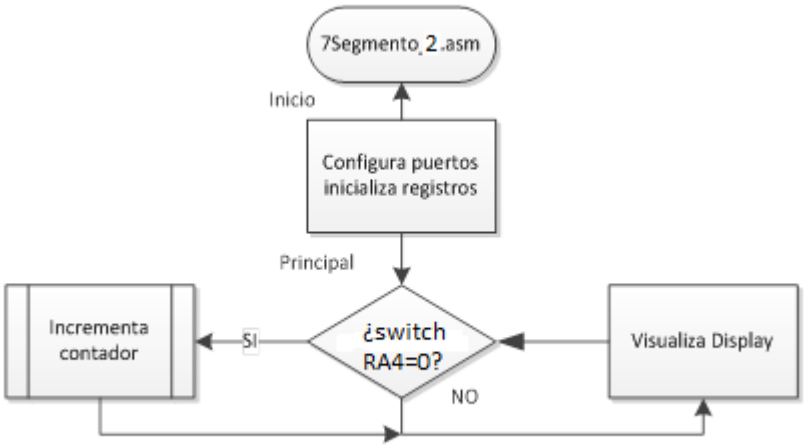


Figura 14. Diagrama de flujo para programa 7Segmentos_2.asm

Inicio: La subrutina inicio configura dos registros que almacenan los valores para las unidades y las decenas llamados UNIDADES y DECENAS y a la vez los inicializa en cero. También configura el puerto B como salida para mostrar los valores en el display de 7 segmentos y el puerto A como salida los pines RA0 y RA1 que son los que activan y desactivan los transistores 2N3904 y lograr la multiplexación. El pin RA4 como entrada para censar el pulsador de incremento.

Principal: Verifica si el switch RA4 está pulsado descarta los rebotes y si está pulsado hace un incremento del registro UNIDADES y verifica si se desborda con el valor de 9 para luego reiniciarlo en cero pero a la vez incrementa el valor del registro Decenas y también verifica si este se desborda en 9 para luego reiniciarlo en cero y volver a empezar el conteo. También esta subrutina se encarga de visualizar el valor de las unidades y de las decenas en el display de 7 segmentos. En este programa el incremento depende si el switch RA4 está pulsado.

2.13 Programa 1 para circuito “Display LCD”. Figura 5.

Para este programa se hace uso de las librerías “LCD_4BIT.INC”, “LCD_MENS.INC” y “RETARDOS.INC”. Las librería LCD_4BIT se encarga de configurar el LCD en modo 4 bit y su correcto envió de datos y la librería LCD_MENS contiene las subrutinas para el envío correcto de mensajes fijos o en movimientos. El propósito de este programa es mostrar un mensaje en el LCD. En la primera línea se muestra “UNAN MANAGUA” y en la segunda línea se muestra

"ING. ELECTRONICA". Su código fuente se llama DisplayLCD_1.asm y está en la sección de anexos C. y su diagrama de flujo se puede ver en la figura 15.

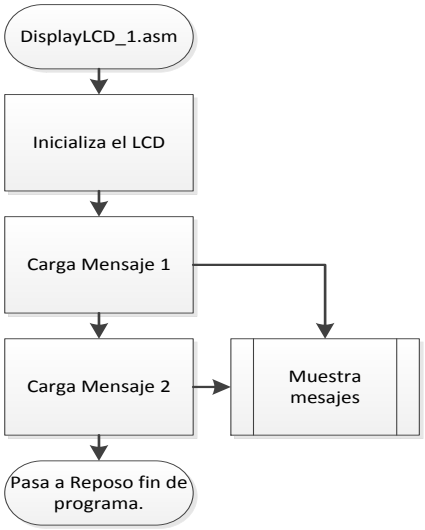


Figura 15. DisplayLCD_1 muestra dos mensajes

Inicializa LCD: Esta subrutina se encarga enviar las palabras claves que se necesitan para un correcto funcionamiento del display LCD que recomienda el fabricante (LM016L, 1998, págs. 23,24,25). Además reserva los registros que serán utilizados por las diferentes subrutinas.
Carga Mensaje1: Esta subrutina se encarga de cargar el mensaje "UNAN MANAGUA" en los registros para luego mostrarlos en el display LCD.
Carga Mensaje2: Esta subrutina se encarga de cargar el mensaje "ING. ELECTRONICA". En los registros para luego mostrarlos en el display LCD.
Pasa a reposo: Esta es una subrutina que utiliza la instrucción sleep del micro quedando en estado de reposo.

2.14 Programa 2 para circuito "Display LCD". Figura 5.

Para este programa se utilizaron las mismas librerías del programa anterior. Este programa muestra un mensaje moviéndose de derecha a izquierda. Esto se debe a que el mensaje a visualizar sobrepasa la capacidad del display de 16 caracteres. El mensaje mostrado es "Estudia programación de microcontroladores" y se crea un bucle infinito para estar siempre mostrando el mismo mensaje. Su código fuente se llama DisplayLCD_2.asm y se encuentra en anexos C. Su diagrama de flujo puede verse en la figura 16.

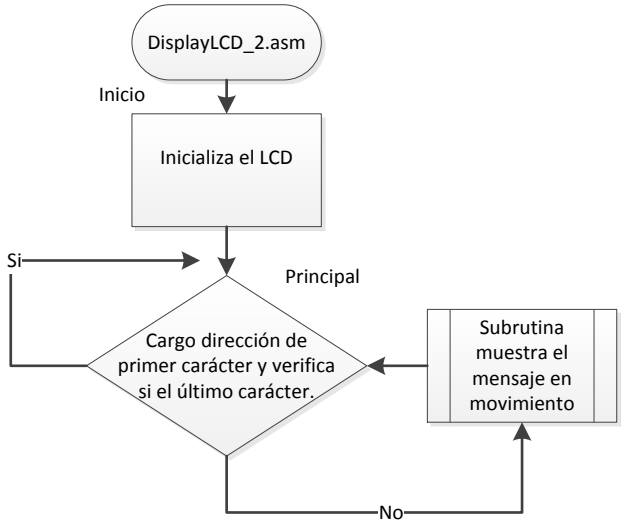


Figura 16. Diagrama de bloques DisplayLCD_2.

Inicio: Esta subrutina se encarga enviar las palabras claves que se necesitan para un correcto funcionamiento del display LCD que recomienda el fabricante (LM016L, 1998, págs. 23,24,25). Además reserva los registros que serán utilizados por las diferentes subrutinas.

Principal: Esta subrutina carga cada carácter en los registros para luego ir mostrándolos en el display LCD. También verifica un registro contador el cual cuenta el número de caracteres mostrado. Si el número de caracteres es 16 entonces la subrutina desplaza el mensaje hacia la izquierda creando el efecto de mensaje en movimiento.

2.15 Programa 1 para circuito "Teclado hexadecimal". Figura 7.

En este programa se utilizaron las librerías "TECLADO.INC", "LCD_4BIT" Y "RETARDOS.INC". La librería TECLADO inicializa el teclado verificando que no esté ninguna tecla pulsada y activando la resistencia de Pull-Up de microcontrolador además de otras funciones importantes. En pantalla aparece el código de las teclas que se van pulsando. Cuando llega al final de la primera línea, pasa a la segunda línea. Cuando llega al final de la segunda línea borra todo y comienza de nuevo al principio de la línea 1. Se utilizó el recurso de interrupciones por cambio de estado del puerto B del microcontrolador para detectar cuando una tecla esta pulsada.

También se hizo uso del módulo "Display LCD" para visualizar los caracteres. Se muestra a continuación su diagrama de flujo. Figura 17. El código fuente se puede ver en la sección de anexos C con el nombre de Teclado_1.asm

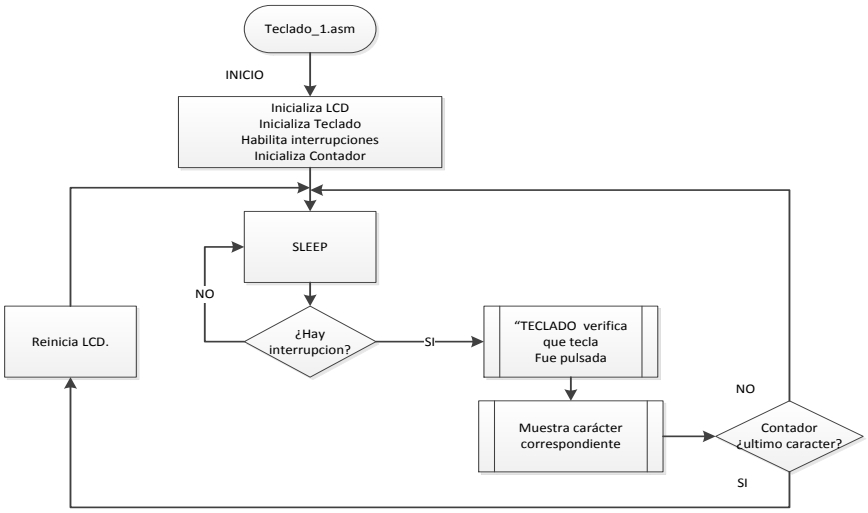


Figura 17.Diagrama de bloques del programa Teclado_1

Inicio: Consta de cuatro subrutinas. La subrutina *Inicializa LCD* configura el LCD como se explicó en los programas para el módulo LCD. La subrutina *Inicializa Teclado* se encarga de configurar las resistencias de pull-up para la exploración del teclado, verifica si no hay una tecla pulsada antes de iniciar el programa. La subrutina *Habilita Interrupciones* configura el tipo de interrupción que para este caso es interrupción por cambio de estado. La subrutina *Inicializa contador* se encarga de inicializar los registros contadores en cero.

Sleep: Al estar el microcontrolador en estado de reposo solo están habilitadas la interrupciones, si se produce una interrupción se llama a la subrutina *Teclado verifica* el cual censa que tecla fue pulsada para llamar a otra subrutina que se encarga de mostrar en el LCD la tecla correspondiente. También se incrementa un contador el cual sirve para verificar si va haber un cambio de línea de la línea 1 a la línea 2 del lcd. Si es final de línea 2 el programa vuelve a reiniciarse al último estado antes del sleep.

2.16 Programa 2 para circuito "Teclado hexadecimal". Figura 7.

El objetivo de este programa es verificar si una clave es correcta para luego activar un dispositivo. El programa pide el ingreso de la clave de cinco dígitos (ABCDF para este programa), en la primera fila del LCD aparece el mensaje "Ingrese CLAVE" y en la segunda línea aparece asteriscos por cada dígito ingresado. Si la clave digitada es correcta aparece "Clave CORRECTA" se enciende el led conectado al puerto RB3 por unos segundos luego se apaga y el proceso se repite. Si la clave ingresada es incorrecta se muestra "Clave incorrecta" y el proceso se repite.

Para este programa se utilizaron las mismas librería que el programa anterior más la librería "LCD_MENS.INC" que es la encargada de controlar los mensajes a mostrar, junto con el módulo LCD. La subrutina principal es "Servicio Interrupción" y su diagrama de flujo se muestra en la figura 18. Su código fuente se llama Teclado_2.asm puede verse en la sección de anexos C.

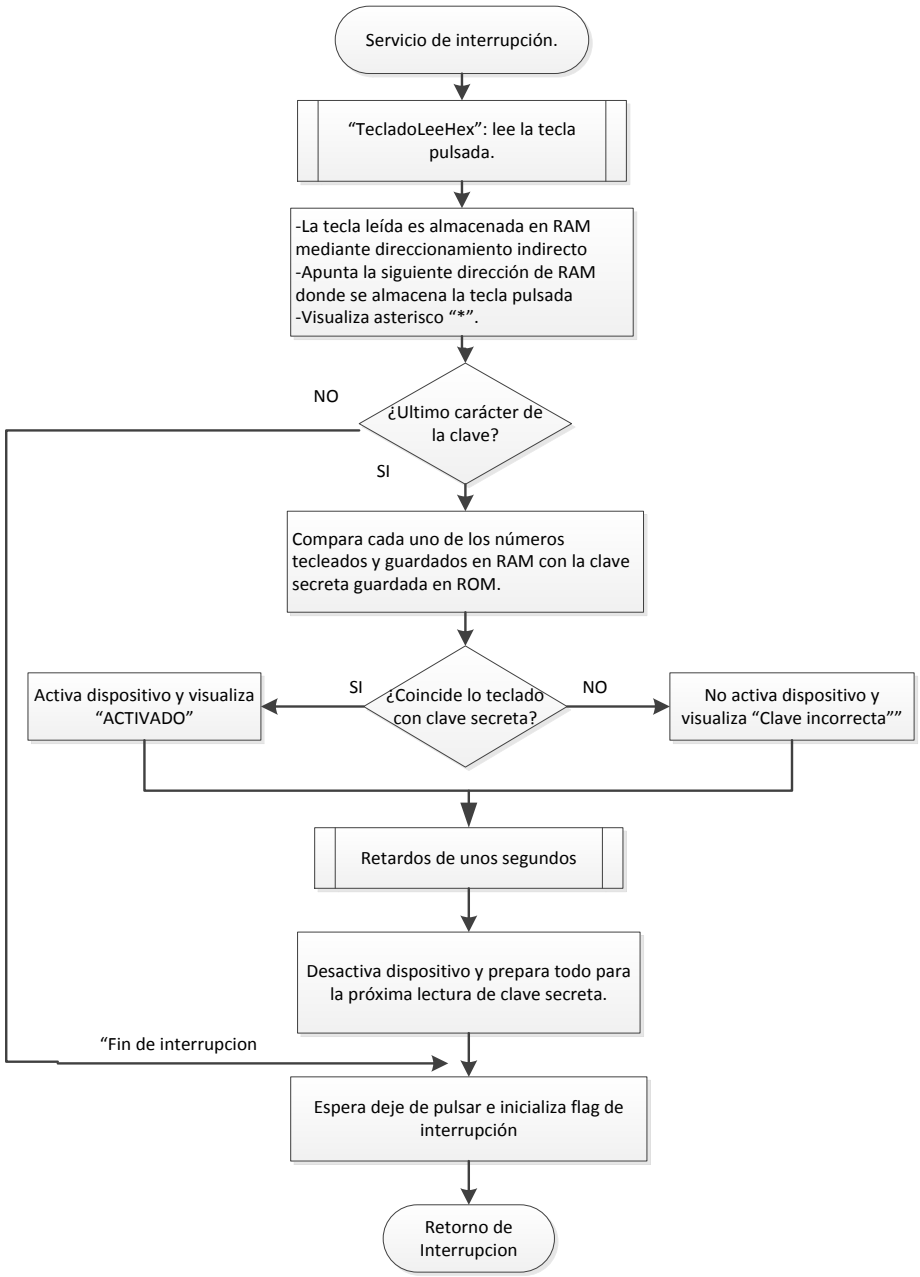


Figura 18. Diagrama de bloques de subrutina Servicio Interrupción".

Teclado lee Hex: Se encarga de leer el valor de la tecla pulsada. Luego los almacena en los registro de propósitos generales y se encargar de mostrar asteriscos en el LCD.

Ultimo carácter de la clave: Verifica si ya ingresaron los todos los dígitos de la clave (el cual depende del tamaño de la clave ya sea de 3,4 o 5 dígitos). Si ya están ingresados todos los dígitos compara estos dígitos con los dígitos de la clave guardada en la ROM del programa. Si los dígitos no coinciden con la clave

guardada se muestra un mensaje de Clave incorrecta. Si coinciden se visualiza un mensaje de Activado. Luego reinicia todo el programa

2.17 Programa 1 para circuito "Bus I2c". Figura 9.

El dispositivo seleccionado para este programa es la memoria 24C04, que tiene una capacidad de 512 byte dividida en dos páginas de 256 byte que van de la posición 0 hasta FF, la dirección configurable en el circuito es 000 y su dirección fija para este dispositivo es 1010, los pasos a seguir para comunicarse con este dispositivo son los siguientes:

1. START: Esta se da cuando la señal SDA pasa a bajo mientras la señal SCL está en alto y marca el inicio de la comunicación.
2. PALABRA CONTROL: Es una serie de 8 bit dividida en cuatro partes para esta memoria en específico:
    - Bit R/W el menos significativo si está en 0 se escribe en la eeprom, si está en 1 se lee en la eeprom.
    - A0 es el bit de selección de página dentro de la memoria, si está en 0 selecciona la primera página, si está en 1 selecciona la segunda página. Este sería el bit 1 de la palabra de control.
    - A1, A2 esto bit son la dirección configurable que podría ser de 00 hasta 11 para un total cuatro memorias C04. Y corresponden a los bits 2 y 3 de la palabra de control y los pines 2 y 3 de la eeprom. En nuestro trabajo está fijada en 00.
    - Dirección fija esta parte es obligatoria para esta memoria y no se puede cambiar para las memoria 24Cxx es 1010 y corresponden a los bits 4 al 7 de la palabra de control.

3 Bit de Acknowledge (ACK): es un pulso que envía el dispositivo esclavo para indicar si los bits fueron transmitidos con éxito, si está en bajo la operación fue exitosa si está en alto la operación falló y tendría que reiniciarse.

4 Dirección dentro de la página: es la que selecciona la posición dentro de la página seleccionada donde se va leer o escribir y va desde 00 a FF.

5 DATOS: son los datos que se envían ordenados en bytes o paquetes de 8 bits.

6 STOP indica el fin de la comunicación.

La librería BUS_I2C.INC contiene las subrutinas que cumplen con estos requisitos y está ajustada a una velocidad de transmisión de 100 Kbps aproximadamente. La librería M24CXX.INC contiene las subrutinas necesarias para escribir o leer mensajes en la memoria 24C04.

El siguiente programa tiene como propósito escribir y leer en una memoria eeprom serie 24C04 y está dividido en dos partes que son:

1. Escribe un mensaje grabado en el programa del PIC16F84A a partir de la posición 0 de la primera página. El mensaje es "UNAN MANAGUA".
2. Lee el mensaje grabado anteriormente en la memoria 24C04 y lo visualiza en la pantalla del módulo LCD. Ver diagrama de flujo en figura 19. Su

código fuente se puede verificar en la sección de anexos C con el nombre BusI2C_1.asm

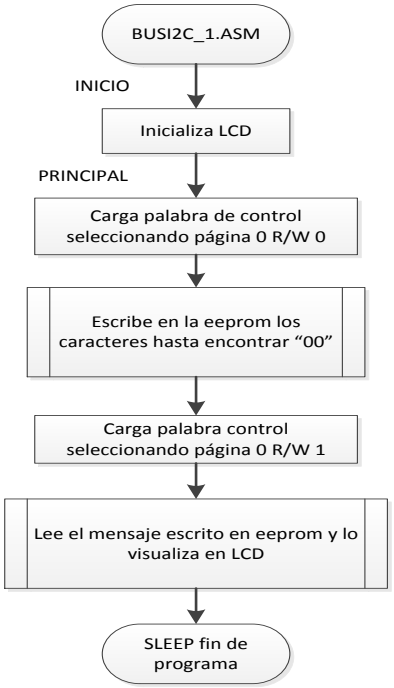


Figura 19. Para problema BUSI2C_1.ASM

Inicio: Configura el LCD según especifica el fabricante (Ver programas para módulos LCD.

Principal: El microcontrolador envía una serie de datos que son obligados para esta memoria según especifica el fabricante (24C04, 2004, págs. 5,6,7). El cual es una palabra de control que especifica si la acción es de escritura o de lectura. Esta subrutina hace uso de la de la librería BUSI2C.INC el cual se encarga de cumplir con las normas para este protocolo como puede verificarse en la sección de anexos donde están las librerías.

2.18 Programa 2 para circuito BUS I2C. Figura 9

El propósito de este programa es leer el mensaje grabado en la memoria eeprom 24C04. Este mensaje fue previamente grabado en la eeprom con el software ICPROG. Se escribió en la segunda página para fines didácticos de la librería M24CXX.INC. Figura 20. Muestra el diagrama de flujo. Se puede ver su código fuente en la sección de anexos C llamado BUSI2C.asm

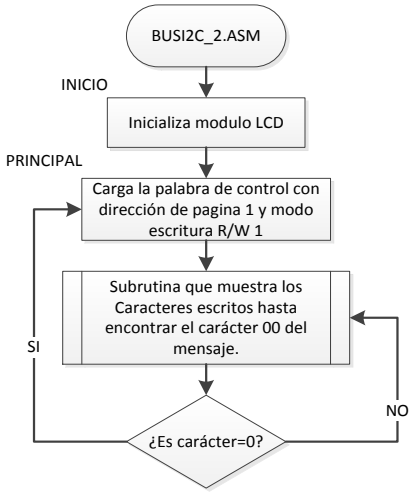


Figura 20 para problema BUSI2C_2.ASM

Inicio: Inicializa el LCD como se ha venido explicando en los programas que utilizan este dispositivos.
Principal: Utiliza las subrutinas que se encuentran en las librerías BUSI2C.INC y M24CXX.INC para enviar las palabras de control cumpliendo con el protocolo I2C para este ejemplo se configura el inicio de lectura en la página 2 de la memoria. También es importante ver que la subrutina *¿Es carácter=0?* Se encarga de verificar si el carácter leído es el carácter nulo (0 no como ascii) que determina si es fin de mensaje. Otro aspecto importante es la subrutina que se encarga de mostrar los caracteres verifica si sobrepasa los dieciséis caracteres para empezar a desplazar el mensaje de izquierda a derecha como se puede verificar en la librería M24CXX.INC.

2.19 Programa 1 para lectura escritura de eeprom interna.

Los microcontroladores PIC poseen una memoria interna de 64 byte y en muchos proyectos es útil su utilización para guardar datos que mantengan su valor aun después de desconectado el dispositivo, otros la utilizan para guardar la información del firmware de sus programas etc.

En este programa se utiliza para guardar un contador que se incrementa cada vez que una persona acciona y pulsador. Esto puede ser utilizado en una oficina de atención al cliente por ejemplo donde a las personas se les da un número para su respectiva atención.

Las librerías utilizadas son EEPROM.INC es la que se encarga de leer el dato o escribirlo en la memoria eeprom y configura sus respectivos registros. BIN_a_BCD.INC esta librería convierte un número binario puro a código BCD esto es necesario porque la librería LCD_4BIT.INC contiene una subrutina llamada LCD_BYTE y muestra su valor en el LCD. La librería LCD_MENS que se encarga de los mensajes y RETARDOS que se ha venido utilizando en todos los programas. Para verificar este programa se activa el circuito de la figura 5. En la interface entrenadora. La figura 21. Muestra el diagrama de flujo para este

programa y su código fuente puede verse en la sección de anexos C nombrado como EEPROM_1.asm. Para incrementar el contador es necesario activar el switch SRA4 en la interfaz.

Para este programa solo hay que habilitar el módulo LCD y el switch SRA4.

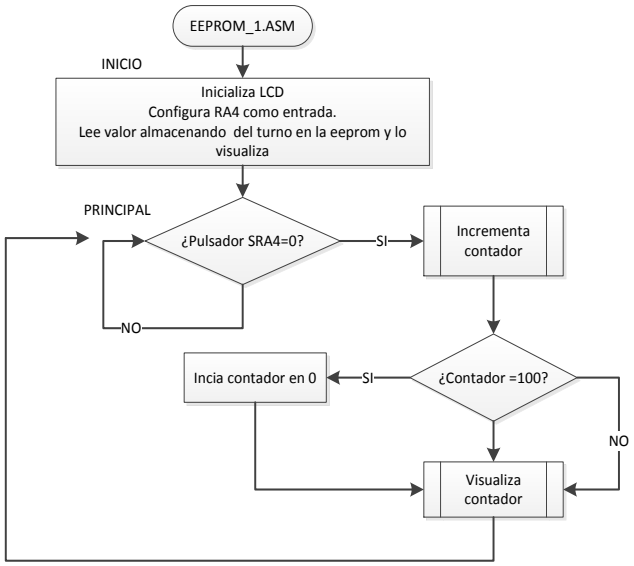


Figura 21. Para el programa EEPROM_1.

Inicio: Inicializa el LCD. Configura le puerto RA4 como entrada. La *subrutina Lee valor almacenado* lee el valor que se encuentra en el registro llamado Turno que se encuentra en la posición cero de la eeprom interna esta cumple con las normas establecida por el fabricante del PIC en la hoja de datos. (PIC16F84A, 2001, págs. 13,14.)

Principal: Verifica si el switch RA4 está pulsado. Si está pulsado incrementa el registro contador que se encuentra en la eeprom interna del PIC y a la vez comprueba si llego a 100 para reiniciarlo en cero a la vez que lo está visualizando en el LCD

CAPITULO 3. IMPLEMENTACION Y PRUEBA DE LA INTERFAZ ENTRENADORA.

3.1 IMPLEMENTACION

Para la implementación de la interfaz entrenadora se hizo uso de software Isis Proteus donde se colocaron todos los componentes que se diseñaron para la interfaz entrenadora dejando solamente el circuito del Teclado Hexadecimal que se hizo en una tarjeta aparte. Ver figura 22.

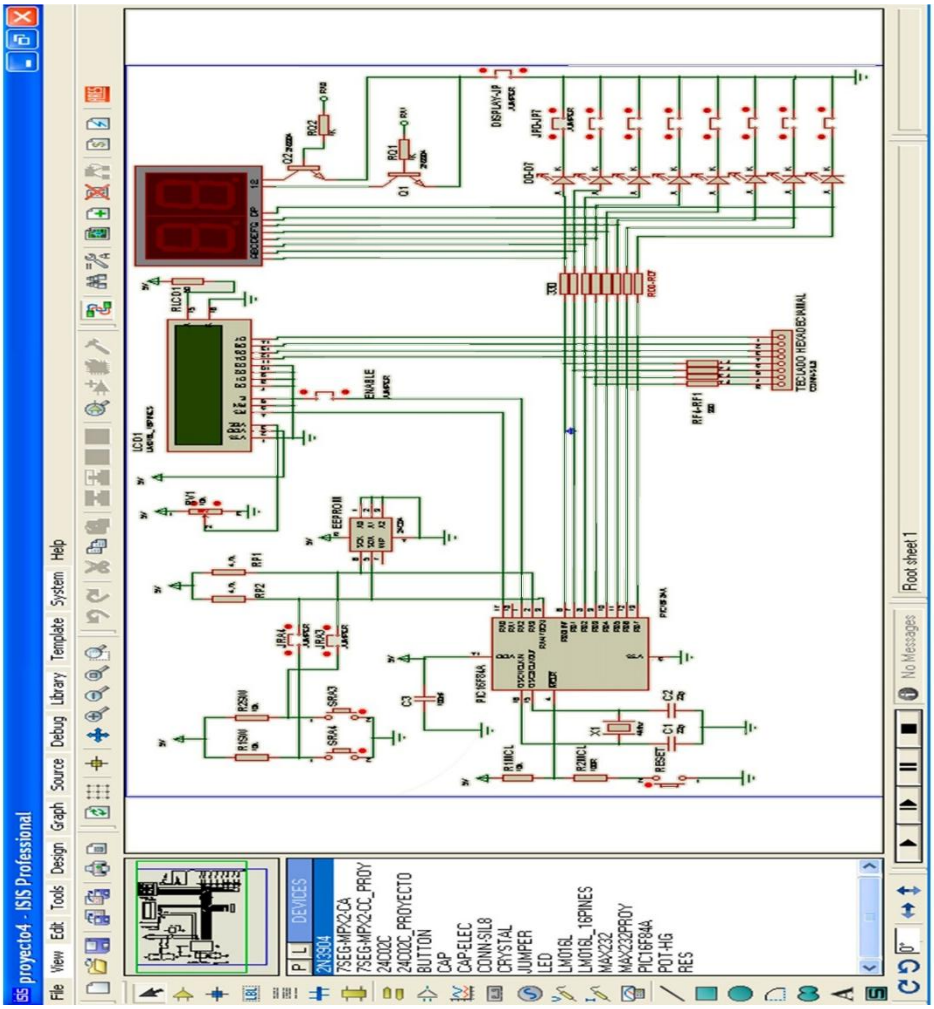


Figura 22 Diseño de la Interfaz Entrenadora en software Isis Proteus.

3.2 TECLADO HEXADECIMAL.

Para el teclado hexadecimal se colocaron los 16 switch, conectados en cuatro filas por cuatro columnas. Estas a la vez se conectan a un conector SIL8 para que pueda ser conectado a la tarjeta de la interfaz por medio de un bus de 8 líneas. Figura 23.

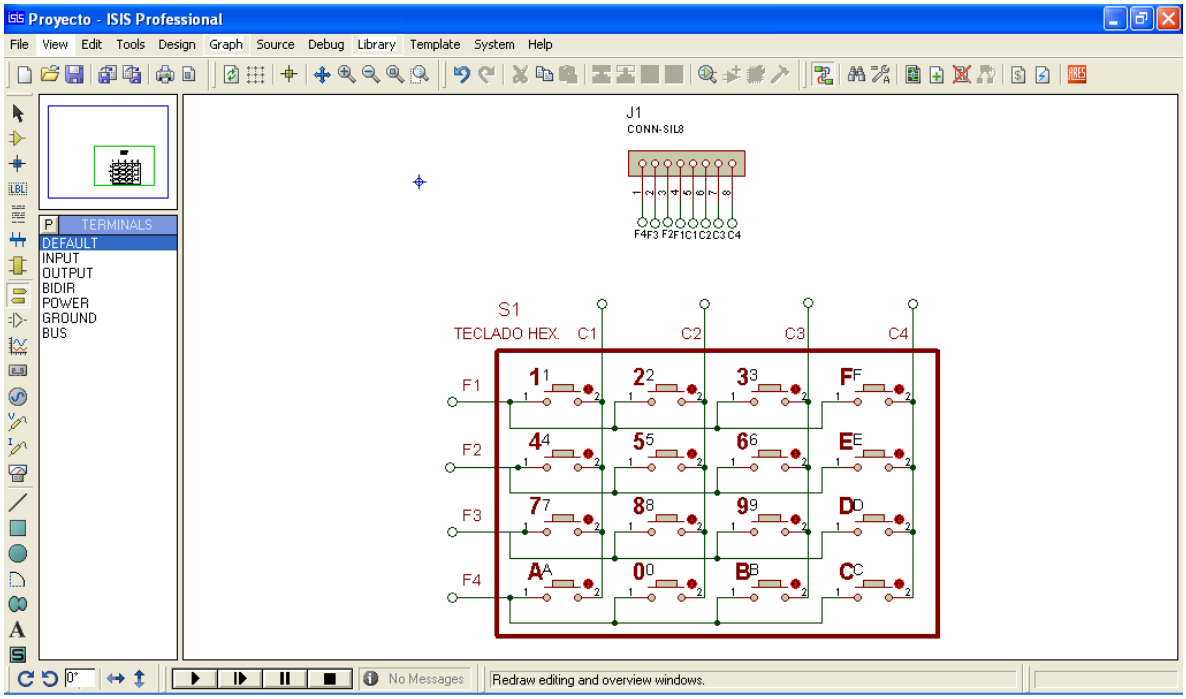


Figura 23.Teclado Hexadecimal en Isis Proteus.

3.3 DISEÑO DEL PCB DE LA INTERFAZ

El Isis Proteus tiene asociado el software Ares Proteus el cual se encarga de generar el ruteo de las conexiones para el circuito impreso y nos permite ver el circuito en 3D que es como quedaría de manera real. En la figura 24. Se muestra el circuito de la interfaz generada por el Ares Proteus en 3D. Y la figura 25 muestra el circuito impreso generado también por el Ares Proteus.

Las dimensiones de la tarjeta de la interfaz son de 20.03 cm por 15.03 cm y las dimensiones de la tarjeta del teclado hexadecimal son de 6 cm por 10 cm

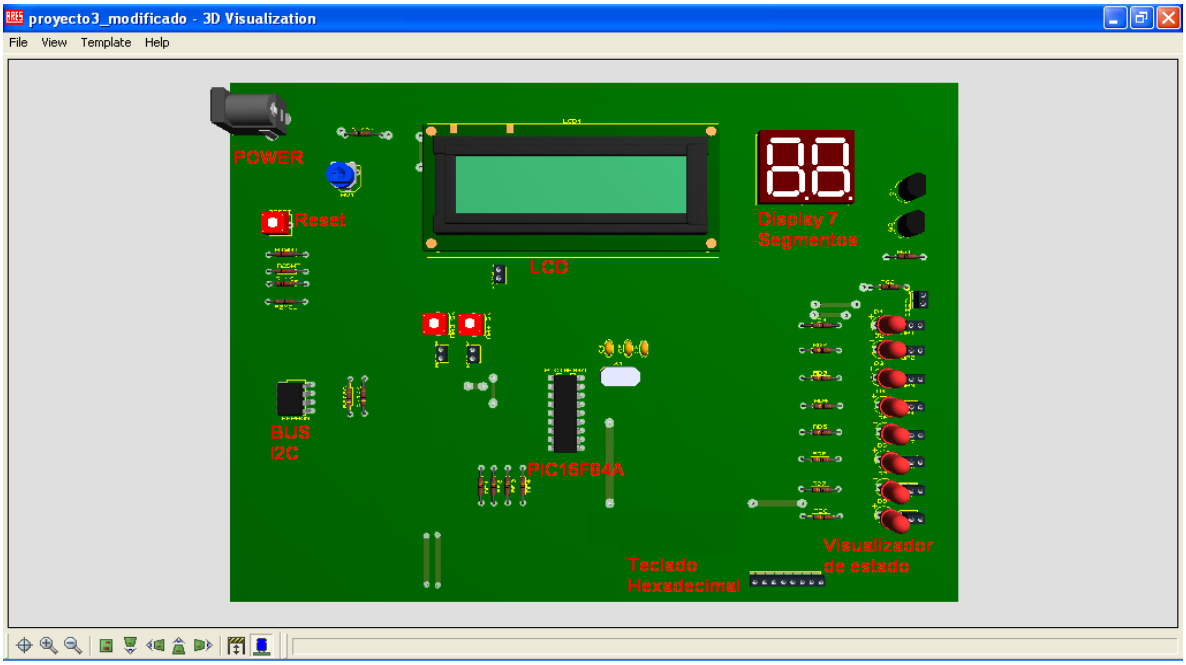


Figura 24. Interfaz Entrenadora en 3D generada por Ares Proteus.

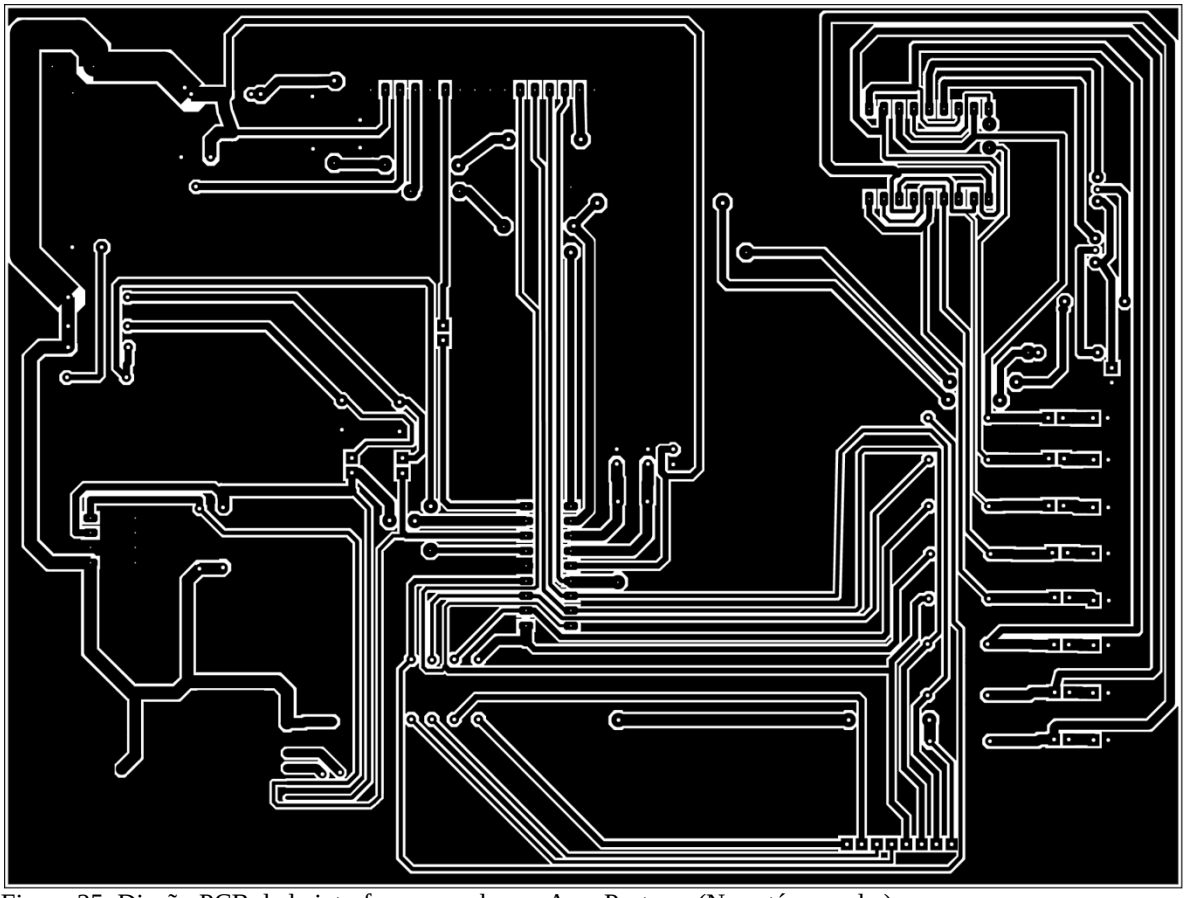


Figura 25. Diseño PCB de la interfaz generada por Ares Proteus. (No está a escala,)

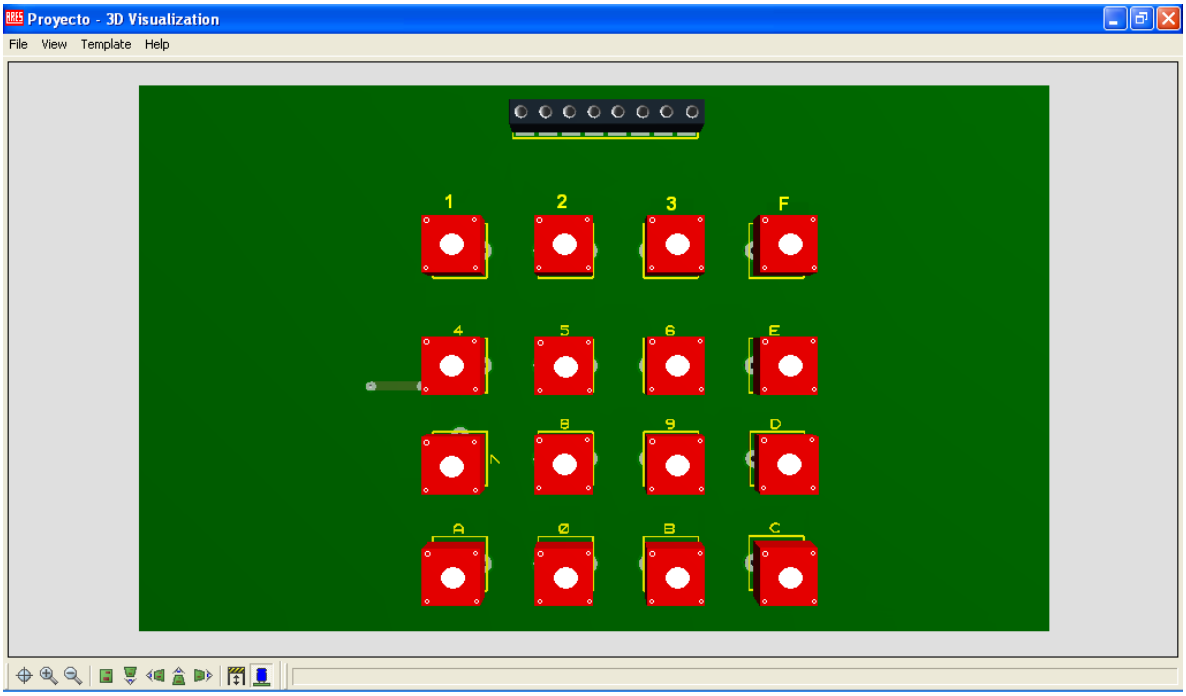


Figura 26 Teclado hexadecimal en 3D generado por Ares Proteus.

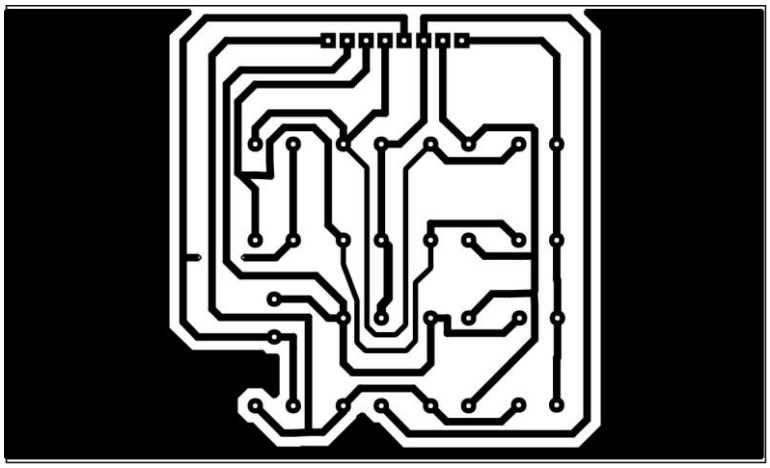


Figura 27. PCB del Teclado hexadecimal generado por Ares Proteus.

3.4 QUEMADO DEL CIRCUITO IMPRESO.

El método utilizado para el quemado del circuito impreso fue “método de termo transferencia” o “método del planchado”.

Este método consiste en imprimir en un papel fotográfico o papel Glossy el diseño del pcb generado por el Ares Proteus de la figura 25 y 27. La impresión tiene que ser hecha con impresora láser o de tóner, por la característica antiácida de este material. Una vez impresa se procede a juntar la cara de la tarjeta virgen de cobre con la cara de la impresión del diseño luego con una plancha se aplica calor al papel hasta que el dibujo se pegue a la superficie de cobre de la tarjeta virgen. Si ya está pegado significa que ya se transmitió la impresión al cobre y está listo para el siguiente paso que es el quemado con el ácido.

El ácido utilizado fue el percloruro de hierro, este se deposita en un recipiente plano asegurando que cubra toda la placa de cobre. El resultado del quemado de las placas se muestra en la figura 28 y 29.

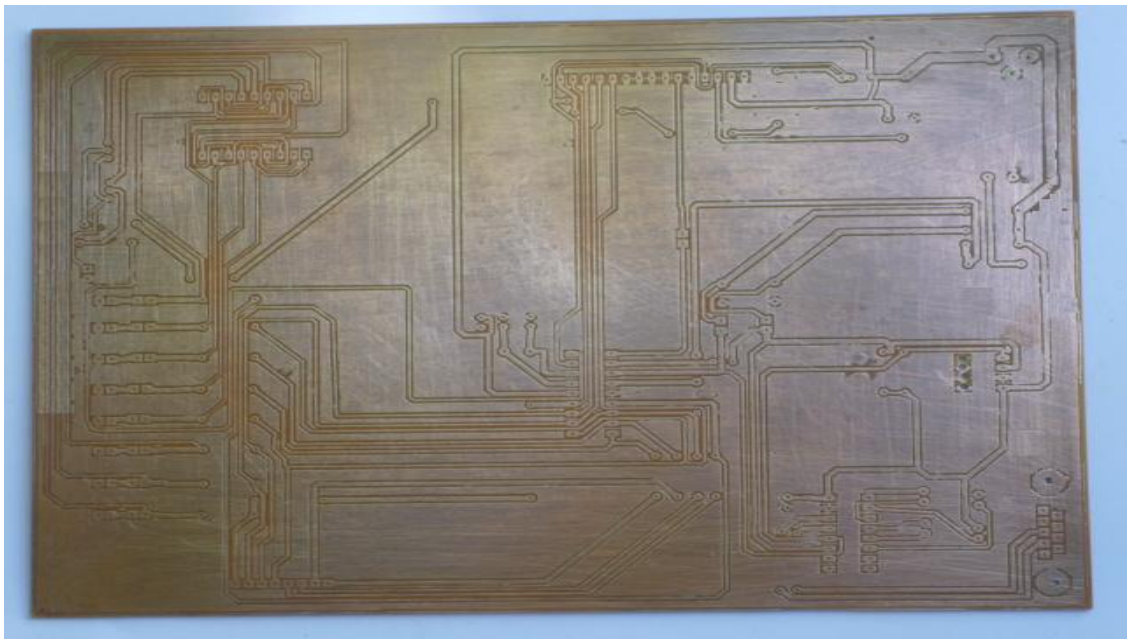


Figura 28. Placa de la interfaz recién quemada por el ácido.

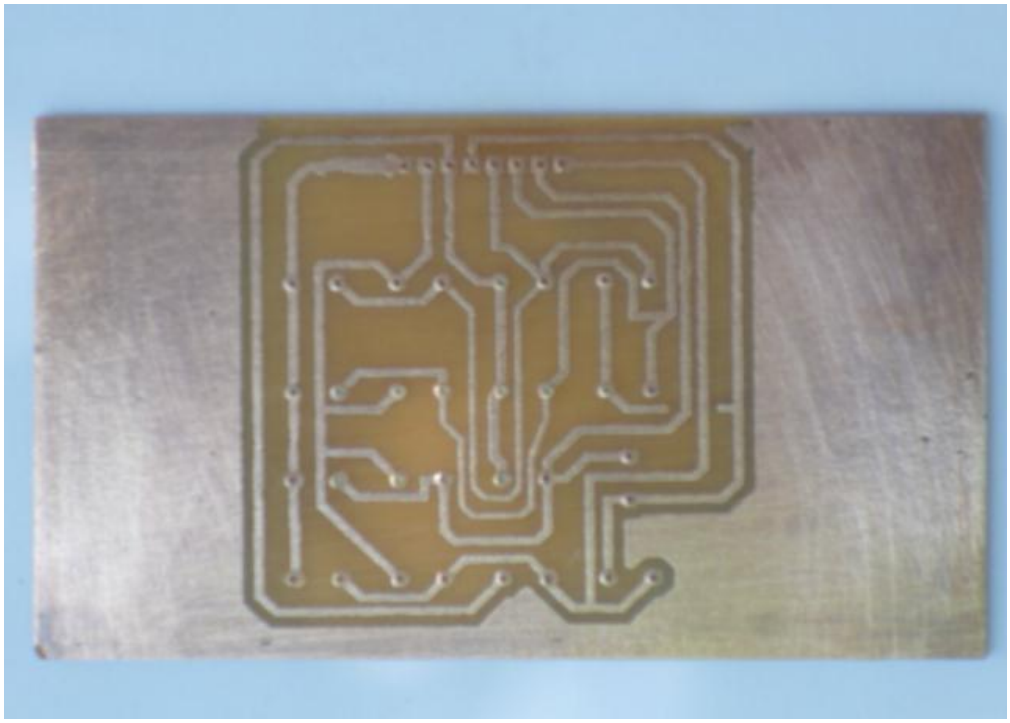


Figura 29. Placa del teclado hexadecimal quemada con el ácido.

3.5 MONTAJE DE LA TARJETA ENTRENADORA Y TECLADO HEXADECIMAL.

Una vez quemada con el ácido la placa de la Interfaz y del teclado se procedió a perforar las placas con una broca de tamaño 1/32 ‘’. Se montaron y soldaron todos los componentes quedando como se muestra en la figura.30 y 31.

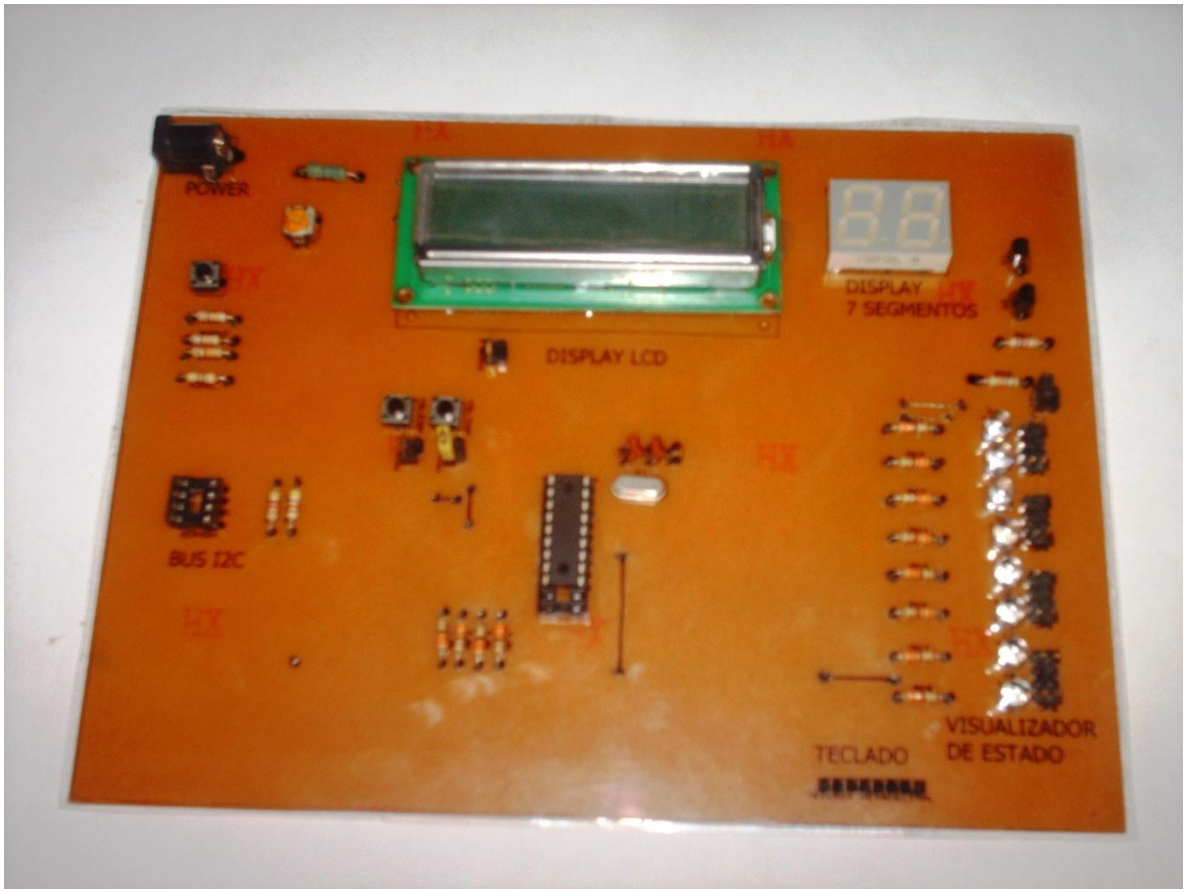


Figura 30. Interfaz entrenadora con sus componentes.

Figura 31. Teclado hexadecimal.

3.6 GUIA DE USUARIO. PRUEBA DE LA INTERFAZ.

Para probar la interfaz entrenadora es necesario contar con los siguientes software: MPLAB IDE que sirve para compilar los programas y generar los archivos con extensión .HEX, ICPROG que sirve para grabar los archivos .hex en el PIC y trabaja con el clonador JDM. Por esta razón en nuestro trabajo hemos incluido in disco CD que contiene los softwares mencionados, los códigos fuentes de los programas propuestos, los archivos .hex y otros archivos que son necesarios para este trabajo. A continuación se describen los pasos para un correcto uso de la interfaz entrenadora.

3.6.1 PASOS PARA COMPILAR LOS PROGRAMAS Y GENERAR EL ARCHIVO .HEX

En la carpeta MPLAB IDE que se encuentra en el CD de este trabajo se encuentra el instalador para este compilador. El asistente muestra el progreso de la instalación de principio a fin.

Una vez que el programa está correctamente instalado seguimos los siguientes pasos para un uso correcto de MPLAB IDE.

1. Crear una carpeta nueva con el nombre PIC16F84A en Mis Documentos, Escritorio o disco local C. La ubicación más recomendable es disco local C. y dentro de esta capeta crear otra nueva carpeta con el nombre del programa, por ejemplo Visualizador_1
2.  Iniciar el programa dando clic en el icono que se encuentra en el escritorio.
3. Una vez abierto el programa nos dirigimos a pestaña *Configure - Select Device* y seleccionamos el PIC16F84A.

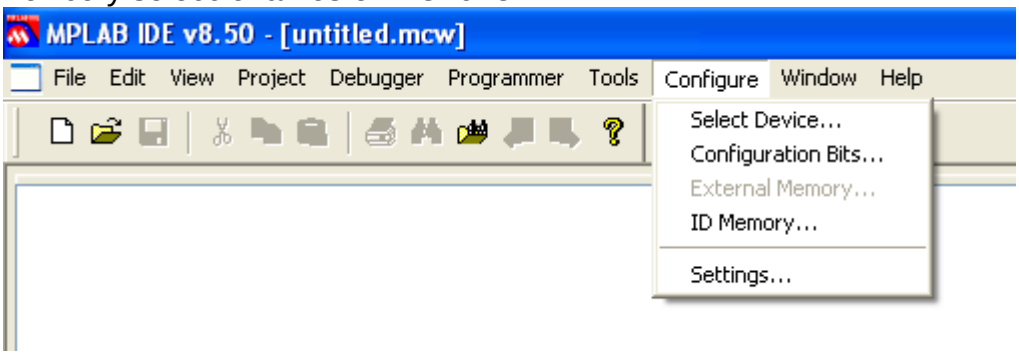


Figura 32. Selección del PIC16F84 en MPLAB.

4. Es conveniente activar el simulador para esto ir al menú Debugger Select Tool MPLAB SIM

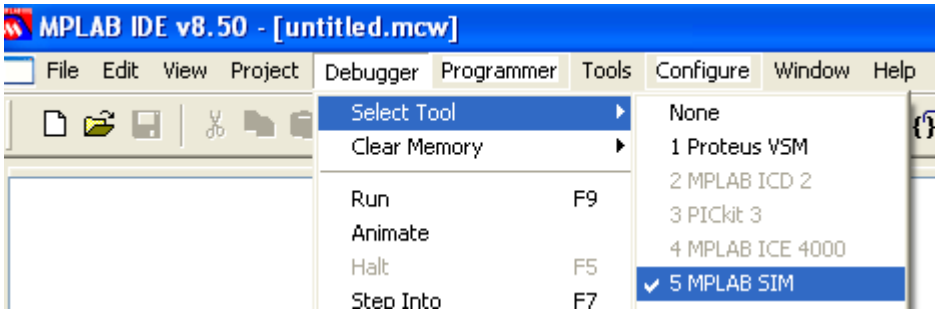


Figura 33. Selección del simulador MPLAB SIM.

5. Para configurar la frecuencia de reloj seleccionamos la pestaña *Debugger Settings Osc/Trace 4 MHz.*

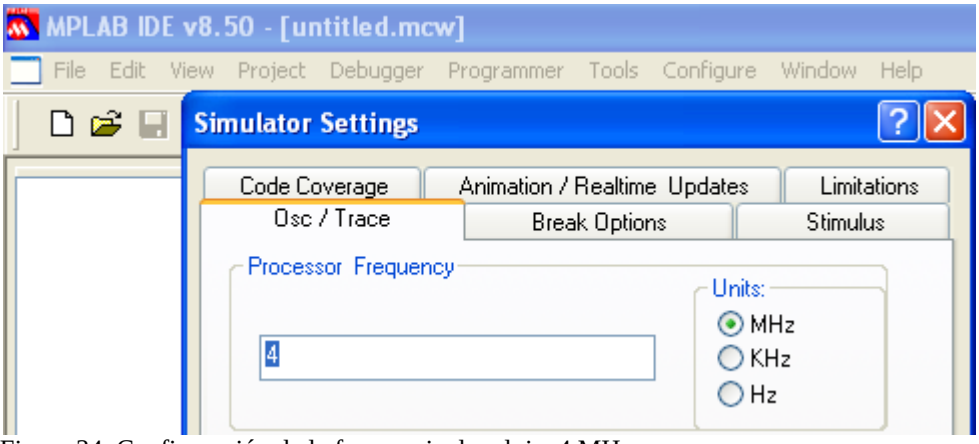


Figura 34. Configuración de la frecuencia de reloj a 4 MHz.

6. Para visualizar el número de líneas seleccionamos el menú *Edit Properties File Type Line Numbers.*

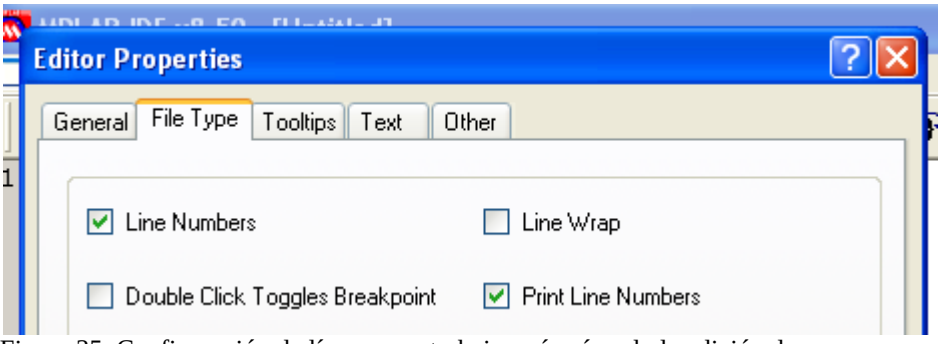


Figura 35. Configuración de líneas para trabajar más cómodo la edición de programas.

7. Una vez configurado el MPLAB le damos clic al icono *Nuevo o File New o control+N* y nos aparece el editor de texto esperando ingresemos los códigos para el nuevo programa.

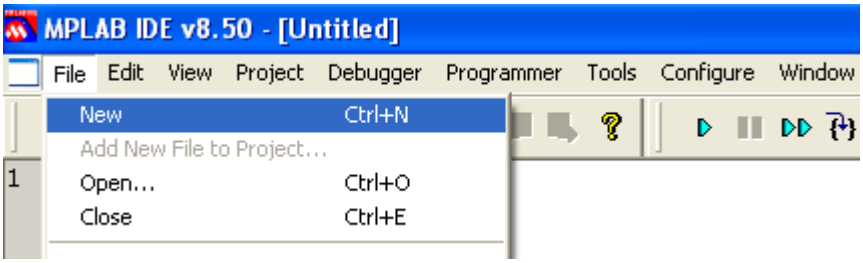


Figura 36. Seleccionar un nuevo programa o código fuente.

8. Para guardar el código fuente recién ingresado se debe ir al menú *File Save As* y buscar la ruta de la carpeta del programa donde se va a guardar y es importante que el nombre del programa finalice con la extensión .ASM. por ejemplo Visualizador_1.asm. También es importante que la o las librerías estén en la misma ruta donde se guarda el programa de lo contrario el compilador le generara un error. Estas librerías tienen la extensión .INC

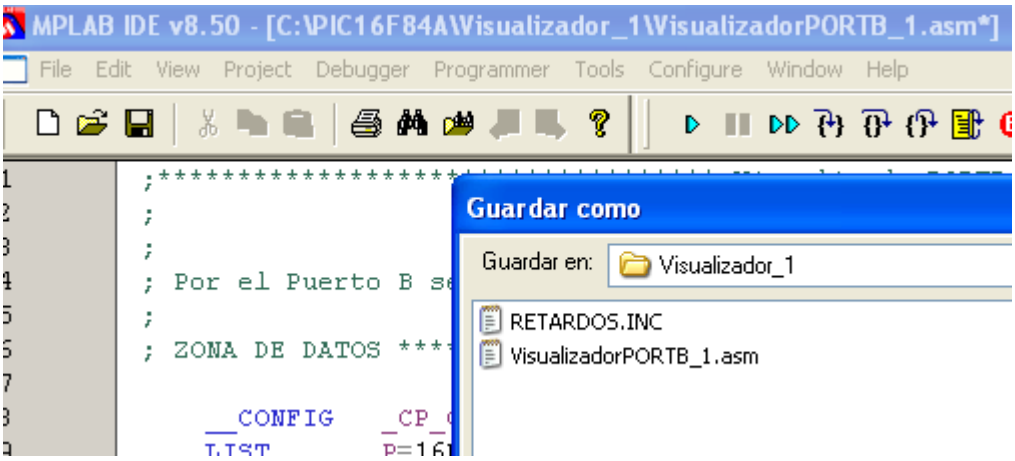


Figura 37. El programa debe ser guardado junto con la librería utilizada.

9. El siguiente paso consiste en compilar el código fuente, pero antes es necesario ver que en la carpeta donde está el código fuente y la librería no hay más archivos. Después de compilar el programa aparecen varios archivos que son generados por MPLAB entre ellos el .HEX Para compilar el programa vamos al menú *Project Quickbuild Visualizador_1.asm*

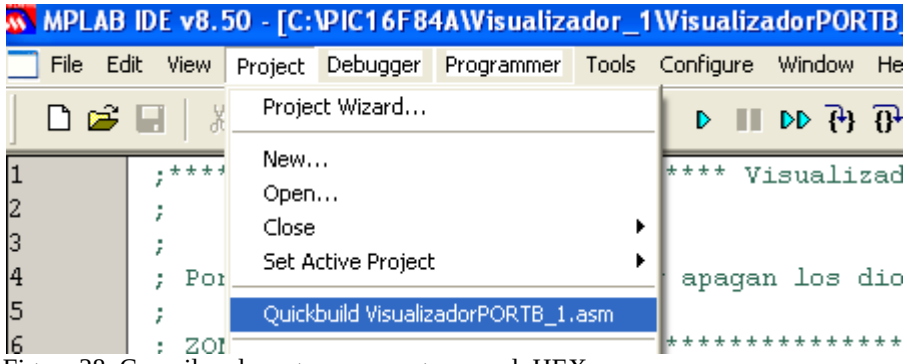


Figura 38. Compilar el programa para generar el .HEX

Si todo el proceso está correcto el compilador nos dice *Build Succeeded* sino el compilador genera los errores encontrados y la línea donde se encuentra dicho error para una mejor ubicación.

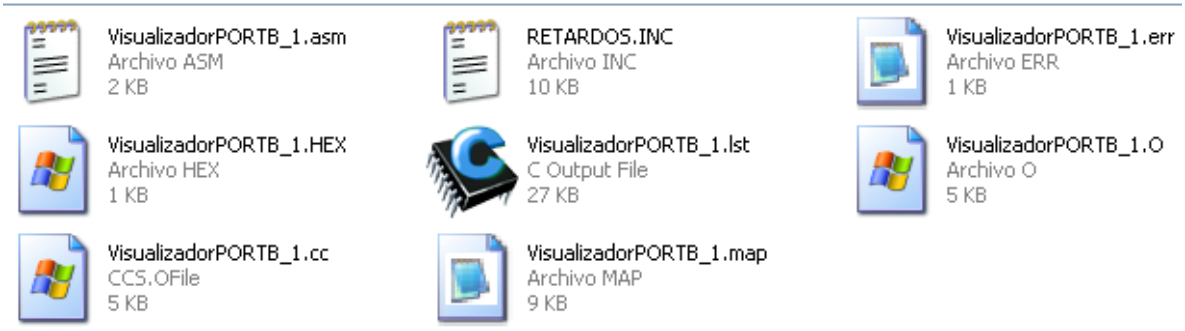


Figura 39. El MPLAB genera 6 archivos en la carpeta donde estaba el código fuente y la librería.

Para el caso que los programas ya están editados en el disco. Se debe ir al icono de abrir archivo y buscar la ruta donde están los archivos. Si se modifica el contenido de un programa se deben guardar los cambios antes de compilar el programa.

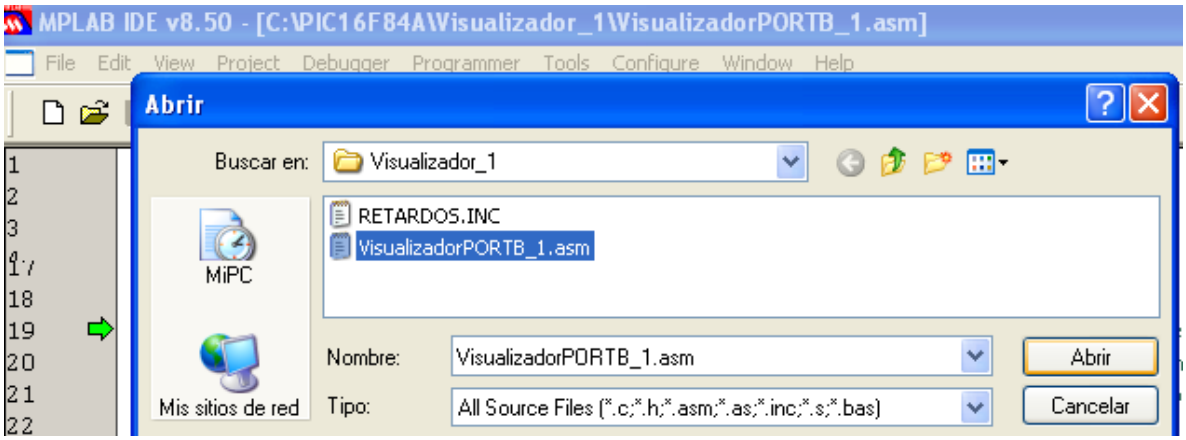


Figura 40. Se puede abrir un archivo existente y modificar su contenido.

3.6.2 PASOS PARA GRABAR LOS PROGRAMAS EN EL PIC16F84A.

El grabador que se utilizó para grabar el programa en el PIC es el JDM PROGRAMER y el software es el ICPROG.

Este programa se encuentra en el disco en la carpeta ICPROG. Se debe copiar la carpeta a Mis Documentos, Escritorio o disco local C. Damos clic en el icono del programa y se procede a configurar.

Paso 1: El software ICPROG se configuró como se muestra en la siguiente figura.

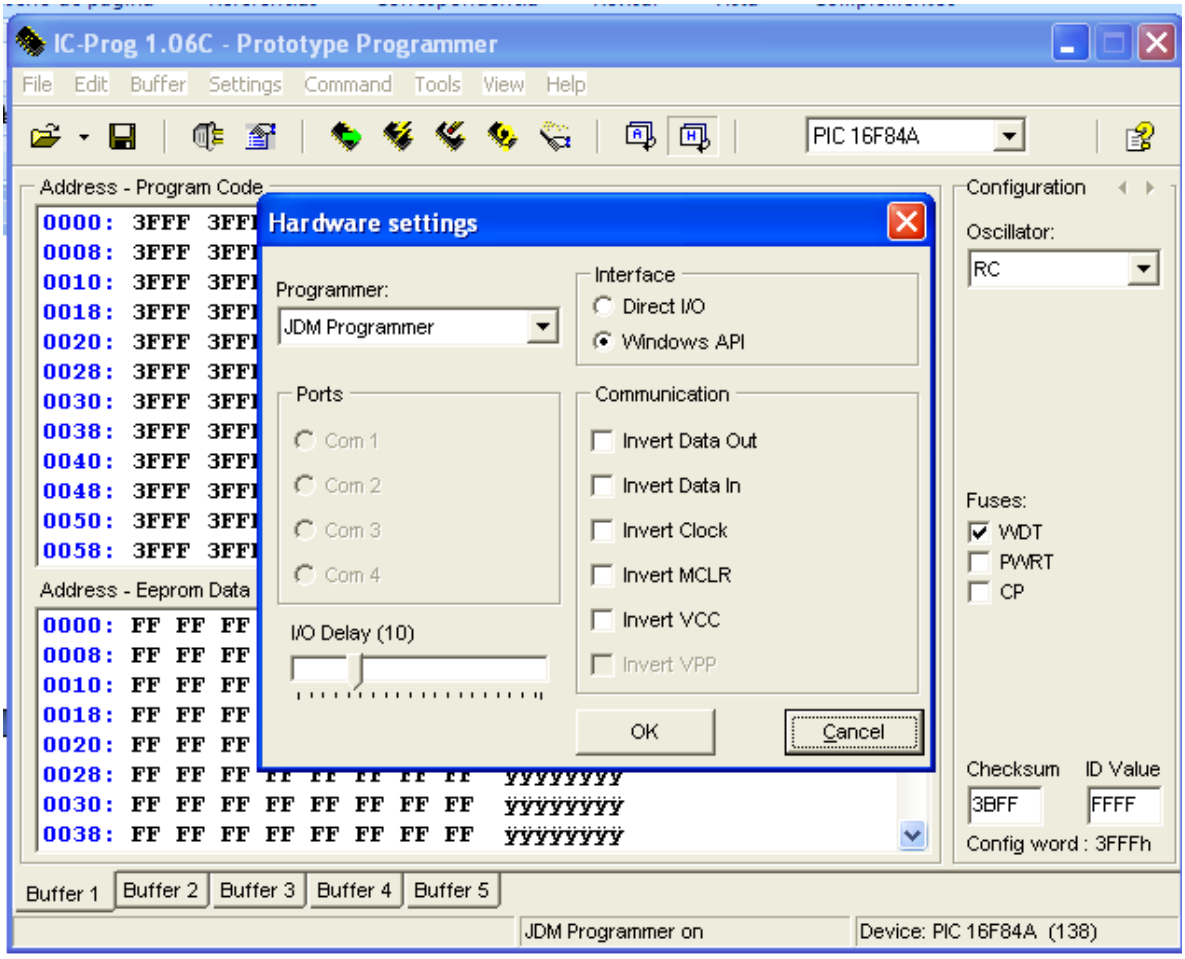


Figura 41. Configuración del software ICPROG

Si el programa presenta errores a la hora de grabar el programa, se puede modificar el valor de la casilla I/O Delay (10). En este caso tiene un valor por defecto de 10 pero puede subirse o bajarse de acuerdo a la PC que se utilice. En nuestro caso hubo problemas en una PC con el valor de 10 y se modificó a 5 y el problema se resolvió.

Paso 2: Una vez configurado el ICPROG se escoge el dispositivo a utilizar dando clic en la casilla de selección de dispositivo donde el programa muestra una gran cantidad de dispositivos con los que tiene compatibilidad. En nuestro caso seleccionamos el PIC16F84A como muestra la siguiente figura.

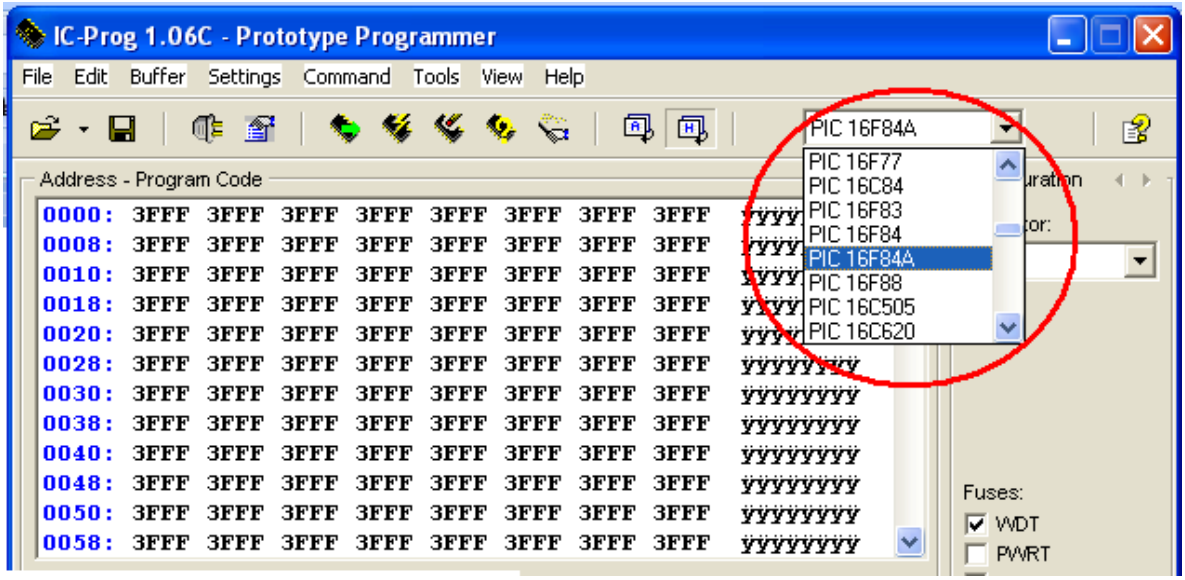


Figura 42. Selección de dispositivo. El PIC16F84A para este trabajo.

Paso 3: Luego de haber seleccionado el dispositivo procedemos a conectar el grabador JDM al puerto DB9 de la computadora con el PIC insertado en el socket de este clonador.


Figura 43. Grabador JDM para grabar el PIC se debe conectar al puerto serie COM1.

Paso 4: Con el grabador y el PIC conectado a la computadora se procede a borrar el contenido del PIC dando clic en el icono de borrado de dispositivo. En la siguiente figura se muestra el icono de borrado de dispositivo encerrado en un círculo para facilitar su ubicación.

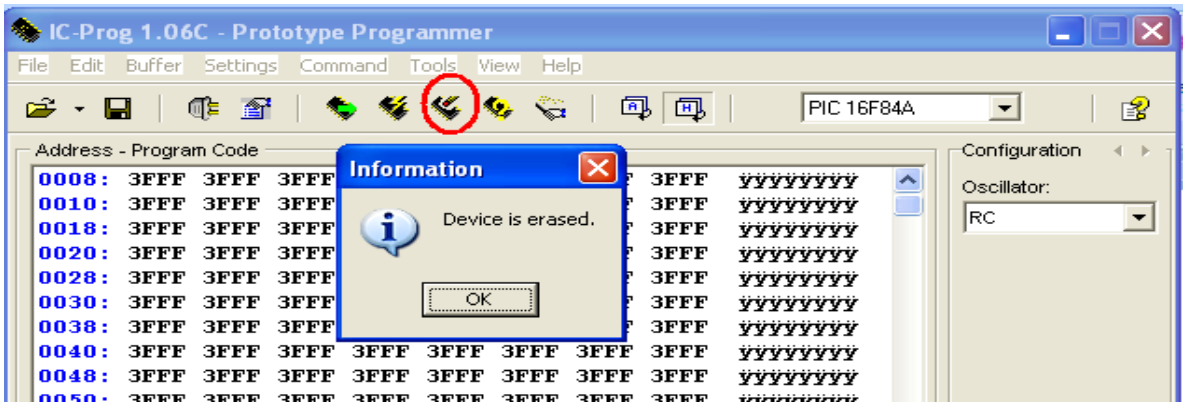


Figura 44. Borrado de dispositivo.

Paso 5: Una vez borrado contenido del dispositivo procedemos a abrir el archivo que se grabará. En este caso los que contienen la extensión .HEX que fueron generados por el software MPLAB IDE a la hora de compilar los programas.

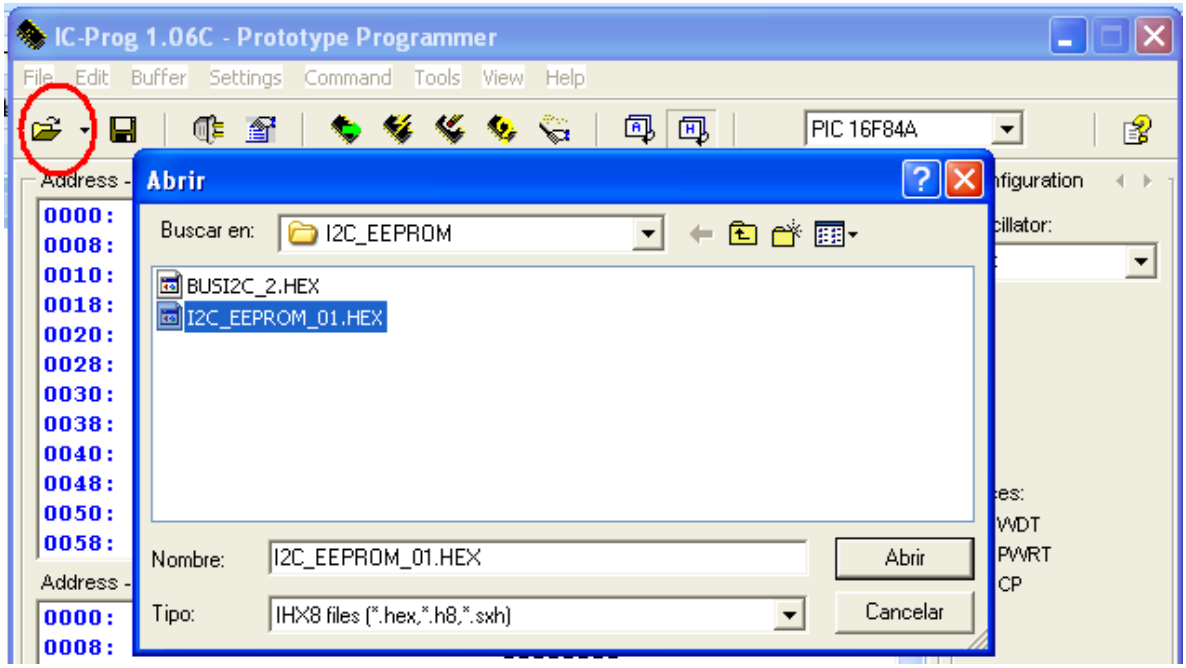


Figura 45. Los archivos a cargar son los que contienen la extensión .HEX

Paso 6: Finalmente se da clic en icono de programar todo para que el software ICPROG envíe la información al grabador por el puerto serial. En la siguiente figura se encierra en un círculo el icono de grabar todo para facilitar su ubicación se espera a que el programa termine de grabar. Si la operación fue exitosa se retira el clonador y el PIC y se procede a montar en la interfaz entrenadora. Si la operación fue fallida se repiten los pasos teniendo en cuenta la explicación para la figura 41.

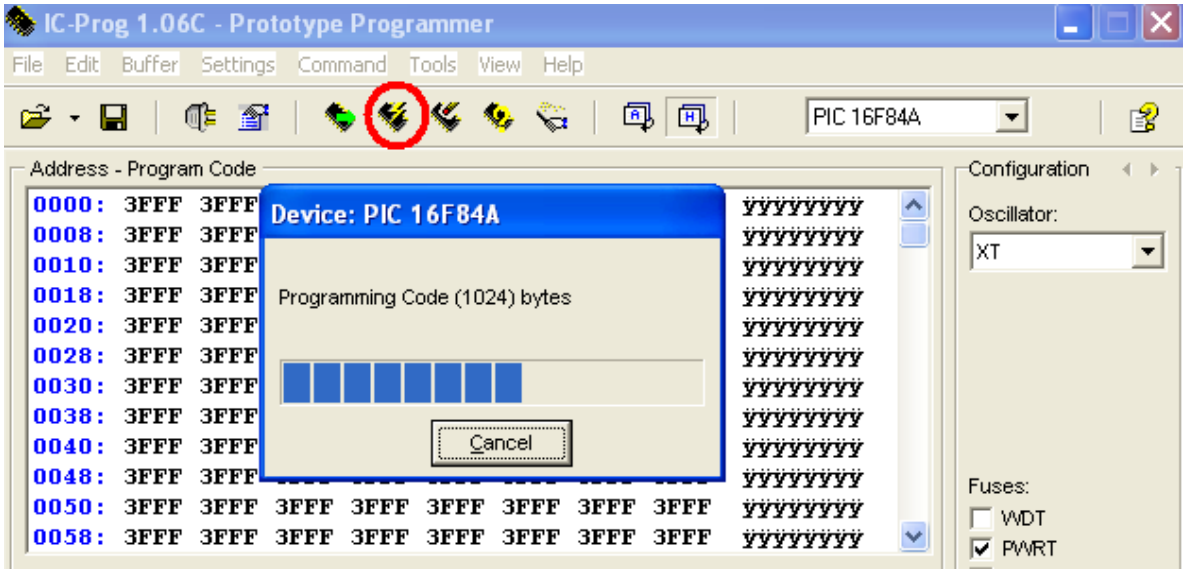


Figura 46. Proceso de grabación del PIC16F84A.

Paso 7: Este paso sirve para leer el contenido del PIC. Antes de leer el contenido se debe asegurar que el clonador JDM y el PIC estén conectados al puerto serie COM1 de la computadora. Luego hacer clic en el icono de leer todo como se muestra en la figura 47.

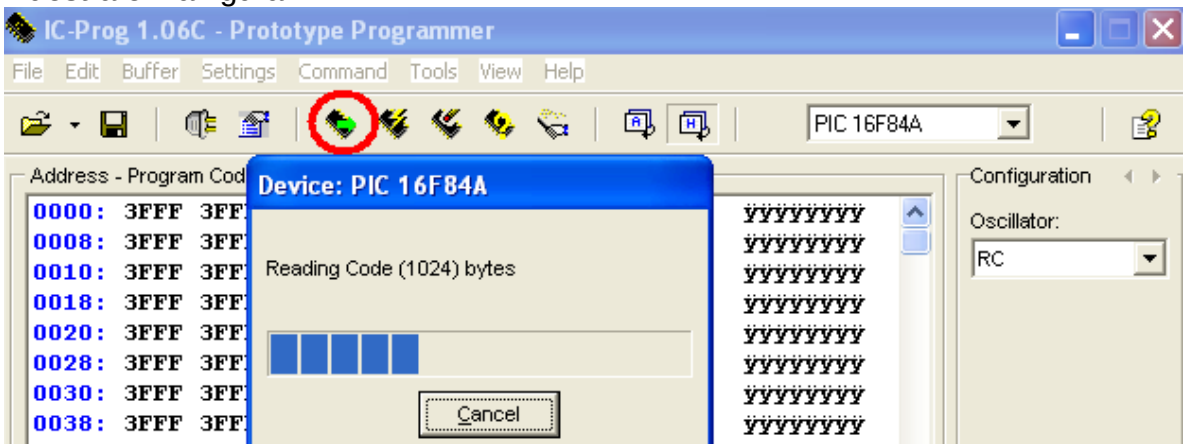


Figura 47. Procedimiento para leer el contenido del microcontrolador.

3.6.3 PASOS PARA GRABAR LA EEPROM CON EL GRABADOR JDM.

1. Para grabar memoria nos ubicamos en *Settings Device I2C Eeprom 24c04*

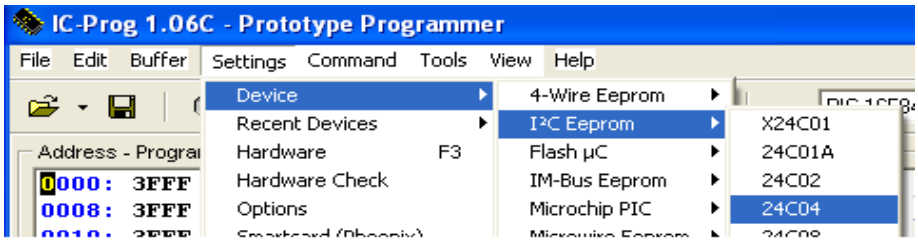


Figura 48. Selección de la memoria eeprom en el ICPROG.

2. Para editar el contenido de la memoria es preferible ubicarse en la columna de caracteres ascii y con el teclado digitar el mensaje. En la figura 49 se muestra encerrado en un círculo la zona de caracteres ascii.

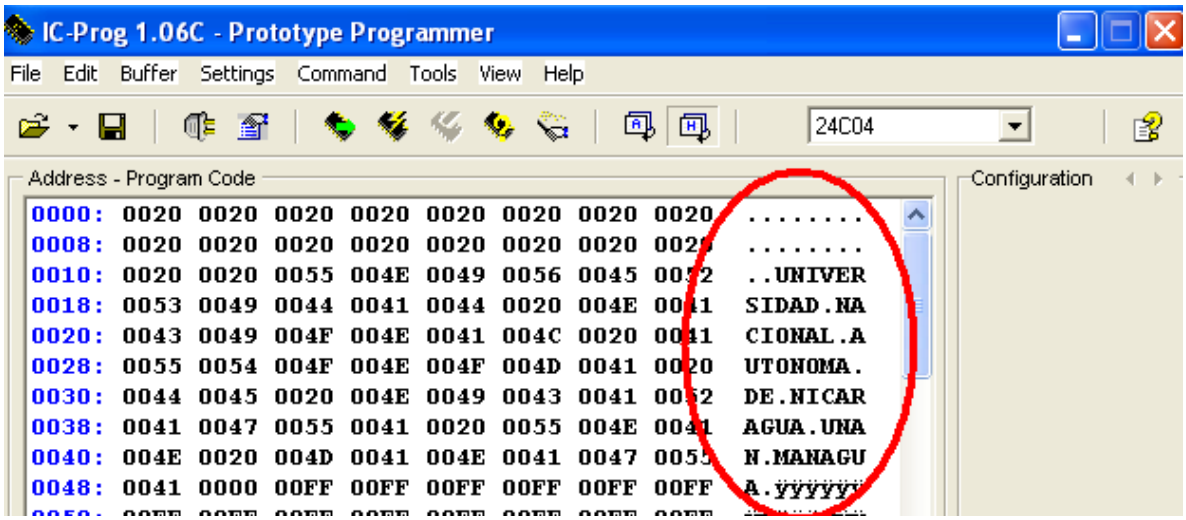


Figura 49. Zona de caracteres ascii el mensaje empieza con 16 espacios y termina con el carácter A.

3. Es importante que después del último carácter del mensaje se debe ingresar el valor 00 en hexadecimal no el carácter '0' para esto damos doble clic en la posición de la memoria inmediatamente después del último carácter. Para el caso de la figura 50. el último carácter es 'A', 0041 en hexadecimal, en la siguiente posición se ingresa 00 como se muestra en la figura.

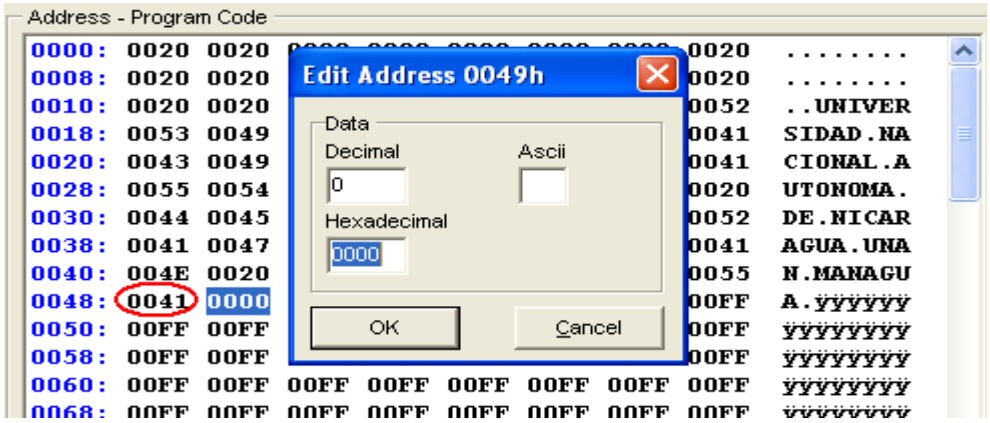


Figura 50. Se ingresa el valor 0000 para que la subrutina de mensajes sepa que es el último carácter a mostrar.

4. Una vez editado el contenido de la memoria se ubica la memoria en el socket correspondiente del programador JDM se procede a grabar la eeprom de la misma manera que se grabó en PIC. Ver figura 47.

5. Para leer el contenido de la eeprom se procede de la misma manera que el PIC solo que el dispositivo seleccionado debe estar en *Device 24C04* (Ver paso 7 de la sección 3.6.2)

3.6.4 PASOS A SEGUIR PARA EL USO DE LA INTERFAZ

1. Asegurarse que el circuito no esté alimentado.
2. Deshabilitar todos los módulos quitando los jumperes de habilitación de cada circuito, el bus I2C se deshabilita quitando la memoria eeprom del socket. Ver figura 51.
3. Elegir el circuito o módulo que se pondrá a prueba colocando su respectivo jumper de habilitación. Ver figura 51.
4. Montar el PIC16f84A grabado con su programa. Asegúrese que la muesca del PIC coincida con la muesca del socket.
5. Si los caracteres del lcd no se ven claramente se puede ajustar el contraste con el potenciómetro lcd contraste como se muestra en la figura 51.
6. Alimentar el circuito.

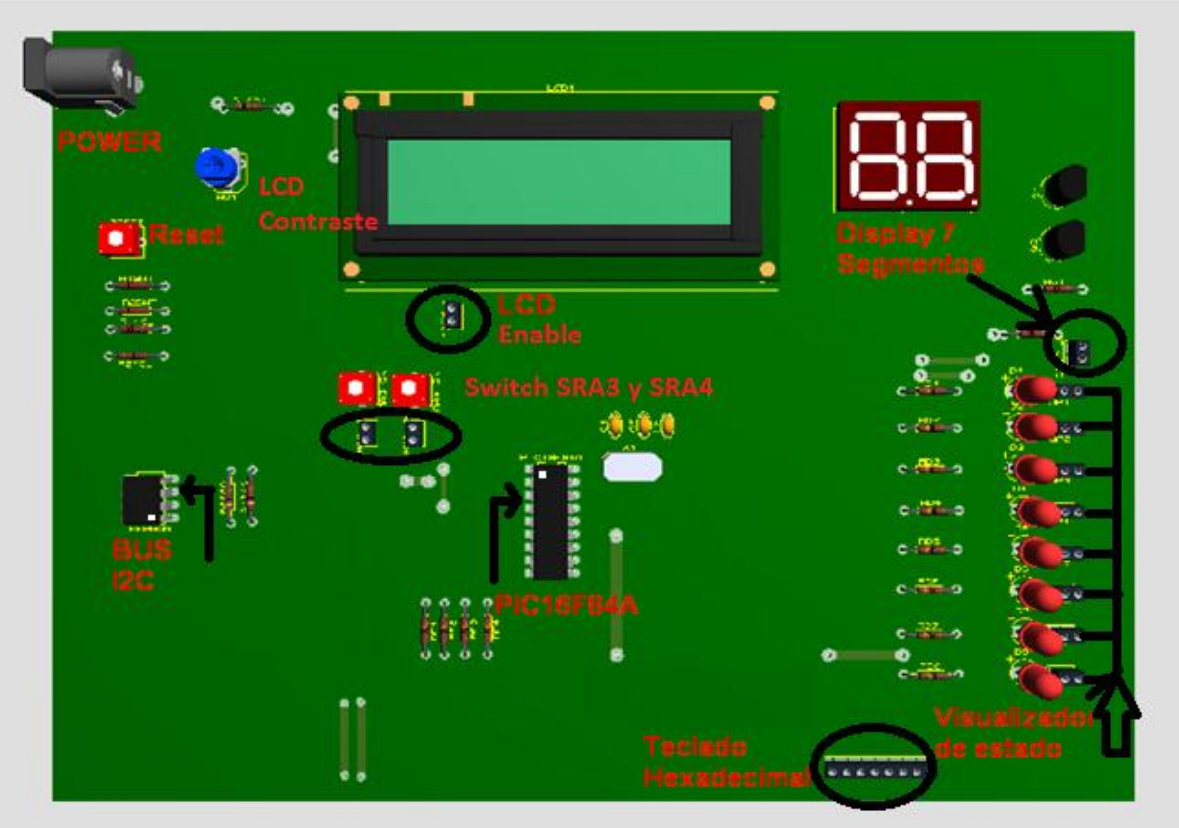


Figura 51. Se muestran los jumperes de habilitación/deshabilitación de cada circuito.

3.6.5 PRACTICAS DE EVALUACION DE LA INTERFAZ ENTRENADORA.

Nota 1: Para poder hacer estas prácticas es necesario tener instalados y configurados los programas MPLAB IDE y ICPROG. Como se indica el manual para la interfaz (Sección 3.6). Estos programas se encuentran en el disco que acompañan este trabajo.

Nota 2: Copie la carpeta PIC16F84A que se encuentra en el disco CD en la ruta disco local C:

PRACTICA 1.

Objetivo: El usuario aprenderá a grabar el PIC con el clonador JDM y el software ICPROG también aprenderá a usar el primer módulo Visualizador de estado.

Abrir la carpeta HEX que se encuentra en la ruta C:\PIC16F84A\HEX y con el clonador JDM y el ICPROG grabe el archivo Visualizador_1.hex en el PIC y monte en la Interfaz entrenadora habilitando el módulo Visualizador de estado. Tenga en cuenta los pasos de la sección 3.6.4

PRACTICA 2.

Objetivo: El usuario será capaz de editar un programa con el software MPLAB IDE y podrá ver los resultados en la interfaz entrenadora.

Inicie el programa MPLAB IDE, abra el archivo Visualizador_1.asm que se encuentra en la ruta C:\PIC16F84A\Visualizador_1 y cambie los llamados a subrutinas:

call Retardo_5s por ***call Retardo_2s*** las dos veces que aparece.

y cambie ***call Retardo_2s*** por ***call Retardo_1s***

Guarde los cambios y compile el programa. Ver sección 3.6.1 paso 9 del manual.

Luego grabe al archivo resultante Visualizador_1.hex que se encuentra en esta misma carpeta del programa con el clonador JDM y el ICPROG. Ver sección 3.6.2 y pruébelo en la interfaz entrenadora.

PRACTICA 3.

Objetivo: El propósito de esta práctica es el uso del módulo LCD su habilitación/Deshabilitación.

Grabar en el PIC el archivo DisplayLCD_1.hex que se encuentra en la carpeta HEX y montar en la interfaz entrenadora.

PRACTICA 4.

Objetivo: El usuario será capaz de editar los mensajes que están en un programa ejemplo y ver los resultados en el módulo LCD.

Iniciar el MPLAB IDE y abrir el archivo DisplayLCD_1.asm que se encuentra en la ruta C:\PIC16F84A\DisplayLCD_1 y cambiar el mensaje0 que dice “UNAN MANAGUA” por otro mensaje que no sobrepase los 16 caracteres incluyendo los espacios (Tener cuidado de no borrar las comillas). Puede cambiar el mensaje1 por otro mensaje con el mismo cuidado. Guarde los cambios compile el programa y grabe el archivo DisplayLCD.hex resultante en el PIC. Y pruebe en la interfaz.

PRACTICA 5.

Objetivo: El propósito principal de esta práctica es que el usuario aprenda a grabar mensajes en la memoria eeprom 24C04 y a la vez podrá usar el módulo BUSI2C.

Grabe en el PIC el programa BusI2C_2.hex. Que se encuentra en la carpeta HEX. Probar en la interfaz el módulo BUSI2C colocando la memoria eeprom en el socket correspondiente. Luego desmontar el circuito y modificar el contenido de la eeprom 24C04 en el ICPROG. Siguiendo los siguientes pasos.

1. Lea el dato de la eeprom como se explica en la sección 3.6.3 paso 5

2. Ubíquese en la página 1 (segunda página) de la eeprom 24C04 luego en la zona de caracteres modifique el mensaje por cualquier otro mensaje. El mensaje puede ser de cualquier tamaño solo hay que asegurarse que termine en el valor hexadecimal 0000 (no confundir con el carácter ‘0’) ver figura 52.

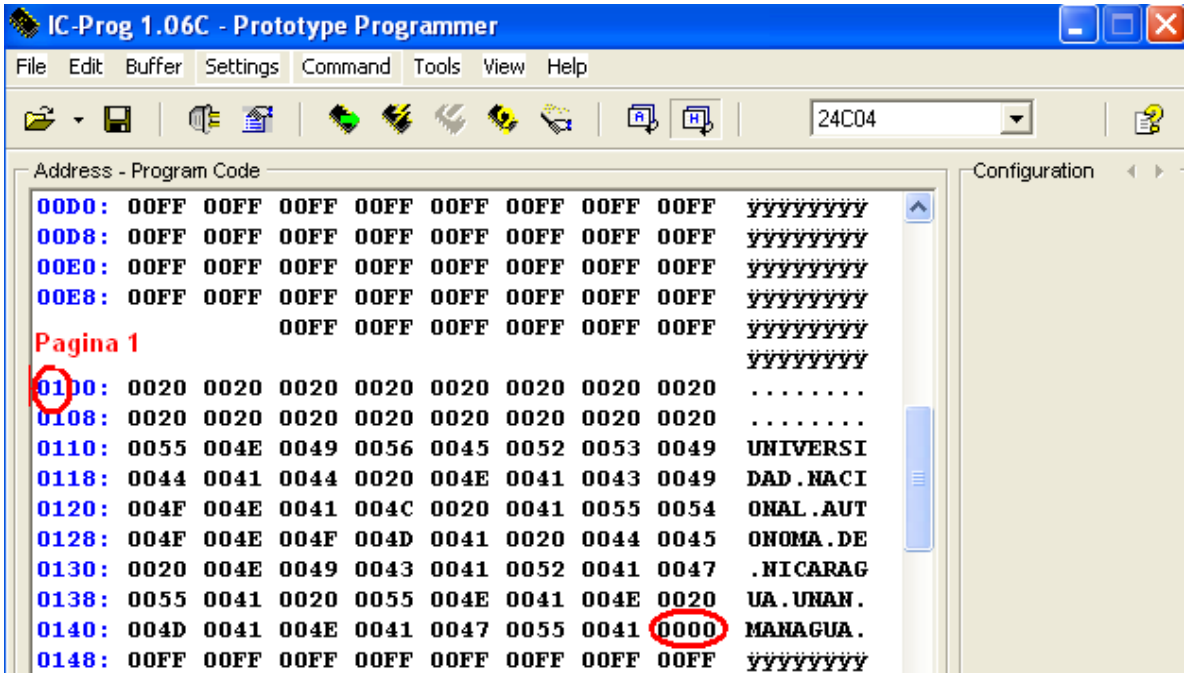


Figura 52. Final de mensaje.

3. Grabe este dato en la eeprom y vuelva a correr el programa colocando la eeprom en la interfaz (el programa en el PIC no es necesario volver a grabarlo).

Practica 6.

Objetivo: Al realizar esta práctica el estudiante será capaz de cambiar la contraseña del programa Teclado_2.asm y así podrá implementar este programa en un proyecto que requiera este tipo de utilidad.

Con el software MPLAB IDE abra el archivo Teclado_2.asm que se encuentra en la ruta C:\PIC16F84A\Teclado_2 luego ubique el código que se muestra en la figura 53.

```
;
LeeClaveSecreta
        addwf   PCL,F
ClaveSecreta
        DT      0XA,0XB,0XC,0XD          ; Ejemplo de clave secreta.
        DT      0XE,0XF
FinClaveSecreta
```

Figura 53. Código que contiene la clave secreta.

Cambie los valores A, B, C, D, E, F por cualquier otro valor hexadecimal. Guarde los cambios compile el programas y grabe el archivo .hex resultante en el PIC y compruebe que la clave ha cambiado.

Practica 7

Objetivo: Con esta práctica se podrá ver como el contador puede ajustarse a cualquier valor que este entre 0 y 99 para este caso específico el contador llegara hasta 59. El cual puede utilizarse en aplicaciones como ejemplo un reloj.

Con el MPLAB IDE ubique el archivo 7Segmento_1.asm que se encuentra en la ruta C:\PIC16F84A\7Segmento_1 y ubique el siguiente código de la subrutina S10:

```
s10                         ; rutina de incremento x 10
   clrf     Dig2            ;pone a cero las unidades
   incf     Dig1,f          ;incrementa el contador de decenas
   movf     Dig1, w         ;carga en registro w el conteo de las decenas
   xorlw    0x0A            ;si w era 10, entonces quedara en cero
   btfsc    STATUS,Z        ;si es cero, el flag z queda alto
   clrf     Dig1            ; Pone a cero Dígito decenas.
   return
```

Cambie el código que está en rojo por este otro:

```
xorlw       0x06
```

Compile el programa y grabe el archivo resultante en el PIC y compruebe que el contador va de 0 a 59.

CONCLUSIONES

- La encuesta nos permitió comprobar la importancia de este trabajo para los estudiantes de ingeniería electrónica de la UNAN Managua para fines didácticos.
- Se realizó el montaje de una interfaz entrenadora para el PIC16F84A para los estudiantes de la carrera de ingeniería electrónica en las asignaturas de Microprocesadores y electrónica digital
- Se diseñó los diferentes módulos de la interfaz entrenadora y se deja la opción de mejorar y agregar nuevos módulos como el módulo del Puerto serie RS232 y otros según el criterio del usuario.
- Se diseñaron y se probaron los diferentes programas para las prácticas de laboratorio que los estudiantes realizarían en la interfaz entrenadora.
- La creación de esta interfaz nos enseña lo útil de esta herramienta en el ámbito de los microcontroladores y electrónica digital nos da una perspectiva clara del funcionamiento del dispositivo desde su interior y su interacción con el mundo externo representado por los periféricos.
- El uso de resistencias limitadoras es importante ya que protege al microcontrolador de exceder los valores límites de corriente ya sea en modo fuente o modo sumidero. Que se encuentra entre 20 y 25 mA. (PIC16F84A, 2001, pág. 49)
- Los transistores 2n3904 tienen tiempos de conmutación de 30 hasta 50 nanosegundos, suficiente para la multiplexación el cual se hace como mínimo 25 milisegundos. Además su máxima corriente en el colector de 200 mA nos garantiza la conducción de la corriente de los display de 7 segmentos el cual tiene un máximo de 80 mA. (2n3904, 2011, págs. 2,3.)
- El uso de una matriz 4x4 para el teclado permitió el ahorro de pines ya que con solamente 8 pines del puerto se lograron censar 16 switches
- El utilizar una memoria eeprom 24C04 en el bus I2C permitió grabar mensajes muy largos (512 caracteres) que serían imposibles guardarlos en el propio microcontrolador por sus limitada memoria interna. Además que son fáciles de conseguir y de muy reducido costo.
- El display LCD contiene 2 filas por 16 columnas el cual nos permite ver 16 caracteres de manera fija pero da la opción de ver mensajes en movimiento esto permite visualizar mensajes muy largos algo que es muy útil a la hora de mostrar mensajes de este tipo.
- Se entregó la interfaz entrenadora del PIC16F84A al docente encargado del laboratorio de microcontroladores para su respectiva requisa, validación y recomendaciones en la implementación de la interfaz en la clase de electrónica digital y Microprocesadores y los resultados fueron positivos.

RECOMENDACIONES

Con la demostración de nuestro trabajo nuestra investigación nos dio muchas opciones que nos permitiera llegar más profundamente en el campo de los microcontroladores y su programación.

- Se sugiere hacer varias interfaces de este mismo tipo (por ejemplo 4 interfaces) esto permitiría hacer 4 o 5 grupos de estudiantes y se puedan tener un mejor dominio de los temas o clases que se les imparte.

- Se sugiere hacer otra versión de esta interfaz orientada en el aumento del número de módulos que no se pudieron mostrar en esta interfaz entrenadora del PIC16F84A como son:

- Motores Paso a Paso y Motores de corriente continua.
- Varias aplicaciones utilizando la tecnología del protocolo serial Bus I2C (Circuito Bus Inter Integrado) desarrollado por la empresa Phillips. Es decir que se puede conectar una gran cantidad de dispositivos pero los más importantes son el DS1624(el termómetro digital); DS1307(el reloj y calendario de tiempo real); SAA1064(driver de display del tipo LED de 4 dígitos); PCF8534(interface entre un puerto de 8 líneas y el bus I2C); PCF8591(conversor ADC de cuatro canales de 8 bits más 1 canal DAC); PCF8576(driver para display LCD); LM76(termómetro digital y termostato) y 24LC256(memoria EEPROM serie de 32Kbytes).
- Comunicación con el ordenador utilizando el Puerto serie RS232.
- Servomotores de Radiocontrol.

Todos estos dispositivos son de enorme utilidad para este tipo de microcontrolador como es el PIC16F84A. Pero se condenso la información y los módulos más importantes e interesantes para los estudiantes de la clase de Microprocesadores y Periféricos

BIBLIOGRAFÍA

24C04, D. (2004). *Microchip Technology.* Obtenido de Microchip: http://ww1.microchip.com/downloads/en/DeviceDoc/11183F.pdf

2n3904, D. (2011). *Fairchild Semiconductor.* Obtenido de Fairchild Semiconductor: http://www.fairchildsemi.com/ds/2N/2N3904.pdf

Angulo josé, A. I. (2005). *Microcontroladores PIC. Diseño práctico de aplicaciones.* Chile: J.H Corp.

Fernández, H. A. (2010). *Diseño lógico. Fundamentos en electrónica digital.* Ediciones de la U.

LM016L, D. (1998). *Hitachi*. Obtenido de Hitachi: http://web.mit.edu/6.115/www/datasheets/44780.pdf

Mackenzie, P. (2007). *Microcontrolador 8051.* Mexico: Pearson Educación.

Mendoza, J. R. (2011). *Diseño y simulacion de sistemas microcontroladores en lenguaje C.* Colombia.

MicroPIC. (2014). *MicroPIC Página web*. Obtenido de MicroPIC: http://www.micropic.es/mpblog/

Palacios, R. L. (2009). *Microcontrolador PIC16F84A Desarrollo de Proyectos.* Colombia: Alfa y Omega.

PIC16F84A, D. (2001). *Microchip Technology.* Obtenido de Microchip: http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf

Robert Boylestad, L. N. (2009). *Electrónica: Teoría de circuitos y Dispositivos Electrónicos.* México: Person Educación.

Ronald Tocci, N. W. (2007). *Sistemas Digitales Principios y Aplicaciones.* México: Person Educación.

Ucontrol. (2014). *Ucontrol Electrónica en general PIC en particular*. Obtenido de Ucontrol: http://www.ucontrol.com.ar/

Valdés, P. (2007). *Microcontroladores Fundamentos y Aplicaciones con Pic.* España: 3Q.

GLOSARIO.

- **BUS I2C:** I²C es un bus de comunicaciones en serie. Su nombre viene de Inter-Integrated Circuit (Inter-Circuitos Integrados). La versión 1.0 data del año 1992 y la versión 2.1 del año 2000, su diseñador es Philips. La velocidad es de 100 kbit/s en el modo estándar, aunque también permite velocidades de 3.4 Mbit/s. Es un bus muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados (Embeded Systems) y generalizando más para comunicar circuitos integrados entre sí que normalmente residen en un mismo circuito impreso. La principal característica de I²C es que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj
- **DISPLAY DE SIETE SEGMENTOS:** El visualizador de siete segmentos (llamado también display por calco del inglés, aunque no recomendado en español) es una forma de representar números en equipos electrónicos. Está compuesto de siete segmentos que se pueden encender o apagar individualmente. Cada segmento tiene la forma de una pequeña línea. Se podría comparar a escribir números con cerillas o fósforos de madera. En un visualizador de 7 segmentos se representan los dígitos 0 a 9 iluminando los segmentos adecuados.
- **EEPROM:** EEPROM o E²PROM son las siglas de Electrically Erasable Programmable Read-Only Memory (ROM programable y borrada eléctricamente). Es un tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente, a diferencia de la EPROM que ha de borrarse mediante un aparato que emite rayos ultravioleta. Son memorias no volátiles.
- **Interfaz:** En electrónica, telecomunicaciones y hardware, una interfaz es el puerto (circuito físico) a través del que se envían o reciben señales desde un sistema o subsistemas hacia otros. No existe una interfaz universal, sino que existen diferentes estándares (Interfaz USB, interfaz SCSI, etc.) que establecen especificaciones técnicas concretas (características comunes), con lo que la interconexión sólo es posible utilizando la misma interfaz en origen y destino. Así también, una interfaz puede ser definida como un intérprete de condiciones externas al sistema, a través de transductores y otros dispositivos, que permite una comunicación con actores externos, como personas u otros sistemas, a través de un protocolo común a ambos. Una interfaz es una Conexión física y funcional entre dos aparatos o sistemas independientes.
- **Jumper**: O puente es un elemento que permite interconectar dos terminales de manera temporal sin tener que efectuar una operación que requiera una herramienta adicional. Dicha unión de terminales cierra el circuito eléctrico del que forma parte.

- **LED:** Un LED (light-emitting diode o diodo emisor de luz) es un componente opto electrónico pasivo y, más concretamente, un diodo que emite luz. Al igual que la inmensa mayoría de diodos, el diodo LED sólo emitirá luz -sólo funcionará- si lo polarizamos de forma directa, colocando el positivo en el ánodo y el negativo en el cátodo. La electroluminiscencia (la capacidad para liberar energía en forma de fotones) y el color de la luz emitida dependen del semiconductor del que están fabricados.
- **Lenguaje ensamblador:** Es un lenguaje de programación de bajo nivel para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables. Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura dada de CPU y constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador.
- **Microcontrolador**: Un microcontrolador es un circuito integrado programable que contiene todos los componentes necesarios para controlar el funcionamiento de una tarea determinada, como el control de una lavadora, un teclado de ordenador, una impresora, un sistema de alarmas, etc. Para esto, el microcontrolador utiliza muy pocos componentes asociados. Un sistema con microcontrolador debe disponer de una memoria donde se almacena el programa que gobierna el funcionamiento del mismo que, una vez programado y configurado, solo sirve para realizar la tarea asignada. A la vez un microcontrolador es un sistema cerrado, lo que quiere decir que en un solo circuito integrado se encierra un sistema programable completo.
- **Microprocesador:** Un microprocesador es básicamente un chip que contiene la CPU (Central Proccesing Unit) que se encarga de controlar todo el sistema. Un sistema digital basado en un microprocesador es un sistema abierto ya que su configuración difiere según a la aplicación a la que se destine. Se puede acoplar los módulos necesarios para configurarlo con las características que se desee. Para ello saca al exterior las líneas de sus buses de datos, direcciones, y control de modo que permita su conexión con la memoria y los módulos de entrada/salida. Finalmente resulta en un sistema implementado por varios circuitos integrados dentro de una misma placa de circuito impreso.
- **Pantalla LCD:** Una pantalla de cristal líquido o LCD (sigla del inglés liquid crystal display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. A menudo se utiliza en dispositivos electrónicos de pilas, ya que utiliza cantidades muy pequeñas de energía eléctrica.
- **PIC16F84A:** El (Peripheral Interface Controller) PIC16F84A es un microcontrolador a 8 bits de la familia PIC perteneciente a la Gama Media (según la clasificación dada a los microcontroladores por la misma empresa fabricante) Microchip. Se trata de uno de los microcontroladores más populares del mercado actual, ideal para principiantes, debido a su

arquitectura de 8 bits, 18 pines, y un conjunto de instrucciones RISC muy amigable para memorizar y fácil de entender.

- **RESISTENCIAS PULL-UP:** En electrónica se denomina pull-up a la acción de elevar la tensión de salida de un circuito lógico, a la tensión que, por lo general mediante un divisor de tensión, se pone a la entrada de un amplificador con el fin de desplazar su punto de trabajo
- **TECLADO HEXADÉCIMAL:** Un teclado matricial está constituido por una matriz de pulsadores dispuestos en filas y columnas. Cada tecla está conectada a una fila y a una columna. Cuando una tecla es pulsada, queda en contacto una fila con una columna y, si no hay tecla alguna presionada, las filas están desconectadas de las columnas.
- **TRANSISTOR 2N3904:** El transistor 2N3904 es uno de los más comunes transistores NPN generalmente usado para amplificación. Está diseñado para funcionar a bajas intensidades, bajas potencias, tensiones medias, y puede operar a velocidades razonablemente altas. Se trata de un transistor de bajo costo, muy común, y suficientemente robusto como para ser usado en experimentos electrónicos. Es un transistor de 200 miliamperios, 40 voltios, 625 mili vatios, con una frecuencia de transición de 300 MHz, con una beta de 100. Es usado primordialmente para la amplificación analógica.
- **TRANSISTOR:** El transistor es un dispositivo electrónico semiconductor utilizado para producir una señal de salida en respuesta a otra señal de entrada. 1 Cumple funciones de amplificador, oscilador, conmutador o rectificador. El término «transistor» es la contracción en inglés de transfer resistor («resistencia de transferencia»). Actualmente se encuentran prácticamente en todos los aparatos electrónicos de uso diario: radios, televisores, reproductores de audio y video, relojes de cuarzo, computadoras, lámparas fluorescentes, tomógrafos, teléfonos celulares.

ANEXOS.

A.- Encuesta

Encuesta realizada a los estudiantes egresados de la carrera ingeniería electrónica de la UNAN Managua entre el año 2007 – 2014.

Año en que terminó sus estudios_____

1. ¿Conoce los dispositivos microcontroladores? (Si-No).
2. ¿Ha tenido oportunidad de trabajar con una interfaz entrenadora Virtual para microcontroladores?(Si-no)
3. Si su respuesta anterior es sí que tipo: PIC de Microchip, AVR de Atmel, 8051 Intel, ninguno de los anteriores.
4. ¿Ha tenido oportunidad de trabajar con una interfaz entrenadora real para microcontroladores?(Si-No)
5. Si su respuesta Anterior es sí que tipo: PIC de Microchip, AVR de Atmel, 8051 Intel, ninguno de los anteriores.
6. ¿Utilizó el lenguaje ensamblador para programar estos microcontroladores? (Si-No)
7. Si su respuesta anterior es Si quedó claro de cómo está organizado internamente el microcontrolador. (Si-No).
8. ¿Utilizo el lenguaje C para programar estos microcontroladores?(Si-No)
9. Si su respuesta anterior es Si quedó claro de cómo está organizado internamente el microcontrolador. (Si-No).
10. ¿Conoce el proceso de “quemado” o “grabado” de los microcontroladores? (Si-No)
11. ¿Según su criterio piensa que con solo virtualizar las prácticas de microcontroladores dominará totalmente estos dispositivos?(Si-No)
12. ¿Estaría de acuerdo que se trabaje en los laboratorios de electrónica con una interfaz entrenadora para microcontroladores como el PIC16F84A y con un lenguaje de programación básico como el ensamblador?(Si-No)

Tabla de frecuencia

Tabla A.1 ¿Conoce los dispositivos microcontroladores?

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	si	14	100,0	100,0	100,0

Conclusiones: Los dispositivos microcontroladores son ampliamente conocidos por los estudiantes, algo que es muy importante porque demuestra la importancia de investigar más sobre estos dispositivos.

Tabla A.2 ¿Ha tenido oportunidad de trabajar con una interfaz entrenadora Virtual para microcontroladores?

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	si	14	100,0	100,0	100,0

Conclusiones: El cien por ciento ha trabajado con una interfaz virtual de simulación esto es muy importante ya que al trabajar con una interfaz real es necesario que los trabajos primero se simulen en interfaz virtual.

Tabla A.3 Si su respuesta es sí que tipo

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	8051 Intel	4	28,6	28,6	28,6
	PIC de Microchip	10	71,4	71,4	100,0
	Total	14	100,0	100,0	

Conclusiones: Los dispositivos microcontroladores más utilizados por los estudiantes son los de PIC de Microchip 71,4 %. Esto demuestra la importancia de investigar más sobre estos microcontroladores de esta marca.

Tabla A.4 ¿Ha tenido oportunidad de trabajar con una interfaz entrenadora real para microcontroladores?

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	no	14	100,0	100,0	100,0

Conclusiones: Los estudiantes no han trabajado en los laboratorios con una interfaz real. Lo que muestra la necesidad de una.

Tabla A.5 Si su respuesta es sí que tipo

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	ninguno	14	100,0	100,0	100,0

Conclusiones: Al no haber trabajado con una interfaz real no pueden elegir ninguna opción aunque este no necesariamente es un desconocimiento de estos dispositivos.

Tabla A.6 ¿Utilizó el lenguaje de ensamblador para programar estos microcontroladores?

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	no	3	21,4	21,4	21,4
	si	11	78,6	78,6	100,0
	Total	14	100,0	100,0	

Conclusiones: El lenguaje ensamblador ha sido muy utilizado por tanto no es desconocidos por los estudiantes.

Tabla A.7 Si su respuesta anterior es Si quedo claro de cómo está organizado internamente el microcontrolador

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	no	2	14,3	14,3	14,3
	ninguno	3	21,4	21,4	35,7
	si	9	64,3	64,3	100,0
	Total	14	100,0	100,0	

Conclusiones: El 64,3 % de los estudiantes que han programado en ensamblador (78,6%) queda claro de como trabaja internamente el Microcontrolador o sea un ochenta por ciento de los que programa en ensamblador.

Tabla A.8 ¿Utilizó el lenguaje C para programar estos microcontroladores?

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	si	14	100,0	100,0	100,0

Conclusiones: El lenguaje C es el más utilizado por los estudiantes.

Tabla A.9 Si su respuesta anterior es Si quedo claro de cómo está organizado internamente el microcontrolador

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	no	4	28,6	28,6	28,6
	si	10	71,4	71,4	100,0
	Total	14	100,0	100,0	

Conclusiones: El programar en lenguaje C el dominio del tema es menor 71,4. Por tanto es recomendable usar lenguaje ensamblador por lo menos para microcontroladores básicos.

Tabla A.10 ¿Conoce el proceso de "quemado" o grabado de los microcontroladores?

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	si	14	100,0	100,0	100,0

Conclusiones: El proceso de quemado es conocido por los estudiantes por lo que este proceso no representa ninguna dificultad a la hora de trabajar con una interfaz real.

Tabla A.11 ¿Según su criterio piensa que con solo virtualizar las prácticas de microcontroladores dominaran totalmente estos dispositivos?

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	no	14	100,0	100,0	100,0

Conclusiones: Todos los estudiantes están de acuerdo en que no basta solamente con virtualizar las prácticas de microcontroladores

Tabla A.12 ¿Estarías de acuerdo que se trabaje en los laboratorios de electrónicas con una interfaz entrenadora real para microcontroladores PIC 16F84A y un lenguaje de programación básico como el ensamblador?

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	si	14	100,0	100,0	100,0

Conclusiones: El 100% de los estudiantes están de acuerdo en que se implemente una interfaz entrenadora real en los laboratorios de electrónica.

B.- LIBRERIAS UTILIZADAS EN LOS PROGRAMAS.

B.1. BIN_BCD.INC

```
;******************************** Librería "BIN_BCD.INC" ********************************
;
;
; Un número binario natural de 8 bits es convertido a BCD. El resultado se guarda en tres
; posiciones de memorias llamadas: BCD_Centenas, BCD_Decenas y BCD_Unidades.
;
; El procedimiento utilizado es mediante restas de 10, tal como se explicó en el capítulo 9.
;
; Entrada:      En el registro W el número binario natural a convertir.
; Salidas:      En (BCD_Centenas), (BCD_Decenas) y (BCD_Unidades).
;               En el registro W también las decenas (nibble alto) y unidades (nibble bajo).

; Subrutina "BIN_a_BCD" ----------------------------------------------------------------

        CBLOCK                  ; En las subrutinas no se debe fijar la dirección
        BCD_Centenas            ; de la RAM de usuario. Se toma a continuación de
        BCD_Decenas             ; la última asignada.
        BCD_Unidades
        ENDC
;
BIN_a_BCD
        clrf    BCD_Centenas    ; Carga los registros con el resultado inicial.
        clrf    BCD_Decenas     ; En principio las centenas y decenas a cero.
        movwf   BCD_Unidades    ; Se carga el número binario a convertir.
BCD_Resta10
        movlw   .10             ; A las unidades se les va restando 10 en cada
        subwf   BCD_Unidades,W  ; pasada. (W)=(BCD_Unidades) -10.
        btfss   STATUS,C        ; ¿C = 1?, ¿(W) positivo?, ¿(BCD_Unidades)>=10?
        goto    BIN_BCD_Fin     ; No, es menor de 10. Se acabó.
BCD_IncrementaDecenas
        movwf   BCD_Unidades    ; Recupera lo que queda por restar.
        incf    BCD_Decenas,F   ; Incrementa las decenas y comprueba si ha
llegado
        movlw   .10             ; a 10. Lo hace mediante una resta.
        subwf   BCD_Decenas,W   ; (W)=(BCD_Decenas)-10).
        btfss   STATUS,C        ; ¿C = 1?, ¿(W) positivo?, ¿(BCD_Decenas)>=10?
        goto    BCD_Resta10     ; No. Vuelve a dar otra pasada, restándole 10 a
BCD_IncrementaCentenas          ; las unidades.
        clrf    BCD_Decenas     ; Pone a cero las decenas
        incf    BCD_Centenas,F  ; e incrementa las centenas.
        goto    BCD_Resta10     ; Otra pasada: Resta 10 al número a convertir.
BIN_BCD_Fin
        swapf   BCD_Decenas,W           ; En el nibble alto de (W) también las
                                        ; decenas
```

```
        addwf   BCD_Unidades,W          ; En el nibble bajo de (W) las unidades.
        return                          ; Vuelve al programa principal.

; La directiva "END" se debe poner en el programa principal no aquí.
```

B.2. BUS_I2C.INC

```
;***************************** Librería "BUS_I2C.INC" ***********************************
;
; Estas subrutinas permiten realizar las tareas básicas de control del bus serie I2C,
; por parte de un solo microcontrolador maestro.
;
; ZONA DE DATOS ********************************************************************
;
        CBLOCK
        I2C_ContadorBits            ; Cuenta los bits a transmitir o a recibir.
        I2C_Dato                    ; Dato a transmitir o recibido.
        I2C_Flags                   ; Guarda la información del estado del bus I2C.
        ENDC

#DEFINE I2C_UltimoByteLeer  I2C_Flags,0
;   - (I2C_UltimoByteLeer)=0, NO es el último byte a leer por el maestro.
;   - (I2C_UltimoByteLeer)=1, SÍ es el último byte a leer por el maestro.

; La definición de las líneas SCL y SDA del bus I2C se puede cambiar según las
; necesidades del hardware.

#DEFINE SCL  PORTA,3                ; Línea SCL del bus I2C.
#DEFINE SDA  PORTA,4                ; Línea SDA del bus I2C.
;
; Subrutina "SDA_Bajo" -------------------------------------------------------------------------
;
SDA_Bajo
        bsf     STATUS,RP0          ; Configura la línea SDA como salida.
        bcf     SDA
        bcf     STATUS,RP0
        bcf     SDA                 ; SDA en bajo.
        return
;
; Subrutina "SDA_AltaImpedancia" ---------------------------------------------------
;
SDA_AltaImpedancia
        bsf     STATUS,RP0          ; Configura la línea SDA entrada.
        bsf     SDA                 ; Lo pone en alta impedancia y, gracias a la
        bcf     STATUS,RP0          ; Rp de esta línea, se mantiene a nivel alto.
        return
;
; Subrutina "SCL_Bajo" -------------------------------------------------------------
```

```
;
SCL_Bajo
        bsf     STATUS,RP0
        bcf     SCL                     ; Configura la línea SCL como salida.
        bcf     STATUS,RP0
        bcf     SCL                     ; La línea de reloj SCL en bajo.
        return
;
; Subrutina "SCL_AltaImpedancia" -------------------------------------------------------
;
SCL_AltaImpedancia
        bsf     STATUS,RP0              ; Configura la línea SCL entrada.
        bsf     SCL                     ; Lo pone en alta impedancia y, gracias a la Rp
        bcf     STATUS,RP0              ; de esta línea, se mantiene a nivel alto.
SCL_EsperaNivelAlto
        btfss   SCL                     ; Si algún esclavo mantiene esta línea en bajo
        goto    SCL_EsperaNivelAlto     ; hay que esperar.
        return
;
; Subrutina "I2C_EnviaStart" -----------------------------------------------------------
;
; Esta subrutina envía una condición de Start o inicio.
;
I2C_EnviaStart
        call    SDA_AltaImpedancia      ; Línea SDA en alto.
        call    SCL_AltaImpedancia      ; Línea SCL en alto.
        call    Retardo_4micros                 ; Tiempo tBUF del protocolo.
        call    SDA_Bajo                ; Flanco de bajada de SDA mientras SCL está alto.
        call    Retardo_4micros                 ; Tiempo tHD;STA del protocolo.
        call    SCL_Bajo                ; Flanco de bajada del reloj SCL.
        call    Retardo_4micros
        return
;
; Subrutina "I2C_EnviaStop" ------------------------------------------------------------
;
; Esta subrutina envía un condición de Stop o parada.
;
I2C_EnviaStop
        call    SDA_Bajo
        call    SCL_AltaImpedancia      ; Flanco de subida de SCL.
        call    Retardo_4micros                 ; Tiempo tSU;STO del protocolo.
        call    SDA_AltaImpedancia      ; Flanco de subida de SDA.
        call    Retardo_4micros                 ; Tiempo tBUF del protocolo.
        return
;
; Subrutina "I2C_EnviaByte" ------------------------------------------------------------
;
; El microcontrolador maestro transmite un byte por el bus I2C, comenzando por el bit
; MSB. El byte a transmitir debe estar cargado previamente en el registro de trabajo W.
```

```
; De la subrutina ejecutada anteriormente I2C_EnviaStart o esta misma I2C_EnviaByte,
; la línea SCL se debe encontrar a nivel bajo al menos durante 5 µs.
;
I2C_EnviaByte
        movwf   I2C_Dato                ; Almacena el byte a transmitir.
        movlw   0x08                    ; A transmitir 8 bits.
        movwf   I2C_ContadorBits
I2C_EnviaBit
        rlf     I2C_Dato,F              ; Chequea el bit, llevándolo previamente al Carry.
        btfsc   STATUS,C
        goto    I2C_EnviaUno
I2C_EnviaCero
        call    SDA_Bajo                ; Si es "0" envía un nivel bajo.
        goto    I2C_FlancoSCL
I2C_EnviaUno
        call    SDA_AltaImpedancia      ; Si es "1" lo activará a alto.
I2C_FlancoSCL
        call    SCL_AltaImpedancia      ; Flanco de subida del SCL.
        call    Retardo_4micros         ; Tiempo tHIGH del protocolo.
        call    SCL_Bajo                ; Termina el semiperiodo positivo del reloj.
        call    Retardo_4micros         ; Tiempo tHD;DAT del protocolo.
        decfsz  I2C_ContadorBits,F      ; Lazo para los ocho bits.
        goto    I2C_EnviaBit
;
        call    SDA_AltaImpedancia      ; Libera la línea de datos.
        call    SCL_AltaImpedancia      ; Pulso en alto de reloj para que el esclavo
        call    Retardo_4micros         ; pueda enviar el bit ACK.
        call    SCL_Bajo
        call    Retardo_4micros
        return
;
; Subrutina "I2C_LeeByte" ---------------------------------------------------------
;
; El microcontrolador maestro lee un byte desde el esclavo conectado al bus I2C. El dato
; recibido se carga en el registro I2C_Dato y lo envía a la subrutina superior a través
; del registro W. Se empieza a leer por el bit de mayor peso MSB.
; De alguna de las subrutinas ejecutadas anteriormente I2C_EnviaStart, I2C_EnviaByte
; o esta misma I2C_LeeByte, la línea SCL lleva en bajo al menos 5 µs.

I2C_LeeByte
        movlw   0x08                    ; A recibir 8 bits.
        movwf   I2C_ContadorBits
        call    SDA_AltaImpedancia      ; Deja libre la línea de datos.
I2C_LeeBit
        call    SCL_AltaImpedancia      ; Flanco de subida del reloj.
        bcf     STATUS,C                ; En principio supone que es "0".
        btfsc   SDA                     ; Lee el bit
        bsf     STATUS,C                ; Si es "1" carga 1 en el Carry.
        rlf     I2C_Dato,F              ; Lo introduce en el registro.
```

```
        call    SCL_Bajo            ; Termina el semiperiodo positivo del reloj.
        call    Retardo_4micros     ; Tiempo tHD;DAT del protocolo.
        decfsz  I2C_ContadorBits,F  ; Lazo para los 8 bits.
        goto    I2C_LeeBit
;
; Chequea si este es el último byte a leer para enviar o no el bit de reconocimiento
; ACK en consecuencia.
;
        btfss   I2C_UltimoByteLeer  ; Si es el último, no debe enviar
                                    ; el bit de reconocimiento ACK.
        call    SDA_Bajo            ; Envía el bit de reconocimiento ACK
                                    ; porque todavía no es el último byte a leer.
        call    SCL_AltaImpedancia  ; Pulso en alto del SCL para transmitir el
        call    Retardo_4micros             ; bit ACK de reconocimiento. Este es
tHIGH.
        call    SCL_Bajo            ; Pulso de bajada del SCL.
        call    Retardo_4micros
        movf    I2C_Dato,W          ; El resultado se manda en el registro de
        return                      ; de trabajo W.

;
```

B.3. EEPROM.INC

```
;****************************** Librería "EEPROM.INC" *********************************
;
;
; Estas subrutinas permiten realizar las tareas básicas de escritura y lectura de la
; memoria EEPROM de datos del PIC.
;
; Subrutina "EEPROM_LeeDato" -----------------------------------------------------------
;
; El microcontrolador lee el dato que hay escrito en la posición de la EEPROM del PIC
apuntada
; por el contenido del registro de trabajo W. El resultado se proporciona en el mismo W.
;
; Entrada: En (W) la dirección de la memoria EEPROM a leer.
; Salida :  En (W) el byte leído.

EEPROM_LeeDato
        bcf     STATUS,RP0          ; Asegura que trabaja con el Banco 0.
        movwf   EEADR               ; Dirección a leer.
        bsf     STATUS,RP0          ; Banco 1.
        bsf     EECON1,RD           ; Orden de lectura.
EEPROM_SigueLeyendo
        btfsc   EECON1,RD           ; El PIC indica que ha terminado la lectura
        goto    EEPROM_SigueLeyendo         ; poniendo en bajo este bit.
        bcf     STATUS,RP0          ; Banco 0.
```

```
        movf    EEDATA,W            ; El byte leído al registro W.
        return

; Subrutina "EEPROM_EscribeDato" ---------------------------------------------------------
;
; Escribe el dato introducido en el registro de trabajo W en la posición de memoria,
;EEPROM del
; PIC apuntada por el registro EEADR.
;
; Como altera el valor del registro INTCON al posicionar el flag GIE, éste se debe guardar
;al
; principio de la subrutina y restaurarlo al final.
;
; Entradas:     En el registro EEADR la dirección de la memoria EEPROM a escribir.
;               En el registro W el byte a escribir.
;
        CBLOCK
        EEPROM_GuardaINTCON
        ENDC

EEPROM_EscribeDato
        bcf     STATUS,RP0          ; Asegura que trabaja con el Banco 0.
        movwf   EEDATA              ; El byte a escribir.
        movf    INTCON,W            ; Reserva el valor anterior de INTCON
        movwf   EEPROM_GuardaINTCON
        bsf     STATUS,RP0          ; Acceso al Banco 1.
        bcf     INTCON,GIE          ; Deshabilita todas las interrupciones.
        bsf     EECON1,WREN         ; Habilita escritura.
;
; El fabricante especifica que hay que seguir esta secuencia para escritura en EEPROM:
;
        movlw   0x55
        movwf   EECON2
        movlw   0xAA
        movwf   EECON2
        bsf     EECON1,WR           ; Inicia la escritura.
EEPROM_TerminaEscribir
        btfsc   EECON1,WR           ; Comprueba que termina de escribir en la
EEPROM.
        goto    EEPROM_TerminaEscribir
        bcf     EECON1,WREN         ; Desautoriza la escritura en EEPROM.
        bcf     EECON1,EEIF         ; Limpia este flag.
        bcf     STATUS,RP0          ; Acceso al Banco 0.
        movf    EEPROM_GuardaINTCON,W ; Restaura el valor anterior de INTCON.
        movwf   INTCON
        return
```

B.4. LCD_4BIT.INC

```
;***************************** Librería "LCD_4BIT.INC" **********************************
;
;
; Estas subrutinas permiten realizar las tareas básicas de control de un módulo LCD de 2
; líneas por 16 caracteres, compatible con el modelo LM016L.
;
; El visualizador LCD está conectado al Puerto B del PIC mediante un bus de 4 bits. Las
; conexiones son:
;   -   Las 4 líneas superiores del módulo LCD, pines <DB7:DB4>  se conectan a las 4
;       líneas superiores del Puerto B del PIC, pines <RB7:RB4>.
;   -   Pin RS del LCD a la línea RA0 del PIC.
;   -   Pin R/W del LCD a la línea RA1 del PIC, o a masa.
;   -   Pin Enable del LCD a la línea RA2 del PIC.
;
; Se utilizan llamadas a subrutinas de retardo de tiempo localizadas en la librería
RETARDOS.INC.
;
; ZONA DE DATOS ********************************************************************

        CBLOCK
        LCD_Dato
        LCD_GuardaDato
        LCD_GuardaTRISB
        LCD_Auxiliar1
        LCD_Auxiliar2
        ENDC

LCD_CaracteresPorLinea      EQU    .16      ; Número de caracteres por línea de la
pantalla.

#DEFINE  LCD_PinRS  PORTA,0
#DEFINE  LCD_PinRW PORTA,1
#DEFINE  LCD_PinEnable      PORTA,2
#DEFINE  LCD_BusDatos       PORTB

; Subrutina "LCD_Inicializa" ------------------------------------------------------
;
; Inicialización del módulo LCD: Configura funciones del LCD, produce reset por software,
; borra memoria y enciende pantalla. El fabricante especifica que para garantizar la
; configuración inicial hay que hacerla como sigue:
;
LCD_Inicializa
        bsf     STATUS,RP0          ; Configura las líneas conectadas al pines RS,
        bcf     LCD_PinRS           ; R/W y E.
        bcf     LCD_PinEnable
        bcf     LCD_PinRW
```

```
        bcf     STATUS,RP0
        bcf     LCD_PinRW           ; En caso de que esté conectado le indica
                                    ; que se va a escribir en el LCD.
        bcf     LCD_PinEnable       ; Impide funcionamiento del LCD poniendo E=0.
        bcf     LCD_PinRS           ; Activa el Modo Comando poniendo RS=0.
        call    Retardo_20ms
        movlw   b'00110000'
        call    LCD_EscribeLCD      ; Escribe el dato en el LCD.
        call    Retardo_5ms
        movlw   b'00110000'
        call    LCD_EscribeLCD
        call    Retardo_200micros
        movlw   b'00110000'
        call    LCD_EscribeLCD
        movlw   b'00100000'         ; Interface de 4 bits.
        call    LCD_EscribeLCD

; Ahora configura el resto de los parámetros:

        call    LCD_2Lineas4Bits5x7 ; LCD de 2 líneas y caracteres de 5x7 puntos.
        call    LCD_Borra           ; Pantalla encendida y limpia. Cursor al principio
        call    LCD_CursorOFF       ; de la línea 1. Cursor apagado.
        call    LCD_CursorIncr      ; Cursor en modo incrementar.
        return

; Subrutina "LCD_EscribeLCD" ---------------------------------------------------------
;
; Envía el dato del registro de trabajo W al bus de dato y produce un pequeño pulso en el
:pin
; Enable del LCD. Para no alterar el contenido de las líneas de la parte baja del Puerto B
:que
; no son utilizadas para el LCD (pines RB3:RB0), primero se lee estas líneas y después se
; vuelve a enviar este dato sin cambiarlo.

LCD_EscribeLCD
        andlw   b'11110000'         ; Se queda con el nibble alto del dato que es el
        movwf   LCD_Dato            ; que hay que enviar y lo guarda.
        movf    LCD_BusDatos,W      ; Lee la información actual de la parte baja
        andlw   b'00001111'         ; del Puerto B, que no se debe alterar.
        iorwf   LCD_Dato,F          ; Enviará la parte alta del dato de entrada
                                    ; y en la parte baja lo que había antes.
        bsf     STATUS,RP0          ; Acceso al Banco 1.
        movf    TRISB,W             ; Guarda la configuración que tenía antes TRISB.
        movwf   LCD_GuardaTRISB
        movlw   b'00001111'         ; Las 4 líneas inferiores del Puerto B se dejan
        andwf   PORTB,F             ; como estaban y las 4 superiores como salida.
        bcf     STATUS,RP0          ; Acceso al Banco 0.
;
        movf    LCD_Dato,W          ; Recupera el dato a enviar.
```

```
        movwf   LCD_BusDatos        ; Envía el dato al módulo LCD.
        bsf     LCD_PinEnable       ; Permite funcionamiento del LCD mediante un
                                    ; pequeño
        bcf     LCD_PinEnable       ; pulso y termina impidiendo el funcionamiento del
                                    ; LCD
        bsf     STATUS,RP0          ; Acceso al Banco 1. Restaura el antiguo valor en
        5r4
        movwf   PORTB               ; Realmente es TRISB.
        bcf     STATUS,RP0          ; Acceso al Banco 0.
        return

; Subrutinas variadas para el control del módulo LCD ----------------------------------------
;
; Los comandos que pueden ser ejecutados son:
;
LCD_CursorIncr                      ; Cursor en modo incrementar.
        movlw   b'00000110'
        goto    LCD_EnviaComando
LCD_Linea1                          ; Cursor al principio de la Línea 1.
        movlw   b'10000000'         ; Dirección 00h de la DDRAM
        goto    LCD_EnviaComando
LCD_Linea2                          ; Cursor al principio de la Línea 2.
        movlw   b'11000000'         ; Dirección 40h de la DDRAM
        goto    LCD_EnviaComando
LCD_PosicionLinea1                  ; Cursor a posición de la Línea 1, a partir de la
        iorlw   b'10000000'         ; dirección 00h de la DDRAM más el valor del
        goto    LCD_EnviaComando    ; registro W.
LCD_PosicionLinea2                  ; Cursor a posición de la Línea 2, a partir de la
        iorlw   b'11000000'         ; dirección 40h de la DDRAM más el valor del
        goto    LCD_EnviaComando    ; registro W.
LCD_OFF                             ; Pantalla apagada.
        movlw   b'00001000'
        goto    LCD_EnviaComando
LCD_CursorON                        ; Pantalla encendida y cursor encendido.
        movlw   b'00001110'
        goto    LCD_EnviaComando
LCD_CursorOFF                       ; Pantalla encendida y cursor apagado.
        movlw   b'00001100'
        goto    LCD_EnviaComando
LCD_Borra                           ; Borra toda la pantalla, memoria DDRAM y pone el
        movlw   b'00000001'         ; cursor a principio de la línea 1.
        goto    LCD_EnviaComando
LCD_2Lineas4Bits5x7                 ; Define la pantalla de 2 líneas, con caracteres
        movlw   b'00101000'         ; de 5x7 puntos y conexión al PIC mediante bus de
;       goto    LCD_EnviaComando    ; 4 bits.

; Subrutinas "LCD_EnviaComando" y "LCD_Caracter" ---------------------------------
;
; "LCD_EnviaComando". Escribe un comando en el registro del módulo LCD. La palabra de
```

```
; Comando ha sido entregada a través del registro W.  Trabaja en Modo Comando.
; "LCD_Caracter". Escribe en la memoria DDRAM del LCD el carácter ASCII introducido a
; a través del registro W. Trabaja en Modo Dato.
;
LCD_EnviaComando
        bcf     LCD_PinRS           ; Activa el Modo Comando, poniendo RS=0.
        goto    LCD_Envia
LCD_Caracter
        bsf     LCD_PinRS           ; Activa el "Modo Dato", poniendo RS=1.
        call    LCD_CodigoCGROM     ; Obtiene el código para correcta visualización.
LCD_Envia
        movwf   LCD_GuardaDato      ; Guarda el dato a enviar.
        call    LCD_EscribeLCD      ; Primero envía el nibble alto.
        swapf   LCD_GuardaDato,W    ; Ahora envía el nibble bajo. Para ello pasa el
                                    ; nibble bajo del dato a enviar a parte alta del byte.
        call    LCD_EscribeLCD      ; Se envía al visualizador LCD.

        btfss   LCD_PinRS           ; Debe garantizar una correcta escritura
                                    ; manteniendo

        call    Retardo_2ms         ; 2 ms en modo comando y 50 µs en modo
                                    ; cáracter.
        call    Retardo_50micros
        return

; Subrutina "LCD_CodigoCGROM" ----------------------------------------------------------
;
; A partir del carácter ASCII número 127 los códigos de los caracteres definidos en la
; tabla CGROM del LM016L no coinciden con los códigos ASCII. Así por ejemplo, el código
; ASCII de la "Ñ" en la tabla CGRAM del LM016L es EEh.
;
; Esta subrutina convierte los códigos ASCII de la "Ñ", "º" y otros, a códigos CGROM para
que
; que puedan ser visualizado en el módulo LM016L.
;
; Entrada:      En (W) el código ASCII del carácter que se desea visualizar.
; Salida:       En (W) el código definido en la tabla CGROM.

LCD_CodigoCGROM
        movwf   LCD_Dato            ; Guarda el valor del carácter y comprueba si es
LCD_EnheMinuscula                   ; un carácter especial.
        sublw   'ñ'                 ; ¿Es la "ñ"?
        btfss   STATUS,Z
        goto    LCD_EnheMayuscula   ; No es "ñ".
        movlw   b'11101110'         ; Código CGROM de la "ñ".
        movwf   LCD_Dato
        goto    LCD_FinCGROM
LCD_EnheMayuscula
        movf    LCD_Dato,W          ; Recupera el código ASCII de entrada.
```

```
        sublw   'Ñ'                     ; ¿Es la "Ñ"?
        btfss   STATUS,Z
        goto    LCD_Grado               ; No es "Ñ".
        movlw   b'11101110'             ; Código CGROM de la "ñ". (No hay símbolo para
        movwf   LCD_Dato                ; la "Ñ" mayúscula en la CGROM).
        goto    LCD_FinCGROM
LCD_Grado
        movf    LCD_Dato,W              ; Recupera el código ASCII de entrada.
        sublw   'º'                     ; ¿Es el símbolo "º"?
        btfss   STATUS,Z
        goto    LCD_FinCGROM            ; No es "º".
        movlw   b'11011111'             ; Código CGROM del símbolo "º".
        movwf   LCD_Dato
LCD_FinCGROM
        movf    LCD_Dato,W              ; En (W) el código buscado.
        return

; Subrutina "LCD_DosEspaciosBlancos" y "LCD_LineaBlanco" ---------------------------------
;
; Visualiza espacios en blanco.

LCD_LineaEnBlanco
        movlw   LCD_CaracteresPorLinea
        goto    LCD_EnviaBlancos
LCD_UnEspacioBlanco
        movlw   .1
        goto    LCD_EnviaBlancos
LCD_DosEspaciosBlancos
        movlw   .2
        goto    LCD_EnviaBlancos
LCD_TresEspaciosBlancos
        movlw   .3
LCD_EnviaBlancos
        movwf   LCD_Auxiliar1           ; (LCD_Auxiliar1) se utiliza como contador.
LCD_EnviaOtroBlanco
        movlw   ' '                     ; Esto es un espacio en blanco.
        call    LCD_Caracter            ; Visualiza tanto espacios en blanco como se
        decfsz  LCD_Auxiliar1,F         ; haya cargado en (LCD_Auxiliar1).
        goto    LCD_EnviaOtroBlanco
        return

; Subrutinas "LCD_ByteCompleto" y "LCD_Byte" ----------------------------------------------
;
; Subrutina "LCD_ByteCompleto", visualiza el byte que almacena el registro W en el
; lugar actual de la pantalla. Por ejemplo, si (W)=b'10101110' visualiza "AE".
;
; Subrutina "LCD_Byte" igual que la anterior, pero en caso de que el nibble alto sea cero
; visualiza en su lugar un espacio en blanco. Por ejemplo si (W)=b'10101110' visualiza "AE"
; y si (W)=b'00001110', visualiza " E" (un espacio blanco delante).
```

```
;
; Utilizan la subrutina "LCD_Nibble" que se analiza más adelante.
;
LCD_Byte
        movwf   LCD_Auxiliar2           ; Guarda el valor de entrada.
        andlw   b'11110000'             ; Analiza si el nibble alto es cero.
        btfss   STATUS,Z                ; Si es cero lo apaga.
        goto    LCD_VisualizaAlto       ; No es cero y lo visualiza.
        movlw   ' '                     ; Visualiza un espacio en blanco.
        call    LCD_Caracter
        goto    LCD_VisualizaBajo

LCD_ByteCompleto
        movwf   LCD_Auxiliar2           ; Guarda el valor de entrada.
LCD_VisualizaAlto
        swapf   LCD_Auxiliar2,W         ; Pone el nibble alto en la parte baja.
        call    LCD_Nibble              ; Lo visualiza.
LCD_VisualizaBajo
        movf    LCD_Auxiliar2,W         ; Repite el proceso con el nibble bajo.
;       call    LCD_Nibble              ; Lo visualiza.
;       return
; Subrutina "LCD_Nibble" ------------------------------------------------------------
;
; Visualiza en el lugar actual de la pantalla, el valor hexadecimal que almacena en el nibble
; bajo del registro W. El nibble alto de W no es tenido en cuenta. Ejemplos:
; - Si (W)=b'01010110', se visualizará "6".
; - Si (W)=b'10101110', se visualizará "E".
;
LCD_Nibble
        andlw   b'00001111'             ; Se queda con la parte baja.
        movwf   LCD_Auxiliar1           ; Lo guarda.
        sublw   0x09                    ; Comprueba si hay que representarlo con letra.
        btfss   STATUS,C
        goto    LCD_EnviaByteLetra
        movf    LCD_Auxiliar1,W
        addlw   '0'                     ; El número se pasa a carácter ASCII sumándole
        goto    LCD_FinVisualizaDigito  ; el ASCII del cero y lo visualiza.
LCD_EnviaByteLetra
        movf    LCD_Auxiliar1,W
        addlw   'A'-0x0A                ; Sí, por tanto, se le suma el ASCII de la 'A'.
LCD_FinVisualizaDigito
        goto    LCD_Caracter            ; Y visualiza el carácter. Se hace con un "goto"
                                        ; para no sobrecargar la pila.
```

B.5. LCD_MENS.INC

```
;***************************** Librería "LCD_MENS.INC" ******************************
;
;
; Librería de subrutinas para el manejo de mensajes a visualizar en un visualizador LCD.

        CBLOCK
        LCD_ApuntaCaracter      ; Indica la posición del carácter a visualizar
                                ; respecto del comienzo de todos los mensajes,
                                ; (posición de la etiqueta "Mensajes").
        LCD_ValorCaracter       ; Código ASCII del carácter a
        ENDC                    ; visualizar.

; Los mensajes tienen que estar situados dentro de las 256 primeras posiciones de la
; memoria de programa, es decir, no pueden superar la dirección 0FFh.

; Subrutina "LCD_Mensaje" ---------------------------------------------------------
;
; Visualiza por pantalla el mensaje apuntado por el registro W.
;
; Los mensajes deben localizarse dentro de una zona encabezada por la etiqueta
"Mensajes" y que
; tenga la siguiente estructura:
;
; Mensajes                      ; ¡Etiqueta obligatoria!
;       addwf   PCL,F
; Mensaje0                      ; Posición inicial del mensaje.
;       DT ".. ..", 0x00        ; Mensaje terminado en 0x00.
; Mensaje1
;       ...
;       ...
; FinMensajes
;
; La llamada a esta subrutina se realizará siguiendo este ejemplo:
;
;       movlw   Mensaje0        ; Carga la posición del mensaje.
;       call    LCD_Mensaje     ; Visualiza el mensaje.
;
LCD_Mensaje
        movwf   LCD_ApuntaCaracter  ; Posición del primer carácter del mensaje.
        movlw   Mensajes            ; Halla la posición relativa del primer carácter
        subwf   LCD_ApuntaCaracter,F ; del mensaje respecto de etiqueta "Mensajes".
        decf    LCD_ApuntaCaracter,F ; Compensa la posición que ocupa "addwf PCL,F".
LCD_VisualizaOtroCaracter
        movf    LCD_ApuntaCaracter,W
        call    Mensajes            ; Obtiene el código ASCII del carácter apuntado.
        movwf   LCD_ValorCaracter   ; Guarda el valor de carácter.
```

```
        movf    LCD_ValorCaracter,F   ; Lo único que hace es posicionar flag Z. En caso
        btfsc   STATUS,Z              ; que sea "0x00", que es código indicador final
        goto    LCD_FinMensaje        ; de mensaje, sale fuera.
LCD_NoUltimoCaracter
        call    LCD_Caracter          ; Visualiza el carácter ASCII leído.
        incf    LCD_ApuntaCaracter,F ; Apunta a la posición del siguiente carácter
        goto    LCD_VisualizaOtroCaracter     ; dentro del mensaje.
LCD_FinMensaje
        return                        ; Vuelve al programa principal.

; Subrutina "LCD_MensajeMovimiento" ----------------------------------------------------
;
; Visualiza un mensaje de mayor longitud que los 16 caracteres que pueden representarse
; en una línea, por tanto se desplaza a través de la pantalla.
;
; En el mensaje debe dejarse 16 espacios en blanco, al principio y al final para
; conseguir que el desplazamiento del mensaje sea lo más legible posible.
;
        CBLOCK
        LCD_CursorPosicion            ; Contabiliza la posición del cursor dentro de la
        ENDC                          ; pantalla LCD

LCD_MensajeMovimiento
        movwf   LCD_ApuntaCaracter    ; Posición del primer carácter del mensaje.
        movlw   Mensajes              ; Halla la posición relativa del primer carácter
        subwf   LCD_ApuntaCaracter,F ; del mensaje respecto de la etiqueta "Mensajes".
        decf    LCD_ApuntaCaracter,F ; Compensa la posición que ocupa "addwf PCL,F".
LCD_PrimeraPosicion
        clrf    LCD_CursorPosicion    ; El cursor en la posición 0 de la línea.
        call    LCD_Borra             ; Se sitúa en la primera posición de la línea 1 y
LCD_VisualizaCaracter                 ; borra la pantalla.
        movlw   LCD_CaracteresPorLinea        ; ¿Ha llegado a final de línea?
        subwf   LCD_CursorPosicion,W
        btfss   STATUS,Z
        goto    LCD_NoEsFinalLinea
LCD_EsFinalLinea
        call    Retardo_200ms         ; Lo mantiene visualizado durante este tiempo.
        call    Retardo_200ms
        movlw   LCD_CaracteresPorLinea-1 ; Apunta a la posición del segundo carácter
visualizado
        subwf   LCD_ApuntaCaracter,F ; en pantalla, que será el primero en la siguiente
        goto    LCD_PrimeraPosicion   ; visualización de línea, para producir el efecto
LCD_NoEsFinalLinea                    ; de desplazamiento hacia la izquierda.
        movf    LCD_ApuntaCaracter,W
        call    Mensajes              ; Obtiene el ASCII del carácter apuntado.
        movwf   LCD_ValorCaracter     ; Guarda el valor de carácter.
        movf    LCD_ValorCaracter,F   ; Lo único que hace es posicionar flag Z. En caso
        btfsc   STATUS,Z              ; que sea "0x00", que es código indicador final
        goto    LCD_FinMovimiento     ; de mensaje, sale fuera.
```

```
LCD_NoUltimoCaracter2
        call    LCD_Caracter            ; Visualiza el carácter ASCII leído.
        incf    LCD_CursorPosicion,F    ; Contabiliza el incremento de posición del
                                        ; cursor en la pantalla.
        incf    LCD_ApuntaCaracter,F    ; Apunta a la siguiente posición por visualizar.
        goto    LCD_VisualizaCaracter   ; Vuelve a visualizar el siguiente carácter
LCD_FinMovimiento                       ; de la línea.
        return                          ; Vuelve al programa principal.
```

B.6. M24CXX.INC

```
;***************************** Librería "M24CXX.INC" **********************************
; Estas subrutinas permiten realizar las tareas de manejo de la memoria EEPROM serie
; 24Cxx que transmite y recibe la información vía serie a través de un bus I2C.
; Subrutina "M24Cxx_InicializaEscritura" -----------------------------------------------
; Prepara la memoria para iniciar su escritura a partir de la posición de memoria fijada
; por la palabra de control  el bit R/w debe estar en cero:
; (M24Cxx_AddressLow), indica posición a escribir dentro del bloque, se debe iniciar en 00h.
        CBLOCK
        M24Cxx_Palabracontrol               ; Guarda el valor de la dirección de memoria a
        M24Cxx_AddressLow           ; escribir o leer.
        M24Cxx_Dato
        ENDC
M24Cxx_InicializaEscritura
        call    I2C_EnviaStart              ; Envía condición de Start.
        movf    M24Cxx_Palabracontrol,w     ; Envía dirección de escritura del
        call    I2C_EnviaByte               ; esclavo.
                                            ; A partir de la dirección apuntada por el
                                            ; registro
        movf    M24Cxx_AddressLow,W         ; M24Cxx_AddressLow.
        call    I2C_EnviaByte
        return


; Subrutina "M24Cxx_InicializaLectura" -------------------------------------------------
; Prepara la memoria para iniciar su lectura a partir de la posición de memoria fijada
; por el registro : M24Cxx_Palabracotrol el bit R/W debe estar a 1
; - (M24Cxx_AddressLow), indica posición a escribir dentro del bloque.
M24Cxx_InicializaLectura
        bcf     I2C_UltimoByteLeer      ; Todavía no ha comenzado a leer ningún dato.
        call    I2C_EnviaStart          ; Envía condición de Start.
        bcf     M24Cxx_Palabracontrol,0     ; Configura temporalmente en modo escritura.
        movf    M24Cxx_Palabracontrol,w     ; Envía dirección de escritura del esclavo.
        bsf     M24Cxx_Palabracontrol,0     ; Recupera el modo lectura.

        call    I2C_EnviaByte               ;

                                            ;

        movf    M24Cxx_AddressLow,W         ; Primera posición de la página seleccionada
```

```
        call    I2C_EnviaByte
        call    I2C_EnviaStop
;
        call    I2C_EnviaStart              ; Envía condición de Start.
        movf    M24Cxx_Palabracontrol,w     ; Indica a la memoria M24Cxx que va a
        call    I2C_EnviaByte               ; proceder a la lectura.
        return

; Subrutina "M24Cxx_Mensaje_a_LCD" -----------------------------------------------------
;
; Lee el mensaje grabado en la memoria M24Cxx y lo visualiza en la pantalla del módulo LCD.
; En la palabra de control se introduce la página de la memoria a partir de cuya primera
; posición se va a leer. La lectura termina cuando encuentre el código 0x00. Así por ejemplo,
; si (Ao)=0 lee el mensaje que comienza en la posición 0 de la página 00h de la memoria,
; que es la posición 0000h absoluta.O si (Ao)=1 lee el mensaje que comienza en la posición 0 de la
; página 01h de la memoria que es posición 0100h absoluta. y asi sucesivamente

        CBLOCK
;       M24Cxx_ValorCaracter        ; Valor ASCII del carácter leído.
        ENDC

;M24Cxx_Mensaje_a_LCD
                                    ; Apunta al inicio de la página correspondiente.
        clrf    M24Cxx_AddressLow
        call    M24Cxx_InicializaLectura
M24Cxx_LeeOtroByte
        call    I2C_LeeByte         ; Lee la memoria M24Cxx.
        movwf   M24Cxx_ValorCaracter ; Guarda el valor de carácter.
        movf    M24Cxx_ValorCaracter,F      ; Lo único que hace es posicionar flag Z. En caso
        btfsc   STATUS,Z            ; que sea "0x00", que es código indicador final
        goto    M24Cxx_FinMensaje   ; del mensaje, sale de la subrutina.
        movf    M24Cxx_ValorCaracter,W      ; Recupera el valor leído.
        call    LCD_Caracter        ; Lo visualiza en la pantalla del LCD.
        incf    M24Cxx_AddressLow,F; Apunta a la siguiente posición.
        goto    M24Cxx_LeeOtroByte
;M24Cxx_FinMensaje
;       call    24Cxx_FinalizaLectura
;       return

; Subrutina "M24Cxx_FinalizaLectura" ---------------------------------------------
;
; Activa el bit I2C_UltimoByteLeer para que la subrutina I2C_LeeByte ponga en alta
; impedancia la línea SDA y pueda ejecutarse posteriormente la condición de Start o Stop
; que fija el protocolo del bus I2C.

M24Cxx_FinalizaLectura
        bsf     I2C_UltimoByteLeer  ; Con estas dos instrucciones se pone en
        call    I2C_LeeByte         ; alta impedancia la línea SDA. No importa el
        call    I2C_EnviaStop       ; resultado de la lectura realizada.
```

```
        return

; Subrutina "M24Cxx_Mensaje_a_LCD" ---------------------------------------------------
;
; Lee un mensaje grabado en la memoria M24Cxx y lo visualiza por el módulo LCD. En caso que
; sea más largo que la longitud de la pantalla se desplaza hacia la izquierda con sensación
; de movimiento. En la palabra de control ssse introduce la página de la memoria a partir de cuya
; primera posición va a leer. La visualización termina cuando encuentre el código
; 0x00. Así por ejemplo si (Ao)= 0 lee el mensaje que comienza en la posición 0 de la
; página 00h de la memoria, que es la posición 0000h absoluta. Y si (Ao)= 1 lee el mensaje que
; comienza en la posición 0 de la página 01h de la memoria, que es la posición 0100h absoluta.
; y así sucesivamente.

        CBLOCK
        M24Cxx_ValorCaracter        ; Valor ASCII del carácter leído.
        M24Cxx_CursorPosicion
        ENDC

M24Cxx_Mensaje_a_LCD
        bcf     I2C_UltimoByteLeer
                                    ; Apunta al inicio de la página correspondiente.
        clrf    M24Cxx_AddressLow
        call    M24Cxx_InicializaLectura
M24Cxx_PrimeraPosicion
        clrf    M24Cxx_CursorPosicion       ; El cursor en la posición 0 de la línea.
        call    LCD_Borra           ; Se sitúa en la primera posición de la línea 1 y
M24Cxx_VisualizaCaracter                    ; borra la pantalla.
        movlw   LCD_CaracteresPorLinea      ; ¿Ha llegado a final de línea?
        subwf   M24Cxx_CursorPosicion,W
        btfss   STATUS,C
        goto    M24Cxx_NoEsFinalLinea
M24Cxx_EsFinalLinea
        call    Retardo_200ms               ; Lo mantiene visualizado durante este tiempo.
        call    Retardo_200ms
        call    M24Cxx_FinalizaLectura
        incf    M24Cxx_AddressLow,F
        call    M24Cxx_InicializaLectura
        goto    M24Cxx_PrimeraPosicion
M24Cxx_NoEsFinalLinea
        call    I2C_LeeByte         ; Obtiene el ASCII del carácter apuntado.
        movwf   M24Cxx_ValorCaracter ; Guarda el valor de carácter.
        movf    M24Cxx_ValorCaracter,F      ; Lo único que hace es posicionar flag Z. En caso
        btfsc   STATUS,Z            ; que sea "0x00", que es código indicador final
        goto    M24Cxx_FinMensaje   ; de mensaje, sale fuera.
M24Cxx_NoUltimoCaracter
        call    LCD_Caracter        ; Visualiza el carácter ASCII leído.
        incf    M24Cxx_CursorPosicion,F     ; Contabiliza el incremento de posición del
                                    ; Cursor en la pantalla.
        goto    M24Cxx_VisualizaCaracter    ; Vuelve a visualizar el siguiente carácter
```

```
M24Cxx_FinMensaje                   ; de la línea.
        call    M24Cxx_FinalizaLectura
        return
;
```

B.7. RETARDOS.INC

```
;***************************** Librería "RETARDOS.INC" ********************************
; Librería con múltiples subrutinas de retardos, desde 4 microsegundos hasta 20 segundos.
; Además se pueden implementar otras subrutinas muy fácilmente.
; Se han calculado para un sistema microcontrolador con un PIC trabajando con un cristal
; de cuarzo a 4 MHz. Como cada ciclo máquina son 4 ciclos de reloj, resulta que cada
; ciclo máquina tarda 4 x 1/4MHz = 1 µs.
; En los comentarios, "cm" significa "ciclos máquina".
; ZONA DE DATOS ******************************************************************

        CBLOCK
        R_ContA                             ; Contadores para los retardos.
        R_ContB
        R_ContC
        ENDC
; RETARDOS de 4 hasta 10 microsegundos ------------------------------------------------
; A continuación retardos pequeños teniendo en cuenta que para una frecuencia de 4 MHZ,
; la llamada a subrutina "call" tarda 2 ciclos máquina, el retorno de subrutina
; "return" toma otros 2 ciclos máquina y cada instrucción "nop" tarda 1 ciclo máquina.
;
Retardo_10micros                    ; La llamada "call" aporta 2 ciclos máquina.
        nop                         ; Aporta 1 ciclo máquina.
        nop                         ; Aporta 1 ciclo máquina.
        nop                         ; Aporta 1 ciclo máquina.
        nop                         ; Aporta 1 ciclo máquina.
        nop                         ; Aporta 1 ciclo máquina.
Retardo_5micros                     ; La llamada "call" aporta 2 ciclos máquina.
        nop                         ; Aporta 1 ciclo máquina.
Retardo_4micros                     ; La llamada "call" aporta 2 ciclos máquina.
        return                      ; El salto del retorno aporta 2 ciclos máquina.
; RETARDOS de 20 hasta 500 microsegundos ----------------------------------------------
Retardo_500micros                   ; La llamada "call" aporta 2 ciclos máquina.
        nop                         ; Aporta 1 ciclo máquina.
        movlw   d'164'              ; Aporta 1 ciclo máquina. Este es el valor de "K".
        goto    RetardoMicros       ; Aporta 2 ciclos máquina.
Retardo_200micros                   ; La llamada "call" aporta 2 ciclos máquina.
        nop                         ; Aporta 1 ciclo máquina.
        movlw   d'64'               ; Aporta 1 ciclo máquina. Este es el valor de "K".
        goto    RetardoMicros       ; Aporta 2 ciclos máquina.
Retardo_100micros                   ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'31'               ; Aporta 1 ciclo máquina. Este es el valor de "K".
        goto    RetardoMicros       ; Aporta 2 ciclos máquina.
Retardo_50micros                    ; La llamada "call" aporta 2 ciclos máquina.
        nop                         ; Aporta 1 ciclo máquina.
```

```
        movlw   d'14'                   ; Aporta 1 ciclo máquina. Este es el valor de "K".
        goto    RetardoMicros           ; Aporta 2 ciclos máquina.
Retardo_20micros                        ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'5'                    ; Aporta 1 ciclo máquina. Este es el valor de "K".
; El próximo bloque "RetardoMicros" tarda:
; 1 + (K-1) + 2 + (K-1)x2 + 2 = (2 + 3K) ciclos máquina.


RetardoMicros
        movwf   R_ContA                         ; Aporta 1 ciclo máquina.
Rmicros_Bucle
        decfsz  R_ContA,F               ; (K-1)x1 cm (cuando no salta) + 2 cm (al saltar).
        goto    Rmicros_Bucle           ; Aporta (K-1)x2 ciclos máquina.
        return                          ; El salto del retorno aporta 2 ciclos máquina.
;
; En total estas subrutinas tardan:
; - Retardo_500micros:  2 + 1 + 1 + 2 + (2 + 3K) = 500 cm = 500 µs. (para K=164 y 4 MHz).
; - Retardo_200micros:  2 + 1 + 1 + 2 + (2 + 3K) = 200 cm = 200 µs. (para K= 64 y 4 MHz).
; - Retardo_100micros:  2     + 1 + 2 + (2 + 3K) = 100 cm = 100 µs. (para K= 31 y 4 MHz).
; - Retardo_50micros :  2 + 1 + 1 + 2 + (2 + 3K) =  50 cm =  50 µs. (para K= 14 y 4 MHz).
; - Retardo_20micros :  2     + 1     + (2 + 3K) =  20 cm =  20 µs. (para K=  5 y 4 MHz).
;
; RETARDOS de 1 ms hasta 200 ms. ----------------------------------------------------------
Retardo_200ms                           ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'200'                  ; Aporta 1 ciclo máquina. Este es el valor de "M".
        goto    Retardos_ms             ; Aporta 2 ciclos máquina.
Retardo_100ms                           ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'100'                  ; Aporta 1 ciclo máquina. Este es el valor de "M".
        goto    Retardos_ms             ; Aporta 2 ciclos máquina.
Retardo_50ms                            ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'50'                   ; Aporta 1 ciclo máquina. Este es el valor de "M".
        goto    Retardos_ms             ; Aporta 2 ciclos máquina.
Retardo_20ms                            ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'20'                   ; Aporta 1 ciclo máquina. Este es el valor de "M".
        goto    Retardos_ms             ; Aporta 2 ciclos máquina.
Retardo_10ms                            ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'10'                   ; Aporta 1 ciclo máquina. Este es el valor de "M".
        goto    Retardos_ms             ; Aporta 2 ciclos máquina.
Retardo_5ms                             ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'5'                    ; Aporta 1 ciclo máquina. Este es el valor de "M".
        goto    Retardos_ms             ; Aporta 2 ciclos máquina.
Retardo_2ms                             ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'2'                    ; Aporta 1 ciclo máquina. Este es el valor de "M".
        goto    Retardos_ms             ; Aporta 2 ciclos máquina.
Retardo_1ms                             ; La llamada "call" aporta 2 ciclos máquina.
        movlw   d'1'                    ; Aporta 1 ciclo máquina. Este es el valor de "M".
;
; El próximo bloque "Retardos_ms" tarda:
; 1 + M + M + KxM + (K-1)xM + Mx2 + (K-1)Mx2 + (M-1) + 2 + (M-1)x2 + 2 =
```

```
; = (2 + 4M + 4KM) ciclos máquina. Para K=249 y M=1 supone 1002 ciclos máquina
; que a 4 MHz son 1002 µs = 1 ms.
;
Retardos_ms
        movwf  R_ContB              ; Aporta 1 ciclo máquina.
R1ms_BucleExterno
        movlw  d'249'               ; Aporta Mx1 ciclos máquina. Este es el valor de "K".
        movwf  R_ContA              ; Aporta Mx1 ciclos máquina.
R1ms_BucleInterno
        nop                         ; Aporta KxMx1 ciclos máquina.
        decfsz  R_ContA,F           ; (K-1)xMx1 cm (cuando no salta) + Mx2 cm (al saltar).
        goto    R1ms_BucleInterno   ; Aporta (K-1)xMx2 ciclos máquina.
        decfsz  R_ContB,F           ; (M-1)x1 cm (cuando no salta) + 2 cm (al saltar).
        goto    R1ms_BucleExterno   ; Aporta (M-1)x2 ciclos máquina.
        return                      ; El salto del retorno aporta 2 ciclos máquina.
;
;En total estas subrutinas tardan:
; - Retardo_200ms:     2 + 1 + 2 + (2 + 4M + 4KM) = 200007 cm = 200 ms. (M=200 y K=249).
; - Retardo_100ms:     2 + 1 + 2 + (2 + 4M + 4KM) = 100007 cm = 100 ms. (M=100 y K=249).
; - Retardo_50ms :     2 + 1 + 2 + (2 + 4M + 4KM) =  50007 cm =  50 ms. (M= 50 y K=249).
; - Retardo_20ms :     2 + 1 + 2 + (2 + 4M + 4KM) =  20007 cm =  20 ms. (M= 20 y K=249).
; - Retardo_10ms :     2 + 1 + 2 + (2 + 4M + 4KM) =  10007 cm =  10 ms. (M= 10 y K=249).
; - Retardo_5ms  :     2 + 1 + 2 + (2 + 4M + 4KM) =   5007 cm =   5 ms. (M=  5 y K=249).
; - Retardo_2ms  :     2 + 1 + 2 + (2 + 4M + 4KM) =   2007 cm =   2 ms. (M=  2 y K=249).
; - Retardo_1ms  :     2 + 1     + (2 + 4M + 4KM) =   1005 cm =   1 ms. (M=  1 y K=249).
;
; RETARDOS de 0.5 hasta 20 segundos ----------------------------------------------------
;
Retardo_20s                         ; La llamada "call" aporta 2 ciclos máquina.
        movlw  d'200'               ; Aporta 1 ciclo máquina. Este es el valor de "N".
        goto   Retardo_1Decima      ; Aporta 2 ciclos máquina.
Retardo_10s                         ; La llamada "call" aporta 2 ciclos máquina.
        movlw  d'100'               ; Aporta 1 ciclo máquina. Este es el valor de "N".
        goto   Retardo_1Decima      ; Aporta 2 ciclos máquina.
Retardo_5s                          ; La llamada "call" aporta 2 ciclos máquina.
        movlw  d'50'                ; Aporta 1 ciclo máquina. Este es el valor de "N".
        goto   Retardo_1Decima      ; Aporta 2 ciclos máquina.
Retardo_2s                          ; La llamada "call" aporta 2 ciclos máquina.
        movlw  d'20'                ; Aporta 1 ciclo máquina. Este es el valor de "N".
        goto   Retardo_1Decima      ; Aporta 2 ciclos máquina.
Retardo_1s                          ; La llamada "call" aporta 2 ciclos máquina.
        movlw  d'10'                ; Aporta 1 ciclo máquina. Este es el valor de "N".
        goto   Retardo_1Decima      ; Aporta 2 ciclos máquina.
Retardo_500ms                       ; La llamada "call" aporta 2 ciclos máquina.
        movlw  d'5'                 ; Aporta 1 ciclo máquina. Este es el valor de "N".
;
; El próximo bloque "Retardo_1Decima" tarda:
; 1 + N + N + MxN + MxN + KxMxN + (K-1)xMxN + MxNx2 + (K-1)xMxNx2 +
;   + (M-1)xN + Nx2 + (M-1)xNx2 + (N-1) + 2 + (N-1)x2 + 2 =
```

```
; = (2 + 4M + 4MN + 4KM) ciclos máquina. Para K=249, M=100 y N=1 supone 100011
; ciclos máquina que a 4 MHz son 100011 µs = 100 ms = 0,1 s = 1 décima de segundo.
;
Retardo_1Decima
        movwf   R_ContC                 ; Aporta 1 ciclo máquina.
R1Decima_BucleExterno2
        movlw   d'100'                  ; Aporta Nx1 ciclos máquina. Este es el valor de "M".
        movwf   R_ContB                 ; Aporta Nx1 ciclos máquina.
R1Decima_BucleExterno
        movlw   d'249'                  ; Aporta MxNx1 ciclos máquina. Este es el valor de "K".
        movwf   R_ContA                 ; Aporta MxNx1 ciclos máquina.
R1Decima_BucleInterno
        nop                             ; Aporta KxMxNx1 ciclos máquina.
        decfsz  R_ContA,F               ; (K-1)xMxNx1 cm (si no salta) + MxNx2 cm (al saltar).
        goto    R1Decima_BucleInterno           ; Aporta (K-1)xMxNx2 ciclos máquina.
        decfsz  R_ContB,F               ; (M-1)xNx1 cm (cuando no salta) + Nx2 cm (al saltar).
        goto    R1Decima_BucleExterno           ; Aporta (M-1)xNx2 ciclos máquina.
        decfsz  R_ContC,F               ; (N-1)x1 cm (cuando no salta) + 2 cm (al saltar).
        goto    R1Decima_BucleExterno2          ; Aporta (N-1)x2 ciclos máquina.
        return                          ; El salto del retorno aporta 2 ciclos máquina.
;

;En total estas subrutinas tardan:
; - Retardo_20s:        2 + 1 + 2 + (2 + 4N + 4MN + 4KMN) = 20000807 cm = 20 s.
;                       (N=200, M=100 y K=249).
; - Retardo_10s:        2 + 1 + 2 + (2 + 4N + 4MN + 4KMN) = 10000407 cm = 10 s.
;                       (N=100, M=100 y K=249).
; - Retardo_5s:         2 + 1 + 2 + (2 + 4N + 4MN + 4KMN) =  5000207 cm =  5 s.
;                       (N= 50, M=100 y K=249).
; - Retardo_2s:         2 + 1 + 2 + (2 + 4N + 4MN + 4KMN) =  2000087 cm =  2 s.
;                       (N= 20, M=100 y K=249).
; - Retardo_1s:         2 + 1 + 2 + (2 + 4N + 4MN + 4KMN) =  1000047 cm =  1 s.
;                       (N= 10, M=100 y K=249).
; - Retardo_500ms:      2 + 1     + (2 + 4N + 4MN + 4KMN) =   500025 cm = 0,5 s.
;                       (N=  5, M=100 y K=249).
;
```

B.8. TECLADO.INC

```
;***************************** Librería "TECLADO.INC" **********************************

;       ======================================================================
; Librería de subrutinas para la gestión de un teclado organizado en una matriz de 4 x 4 y
; conectado al Puerto B según la disposición siguiente y explicada en la figura 19-2 del libro:
;                   RB4 RB5 RB6 RB7
;                     ^   ^   ^   ^
;                   |---|---|---|---|
;       RB0 -->|  0 |  1 |  2 |  3 |
;                   |---|---|---|---|
;       RB1 -->|  4 |  5 |  6 |  7 |
```

```
;                   |----|----|----|----|
;          RB2 -->|  8 |  9 | 10 | 11 |
;                   |----|----|----|----|

;          RB3 -->| 12 | 13 | 14 | 15 |
;                   |----|----|----|----|
;
; Los números que se han dibujado dentro de cada cuadrado son el orden de las teclas
; que no tienen por qué coincidir con lo serigrafiado sobre ellas. El paso del número de orden
; de la tecla al valor que hay serigrafiado sobre la misma se hace con una tabla de conversión.
;
; ZONA DE DATOS *****************************************************************
;
          CBLOCK
          Tecl_TeclaOrden                   ; Orden de la tecla a chequear.
          ENDC

Tecl_UltimaTecla        EQU     d'15'       ; Valor de orden de la última tecla utilizada.

; Subrutina "Teclado_LeeHex" ****************************************************
;
; Cada tecla tiene asignado un número de orden que es contabilizado en la variable
; Tecl_TeclaOrden. Para convertir a su valor según el tipo de teclado en concreto se
; utiliza una tabla de conversión.
; A continuación se expone la relación entre el número de orden de la tecla y los
; valores correspondientes para el teclado hexadecimal más utilizado.
;
;          ORDEN DE TECLA:          TECLADO HEX. UTILIZADO:
;               0   1   2   3               1  2  3  F
;               4   5   6   7               4  5  6  E
;               8   9  10  11               7  8  9  D
;              12  13  14  15               A  0  B  C
; Así, en este ejemplo, la tecla "7" ocupa el orden 8, la tecla "F" ocupa el orden 3 y la
; tecla "9" el orden 10.
; Si cambia el teclado también hay cambiar de tabla de conversión.
; Entrada:      En (W) el orden de la tecla pulsada.
; Salida:       En (W) el valor hexadecimal para este teclado concreto.
;
Teclado_LeeHex
          call      Teclado_LeeOrdenTecla       ; Lee el Orden de la tecla pulsada.
          btfss     STATUS,C            ; ¿Pulsa alguna tecla?, ¿C=1?
          goto      Tecl_FinLeeHex              ; No, por tanto sale.
          call      Tecl_ConvierteOrdenEnHex    ; Lo convierte en su valor real mediante tabla.
          bsf       STATUS,C            ; Vuelve a posicionar el Carry, porque la
Tecl_FinLeeHex                            ; instrucción "addwf PCL,F" lo pone a "0".
          return
;
Tecl_ConvierteOrdenEnHex                  ; Según el teclado utilizado resulta:
          addwf   PCL,F
```

```
        DT      1h,2h,3h,0Fh            ; Primera fila del teclado.
        DT      4h,5h,6h,0Eh            ; Segunda fila del teclado
        DT      7h,8h,9h,0Dh            ; Tercera fila del teclado.
        DT      0Ah,0h,0Bh,0Ch                  ; Cuarta fila del teclado.
Teclado_FinTablaHex
;
; Esta tabla se sitúa al principio de la librería con el propósito de que no supere la
; posición 0FFh de memoria ROM de programa. De todas formas, en caso que así fuera
; visualizaría el siguiente mensaje de error en el proceso de ensamblado:
;
        IF (Teclado_FinTablaHex > 0xFF)
                ERROR           "Atención: La tabla ha superado el tamaño de la página de los"
                MESSG           "primeros 256 bytes de memoria ROM. NO funcionará
correctamente."
        ENDIF

; Subrutina "Teclado_Inicializa" -------------------------------------------------------
;
; Esta subrutina configura las líneas del Puerto B según la conexión del teclado realizada
; y comprueba que no hay pulsada tecla alguna al principio.

Teclado_Inicializa
        bsf     STATUS,RP0              ; Configura las líneas del puerto:
        movlw   b'11110000'             ; <RB7:RB4> entradas, <RB3:RB0> salidas
        movwf   PORTB
        bcf     OPTION_REG,NOT_RBPU     ; Habilita resistencia de Pull-Up del Puerto B.
        bcf     STATUS,RP0              ; Acceso al banco 0.
;       call    Teclado_EsperaDejePulsar
;       return
; Subrutina "Teclado_EsperaDejePulsar" -------------------------------------------------
;Permanece en esta subrutina mientras siga pulsada la tecla.
Teclado_Comprobacion EQU    b'11110000'
Teclado_EsperaDejePulsar:
        movlw   Teclado_Comprobacion; Pone a cero las cuatro líneas de salida del
        movwf   PORTB               ; Puerto B.
Teclado_SigueEsperando
        call    Retardo_20ms        ; Espera a que se estabilicen los niveles de tensión.
        movf    PORTB,W             ; Lee el Puerto B.
        sublw   Teclado_Comprobacion; Si es lo mismo que escribió es que ya no pulsa
        btfss   STATUS,Z            ; tecla alguna.
        goto    Teclado_SigueEsperando
        return
; Subrutina "Teclado_LeeOrdenTecla" ----------------------------------------------------
; Lee el teclado, obteniendo el orden de la tecla pulsada.
; Salida:       En (W) el número de orden de la tecla pulsada. Además Carry se pone a "1" si
;               se pulsa una tecla ó a "0" si no se pulsa tecla alguna.
Teclado_LeeOrdenTecla:
        clrf    Tecl_TeclaOrden     ; Todavía no ha empezado a chequear el teclado.
        movlw   b'11111110'         ; Va a chequear primera fila.
```

```
Tecl_ChequeaFila                ; (Ver esquema de conexión).
        movwf  PORTB            ; Activa la fila correspondiente.
Tecl_Columna1
        btfss   PORTB,4         ; Chequea la 1ª columna buscando un cero.
        goto    Tecl_GuardaValor ; Sí, es cero y por tanto guarda su valor y sale.
        incf    Tecl_TeclaOrden,F ; Va a chequear la siguiente tecla.
Tecl_Columna2                   ; Repite proceso para las siguientes
        btfss   PORTB,5         ; columnas.
        goto    Tecl_GuardaValor
        incf    Tecl_TeclaOrden,F
Tecl_Columna3
        btfss   PORTB,6
        goto    Tecl_GuardaValor
        incf    Tecl_TeclaOrden,F
Tecl_Columna4
        btfss   PORTB,7
        goto    Tecl_GuardaValor
        incf    Tecl_TeclaOrden,F
;
; Comprueba si ha chequeado la última tecla, en cuyo caso sale. Para ello testea si
; el contenido del registro Tecl_TeclaOrden es igual al número de teclas del teclado.
;
Tecl_TerminaColumnas
        movlw   Tecl_UltimaTecla
        subwf   Tecl_TeclaOrden,W ; (W) = (Tecl_TeclaOrden)-Tecl_UltimaTecla.
        btfsc   STATUS,C        ; ¿C=0?, ¿(W) negativo?, ¿(Tecl_TeclaOrden)<15?
        goto    Tecl_NoPulsada  ; No, se ha llegado al final del chequeo.
        bsf     STATUS,C        ; Sí. Va a chequear la siguiente fila.
        rlf     PORTB,W         ; Apunta a la siguiente fila.
        goto    Tecl_ChequeaFila
Tecl_NoPulsada
        bcf     STATUS,C        ; Posiciona C=0, indicando que no ha pulsado
        goto    Tecl_FinTecladoLee ; tecla alguna y sale.
Tecl_GuardaValor
        movf    Tecl_TeclaOrden,W ; El orden de la tecla pulsada en (W) y sale.
        bsf     STATUS,C        ; Como hay tecla tecla pulsada, pone C=1.
Tecl_FinTecladoLee
        return

;       ================================================================
```

C. CODIGOS DE PROGRAMAS

C1. Visualizador_1.asm

```
;********************************** Visualizador_1.asm *********************************
;
;
; Por el Puerto B se encienden y apagan los diodos led.
;
; ZONA DE DATOS ***************************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A             ; Procesador utilizado.
        INCLUDE  <P16F84A.INC>          ; Definición de algunos operandos utilizados.
        CBLOCK 0x0C                     ; Se crea este bloque de constantes para la
        ENDC                            ; librería de retardos.



; ZONA DE CÓDIGOS *************************************************************************

        ORG    0                        ; El programa comienza en la dirección 0.
Inicio
        bsf    STATUS,RP0   ; Pone a 1 el bit 5 del STATUS. Acceso al Banco 1.
        clrf   TRISB        ; Las líneas del Puerto B configuradas como salidas.
        bcf    STATUS,RP0   ; Pone a 0 el bit 5 de STATUS. Acceso al Banco 0.
Principal
        movlw  b'11111111'  ; Carga el registro w con el literal b'11111111'
        movwf  PORTB        ; Carga este valor en el puerto encendiendo todos los led.
        call   Retardo_5s   ; Mantiene encendido por 5 segundos
        clrf   PORTB        ; Apaga todos los led
        call   Retardo_5s   ; Los mantiene apagado por 5 segundos.
        movlw  b'00000001'  ; Carga en w el literal b'00000001'
        movwf  PORTB        ; Carga el puerto B con este literal para encender el led
                            ; conectado al pin RB0
Rotar
        call   Retardo_2s   ; Mantiene encendido el led por dos segundos.
        bcf    STATUS,C     ; Antes de rotar se asegura que bit carry este a "0"
        rlf    PORTB,f      ; Hace una rotación hacia la izquierda para encender
                            ; el siguiente led (RB1)
        btfss  STATUS,C     ; Comprueba si ya se encendieron todos los led
        goto   Rotar        ; Si no se han encendidos todos vuelve a rotar.
        INCLUDE <RETARDOS.INC>  ; Incluye la librería "RETARDOS"
        END                             ; Fin del programa.

;       ================================================================
```

C2. Visualizador_2.asm

```
;********************************** Visualizador_2.asm ***********************************
;
;
; Por el Puerto B se encienden y apagan los diodos led. El encendido se controla
; por medio de los pulsadores conectados a los pines RA4,RA3.
;
; ZONA DE DATOS ******************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A             ; Procesador utilizado.
        INCLUDE  <P16F84A.INC>                  ; Definición de algunos operandos utilizados.
        #DEFINE   SW_RA4 PORTA,4
        #DEFINE   SW_RA3 PORTA,3
        CBLOCK 0x0C                     ; Se crea este bloque de constantes por la
        ENDC                            ; librería de retardos.

; ZONA DE CÓDIGOS ****************************************************************

        ORG    0                        ; El programa comienza en la dirección 0.
Inicio
        bsf    STATUS,RP0               ; Pone a 1 el bit 5 del STATUS. Acceso al Banco 1.
        clrf   TRISB                    ; Las líneas del Puerto B configuradas como salidas.
        movlw  b'00011000'              ;
        movwf  TRISA                    ; Configura los pines RA4 Y RA3  como entrada.
        bcf    STATUS,RP0               ; Pone a 0 el bit 5 de STATUS. Acceso al Banco 0.
        clrf   PORTB                    ; Inicializa el Puerto B en cero
Principal
        btfss  SW_RA4                   ;Verifica si pin RA4 esta pulsado.
        goto   Incrementa       ; Si esta pulsado va hacia la subrutina Incrementa
SwitchRA3
        btfss  SW_RA3           ;Verifica si RA3 esta pulsado.
        goto   Complementa      ;Si esta pulsado va hacia la subrutina Complementa
        goto   Principal        ;Bucle para censar los switch.
Incrementa
        call   Retardo_20ms     ;Retardo anti rebote.
        btfsc  SW_RA4           ;Verifica nuevamente para ver si no es rebote.
        goto   SwitchRA3        ; Si es rebote entonces verifica RA3.
        incf   PORTB,f          ; Incrementa el contenido del puerto B.
EsperaRA4
        btfss  SW_RA4           ; Bucle espera que se libere el pulsador.
        goto   EsperaRA4
        goto   Principal
Complementa
        call   Retardo_20ms     ; Retardo anti rebote
        btfsc  SW_RA3           ;Verifica si no es rebote.
        goto   Principal        ; Si es rebote vuelve a  principal
        comf   PORTB,F          ;Complementa el valor del puerto B
EsperaRA3
        btfss  SW_RA3
        goto   EsperaRA3
        goto   Principal
        INCLUDE <RETARDOS.INC>  ; Incluye la librería "RETARDOS"
        END                             ; Fin del programa.
```

```
;       ================================================================
```

C3. 7Segmento_1.asm

```
;********************************** 7segmento_1.asm ***********************************
;
;
;
;
;
; ZONA DE DATOS ********************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A             ; Procesador utilizado.
        INCLUDE  <P16F84A.INC>          ; Definición de algunos operandos utilizados.
        CBLOCK         0x0C             ;
        Dig1                            ; Registro que almacena las decenas
        Dig2                            ; Registro que almacena las unidades
        Rota                            ; Registro de control para la rotación
        Contador                        ; Contador para el bucle
        ENDC


; ZONA DE CÓDIGOS ******************************************************************

        ORG    0               ; El programa comienza en la dirección 0.
Inicio
   bsf         STATUS,RP0      ;Configura puertos
   movlw       b'00010000'     ;configura pin RA4 como entrada
   movwf       TRISA           ;
   clrf        TRISB           ;configura el puerto B como salida
   bcf         STATUS,RP0      ;selecciona el banco de memoria bajo
   clrf        Dig1            ;inicializa contadores
   clrf        Dig2
   movlw       .100            ;inicializa contador en 100
   movwf       Contador
   movlw       b'00000000'     ;envía ceros a los transistores para apagar
   movwf       PORTA           ;los display
Principal
   decf        Contador,f      ;
   btfsc       STATUS,Z        ;verifica si contador es cero.
   call        Subir           ;llama la rutina de incremento
   movlw       b'00000010'     ;iniciar un 1 en el registro de rotación
   movwf       Rota
   movlw       Dig1            ;con el registro selector (fsr) se apunta
   movwf       FSR             ;al primer dato que se va a mostrar
Disp
   movlw       00h             ;colocar en cero el dato del display
   movwf       PORTB           ;para apagarlos
   movf        Rota, w         ;pasa rotación al registro de trabajo
   movwf       PORTA           ;enciende el transistor (display)
   movf        INDF, w         ;lee el dato del registro apuntado por fsr
   call        Tabla           ;genera el digito de 7 segmentos
   movwf       PORTB           ;envía el digito al puerto b
   call        Retardo_5ms     ;retardo de 5ms para visualización
   btfsc       Rota, 0         ;controla si terminaron las dos rotaciones
   goto        Principal       ;si termino, vuelve desde el comienzo
```

```
   bcf         STATUS,C     ;carry en cero para no afectar las rotaciones
   rrf         Rota,f       ;desplaza el 1 que enciende los displays
   incf        FSR,f        ;incrementa el puntero. Apunta el próximo
   goto        Disp         ;digito a mostrar

Tabla
   addwf       PCL,F        ;genera los números sobre el display
   retlw       b'00111111'  ;genera el 0
   retlw       b'00000110'  ;genera el 1
   retlw       b'01011011'  ;genera el 2
   retlw       b'01001111'  ;genera el 3
   retlw       b'01100110'  ;genera el 4
   retlw       b'01101101'  ;genera el 5
   retlw       b'01111100'  ;genera el 6
   retlw       b'00000111'  ;genera el 7
   retlw       b'01111111'  ;genera el 8
   retlw       b'01100111'  ;genera el 9
Subir                       ; rutina de incremento
   movlw       .100
   movwf       Contador     ;reinicia registro contador
   incf        Dig2,f       ;incrementa el contador de unidades
   movf        Dig2,w       ;carga en registro de trabajo (w) conteo de unidades
   xorlw       0x0A         ;si w era 10, entonces quedara en cero
   btfsc       STATUS,Z     ;si es cero, el flag z queda alto
   call        s10          ;incrementa las decenas
   return

s10                         ; rutina de incremento x 10
   clrf        Dig2         ;pone a cero las unidades
   incf        Dig1,f       ;incrementa el contador de decenas
   movf        Dig1, w      ;carga en registro w el conteo de las decenas
   xorlw       0x0A         ;si w era 10, entonces quedara en cero
   btfsc       STATUS,Z     ;si es cero, el flag z queda alto
   clrf        Dig1         ; Pone a cero Dígito decenas.
   return

   INCLUDE     <RETARDOS.INC>
   END
```

C4. 7Segmento_2.asm

```
;********************************** 7segmento_2.asm ***********************************
;
; Por el Puerto B se encienden y apagan los diodos led.
;
; ZONA DE DATOS ****************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST            P=16F84A            ; Procesador utilizado.
        INCLUDE         <P16F84A.INC>       ; Definición de algunos operandos utilizados.
        CBLOCK          0x0C                ;
        Dig1                                ; Registro que almacena las decenas
        Dig2                                ; Registro que almacena las unidades
        Rota                                ; Registro de control para la rotación
        ENDC
```

```
; ZONA DE CÓDIGOS ********************************************************************

        ORG    0                    ; El programa comienza en la dirección 0.
Inicio
   bsf         STATUS,RP0            ;Configura puertos
   movlw       b'00010000'           ;configura pin RA4 como entrada
   movwf       TRISA                 ;
   clrf        TRISB                 ;configura el puerto B como salida
   bcf         STATUS,RP0            ;selecciona el banco de memoria bajo
   clrf        Dig1                  ;inicializa contadores
   clrf        Dig2
   movlw       b'00000000'           ;envía ceros a los transistores para apagar
   movwf       PORTA                 ;los displays
Principal
   btfss       PORTA, 4              ;chequea el pulsador
  call         Subir           ;llama la rutina de incremento
  movlw        b'00000010'     ;iniciar un 1 en el registro de rotación
  movwf        Rota
  movlw        Dig1            ;con el registro selector (fsr) se apunta
  movwf        FSR             ;al primer dato que se va a mostrar
Disp
  movlw   00h                  ;colocar en cero el dato del display
  movwf   PORTB                ;para apagarlos
  movf    Rota, w              ;pasa rotación al registro de trabajo
  movwf   PORTA                ;enciende el transistor (display)
  movf    INDF, w              ;lee el dato del registro apuntado por fsr
  call    Tabla                ;genera el digito de 7 segmentos
  movwf   PORTB                ;envía el digito al puerto b
  call    Retardo_5ms          ;retardo de 5ms para visualización
  btfsc   Rota, 0              ;controla si terminaron las dos rotaciones
  goto    Principal            ;si terminó, vuelve desde el comienzo
  bcf     STATUS,C             ;carry en cero para no afectar las rotaciones
  rrf     Rota,f               ;desplaza el 1 que enciende los displays
  incf    FSR,f                ;incrementa el puntero. Apunta el próximo
  goto    Disp                 ;digito a mostrar

Tabla
  addwf   PCL,F                ;genera los números sobre el display
  retlw   b'00111111'          ;genera el 0
  retlw   b'00000110'          ;genera el 1
  retlw   b'01011011'          ;genera el 2
  retlw   b'01001111'          ;genera el 3
  retlw   b'01100110'          ;genera el 4
  retlw   b'01101101'          ;genera el 5
  retlw   b'01111100'          ;genera el 6
  retlw   b'00000111'          ;genera el 7
  retlw   b'01111111'          ;genera el 8
  retlw   b'01100111'          ;genera el 9

Subir                          ; rutina de incremento

   incf    Dig2,f              ;incrementa el contador de unidades
   movf    Dig2,w              ;carga en registro de trabajo (w) conteo de unidades
   xorlw   0x0A                ;si w era 10, entonces quedara en cero
```

```
   btfsc   STATUS,Z              ;si es cero, el flag z queda alto
   call    s10                   ;incrementa las decenas
   call    Retardo_100ms         ;retardo de 100ms

   return

s10                              ; rutina de incremento x 10
   clrf    Dig2                  ;pone a cero las unidades
   incf    Dig1,f                ;incrementa el contador de decenas
   movf    Dig1, w               ;carga en registro w el conteo de las decenas
   xorlw   0x0A                  ;si w era 10, entonces quedara en cero
   btfsc   STATUS,Z              ;si es cero, el flag z queda alto
   clrf    Dig1                  ;Pone a cero Dígitos decenas.
   return

   INCLUDE     <RETARDOS.INC>
   END
```

C5. DisplayLCD_1.asm

```
;************************************ Display LCD_1.asm ************************************
;
;=======================================================================
;
;       En el LCD se visualiza el mensaje "UNAN-MANAGUA" en la línea 1
;       y en la línea 2 se visualiza "ING. ELECTRONICA"
; ZONA DE DATOS ********************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A
        INCLUDE <P16F84A.INC>

        CBLOCK  0x0C
        ENDC

; ZONA DE CÓDIGOS ******************************************************************

        ORG    0
Inicio
        call      LCD_Inicializa
        movlw     Mensaje0        ; Apunta dónde se encuentra el mensaje.
        call      LCD_Mensaje     ; Visualiza el mensaje.
        call      LCD_Linea2      ; Posiciona el Cursor en la línea 2
        movlw     Mensaje1        ; Apunta a Mensaje 1
        call      LCD_Mensaje;
sleep                     ; Pasa a modo bajo consumo.

; Mensajes --------------------------------------------------------------------------
;
Mensajes
        addwf   PCL,F
Mensaje0
        DT "UNAN-MANAGUA   ", 0x00
Mensaje1
        DT "ING. ELECTRONICA", 0x00
```

```
        INCLUDE  <LCD_4BIT.INC>
        INCLUDE  <LCD_MENS.INC>
        INCLUDE  <RETARDOS.INC>
        END
```

C6. DisplayLCD_2.asm

```
;***********************************DisplayLCD_2.asm ************************************
;
;
; El módulo LCD visualiza un mensaje largo (más de 16 caracteres) que se desplaza a lo largo
; de la pantalla. Se utiliza la subrutina LCD_MensajeMovimiento de la librería LCD_MENS.INC.
;
; ZONA DE DATOS ********************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST    P=16F84A
        INCLUDE  <P16F84A.INC>

        CBLOCK  0x0C
        ENDC

; ZONA DE CÓDIGOS ******************************************************************

        ORG    0
Inicio
        call    LCD_Inicializa          ; Prepara la pantalla.
Principal
        movlw   Mensaje0                ; Apunta al mensaje.
        call    LCD_MensajeMovimiento
        goto    Principal               ; Repite la visualización.

; "Mensajes" ----------------------------------------------------------------------
;
Mensajes
        addwf   PCL,F
Mensaje0                                ; Posición inicial del mensaje.
        DT "              "             ; Espacios en blanco al principio para mejor
        DT "Estudia programacion "      ; visualización.
        DT "de MICROCONTROLADORES."
        DT "              ", 0x0        ; Espacios en blanco al final.
;
        INCLUDE  <LCD_MENS.INC>         ; Subrutina LCD_MensajeMovimiento.
        INCLUDE  <LCD_4BIT.INC>         ; Subrutinas de control del LCD.
        INCLUDE  <RETARDOS.INC>         ; Subrutinas de retardos.
        END                             ; Fin del programa.

;************************************************************************
```

C7. Teclado_1.asm

```
;************************************ Teclado_1.asm ********************************
;
; ZONA DE DATOS ***************************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A
        INCLUDE  <P16F84A.INC>

        CBLOCK  0x0C
        GuardaTecla                 ; Guarda el valor de la tecla pulsada.
        ContadorCaracteres          ; Guarda el número de caracteres pulsados.
        ENDC

CaracteresUnaLinea    EQU    d'16'    ; Número de caracteres de una línea.
CaracteresDosLineas   EQU    d'32'    ; Número de caracteres de dos líneas.

; ZONA DE CÓDIGOS *************************************************************************

        ORG    0
        goto   Inicio
        ORG    4
        goto   ServicioInterrupcion
Inicio
        call   LCD_Inicializa
        call   Teclado_Inicializa           ; Configura líneas del teclado.
        movlw  b'10001000'                  ; Habilita la interrupción RBI y la general.
        movwf  INTCON
        clrf   ContadorCaracteres           ; En principio no hay caracteres escritos.
Principal
        sleep                               ; Espera en modo bajo consumo que pulse tecla.
        goto   Principal

; Subrutina "ServicioInterrupcion"

ServicioInterrupcion
        call   Teclado_LeeHex               ; Obtiene el valor de la tecla pulsada.
        movwf  GuardaTecla                  ; Reserva el valor de la tecla pulsada.
        movlw  CaracteresUnaLinea           ; Comprueba si ha llegado al número máximo de
        subwf  ContadorCaracteres,W         ; caracteres de una línea.
        btfsc  STATUS,Z
        call   LCD_Linea2                   ; Se sitúa en la segunda línea.
        movlw  CaracteresDosLineas          ; Comprueba si ha llegado al número máximo de
        subwf  ContadorCaracteres,W         ; caracteres de dos líneas.
        btfss  STATUS,Z
        goto   EscribeCaracter              ; No, por tanto, sigue escribiendo.
        call   LCD_Borra                    ; Sí, borra pantalla e inicializa el contador.
        clrf   ContadorCaracteres
EscribeCaracter
        incf   ContadorCaracteres,F         ; Un nuevo carácter escrito.
        movf   GuardaTecla,W
        call   LCD_Nibble                   ; Visualiza el carácter en pantalla.
        call   Teclado_EsperaDejePulsar     ; Para que no se repita el mismo carácter.
        bcf    INTCON,RBIF                  ; Limpia flag.
        retfie
```

```
        INCLUDE  <TECLADO.INC>
        INCLUDE  <LCD_4BIT.INC>
        INCLUDE  <RETARDOS.INC>
        END

;*******************************************************************************
```

C 8. Teclado_2.asm

```
;************************************ Teclado_2.asm *********************************
;
; ZONA DE DATOS ********************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A
        INCLUDE  <P16F84A.INC>
        CBLOCK  0x0C
        ENDC

#DEFINE  LongitudClave         (FinClaveSecreta-ClaveSecreta)
#DEFINE  Dispositivo_Salida    PORTB, 3

; ZONA DE CÓDIGOS ******************************************************************

        ORG    0
        goto    Inicio
        ORG    4
        goto    ServicioInterrupcion

Mensajes
        addwf   PCL,F
MensajeTeclee
        DT      "Ingrese CLAVE:", 0x00
MensajeClaveCorrecta
        DT      "Clave CORRECTA", 0x00
MensajeActivado
        DT      "ACTIVADO", 0x00
MensajeClaveIncorrecta
        DT      "Clave INCORRECTA", 0x00
;
LeeClaveSecreta
        addwf   PCL,F
ClaveSecreta
        DT      0XA,0XB,0XC,0XD             ; Ejemplo de clave secreta.
        DT      0XE,0XF
FinClaveSecreta

Inicio
        call    LCD_Inicializa
        bsf     STATUS,RP0
        bcf     Dispositivo_Salida          ; Define como salida.
        bcf     STATUS,RP0
        call    Teclado_Inicializa          ; Configura las líneas del teclado.
        call    InicializaTodo              ; Inicializa el resto de los registros.
```

```
        movlw   b'10001000'             ; Habilita la interrupción RBI y la general.
        movwf   INTCON
Principal
        sleep                           ; Espera en modo bajo consumo que pulse alguna
tecla.
        goto    Principal

; Subrutina "ServicioInterrupcion"

        CBLOCK
        ContadorCaracteres
        GuardaClaveTecleada
        ENDC

ServicioInterrupcion
        call    Teclado_LeeHex          ; Obtiene el valor hexadecimal de la tecla pulsada.
;
; Según va introduciendo los dígitos de la clave, estos van siendo almacenados a partir de
; las posiciones RAM "ClaveTecleada" mediante direccionamiento indirecto y utilizando el
; FSR como apuntador. Por cada dígito leído en pantalla se visualiza un asterisco.
;
        movwf   INDF                    ; Almacena ese dígito en memoria RAM con
                                        ; con direccionamiento indirecto apuntado por FSR.
        movlw   '*'                     ; Visualiza asterisco.
        call    LCD_Caracter
        incf    FSR,F                   ; Apunta a la próxima posición de RAM.
        incf    ContadorCaracteres,F    ; Cuenta el número de teclas pulsadas.
        movlw   LongitudClave           ; Comprueba si ha introducido tantos caracteres
        subwf   ContadorCaracteres,W    ; como longitud tiene la clave secreta.
        btfss   STATUS,C                ; ¿Ha terminado de introducir caracteres?
        goto    FinInterrupcion         ; No, pues lee el siguiente carácter tecleado.
;
; Si ha llegado aquí es porque ha terminado de introducir el máximo de dígitos. Ahora
; procede a comprobar si la clave es correcta. Para ello va comparando cada uno de los
; dígitos almacenados en las posiciones RAM a partir de "ClaveTecleada" con el valor
; correcto de la clave almacenado en la posición ROM "ClaveSecreta".
;
; Para acceder a las posiciones de memoria RAM a partir de "ClaveTecleada" utiliza
; direccionamiento indirecto siendo FSR el apuntador.
;
; Para acceder a memoria ROM "ClaveSecreta" se utiliza direccionamiento indexado con el
; el registro ContadorCaracteres como apuntador.
;
        call    LCD_Borra               ; Borra la pantalla.
        clrf    ContadorCaracteres      ; Va a leer el primer carácter almacenado en ROM.
        movlw   ClaveTecleada           ; Apunta a la primera posición de RAM donde se ha
        movwf   FSR                     ; guardado la clave tecleada.
ComparaClaves
        movf    INDF,W                  ; Lee la clave tecleada y guardada en RAM.
        movwf   GuardaClaveTecleada     ; La guarda para compararla después.
        movf    ContadorCaracteres,W    ; Apunta al carácter de ROM a leer.
        call    LeeClaveSecreta         ; En (W) el carácter de la clave secreta.
        subwf   GuardaClaveTecleada,W   ; Se comparan.
        btfss   STATUS,Z                ; ¿Son iguales?, ¿Z=1?
        goto    ClaveIncorrecta         ; No, pues la clave tecleada es incorrecta.
        incf    FSR,F                   ; Apunta a la próxima posición de RAM.
```

```
        incf    ContadorCaracteres,F        ; Apunta a la próxima posición de ROM.
        movlw   LongitudClave               ; Comprueba si ha comparado tantos caracteres
        subwf   ContadorCaracteres,W        ; como longitud tiene la clave secreta.
        btfss   STATUS,C                    ; ¿Ha terminado de comparar caracteres?
        goto    ComparaClaves               ; No, pues compara el siguiente carácter.
ClaveCorrecta                               ; La clave ha sido correcta. Aparecen los mensajes
        movlw   MensajeClaveCorrecta        ; correspondientes y permite la activación
        call    LCD_Mensaje                 ; del dispositivo  durante unos segundos.
        call    LCD_Linea2
        movlw   MensajeActivado
        call    LCD_Mensaje
        bsf       Dispositivo_Salida        ; Activa el dispositivo  unos segundos.
        goto    Retardo
ClaveIncorrecta
        movlw   MensajeClaveIncorrecta
        call    LCD_Mensaje
Retardo
        call    Retardo_2s
        call    Retardo_1s
InicializaTodo
        bcf       Dispositivo_Salida        ; Desactiva dispositivo.
        clrf    ContadorCaracteres          ; Inicializa este contador.
        movlw   ClaveTecleada               ; FSR apunta a la primera dirección de la RAM
        movwf   FSR                         ; donde se va a almacenar la clave tecleada.
        call    LCD_Borra                   ; Borra la pantalla.
        movlw   MensajeTeclee               ; Aparece el mensaje para que introduzca la clave.
        call    LCD_Mensaje
        call    LCD_Linea2                  ; Los asteriscos se visualizan en la segunda línea.
FinInterrupcion
        call    Teclado_EsperaDejePulsar
        bcf     INTCON,0
        retfie

        INCLUDE  <TECLADO.INC>
        INCLUDE  <LCD_4BIT.INC>
        INCLUDE  <LCD_MENS.INC>
        INCLUDE  <RETARDOS.INC>

; Las posiciones de memoria RAM donde se guardará la clave leída se definen al final, después
; de los Includes, ya que van a ocupar varias posiciones de memoria mediante el
; direccionamiento indirecto utilizado.

        CBLOCK
        ClaveTecleada
        ENDC

        END                                 ; Fin del programa.

;*******************************************************************************************
```

C9. BusI2C_1.asm

```
;********************************** BUSI2C_01.asm ******************************
; El mensaje tendrá menos de 16 caracteres, que es la máxima longitud de la pantalla.
;
; ZONA DE DATOS ***************************************************************************

        __CONFIG   _CP_OFF &_ WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A
        INCLUDE <P16F84A.INC>

        CBLOCK         0x0C
        ENDC


; ZONA DE CÓDIGOS *************************************************************************

        ORG    0
Inicio
        call       LCD_Inicializa
Principal
        movlw  b'10100000'              ; Observe que la palabra de control termina en 0
                                        ; ya que va a escribir
        movwf  M24Cxx_Palabracontrol    ; Escribe a partir de la dirección 0 de esta
        call       M24Cxx_EscribeMensaje    ; página de la memoria.
        movlw  b'10100001'              ; Observe que la palabra de control termina en 1
                                        ; ya que va a leer
        movwf  M24Cxx_Palabracontrol    ; Lee a partir de la dirección 0 de esta página
        call       M24Cxx_Mensaje_a_LCD     ; de la memoria.
        sleep                           ; Pasa a reposo.

; Subrutina "M24cxx_EscribeMensaje"
; Escribe la memoria 24Cxx con el mensaje grabado al final de este programa a partir
; de la posición 0 de la página de la memoria 24Cxx, señalada por el contenido del
; registro de la "palabra de control"
; El tamaño de memoria que se puede grabar por este procedimiento es de 64 bytes como máximo.

        CBLOCK
        ValorCaracter                   ; Valor ASCII del carácter a escribir.
        ENDC

M24Cxx_EscribeMensaje

        clrf       M24Cxx_AddressLow    ; Inicia en la posición 0
        call       M24Cxx_InicializaEscritura
M24Cxx_EscribeOtroByte
        movf       M24Cxx_AddressLow,W
        call       Mensaje              ; Obtiene el código ASCII del carácter.
        movwf  ValorCaracter            ; Guarda el valor de carácter que ha leído del
        call       I2C_EnviaByte        ; mensaje y lo escribe en la 24Cxx.
        call       Retardo_5ms          ; Tiempo de escritura entre byte y byte.
        movf       ValorCaracter,F      ; Lo único que hace es posicionar flag Z. En caso
        btfsc      STATUS,Z             ; de que sea "0x00", que es código indicador final
        goto       FinEscribeMensaje    ; del mensaje sale de la subrutina.
        incf       M24Cxx_AddressLow,F
        goto       M24Cxx_EscribeOtroByte
```

```
FinEscribeMensaje
        call    I2C_EnviaStop           ; Termina la escritura.
        call    Retardo_5ms
        return

Mensaje
        addwf   PCL,F
        DT " UNAN MANAGUA",0x0

        INCLUDE  <BUS_I2C.INC>          ; Subrutinas de control del bus I2C.
        INCLUDE  <M24CXX.INC>           ; Subrutinas de control de la memoria 24Cxx.
        INCLUDE  <RETARDOS.INC>
        INCLUDE  <LCD_4BIT.INC>
        END
```

C10. BusI2C_2.asm

```
;************************************BUSI2C_02.asm ******************************
; ZONA DE DATOS ********************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A
        INCLUDE  <P16F84A.INC>
        CBLOCK          0x0C
        ENDC

; ZONA DE CÓDIGOS ******************************************************************

        ORG    0
Inicio
        call    LCD_Inicializa
Principal
        movlw   b'10100011'             ; R/W en 1 y pagina 1 (segunda página)
        movwf   M24Cxx_Palabracontrol   ; de la memoria.
        call    M24Cxx_Mensaje_a_LCD    ;
        call    Retardo_1s
        goto    Principal

        INCLUDE  <BUS_I2C.INC>
        INCLUDE  <M24CXX.INC>
        INCLUDE  <RETARDOS.INC>
        INCLUDE  <LCD_4BIT.INC>
        END
```

C11. EEPROM_1.asm

```
;********************************** EEPROM_1.asm *********************************
;
; ZONA DE DATOS ***************************************************************************

        __CONFIG   _CP_OFF &  _WDT_OFF & _PWRTE_ON & _XT_OSC
        LIST       P=16F84A
        INCLUDE  <P16F84A.INC>

        CBLOCK   0x0C
        Contador
        ENDC

        ORG    0x2100                ; Corresponde a la dirección 0 de la zona EEPROM
NumeroTurno                          ; de datos.
        DE     0x00                  ; El número de turno se guarda en esta posición
                                     ; memoria EEPROM de datos.
#DEFINE   Pulsador     PORTA,4

; ZONA DE CÓDIGOS *************************************************************************

        ORG    0
Inicio
        call   LCD_Inicializa
        bsf    STATUS,RP0
        bsf    Pulsador              ; La línea del pulsador se configura como entrada.
        bcf    STATUS,RP0
        movlw  NumeroTurno           ; Lee la posición 0x00 de memoria EEPROM de datos
        call   EEPROM_LeeDato        ; donde se guarda el turno.
        movwf  Contador              ; Se guarda.
        call   Visualiza             ; Lo visualiza.
Principal
        btfss  Pulsador              ; Lee el pulsador.
        call   IncrementaVisualiza   ; Salta a incrementar y visualizar el contador.
        goto   Principal

; Subrutina "IncrementaVisualiza" ---------------------------------------------------

IncrementaVisualiza
        call   Retardo_20ms
        btfsc  Pulsador
        goto   Fin_Visualiza
        incf   Contador,F            ; Incrementa el contador y lo visualiza.
Visualiza
        call   LCD_Linea1
        movlw  MensajeSuTurno
        call   LCD_Mensaje
        movlw  .100                  ; Ahora comprueba si el contador supera la
        subwf  Contador,W            ; centena. (W = Contador-100).
        btfsc  STATUS,C              ; ¿C=0?, ¿W es negativo?, ¿Contador < 100?
        clrf   Contador              ; Pone a cero el contador.
        movf   Contador,W
        call   BIN_a_BCD             ; Lo pasa a BCD.
        call   LCD_Byte              ; Lo visualiza
        movlw  NumeroTurno           ; Se escribe en la posición de memoria EEPROM de
```

```
        movwf   EEADR                   ; datos donde se guarda el turno. En este caso en
        movf    Contador,W              ; la posición 00h de la EEPROM.
        call    EEPROM_EscribeDato
EsperaDejePulsar
        btfss   Pulsador
        goto    EsperaDejePulsar
Fin_Visualiza
        return

Mensajes
        addwf   PCL,F
MensajeSuTurno
        DT "Su Turno: ", 0x00

        INCLUDE   <EEPROM.INC>          ; Control de la EEPROM de datos del PIC.
        INCLUDE   <RETARDOS.INC>
        INCLUDE   <BIN_BCD.INC>
        INCLUDE   <LCD_4BIT.INC>
        INCLUDE   <LCD_MENS.INC>
        END

;*****************************************************************************************************************
```

D. CUESTIONARIO

Cuestionario entregado al docente encargado del laboratorio de electrónica.

1. ¿Encontró la interfaz entrenadora del PIC16F84A propicia para fines pedagógicos?
2. ¿Considera que las prácticas presentadas en este trabajo son útiles para la comprensión del PIC16F84A y su implementación con diferentes tipos de periféricos?
3. ¿Comprendió cómo hacer la grabación de los programas en el PIC16F84A y la diferencia entre la grabación de un microcontrolador, como por ejemplo con la grabación de una EEPROM en nuestro caso la 24C04?
4. ¿Según su criterio los módulos de esta interfaz entrenadora son suficientes para cubrir con los temas de la clase de Microprocesadores y Periféricos?
5. ¿Recomienda esta interfaz entrenadora a un estudiante principiante en la materia de Microprocesadores y Periféricos? Y explique por qué.
6. ¿Considera que el lenguaje ensamblador utilizado en esta interfaz sea comprensible y apto para introducir al estudiante en la programación de PICS? Y explique por qué.
7. Si es positiva su repuesta en la pregunta anterior ¿cree que un estudiante sea capaz de dominar un lenguaje de mayor nivel y por tanto a un PIC con mayores recursos? Y explique por qué.

RESPUESTAS. Por el docente encargado del laboratorio de electrónica.

1) Al desarrollar todas las practicas propuestas, llegue a la conclusión que de que el alumno buena asimilar muy bien el uso básico en la programación del PIC16F84A y por ende le servirá de base para otros modelos de PIC. así que definitivamente es apta para fines pedagógicos.

2) Si se logra asimilar el proceso básico en cuanto a quemar el PIC, y echar a andar el prototipo para el uso con diferentes periféricos, en lo que está un poco limitado es en cuanto a la programación del mismo.

3) La única diferencia entre grabar en el PIC y la eeprom solo está en la selección del dispositivo, de ahí casi es similar todo el proceso.

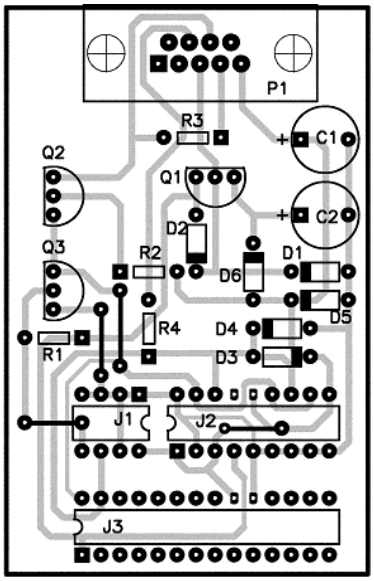
4) Es posible simular muchas otras prácticas con los módulos propuestos, pero creo que si se le diera seguimiento a una versión dos del proyecto, sería bueno incluir otros dispositivos de control, por ejemplo, semáforos, control de motores, agregar módulos de transductores de entrada y salida. etc.

5) Yo lo recomendaría a los alumnos porque no tenemos una herramienta didáctica haciendo uso de PIC dirigía hacia el control de dispositivos.

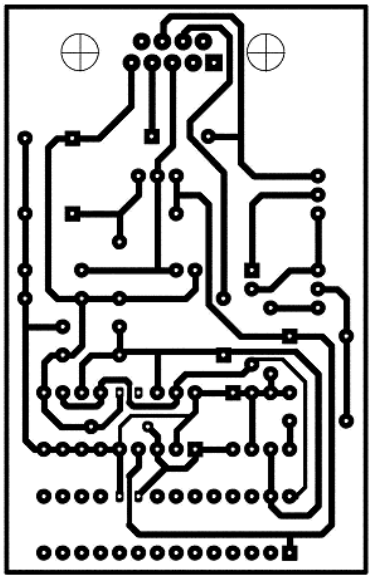
7) Para el manejo y funcionamiento de PIC creo que está apto por qué se puede asimilar de una mejor manera el funcionamiento del mismo, ya para crear aplicaciones más complejas es recomendable usar otros lenguajes de programación.

6) Creo que el uso del lenguaje ensamblador permite al estudiantes comprender a profundidad el funcionamiento del mismo por lo tanto una vez comprendido su funcionamiento se puede explotar mejor los micro controladores usando lenguajes de alto nivel , debido a que permiten crear aplicaciones más complejas.

E. CLONADOR JDM.



Cara de componentes



Cara de pistas

R3=1.5K

C1=22UF/16V

C2-100UF/16V

Q1, Q2-BC237

Q3-BC307

D1, D2, D3, D4-1N4148

D6-8.2V

D5-5.1V

R2-10K

R1-100K

R4-1K

El transistor BC237 Y EL BC 307 Tiene el orden Colector, Base, emisor.2

Mientras que el 2N3904 y 2N3906 tienen el orden Emisor, Base, Colector.