UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE LA COSTA SUR
DEPARTAMENTO DE INGENIERÍAS

Manual de Prácticas de Laboratorio

Microcontroladores

Laboratorio de Electrónica

Elaborado por:

1. ING. JUAN IGNACIO AVELAR MIRANDA
2. ING. JOSÉ VALENTIN AGUIRRE CHAVEZ
3. ING. ISAO PEIRO SUAREZ
4. MTRO. LUIS ALBERTO AMBRIZ LÓPEZ
5. MTRA. ANDREA ALEJANDRA HERNÁNDEZ DEL RIO
6. MTRO. JOSÉ EDUARDO HERNÁNDEZ HARO
7. MTRO. JOSÉ LUIS DOMINGUEZ RUIZ
8. MTRO. JOEL MORAN RODRÍGUEZ
9. DR. JORGE ARTURO PELAYO LÓPEZ
10. DR. DOMINGO VELÁZQUEZ PÉREZ

Presidente de la Academia.

Dr. DOMINGO VELÁZQUEZ PÉREZ

Responsable del Laboratorio de Electrónica.

MTRO. JOSÉ EDUARDO HERNÁNDEZ HARO

Jefe del Departamento de Ingenierías.

DR. DANIEL EDÉN RAMÍREZ ARREOLA

OBJETIVO GENERAL.

El estudiante analizará, diseñará, simulará e implementará circuitos con microcontroladores de 8 a 32 bits, empleando los lenguajes de programación: ensamblador, C y C++ para la configuración de los diversos periféricos que integran los microcontroladores.

CONSIDERACIONES GENERALES

El estudiante debe cumplir con el Reglamento General de Uso de Laboratorios publicado en el “Compendio de reglamentos del Departamento de Ingeniería”.

SEGURIDAD E HIGIENE EN EL USO DEL LABORATORIO

En caso de alguna contingencia (sismo, incendio o cualquier evento que ponga en riesgo su integridad) evacue el laboratorio inmediatamente, siguiendo las normas de seguridad implementadas en los simulacros.

Así mismo es de suma importancia que los usuarios que hagan uso de las instalaciones de los laboratorios, conozcan las ubicaciones de los extintores, botiquines de primeros auxilios y salidas de emergencia.

Es importante resaltar los siguientes puntos referentes a la seguridad e higiene que se deben seguir para el uso de laboratorio y que se encuentran plasmados en el reglamento interno del laboratorio:

1. Mantener y dejar limpia su área de trabajo.
2. No arrojar papeles ni basura al piso.
3. No introducir alimentos y bebidas.
4. No fumar.
5. El alumno deberá dejar su mochila y/o bolsa en los estantes designados para los mismos, respetando todo objeto ajeno que allí se encuentre.
6. Preferentemente no encender radios, grabadoras o cualquier otro aparato que reproduzca música. Sólo lo podrán hacer si se usan audífonos.
7. Está prohibido sentarse sobre las mesas de trabajo o pararse en las sillas.

8. El alumno debe comportarse adecuadamente dentro de las instalaciones del laboratorio, hacer uso apropiado del lenguaje oral y escrito; respetar a sus profesores, compañeras y compañeros de clase.

9. Antes de iniciar las prácticas, el maestro inspeccionará las condiciones físicas del laboratorio y de encontrar situaciones que representen riesgo grave, deberá reportar dicha situación al responsable del laboratorio y/o al asistente o auxiliar del mismo, para que sea corregida, en caso de que no exista la posibilidad de atención inmediata, la práctica será suspendida.

10. Si durante la práctica surgiera una condición que ponga en riesgo grave la Seguridad y Salud de las personas, equipos, materiales o instalaciones se procederá a suspender la práctica debiendo informar de la situación al responsable de laboratorio, asistente o auxiliar del mismo, elaborando por escrito el reporte correspondiente.

11. El profesor deberá cumplir con el uso del equipo de protección personal básico de laboratorio. El Maestro que no cumpla con estos requisitos no podrá realizar la práctica. El Auxiliar notificará la situación al responsable de Laboratorio y/o al Jefe de Departamento quien elaborará un reporte de faltas al reglamento.

12. Es responsabilidad del profesor verificar que antes de iniciar la práctica, todos los alumnos cuenten con el equipo de protección personal y el código de vestimenta necesario para realizar la práctica. El alumno que no cumpla con los requisitos anteriores no podrá realizar la práctica.

13. El profesor deberá asegurarse que los alumnos utilicen adecuadamente el equipo de protección personal durante el desarrollo de la práctica.

14. El profesor llevará un registro de los alumnos que sean observados sin usar su equipo de protección personal o usándolo de manera inadecuada, cada registro contará como una falta al Reglamento del Laboratorio.

15. La acumulación de 4 faltas al Reglamento del Laboratorio implica la suspensión para el alumno de la práctica en el semestre y la no acreditación de esta.

16. El profesor deberá permanecer en el laboratorio durante todo el desarrollo de la práctica.

17. Por razones de Seguridad y Orden está prohibido en el Laboratorio:

    - Correr.
    - Fumar.
    - Usar sombrero, gorra y/o pañoleta en la cabeza.
    - Ingresar personas ajenas a la institución o al grupo que desarrolla la práctica.
    - Usar calzado inadecuado.
    - Usar el cabello largo (las personas con esta característica deberán recoger su cabello y sujetarlo adecuadamente, como medida de prevención contra riesgos).
    - Usar pantalón corto o bermuda.
    - Y en general todo acto y/o conducta que incite al desorden.

18. Todo alumno que sufra una lesión deberá reportarla al maestro encargado de la práctica y de no encontrarse éste, deberá dirigirse con el responsable de Laboratorio y/o asistente de este.

19. Todo trabajador universitario que sufra una lesión deberá reportarla a su jefe inmediato.

20. Todo accidente ocurrido en los laboratorios deberá ser atendido para su control, por la primera persona capacitada y enterada de la situación.

21. Al término de la práctica, el maestro será responsable de supervisar que los alumnos ordenen y limpien su lugar de trabajo. Asegurando que el laboratorio sea entregado a la administración del laboratorio, en condiciones óptimas.

22. La persona que se presente bajo el influjo de alcohol o drogas, que incurra en actos de violencia, daño a la propiedad intencional o negligencia o tome objetos o valores sin autorización será reportado de manera inmediata ante la H. comisión de sanciones del CU Costa Sur.

SEGURIDAD EN LA EJECUCIÓN DE LAS PRÁCTICAS.

Para el desarrollo de las prácticas se pueden presentar los siguientes peligros y su riesgo asociado y es importante que el estudiante los considere y tome las medidas de prevención pertinentes:

No.	Peligro o fuente de energía	Riesgo asociado
1	Manejo de corriente alterna.	Electrochoque, daño a los equipos.
2	Manejo de corriente continua.	Daño a los equipos.

Índice:

Práctica 1

Programación en lenguaje ensamblador

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a usar el software **MPLAB v8.92**.
- Aprender a programar microcontroladores en **lenguaje ensamblador**.
- Aprender a configurar los fusibles de un microcontrolador **PIC16F84A**.
- Simular el microcontrolador mediante el diseño de circuito esquemático en el software **Proteus Professional v8**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **PIC16F84A** [1].
- Software **MPLAB v8.92** [2].
- Software **Proteus Professional v8** [3].
- Programador **PICSTART Plus** [4] o **PICKit 3** [5].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 8 resistencias 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 4MHz.
- 8 LEDs (en caso de que la tarjeta de entrenamiento no los incluya).

Trabajo previo.

- Conocimientos vistos en clase: **Lenguaje ensamblador**.
- Conocimientos vistos en asignaturas previas: **Electrónico analógica y digital**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Identificar el área de trabajo en MPLAB v8.92.

Una vez abierto el software se debe identificar cada uno de los elementos en la GUI como se muestra en la figura 1. De color rojo está la **barra de herramientas**, en color verde el **árbol de proyecto**, en color naranja la **ventana de consumo de memoria**, en color azul el **editor de código**, en morado la **ventana memoria del programa**, donde nos muestra la ubicación del código en la memoria flash, en color magenta la **consola de salida**, donde muestra información, advertencias y errores en la construcción del proyecto.

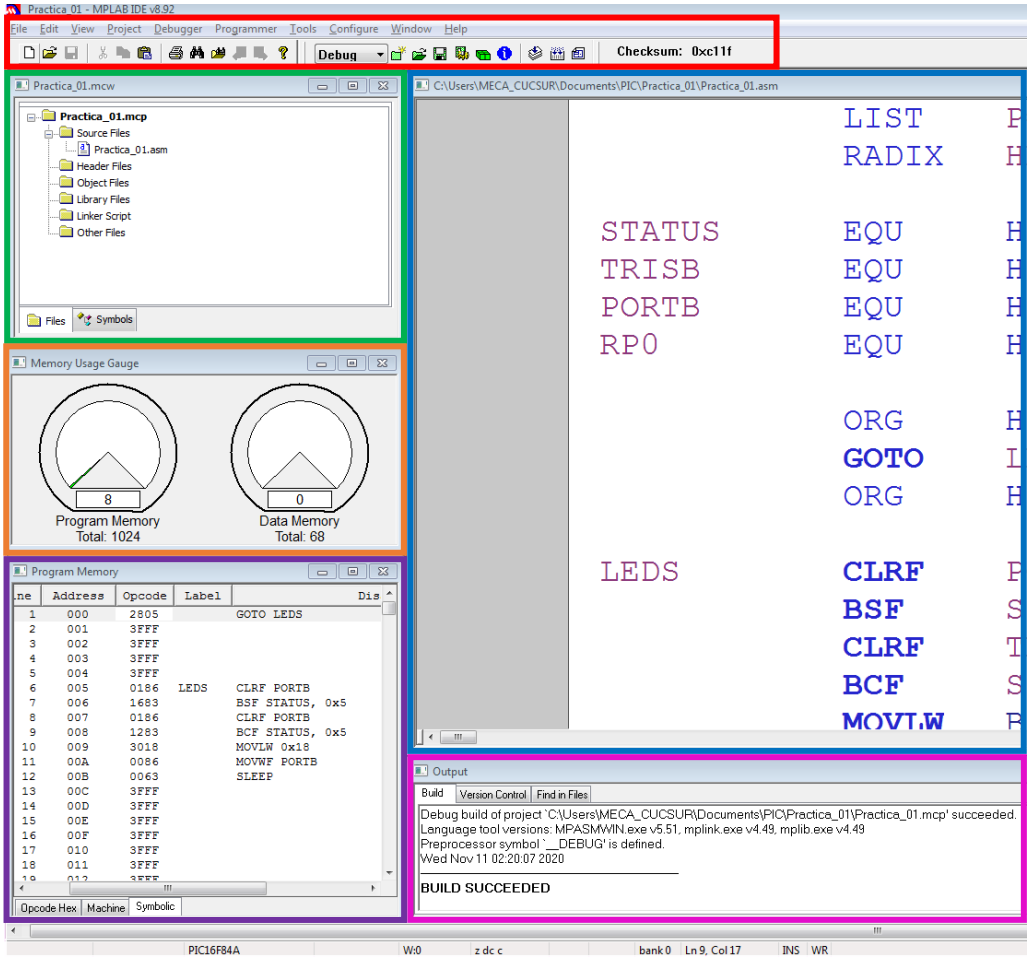

Figura 1 – Interfaz gráfica para MPLAB v8.92.

2) Creación de un proyecto en lenguaje ensamblador.

Seleccionar en la barra de herramientas "**Project** > **Project Wizard…**", en figura 2.

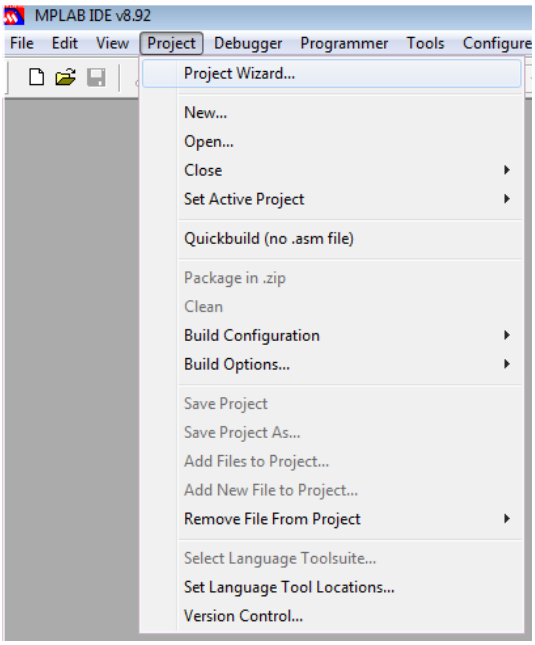

Figura 2 – Menú "**Project**" en la barra de herramientas.

Aparece la ventana "**Project Wizard**", presionar el botón "**Siguiente >**", figura 3.

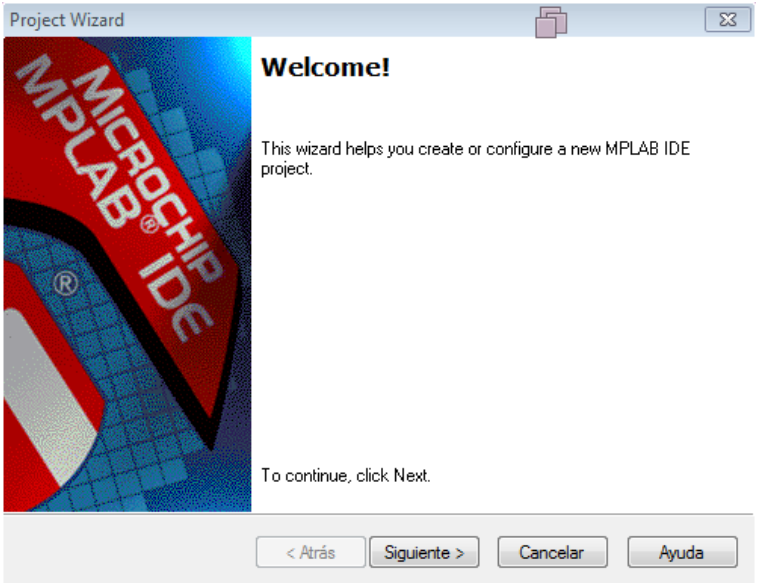

Figura 3 – Ventana de introducción al asistente de proyecto.

En la ventana de selección de dispositivo, elegir el microcontrolador “**PIC16F84A”**, presionar el botón “**siguiente >**”, figura 4.

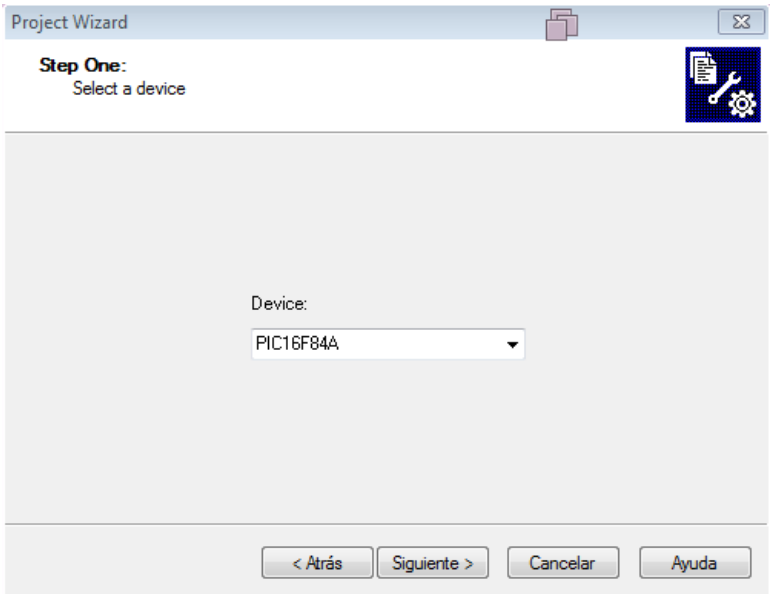

Figura 4 – Ventana de selección de dispositivo.

Seleccionar la herramienta “Microchip MPASM Toolsuite”, y presionar el botón “**Siguiente >**”, figura 5.

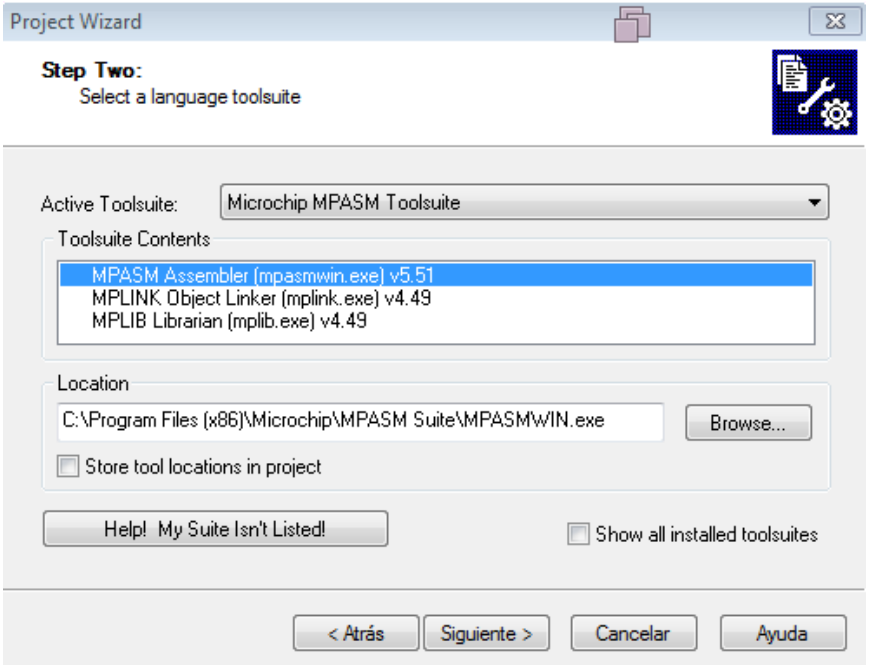

Figura 5 – Ventana para agregar archivos al proyecto.

Presionar el botón "**Browse…**", buscar una ubicación donde guardar el proyecto, crear una carpeta llamada "**Practica_01**", guardar el proyecto como "**Practica_01.mcp**", presionar el botón "**Guardar**", presionar el botón "**Siguiente >**", volver a presionar el botón "**Siguiente >**", figura 6 y 7.

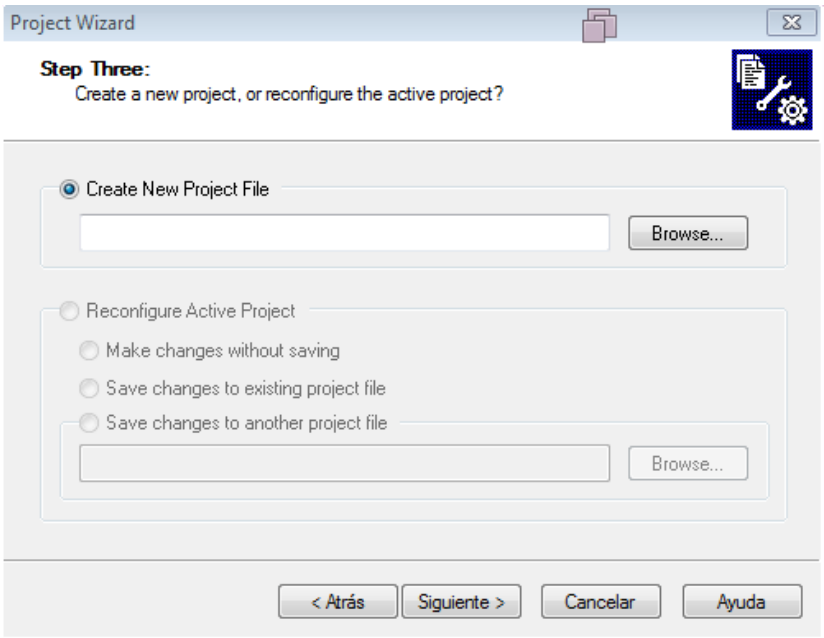

Figura 6 – Ventana de para crear el archivo de proyecto principal.

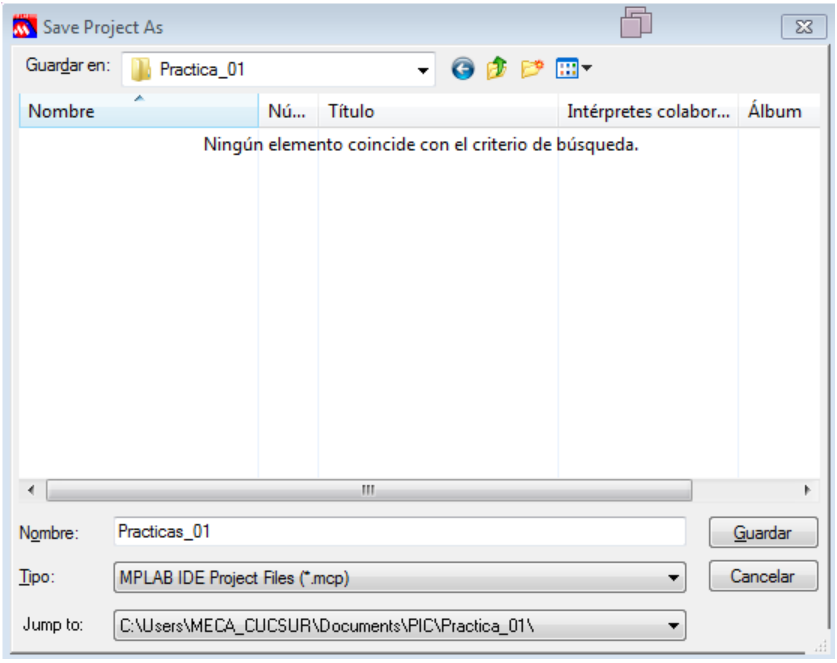

Figura 7 – Ventana de selección de herramientas **EDA**.

Aparece la ventana resumen del proyecto, presionar el botón "**Finalizar**", figura 8.

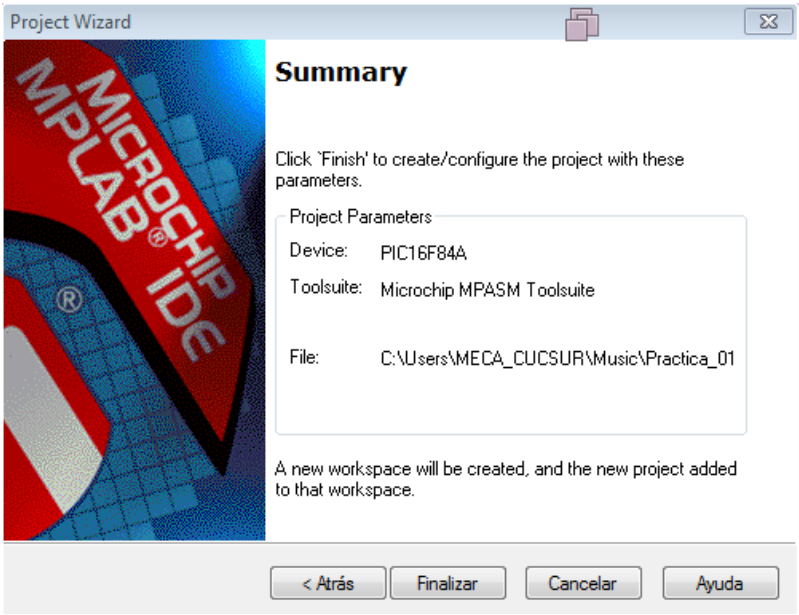

Figura 8 – Resumen del proyecto creado.

En la barra de herramientas seleccionar "**File** > **New**", figura 9.

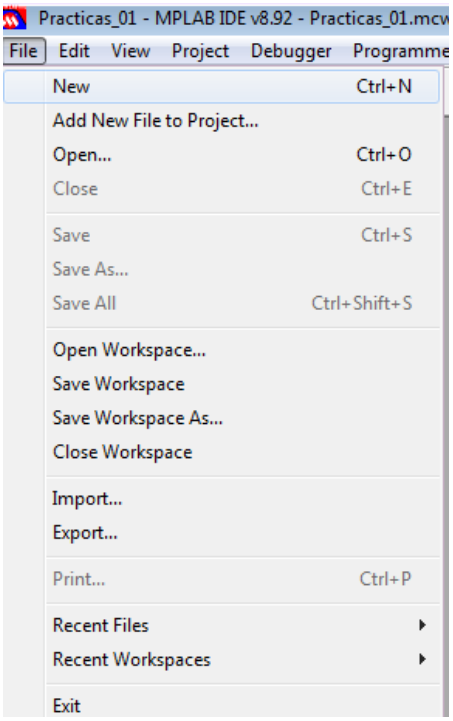

Figura 9 – Menú "New File".

Aparece la ventana del editor de código en figura 10, presionar en el menú "**File > Save As...**".

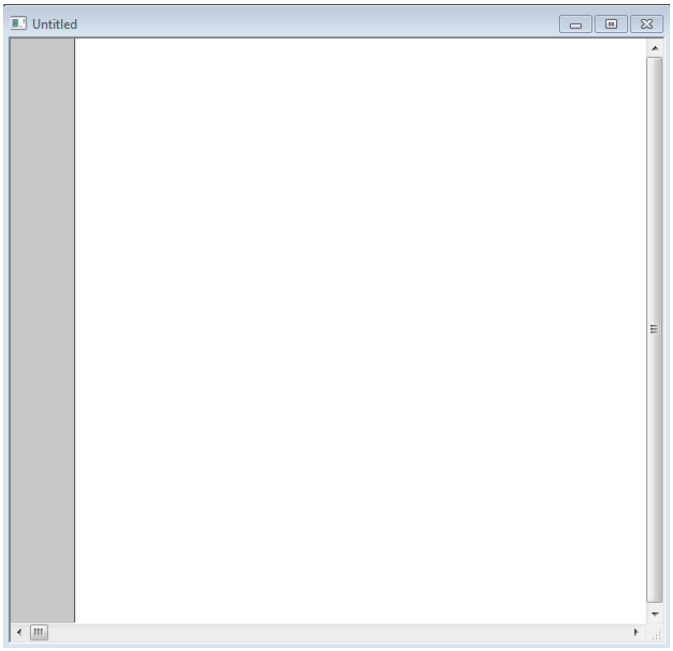

Figura 10 – Ventana de diseño **CAD**.

En la ventana "**Guardar como**", guardar el archivo como "**Practica_01.asm**", figura 11.

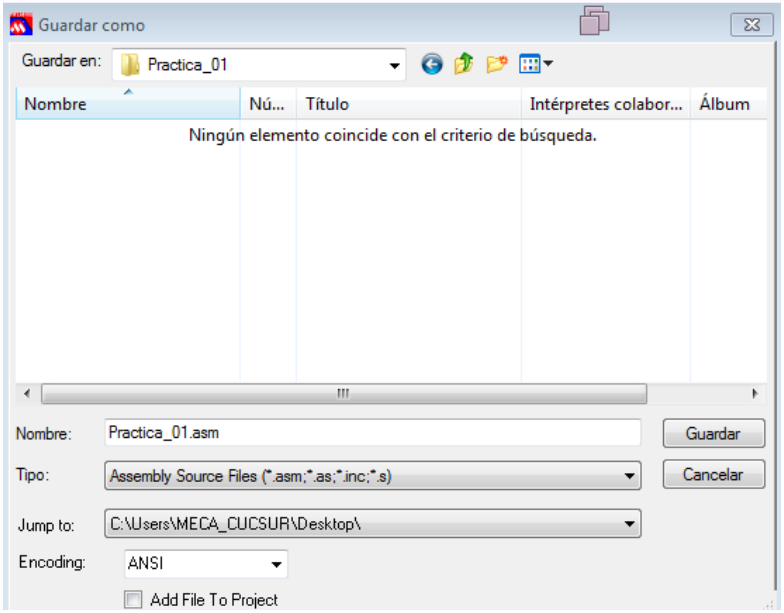

Figura 11 – Extensión "**.ASM"** para código ensamblador.

Escribir el código como se muestra a continuación:

```
;-------------------------------------------------------------------
; Práctica 1: Lenguaje Ensamblador                                  |
; Microcontrolador: PIC16F84A      Frecuencia: 4MHz  Voltaje:5V     |
; Herramienta: MPLAB v8.92 - Microchip MPASM Toolsuite              |
;-------------------------------------------------------------------
;-------------- Microcontrolador a usar --------------------------
                LIST    P=16F84A        ; Usaremos el PIC16F84
                RADIX   HEX             ; Valores hexadecimales
;-------------- Variables a usar ---------------------------------
STATUS          EQU     H'003'          ; Declaramos la variable STATUS
PORTB           EQU     H'006'          ; Declaramos la variable PORTB
TRISB           EQU     H'086'          ; Declaramos la variable TRISB
RP0             EQU     H'005'          ; Declaramos la variable RP0
;-------------- Vector de reset ----------------------------------
                ORG     H'000'          ; Vector de reset del procesador
                GOTO    LEDS            ; Nos manda a la etiqueta LEDS
                ORG     H'005'  ; Se escribe nuestro código en 0x05
;============== Programa Principal ==================================
;-------------- Configuración del microcontrolador ---------------
LEDS            BSF     STATUS,RP0      ; Nos movemos al Banco 1
                CLRF    TRISB   ; Configura el puerto B como salida
                BCF     STATUS,RP0      ; Nos movemos al Banco 0
;-------------- Aplicación encender LEDs -------------------------
                MOVLW   B'01010101'     ; Guardamos el valor en registro W
                MOVWF   PORTB           ; Movemos registro W al puerto B
;============== Fin del Programa ====================================
                END                     ; Le indica fin al ensamblador
```

Código 1 – Primer código en ensamblador.

Guardar los cambios al archivo, seleccionar en el árbol de proyecto la carpeta "**Source Files**", click derecho en la carpeta y seleccionar "**Add Files…",** figura 12.

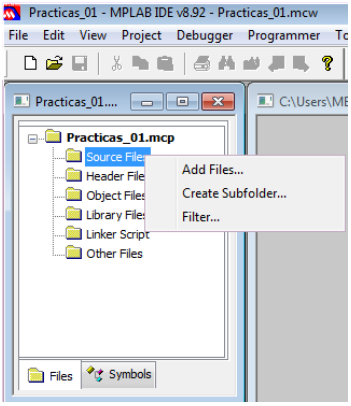

Figura 12 – Carpetas en árbol de proyecto.

Seleccionar el archivo "**Practica_01.asm**", presionar el botón "**Abrir**", figura 13.

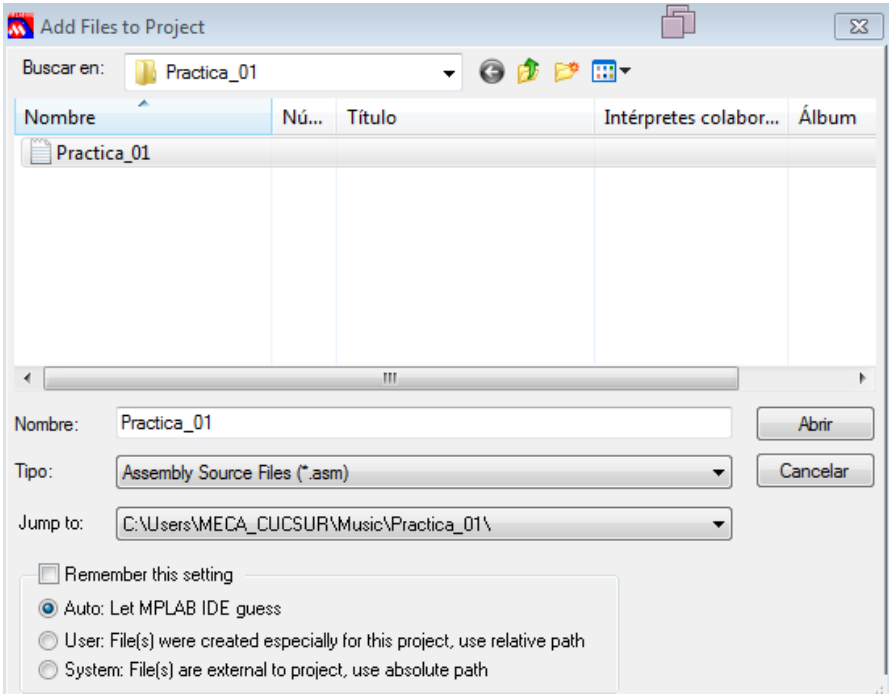

Figura 13 – Ventana "**Add Files to Project**".

Aparecerá el archivo "**Practica_01.asm**" en la carpeta "**Sources Files**", esto indica que se agregó el código al proyecto, figura 14.

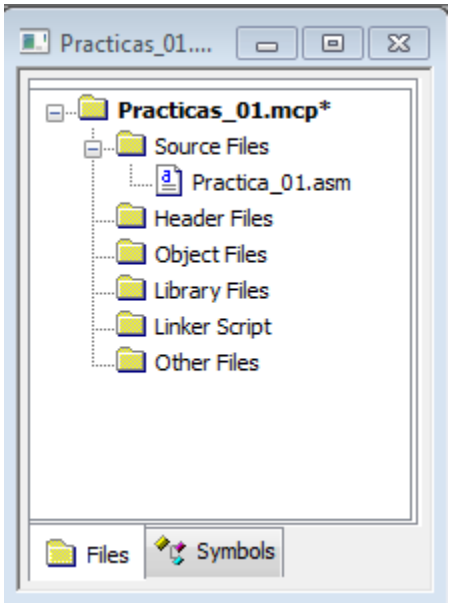

Figura 14 – El archivo "**Practica_01.asm**" ahora forma parte del proyecto.

En la barra de herramientas, presionar el botón "**Build All**" para construir el proyecto y obtener los archivos "**.HEX**", "**.O**", y "**.COF**", figura 15.

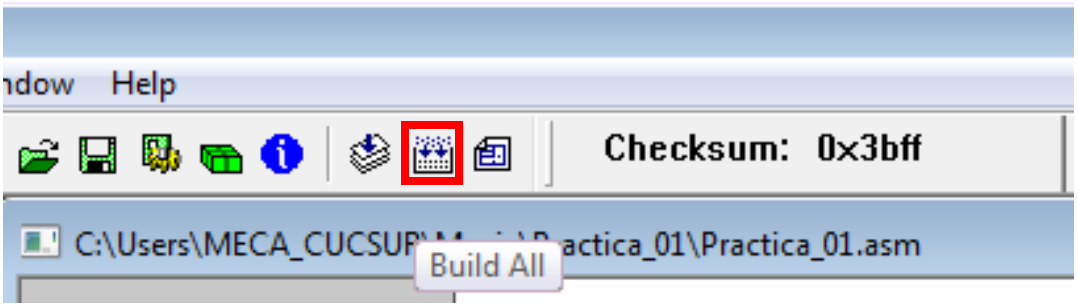

Figura 15 – Botón "**Build All**".

Presionar el botón "Absolute" para comenzar la construcción del proyecto, figura 16.

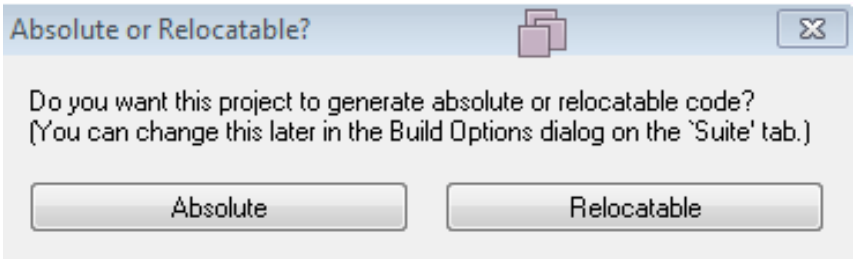

Figura 16 – Ventana de asistente para construir el proyecto.

La consola de salida mostrará el mensaje "**BUILD SUCCEEDED**", indicando que no ocurrió ningún error al construir el proyecto, figura 17.

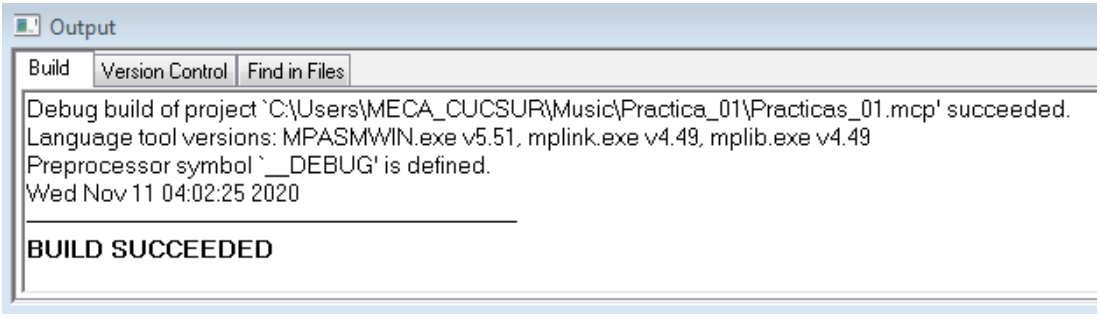

Figura 17 – Ventana de asistente para construir el proyecto.

3) Configurar el microcontrolador PIC16F84A.

En la barra de herramientas seleccionar "**Configure > Configuration Bits…**", figura 18. En esta sección configuran los fusibles internos del microcontrolador, estos modifican el comportamiento básico, solo pueden ser modificados por la tarjeta programadora a la hora de cargar el código al circuito integrado.

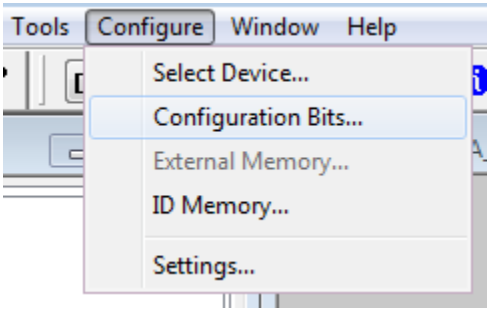

Figura 18 – Menú "**Configuration Bits…**".

Aparecerá la ventana "**Configuration Bits**", desmarcar la casilla "**Configuration Bits set in code"**, en la opción "**Oscillator Selection bits**" seleccionar "**XT oscillator**", en la opción "**Watchdog Timer**" seleccionar "**WDT disable**", por último, cerrar la ventana y presionar el botón "**Build All**", figura 19.

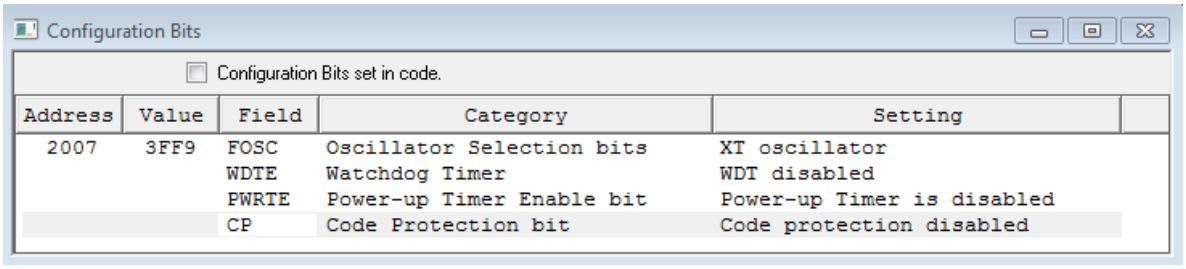

Figura 19 – Ventana "**Configuration Bits**".

- "**Oscillator Selection bits**" configura el tipo de generador de señales de reloj, interno, externo, baja velocidad, alta velocidad, etc.
- "**Watchdog Timer**" activa o desactiva el perro guardián.
- "**Power-up Timer Enable bit**" activa o desactiva el temporizador de encendido.
- "**Code Protection bit**" activa o desactiva la protección de código, esto evita que saquen una copia del código cargado en el microcontrolador.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 20.

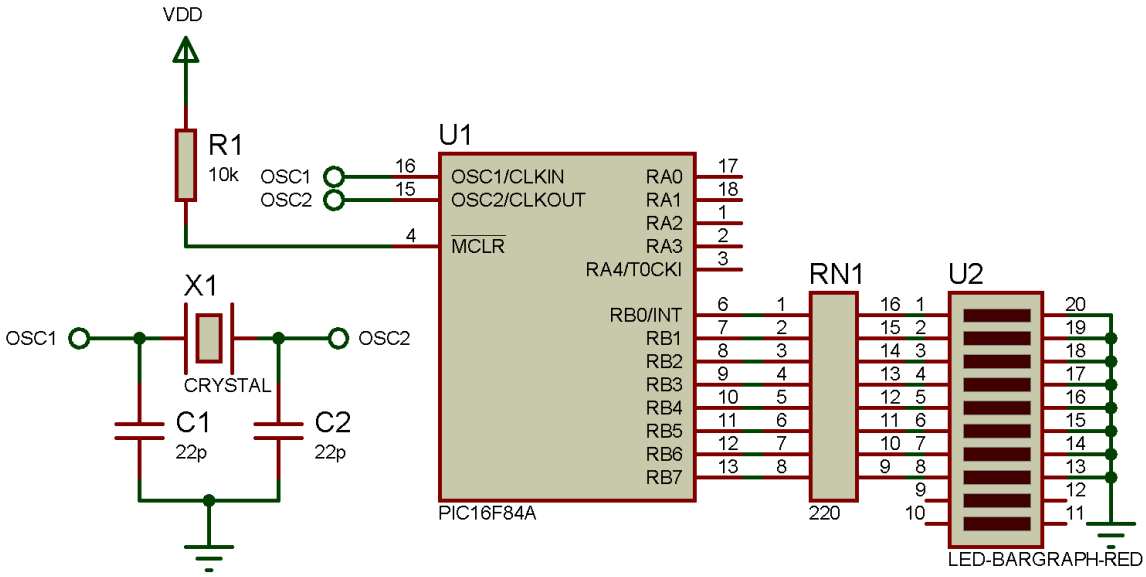

Figura 20 – Diagrama esquemático de práctica 1.

Hacer doble clic en el microcontrolador para ingresar a las propiedades del componente, aparece la ventana "**Edit Component**", figura 21.

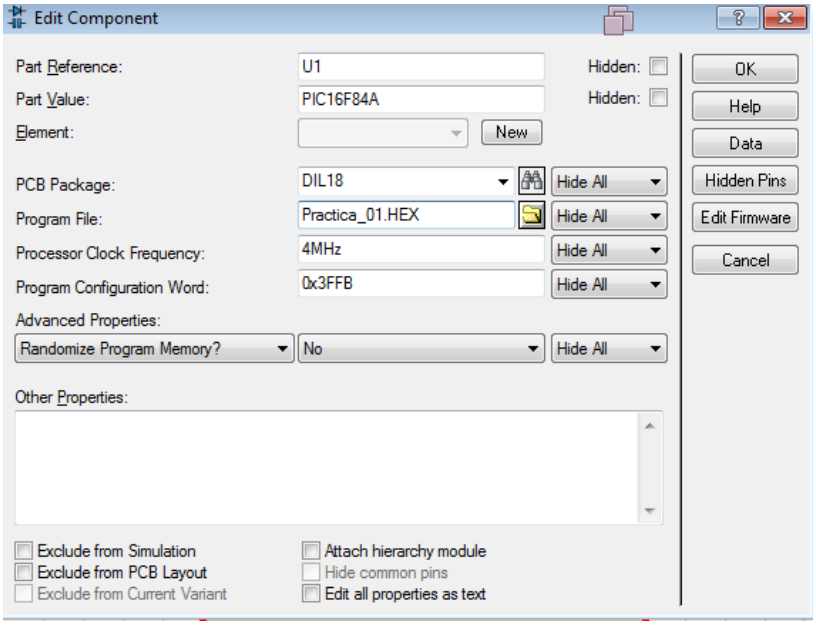

Figura 21 – Ventana "**Edit Component**".

Presionar el botón "**Program File**", seleccionar el archivo "**Practica_01.HEX**", configurar la frecuencia de reloj a 4MHz, presionar el botón "**OK**", presionar el botón RUN para ejecutar la simulación, figura 22.

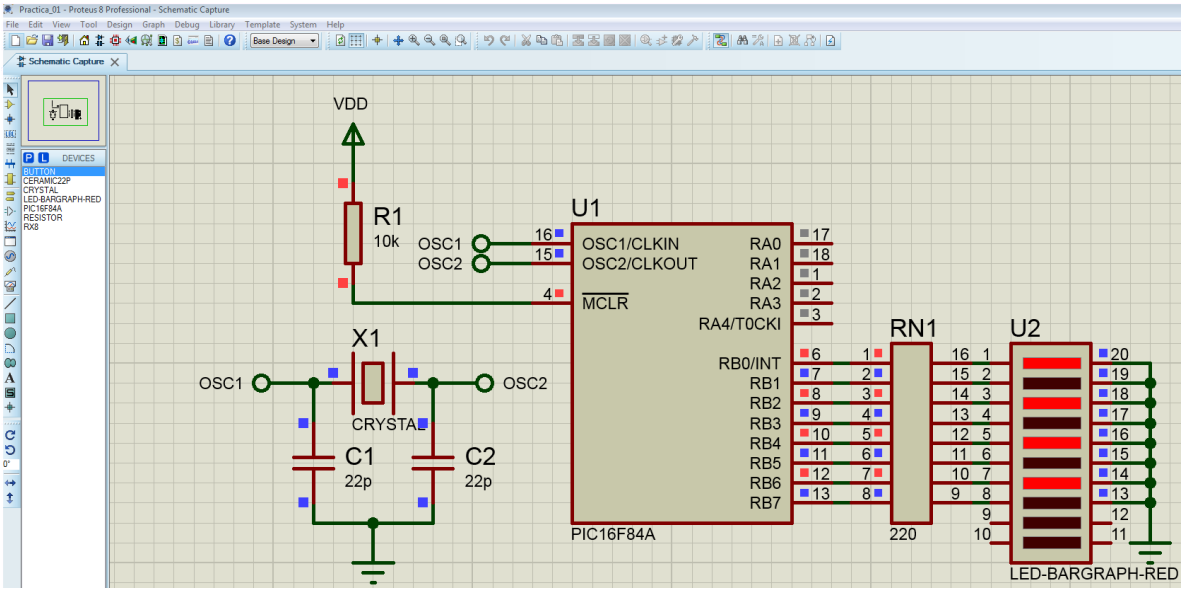

Figura 22 – Resultado de la simulación.

Detener la simulación con el botón STOP, hacer doble clic al microcontrolador, presionar el botón "**Program File**" y seleccionar el archivo "**Practica_01.COF**", presionar el botón "**OK**", figura 23.

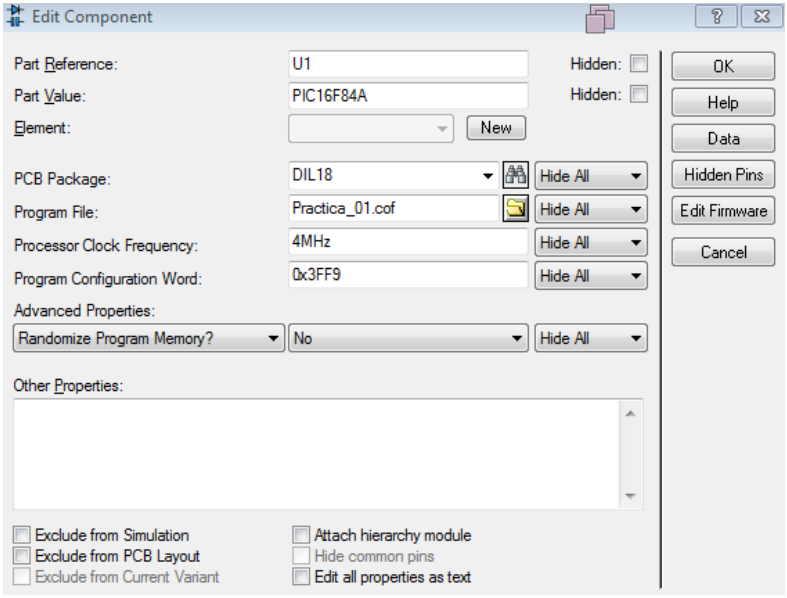

Figura 23 – Cargando archivo "**.COF**" para simulación paso a paso.

Presionar el botón SLOW para ejecutar la simulación paso a paso, aparecerá la ventana de código y la ventana de variables del CPU, figura 24.

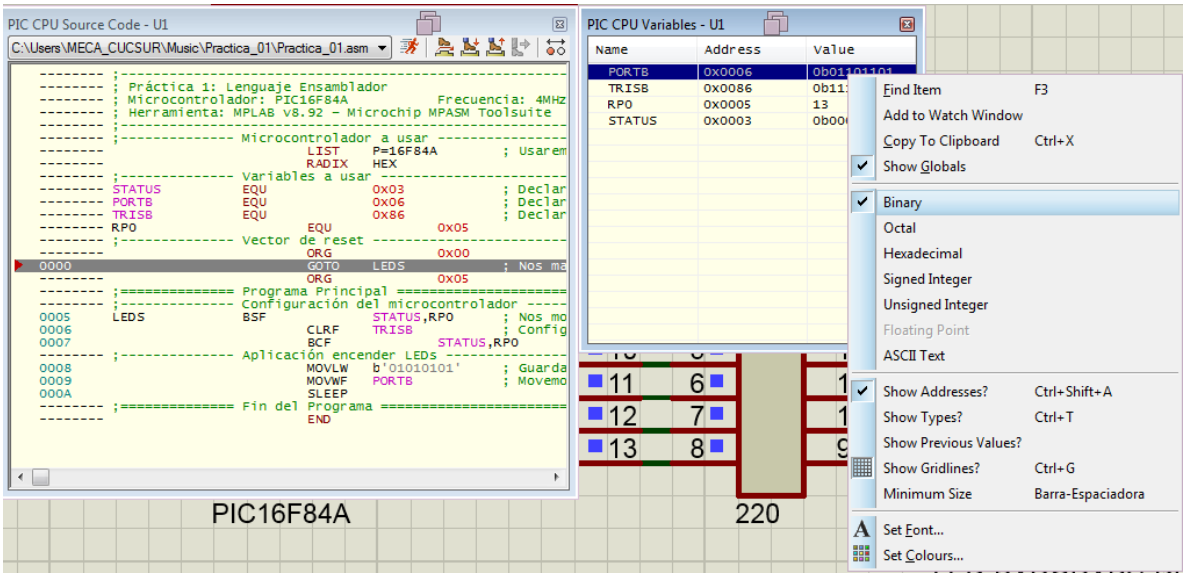

Figura 24 – Ventanas para Debug.

Presionar el botón "**Step into Source Line**" para ejecutar la primera instrucción "**GOTO**", el cursor se moverá a la dirección 0x0005 de la memoria **FLASH**, figura 25.

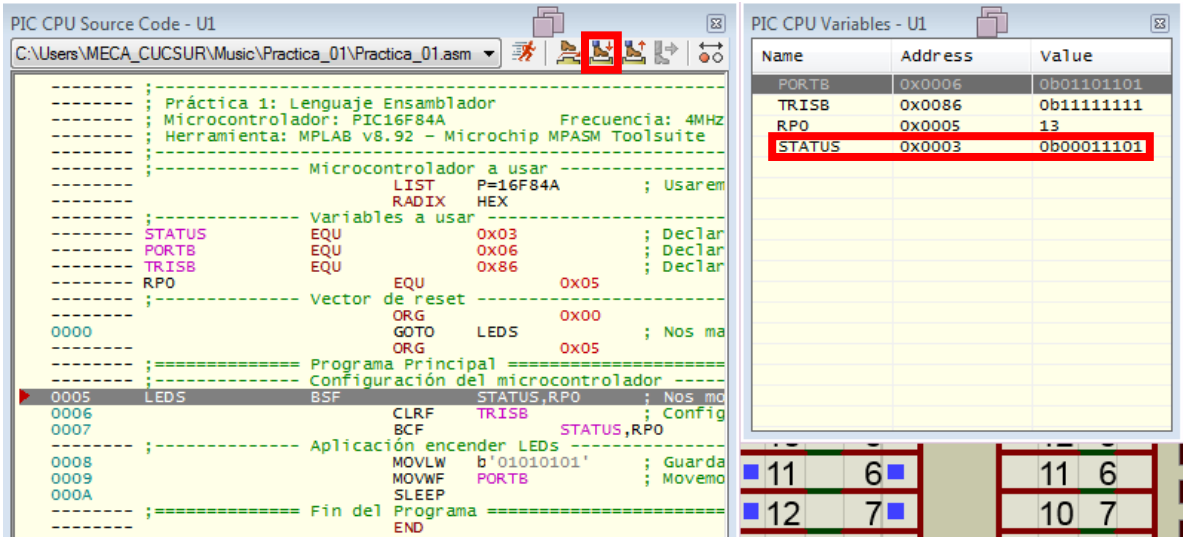

Figura 25 – botón "**Step into Source Line**".

Volver a presionar el botón "**Step into Source Line**" para ejecutar la segunda instrucción "**BSF**", esta instrucción modifica un bit con un "**1**" lógico en un registro, se debe observar que el registro "**STATUS**" cambió de color negro a rojo, esto indica que fue modificado, comparar la figura 25 con la figura 26. El cambio se efectuó en el bit 5 del registro "**STATUS**", ese bit 5 se llama "**RP0**", si se escribe en el un "**0**" lógico, se seleccionará el "**Banco 0**" de la memoria **RAM**, si se escribe en el un "**1**" lógico, se seleccionará el "**Banco 1**" de la memoria **RAM**. Los registros para configurar el microcontrolador se encuentran en el "**Banco 0**", los registros para utilizar el microcontrolador se encuentran en el "**Banco 1**".

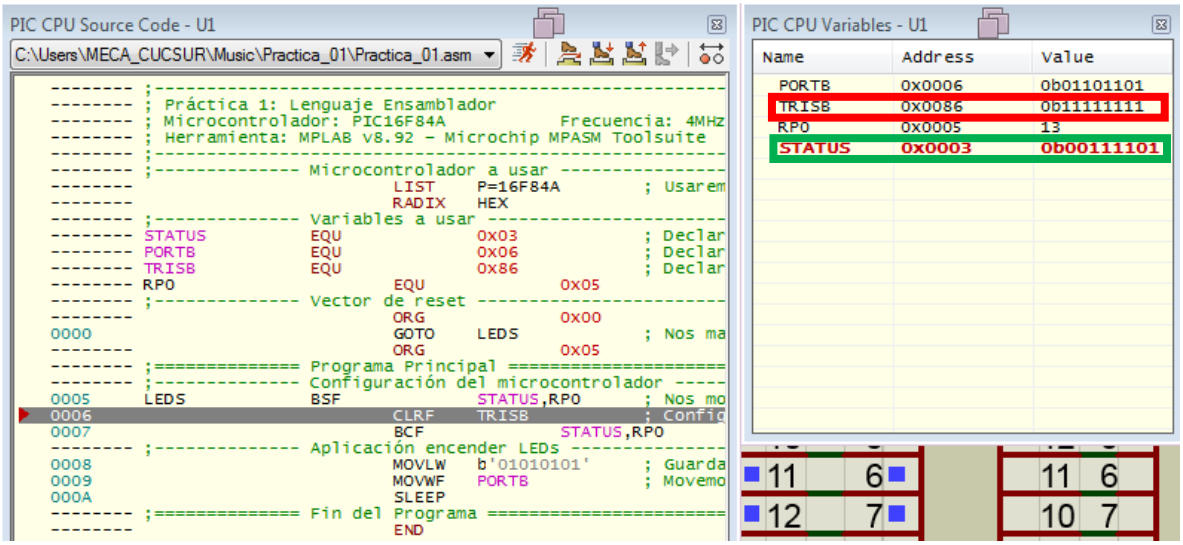

Figura 26 – Se modifica el bit 5 del registro "**STATUS**".

Presionar el botón "**Step into Source Line**" para ejecutar la tercera instrucción "**CRLF**", esta instrucción coloca "**0**" lógicos en los 8 bits del registro "**TRISB**", la función de este registro es controlar la dirección de los datos en los pines digitales, "**0**" lógico configura un pin como salida, "**1**" lógico configura un pin como entrada, figura 27.

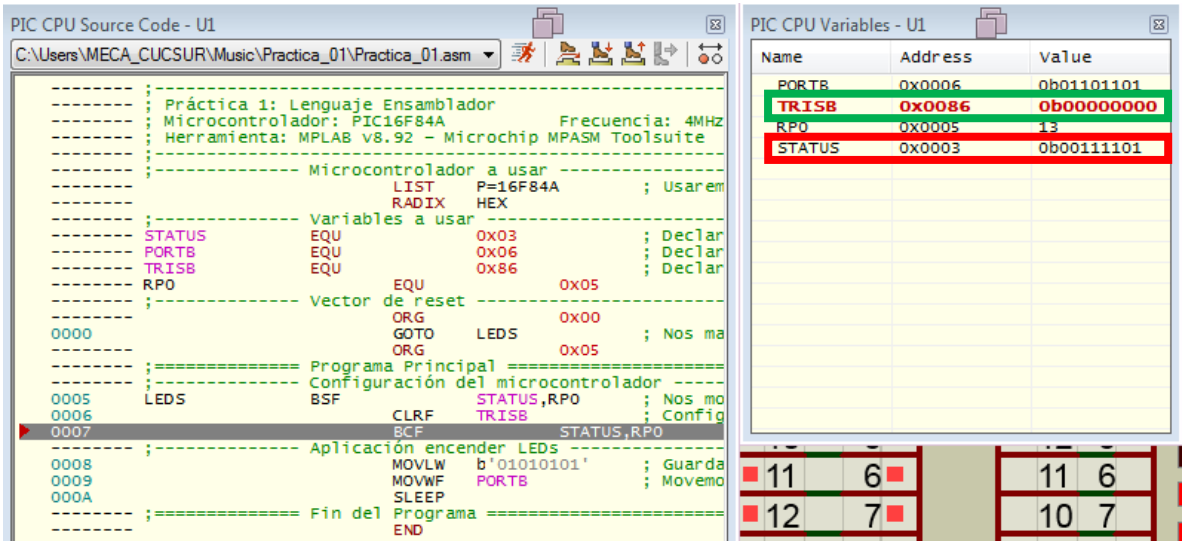

Figura 27 – Configurando el puerto B como salida.

Presionar el botón "**Step into Source Line**" para ejecutar la cuarta instrucción "**BCF**", esta instrucción modifica un bit con un "**0**" lógico en un registro, se debe observar que el bit 5 del registro "**STATUS**" volvió a cambiar, comparar la figura 25, figura 26 y figura 28. Presionar el botón "**Step into Source Line**" para ejecutar la quinta instrucción "**MOVLW**", esta instrucción crea el dato escrito a la derecha y lo almacena en el registro "**W**".

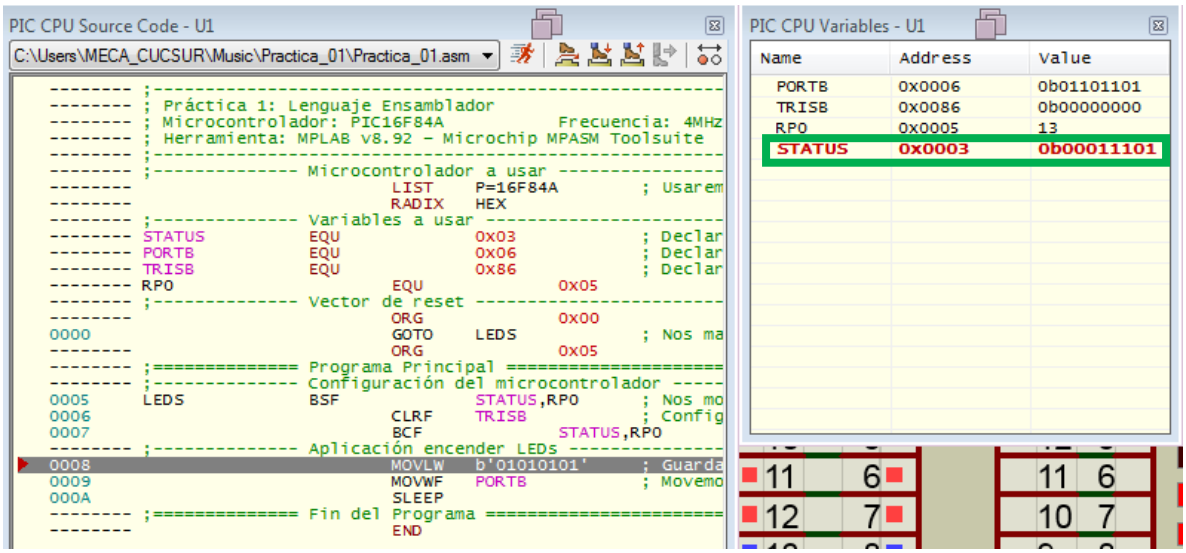

Figura 28 – Cambiar del Banco 1 al Banco 0.

Presionar el botón “**Step into Source Line**” para ejecutar la sexta instrucción “**MOVWF**” como se muestra en la figura 29 y figura 30.

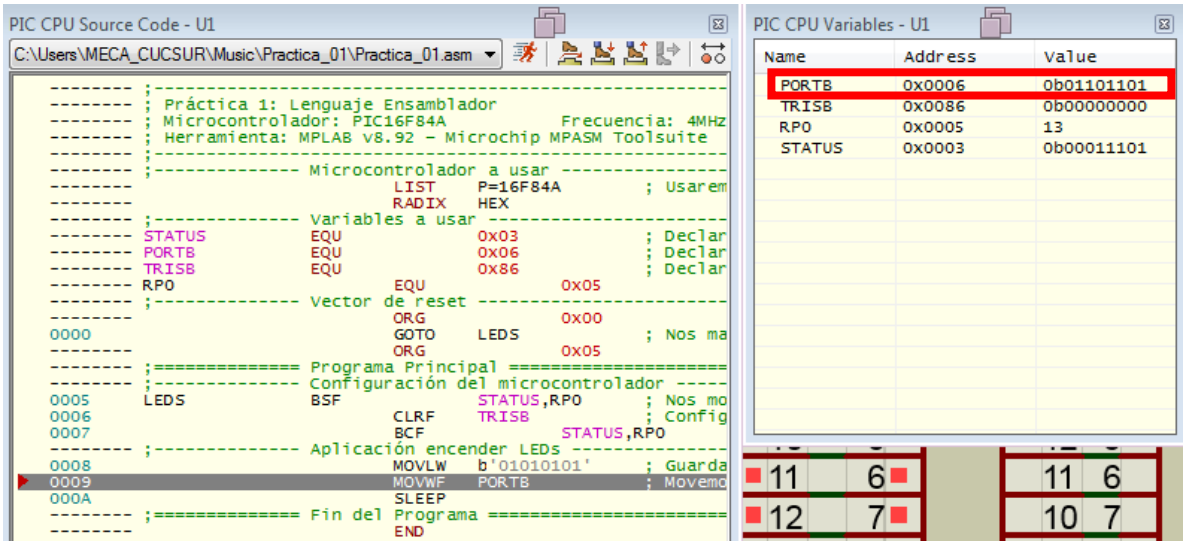

Figura 29 – Puerto B sin modificar.

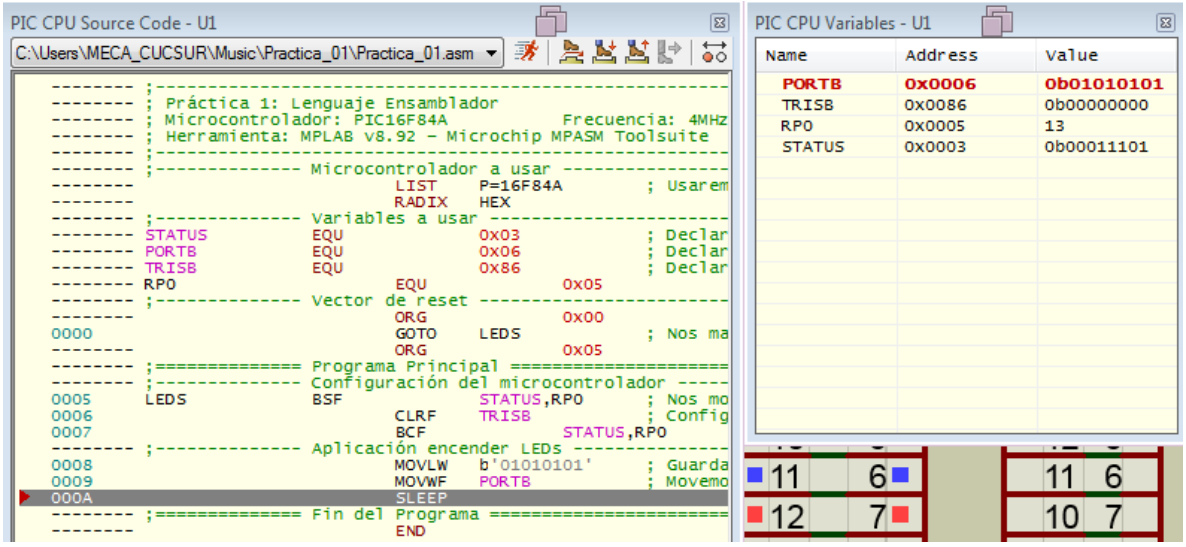

Figura 30 – La instrucción “**MOVWF**” modificó el puerto B.

La instrucción “**MOVWF**” modificó el registro “**PORTB**”, moviendo los datos del registro “**W**” hacia el registro “**PORTB**”, encendiendo los LEDs como se muestra en la figura 31.

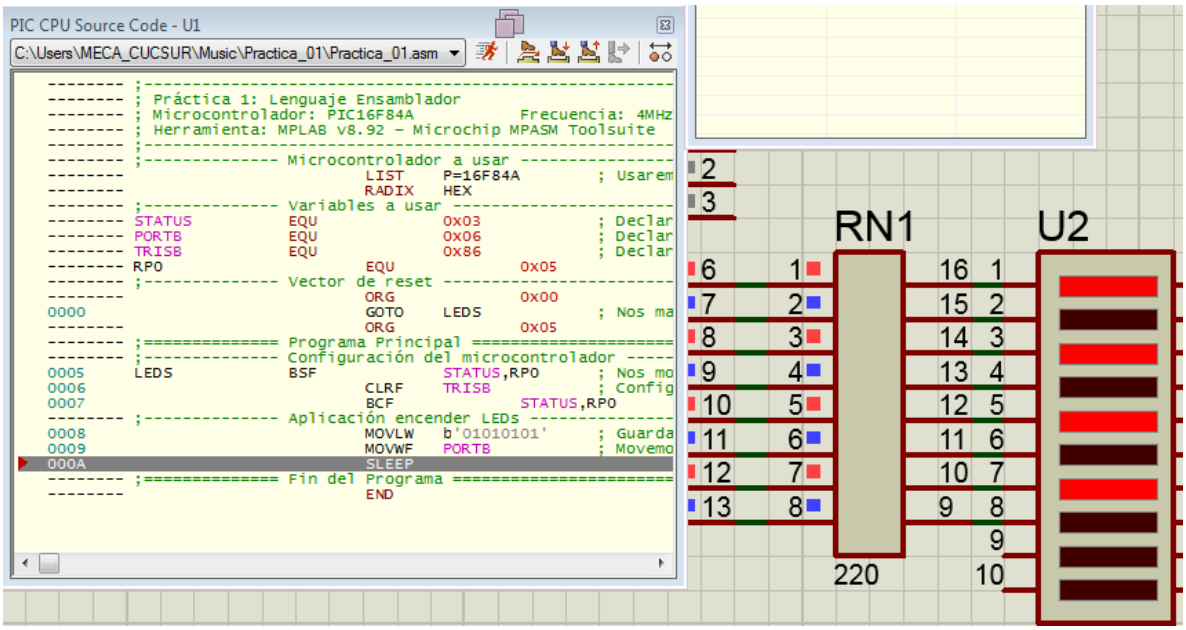

Figura 31 – LEDs encendidos.

5) Programación del PIC16F84A físico.

Tener a la mano un programador **PICSTART Plus** o un **PICKit 3**, conectar el programador con el cable serial RS-232 a la computadora, conectar la fuente de alimentación de 9V al programador, figura 32.

Figura 32 – Kit de programador **PICSTART Plus**.

Colocar el microcontrolador en el socket ZIF cuidando colocar el pin numero 1 con la muesca apuntando hacia arriba, subir la palanca del socket para asegurar el microcontrolador figura 33.

Figura 33 – Microcontrolador en socket ZIF del programador.

En la barra de herramientas, seleccionar "**Programmer > Select Programmer > 1 PICSTART Plus**" como se muestra en la figura 34.

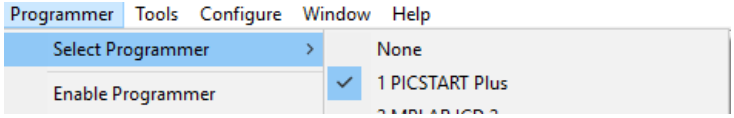

Figura 34 – Seleccionar el programador.

En la barra de herramientas, seleccionar "**Setting**", aparece la ventana "**Programmer**", seleccionar la pestaña "**Communications**" y seleccionar el puerto "COM" a utilizar, figura 35.

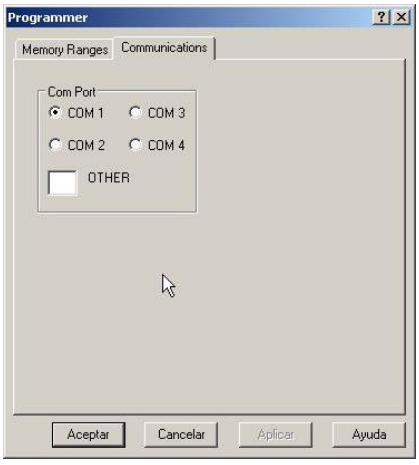

Figura 35 – Elegir el puerto serial para el programador.

Seleccionar "**Programmer > Enable Programmer** para habilitar el programador y establecer la comunicación, figura 36.

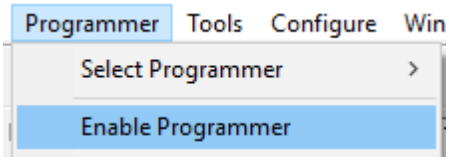

Figura 36 – Comunicar la computadora con el programador.

Una vez habilitado el programador, la barra de herramientas mostrará los botones ordenados de izquierda a derecha, "**Blank Check**", "**Read**", "**Program**", "**Verify**" y "**Erase Flash Device**", para cargar el código al microcontrolador presionar el botón "**Program**", después de unos segundos el texto "**Pass:**" incrementará su valor en 1, cada vez que se realice una programación exitosa, figura 37.

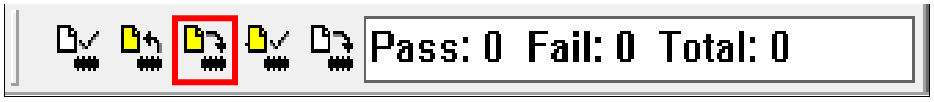

Figura 37 – Botón "**Program**".

Bajar la palanca del zocket ZIF para liberar el microcontrolador, figura 37.

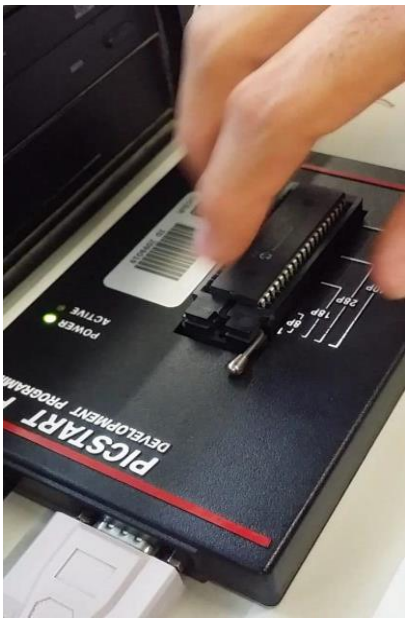

Figura 37 – Liberando el microcontrolador del socket ZIF.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 38.

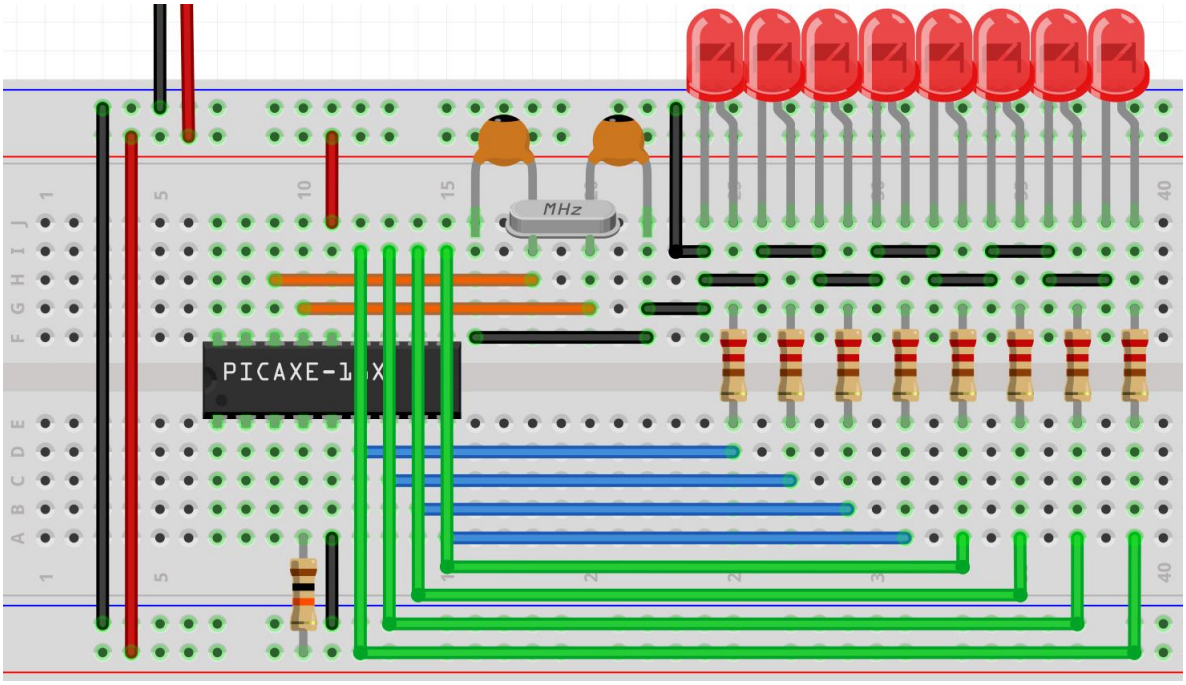

Figura 38 – LEDs conectados al PIC16F84A.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje ensamblador en los **microcontroladores**, aprender a usar el software MPLAB y simular su funcionamiento en Proteus Professional, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 2

Entradas y salidas digitales - PIC16F84A

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a programar las entradas y salidas digitales en **lenguaje ensamblador** usando el software **MPLAB v8.92**.
- Aprender a configurar los fusibles de un microcontrolador **PIC16F84A**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **PIC16F84A** [1].
- Software **MPLAB v8.92** [2].
- Software **Proteus Professional** [3].
- Programador **PICSTART Plus** [4] o **PICKit 3** [5].
- Fuente de alimentación fija de 5V.
- 2 resistencia 10k Ohms.
- 1 resistencia 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 4MHz.
- 1 LED (en caso de que la tarjeta de entrenamiento no los incluya).
- 1 botón pulsador.

Trabajo previo.

- Conocimientos vistos en clase: **Lenguaje ensamblador**.
- Conocimientos vistos en clase: **Hoja de datos del PIC16F84A**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje ensamblador.

Para realizar la práctica 2, seguir los pasos de la práctica 1 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje ensamblador.

Escribir el siguiente código y seguir los pasos de la práctica 1 para la construcción el proyecto y obtener los archivos "**.HEX**" y "**.COF**":

```
;-------------------------------------------------------------------------;
; Práctica 2: Entradas y salidas digitales                                ;
; Microcontrolador: PIC16F84A      Frecuencia: 4MHz  Voltaje:5V           ;
; Herramienta: MPLAB v8.92 - Microchip MPASM Toolsuite                    ;
;-------------------------------------------------------------------------;
;-------------- Archivos de inclusión (Librerías) ----------------------
#INCLUDE "p16f84.inc"  ; Librería del PIC16F84
;-------------- Bits de Configuración ----------------------------------
 __CONFIG _FOSC_XT & _WDTE_OFF & _PWRTE_OFF & _CP_OFF
;-------------- Vector de reset ----------------------------------------
               ORG     H'000'          ; Vector de reset del procesador
               GOTO    CONFIGURACION   ; Nos manda a la etiqueta
               ORG     H'005'          ; CONFIGURACION
;============== Programa Principal ====================================
;-------------- Configuración del microcontrolador ---------------------
CONFIGURACION  BSF     STATUS,RP0      ; Nos movemos al Banco 1
               BSF     TRISA,RA0; Configuramos pin RA0 como entrada
               BCF     TRISB,RB0       ; Configuramos pin RB0 como salida
               BCF     STATUS,RP0      ; Regresamos al Banco 0
               CLRF    PORTA           ; Limpia el registro PORTA
               CLRF    PORTB           ; Limpia el registro PORTB
;-------------- Aplicación entradas y salidas --------------------------
PROGRAMA       BTFSC   PORTA,RA0       ; Si RA0=0 brinca 2 instrucciones
               GOTO    LED_ON          ; Nos manda a etiqueta LED_ON
               GOTO    LED_OFF         ; Nos manda a etiqueta LED_OFF
;-------------- Subrutina LED_ON ---------------------------------------
LED_ON         BSF     PORTB,RB0       ; Enciende el LED en RB0
               GOTO    PROGRAMA        ; Nos manda a la etiqueta PROGRAMA
;-------------- Subrutina LED_OFF --------------------------------------
LED_OFF        BCF     PORTB,RB0       ; Apaga el LED en RB0
               GOTO    PROGRAMA        ; Nos manda a la etiqueta PROGRAMA
;============== Fin del Programa ======================================
               END                     ; Le indica fin al ensamblador
```

Código 2 – Entradas y salidas digitales PIC16F84A.

3) Configurar el microcontrolador PIC16F84A

Seguir los pasos de la práctica 1, para construir el proyecto y realizar la configuración de los fusibles.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 39, seguir los pasos de la práctica 1, para realizar la simulación funcional del circuito:

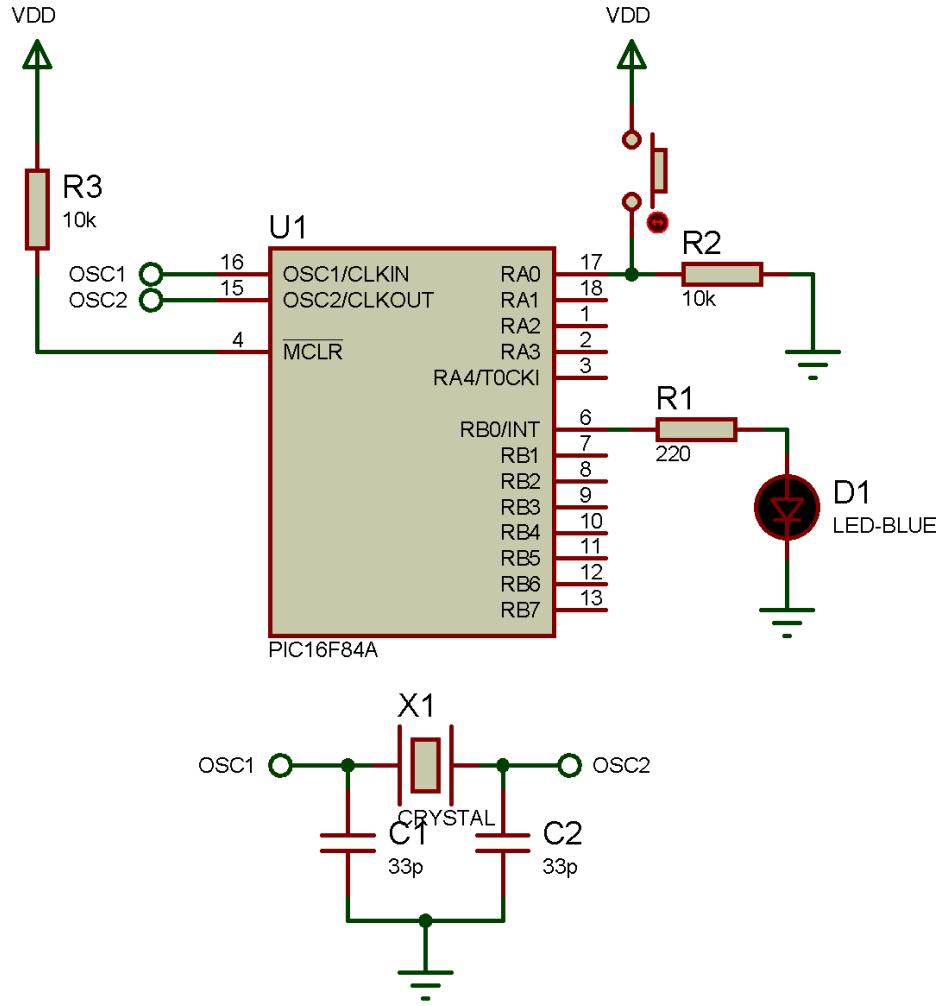

Figura 39 – Diagrama esquemático de práctica 2.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 1, para realizar la programación del **PIC16F84A**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 40.

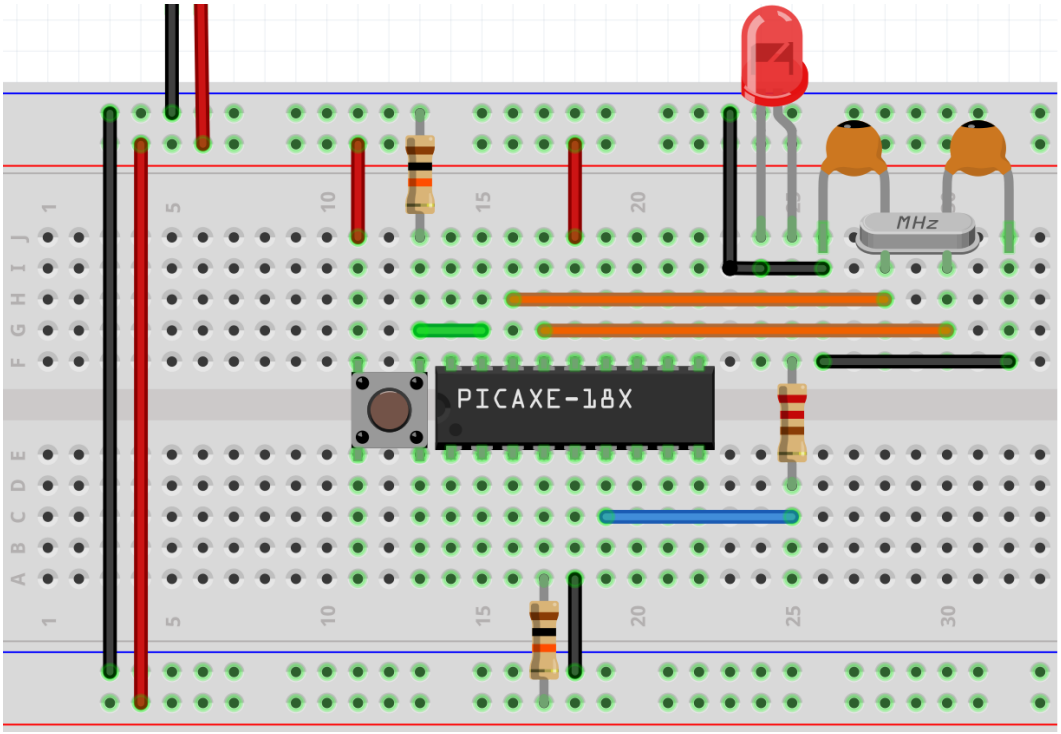

Figura 40 – botón pulsador y LED conectados al PIC16F84A.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje ensamblador en los **microcontroladores**, aprender a usar el software MPLAB y simular su funcionamiento en Proteus Professional, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 3

Retardo con instrucción NOP - PIC16F84A

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a programar un retardo con la instrucción **NOP** en **lenguaje ensamblador** usando el software **MPLAB v8.92**.
- Aprender a configurar los fusibles de un microcontrolador **PIC16F84A**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **PIC16F84A** [1].
- Software **MPLAB v8.92** [2].
- Software **Proteus Professional** [3].
- Programador **PICSTART Plus** [4] o **PICKit 3** [5].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 1 resistencia 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 4MHz.
- 1 LED (en caso de que la tarjeta de entrenamiento no los incluya).

Trabajo previo.

- Conocimientos vistos en clase: **Lenguaje ensamblador**.
- Conocimientos vistos en clase: **Hoja de datos del PIC16F84A**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje ensamblador.

Para realizar la práctica 3, seguir los pasos de la práctica 1 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje ensamblador.

Escribir el siguiente código y seguir los pasos de la práctica 1 para la construcción el proyecto y obtener los archivos "**.HEX**" y "**.COF**":

```
;-------------------------------------------------------------------------;
; Práctica 3: Retardo con instrucción NOP                                 ;
; Microcontrolador: PIC16F84A      Frecuencia: 4MHz  Voltaje:5V           ;
; Herramienta: MPLAB v8.92 - Microchip MPASM Toolsuite                    ;
;-------------------------------------------------------------------------;
;-------------- Archivos de inclusión (Librerías) ----------------------
#INCLUDE "p16f84.inc"                ; Librería del PIC16F84
;-------------- Vector de reset ----------------------------------------
               ORG     0x000         ; Vector de reset del procesador
               GOTO    CONFIGURACION ; Nos manda a la etiqueta
               ORG     0x005         ; CONFIGURACION
;============== Programa Principal ====================================
;-------------- Configuración del microcontrolador ---------------------
CONFIGURACION  BSF     STATUS,RP0    ; Nos movemos al Banco 1
               BCF     TRISB,RB0     ; Configuramos pin RB0 como salida
               BCF     STATUS,RP0    ; Regresamos al Banco 0
               CLRF    PORTB         ; Limpia el registro PORTB
;-------------- Aplicación entradas y salidas --------------------------
BLINKY         BSF     PORTB,RB0     ; RB0 = 1
               CALL    RETARDO_1S    ; Retardo de 1 segundo
               BCF     PORTB,RB0     ; RB0 = 0
               CALL    RETARDO_1S    ; Retardo de 1 segundo
               GOTO    BLINKY        ; Ciclo infinito
;-------------- Subrutina RETARDO_10uS ---------------------------------
RETARDO_10uS   NOP                   ; A 4MHz el CPU le toma ejecutar
               NOP                   ; una instrucción sencilla 1uS,
               NOP                   ; la instrucción CALL consume 2uS,
               NOP                   ; la instrucción NOP 1 uS,
               NOP                   ; la instrucción RETURN 2uS,
               NOP                   ; 6 NOPs = 6uS, entonces sumamos
               RETURN                ; 6uS + 2uS + 2uS = 10uS
;-------------- Subrutina RETARDO_100uS --------------------------------
RETARDO_100uS  CALL    RETARDO_10uS
               CALL    RETARDO_10uS
               CALL    RETARDO_10uS
               CALL    RETARDO_10uS
               CALL    RETARDO_10uS
               CALL    RETARDO_10uS
               CALL    RETARDO_10uS
               CALL    RETARDO_10uS
```

```
                CALL    RETARDO_10uS
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;-------------- Subrutina RETARDO_1mS ---------------------------------
RETARDO_1mS     CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;-------------- Subrutina RETARDO_10mS --------------------------------
RETARDO_10mS    CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO 100uS
```

```
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;-------------- Subrutina RETARDO_100mS -------------------------------
RETARDO_100mS   CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
```

```
                CALL    RETARDO_10uS
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;-------------- Subrutina RETARDO_1S ---------------------------------
RETARDO_1S      CALL    RETARDO_100mS
                CALL    RETARDO_100mS
                CALL    RETARDO_100mS
                CALL    RETARDO_100mS
                CALL    RETARDO_100mS
                CALL    RETARDO_100mS
                CALL    RETARDO_100mS
                CALL    RETARDO_100mS
                CALL    RETARDO_100mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_10mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_1mS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_100uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
```

```
                CALL    RETARDO_10uS
                CALL    RETARDO_10uS
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RETURN
;============== Fin del Programa ======================================
                END                     ; Le indica fin al ensamblador
```

Código 3 – Entradas y salidas digitales PIC16F84A.

3) Configurar el microcontrolador PIC16F84A

Seguir los pasos de la práctica 1, para construir el proyecto y realizar la configuración de los fusibles.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 41, seguir los pasos de la práctica 1, para realizar la simulación funcional del circuito:

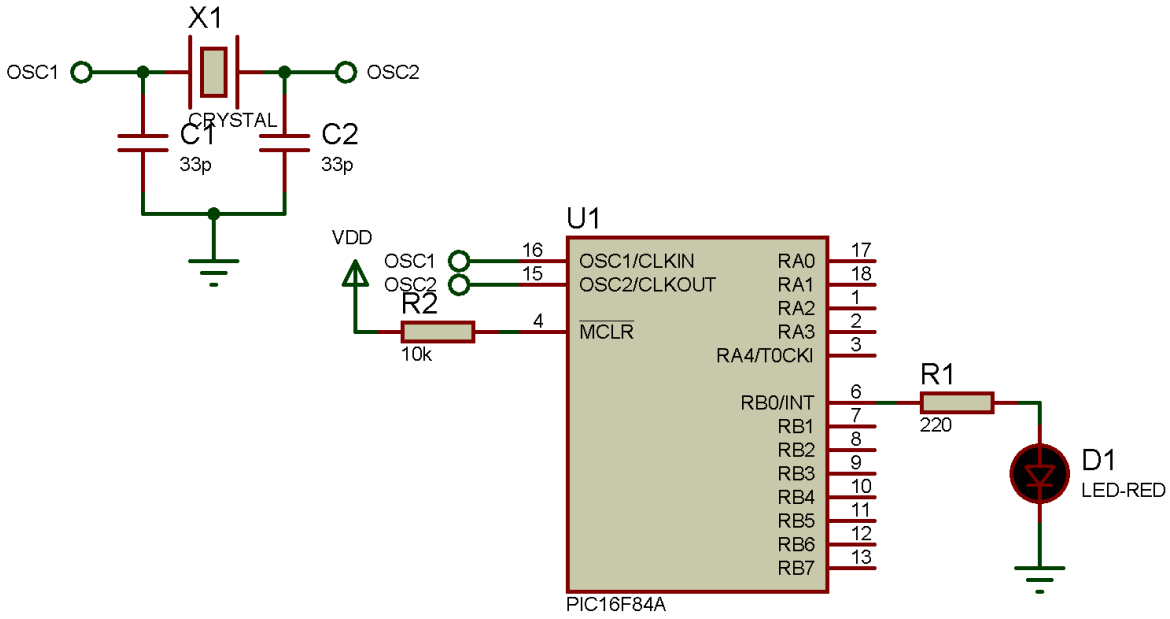

Figura 41 – Diagrama esquemático de práctica 3.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 1, para realizar la programación del **PIC16F84A**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 42.

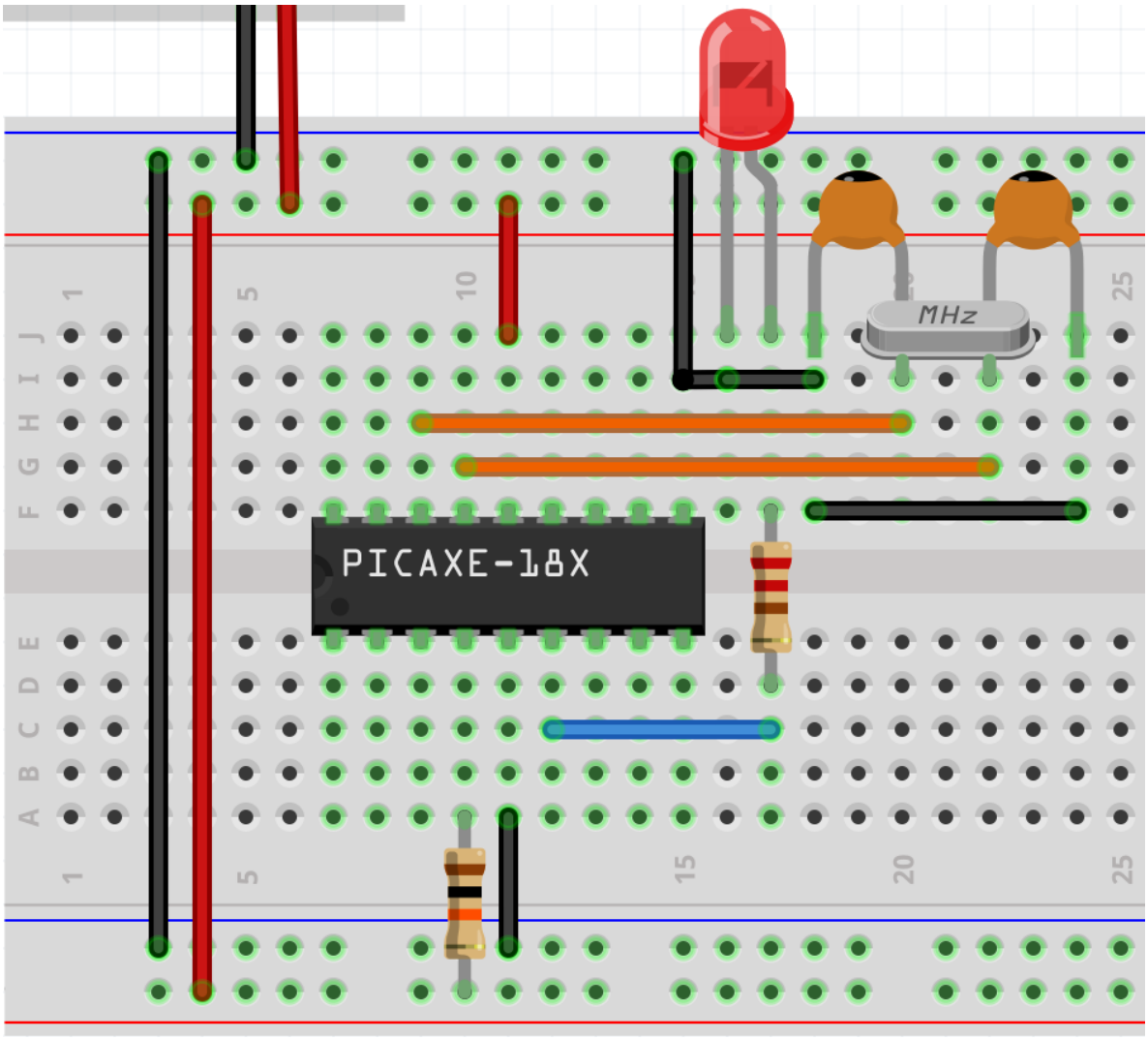

Figura 42 – LED conectado al PIC16F84A.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje ensamblador en los **microcontroladores**, aprender a usar el software MPLAB y simular su funcionamiento en Proteus Professional, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 4

Retardo con algoritmo - PIC16F84A

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a programar un retardo aplicando un algoritmo en **lenguaje ensamblador** usando el software **MPLAB v8.92**.
- Aprender a configurar los fusibles de un microcontrolador **PIC16F84A**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **PIC16F84A** [1].
- Software **MPLAB v8.92** [2].
- Software **Proteus Professional** [3].
- Programador **PICSTART Plus** [4] o **PICKit 3** [5].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 1 resistencia 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 4MHz.
- 1 LED (en caso de que la tarjeta de entrenamiento no los incluya).

Trabajo previo.

- Conocimientos vistos en clase: **Lenguaje ensamblador**.
- Conocimientos vistos en clase: **Hoja de datos del PIC16F84A**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje ensamblador.

Para realizar la práctica 4, seguir los pasos de la práctica 1 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje ensamblador.

Escribir el siguiente código y seguir los pasos de la práctica 1 para la construcción el proyecto y obtener los archivos "**.HEX**" y "**.COF**":

```
;-------------------------------------------------------------------;
; Práctica 4: Algoritmo de retardo                                  ;
; Microcontrolador: PIC16F84A      Frecuencia: 4MHz  Voltaje:5V     ;
; Herramienta: MPLAB v8.92 - Microchip MPASM Toolsuite              ;
;-------------------------------------------------------------------;
;-------------- Archivos de inclusión (Librerías) ------------------
#INCLUDE "p16f84.inc"                ; Librería del PIC16F84
;-------------- Variables a usar -----------------------------------
CONTADOR1       EQU     H'00C'       ; Variable guardada en GPR0
CONTADOR2       EQU     H'00D'       ; Variable guardada en GPR1
CONTADOR3       EQU     H'00E'       ; Variable guardada en GPR2
;-------------- Vector de reset ------------------------------------
                ORG     H'000'       ; Vector de reset del procesador
                GOTO    CONFIGURACION ; Nos manda a la etiqueta
                ORG     H'005'       ; CONFIGURACION
;============== Programa Principal ===================================
;-------------- Configuración del microcontrolador -----------------
CONFIGURACION   BSF     STATUS,RP0   ; Nos movemos al Banco 1
                BCF     TRISB,RB0    ; Configuramos pin RB0 como salida
                BCF     STATUS,RP0   ; Regresamos al Banco 0
                CLRF    PORTB        ; Limpia el registro PORTB
;-------------- Aplicación entradas y salidas ----------------------
BLINKY          BSF     PORTB,RB0    ; RB0 = 1
                CALL    RETARDO_1S   ; Retardo de 1 segundo
                BCF     PORTB,RB0    ; RB0 = 0
                CALL    RETARDO_1S   ; Retardo de 1 segundo
                GOTO    BLINKY       ; Ciclo infinito
;-------------- Subrutina RETARDO_10uS -----------------------------
RETARDO_1S      MOVLW   d'10'        ; W = 10 decimal
                MOVWF   CONTADOR3    ; CONTADOR3 = W
;-------------- Sección R_EXTERNA ----------------------------------
R_EXTERNA       MOVLW   d'100'       ; W = 100 decimal
                MOVWF   CONTADOR2    ; CONTADOR2 = W
;-------------- Sección R_INTERMEDIA -------------------------------
R_INTERMEDIA    MOVLW   d'250'       ; W = 250 decimal
                MOVWF   CONTADOR1    ; CONTADOR1 = W
;-------------- Sección R_INTERNA ----------------------------------
R_INTERNA       DECFSZ  CONTADOR1,1  ; CONTADOR1 = CONTADOR1-1
                GOTO    R_INTERNA    ;
                DECFSZ  CONTADOR2,1  ; CONTADOR2 = CONTADOR2-1
```

```
                GOTO    R_INTERMEDIA
                DECFSZ  CONTADOR3,1     ; CONTADOR3 = CONTADOR3-1
                GOTO    R_EXTERNA       ;
                RETURN                  ; Salimos del RETARDO_1seg
;============== Fin del Programa ======================================
                END                     ; Le indica fin al ensamblador
```

Código 4 – Algoritmo de retardo para PIC16F84A.

3) Configurar el microcontrolador PIC16F84A

Seguir los pasos de la práctica 1, para construir el proyecto y realizar la configuración de los fusibles.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 43, seguir los pasos de la práctica 1, para realizar la simulación funcional del circuito:

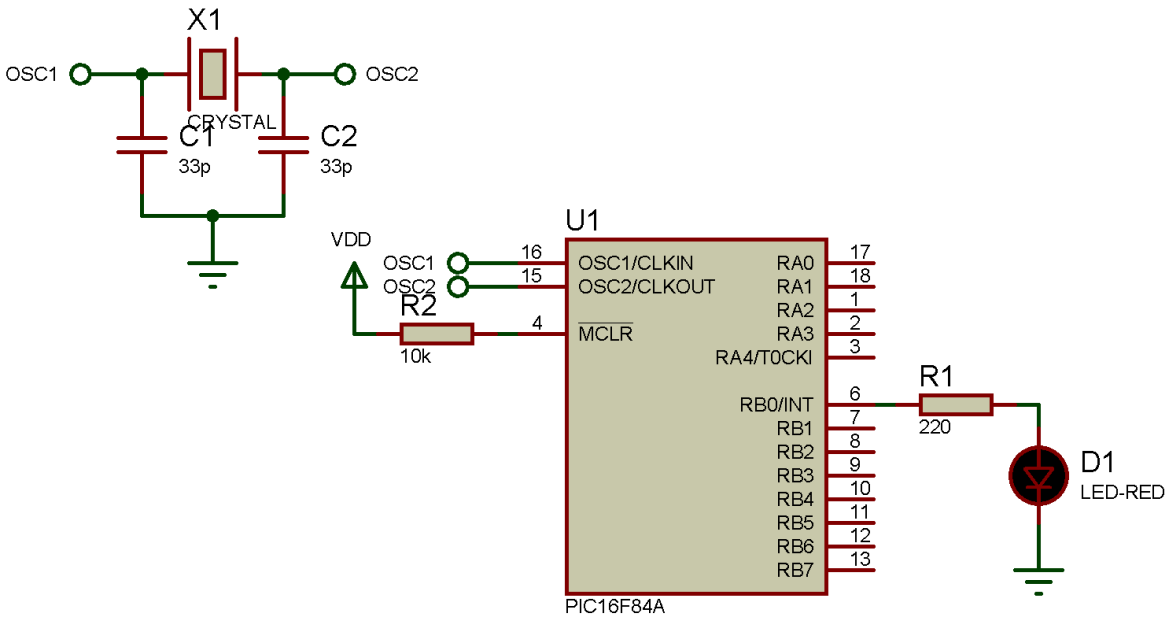

Figura 43 – Diagrama esquemático de práctica 4.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 1, para realizar la programación del **PIC16F84A**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 44.

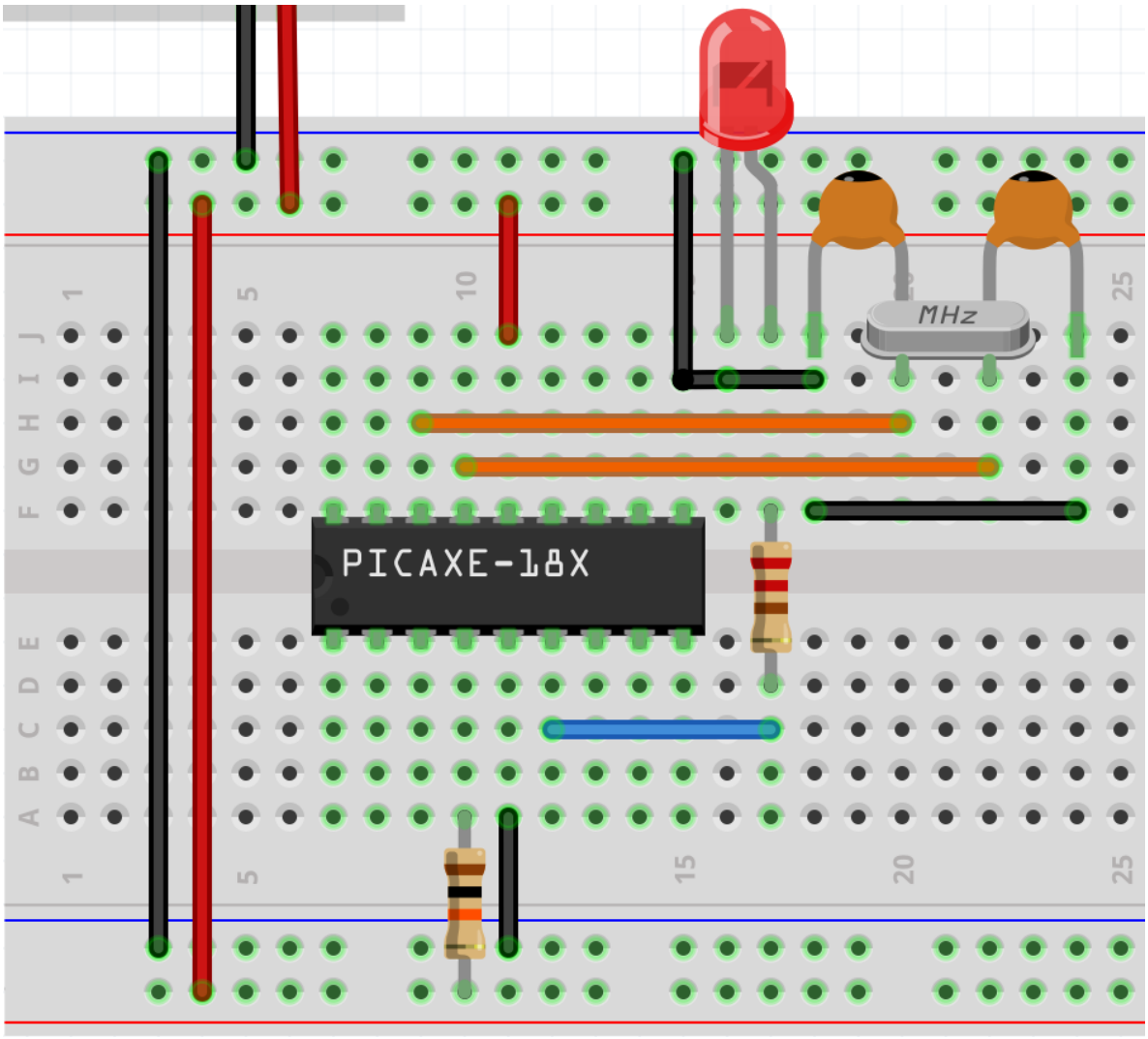

Figura 44 – LED conectado al PIC16F84A.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje ensamblador en los **microcontroladores**, aprender a usar el software MPLAB y simular su funcionamiento en Proteus Professional, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 5

Crear librerías - PIC16F84A

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a crear librerías propias en **lenguaje ensamblador** usando el software **MPLAB v8.92**.
- Aprender a configurar los fusibles de un microcontrolador **PIC16F84A**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **PIC16F84A** [1].
- Software **MPLAB v8.92** [2].
- Software **Proteus Professional** [3].
- Programador **PICSTART Plus** [4] o **PICKit 3** [5].
- Fuente de alimentación fija de 5V.
- 3 resistencia 10k Ohms.
- 8 resistencias 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 4MHz.
- 8 LEDs (en caso de que la tarjeta de entrenamiento no los incluya).
- 3 botones pulsadores (en caso de que la tarjeta de entrenamiento no los incluya).

Trabajo previo.

- Conocimientos vistos en clase: **Lenguaje ensamblador**.
- Conocimientos vistos en clase: **Hoja de datos del PIC16F84A**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje ensamblador.

Para realizar la práctica 5, seguir los pasos de la práctica 1 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje ensamblador.

Escribir los siguientes códigos y seguir los pasos de la práctica 1 para la construcción el proyecto y obtener los archivos "**.HEX**" y ".**COF**":

```
;-------------------------------------------------------------------;
; Práctica 5: Creación de librerías                                 ;
; Microcontrolador: PIC16F84A      Frecuencia: 4MHz  Voltaje:5V     ;
; Herramienta: MPLAB v8.92 - Microchip MPASM Toolsuite              ;
;-------------------------------------------------------------------;
;-------------- Archivos de inclusión (Librerías) ------------------
#INCLUDE "p16f84.inc"                ; Librería del PIC16F84
;-------------- Vector de reset ------------------------------------
               ORG     H'000'        ; Vector de reset del procesador.
               GOTO    CONFIGURACION ; Nos manda a la etiqueta
               ORG     H'005'        ; CONFIGURACION.
;============== Programa Principal ===================================
;-------------- Configuración del microcontrolador -----------------
CONFIGURACION  BSF     STATUS,RP0    ; Nos movemos al Banco 1.
               BSF     TRISA,RA0     ; Pin RA0 como entrada.
               BSF     TRISA,RA1     ; Pin RA1 como entrada.
               CLRF    TRISB         ; Puerto B como salida.
               BCF     STATUS,RP0    ; Regresamos al Banco 0.
               CLRF    PORTA         ; Limpia el registro PORTA.
               CLRF    PORTB         ; Limpia el registro PORTB.
;-------------- Loop Infinito --------------------------------------
PRINCIPAL      BTFSC   PORTA,RA0     ; Si RA0=0 brinca 2 instrucciones
               GOTO    BOTON_RA0     ; si no, se mueve a BOTON_RA0.
               BTFSC   PORTA,RA1     ; Si RA1=0 brinca 2 instrucciones
               CALL    BOTON_RA1     ; si no, se mueve a BOTON_RA1.
               GOTO    PRINCIPAL     ; Se mueve a PRINCIPAL.
;-------------- Aplicaciones ---------------------------------------
BOTON_RA0      CALL    CORRIMIENTO   ; Manda a llamar CORRIMIENTO.
               GOTO    PRINCIPAL     ; Se mueve a PRINCIPAL.
BOTON_RA1      CALL    VISTOSO       ; Manda a llamar VISTOSO.
               GOTO    PRINCIPAL     ; Se mueve a PRINCIPAL.
;============== Fin del Programa =====================================
#INCLUDE <Funciones.inc>             ; Librería subrutina RETARDO.
               END                   ; Le indica fin al ensamblador
```

Código 5 – Archivo principal "**main.ASM**".

```
;-------------- Variables a usar ---------------------------------------
TEMPORAL        EQU     H'00C'          ; Variable guardada en GPR0.
LADO_IZQ        EQU     H'00D'          ; Variable guardada en GPR1.
LADO_DER        EQU     H'00E'          ; Variable guardada en GPR2.
;-------------- Subrutina Corrimiento ----------------------------------
CORRIMIENTO     MOVLW   D'8'            ; W = 8.
                MOVWF   TEMPORAL        ; TEMPORAL = W.
;-------------- Ascendente ---------------------------------
CORRIMIENTO_A   RLF     PORTB,1         ; PORTB <- 1.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                DECFSZ  TEMPORAL        ; TEMPORAL = TEMPORAL-1.
                GOTO    CORRIMIENTO_A   ; Se mueve a CORRIMIENTO_A.
                MOVLW   D'8'            ; W = 8.
                MOVWF   TEMPORAL        ; TEMPORAL = W.
;-------------- Descendente --------------------------------
CORRIMIENTO_B   RRF     PORTB,1         ; PORTB -> 1.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                DECFSZ  TEMPORAL        ; TEMPORAL = TEMPORAL-1.
                GOTO    CORRIMIENTO_B   ; Se mueve a CORRIMIENTO_B.
                RETURN                  ; Sale de la subrutina.
;-------------- Subrutina Vistoso --------------------------------------
VISTOSO         CLRF    PORTB           ; Limpiamos el puerto B.
;-------------- Expandir -----------------------------------
EXPANDIR        MOVLW   B'00011000'     ; W = 00011000.
                MOVWF   PORTB           ; PORTB = W.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                MOVLW   B'00111100'     ; W = 00111100.
                MOVWF   PORTB           ; PORTB = W.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                MOVLW   B'01111110'     ; W = 01111110.
                MOVWF   PORTB           ; PORTB = W.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                MOVLW   B'11111111'     ; W = 11111111.
                MOVWF   PORTB           ; PORTB = W.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
;-------------- Contraer -----------------------------------
CONTRAER        MOVlW   B'01111110'     ; W = 01111110.
                MOVWF   PORTB           ; PORTB = W.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                MOVlW   B'00111100'     ; W = 00111100.
                MOVWF   PORTB           ; PORTB = W.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                MOVlW   B'00011000'     ; W = 00011000.
                MOVWF   PORTB           ; PORTB = W.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                CLRF    PORTB           ; Limpiamos el puerto B.
                CALL    RETARDO_1seg    ; Retardo de 1 segundo.
                RETURN                  ; Sale de la subrutina.
;============== Fin de Funciones ======================================
#INCLUDE <Retardo.inc>                  ; Librería subrutina RETARDO.
                END                     ; Le indica fin al ensamblador.
```

Código 6 – Librería "**Funciones.INC**".

```
;-------------- Variables a usar -------------------------------------
CONTADOR1       EQU     H'00F'          ; Variable guardada en GPR3
CONTADOR2       EQU     H'010'          ; Variable guardada en GPR4
CONTADOR3       EQU     H'011'          ; Variable guardada en GPR5
;-------------- Subrutina RETARDO_10uS -------------------------------
RETARDO_1seg    MOVLW   D'10'           ; W = 10 decimal
                MOVWF   CONTADOR3       ; CONTADOR3 = W
;-------------- Sección R_EXTERNA ------------------------------------
R_EXTERNA       MOVLW   D'100'          ; W = 100 decimal
                MOVWF   CONTADOR2       ; CONTADOR2 = W
;-------------- Sección R_INTERMEDIA ---------------------------------
R_INTERMEDIA    MOVLW   D'250'          ; W = 250 decimal
                MOVWF   CONTADOR1       ; CONTADOR1 = W
;-------------- Sección R_INTERNA ------------------------------------
R_INTERNA       DECFSZ  CONTADOR1,1     ; CONTADOR1 = CONTADOR1-1
                GOTO    R_INTERNA       ;
                DECFSZ  CONTADOR2,1     ; CONTADOR2 = CONTADOR2-1
                GOTO    R_INTERMEDIA
                DECFSZ  CONTADOR3,1     ; CONTADOR3 = CONTADOR3-1
                GOTO    R_EXTERNA       ;
                RETURN                  ; Salimos del RETARDO_1seg
;============== Fin del Programa ======================================
                END                     ; Le indica fin al ensamblador
```

Código 7 – Librería "**Retardo.INC**".

Agregar en la carpeta "**Header Files**" del árbol de proyecto la librería "**Funciones.INC**" y "**Retardo.INC**", presionar el botón "**Build All**", figura 45.

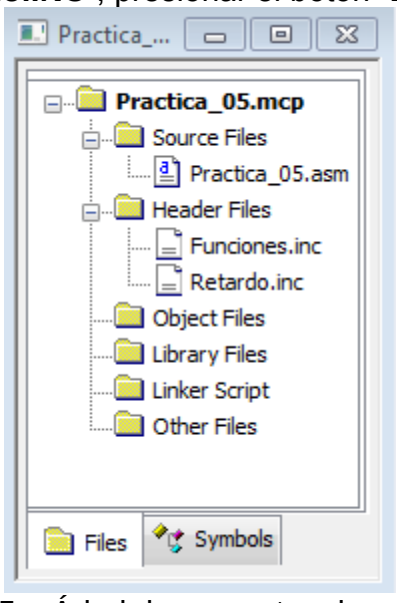

Figura 45 – Árbol de proyectos de práctica 5.

3) Configurar el microcontrolador PIC16F84A

Seguir los pasos de la práctica 1, para construir el proyecto y realizar la configuración de los fusibles.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 46, seguir los pasos de la práctica 1, para realizar la simulación funcional del circuito:

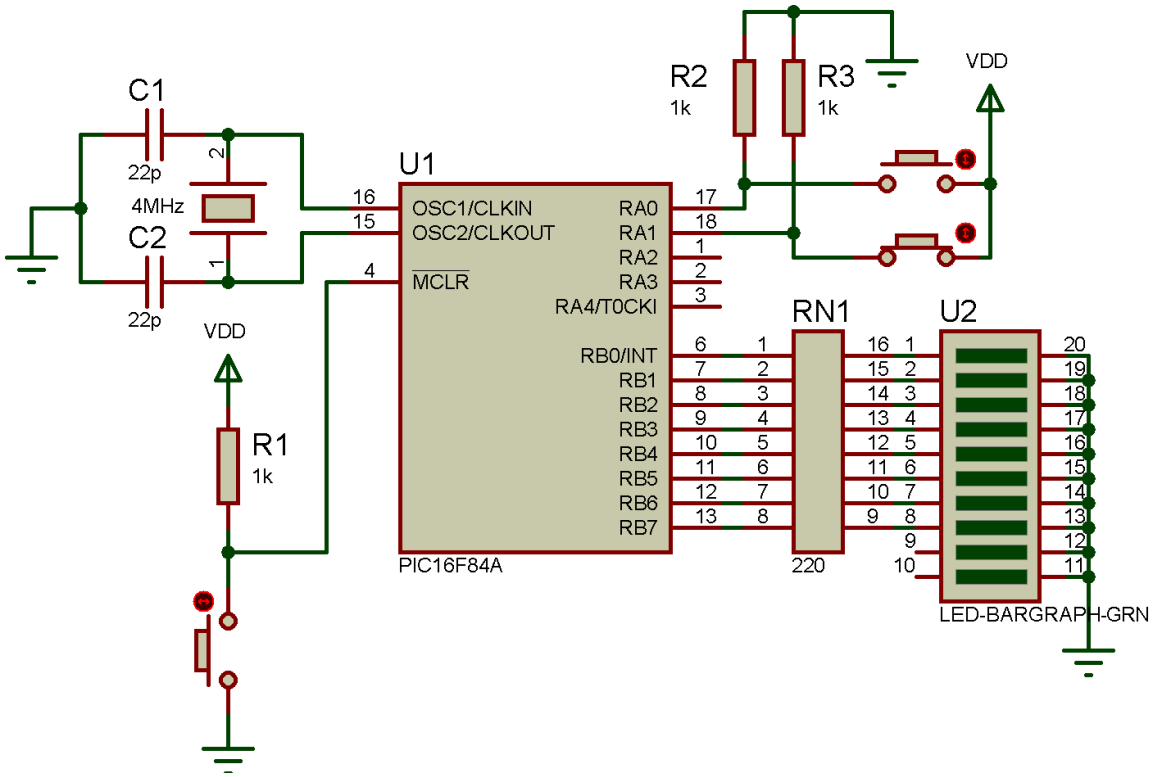

Figura 46 – Diagrama esquemático de práctica 5.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 1, para realizar la programación del **PIC16F84A**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 47.

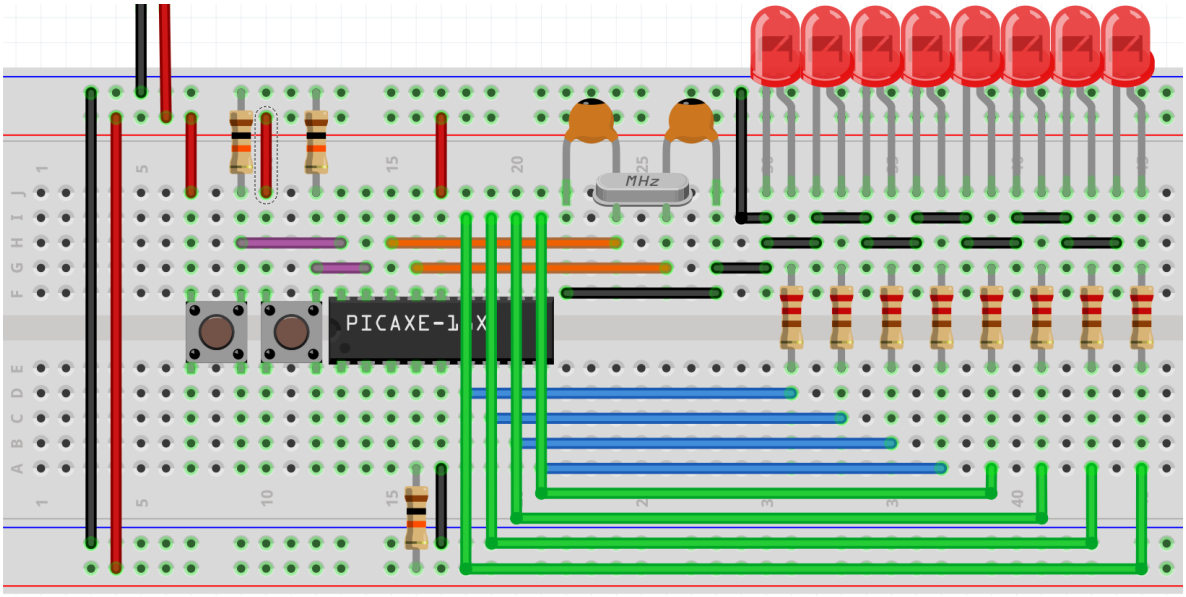

Figura 47 – Corrimiento de LEDs con PIC16F84A.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje ensamblador en los **microcontroladores**, aprender a usar el software MPLAB y simular su funcionamiento en Proteus Professional, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 6

Control ON-OFF - PIC16F84A

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a implementar un controlador ON-OFF en **lenguaje ensamblador** usando el software **MPLAB v8.92**.
- Aprender a configurar los fusibles de un microcontrolador **PIC16F84A**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **PIC16F84A** [1].
- Software **MPLAB v8.92** [2].
- Software **Proteus Professional** [3].
- Programador **PICSTART Plus** [4] o **PICKit 3** [5].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 1 resistencia 1k Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 4MHz.
- 1 bomba de agua sumergible (12Vdc o 120Vac).
- 1 transistor TIP41.
- 1 relevador 5V.

Trabajo previo.

- Conocimientos vistos en clase: **Lenguaje ensamblador**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje ensamblador.

Para realizar la práctica 6, seguir los pasos de la práctica 1 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje ensamblador.

Escribir el siguiente código y seguir los pasos de la práctica 1 para la construcción el proyecto y obtener los archivos "**.HEX**" y "**.COF**":

```
;-------------------------------------------------------------------------;
; Práctica 6: Controlador ON-OFF para nivel de agua                       ;
; Microcontrolador: PIC16F84A      Frecuencia: 4MHz  Voltaje:5V           ;
; Herramienta: MPLAB v8.92 - Microchip MPASM Toolsuite                    ;
;-------------------------------------------------------------------------;
;-------------- Archivos de inclusión (Librerías) ------------------------
#INCLUDE "p16f84.inc"                ; Librería del PIC16F84
;-------------- Vector de reset ------------------------------------------
               ORG     H'000'        ; Vector de reset del procesador.
               GOTO    CONFIGURACION ; Nos manda a la etiqueta
               ORG     H'005'        ; CONFIGURACION.
;============== Programa Principal ======================================
;-------------- Configuración del microcontrolador -----------------------
CONFIGURACION  BSF     STATUS,RP0    ; Nos movemos al Banco 1.
               BSF     TRISA,RA0     ; Pin RA0 como entrada.
               BSF     TRISA,RA2     ; Pin RA2 como entrada.
               BCF     TRISB,RB0     ; Puerto B como salida.
               BCF     STATUS,RP0    ; Regresamos al Banco 0.
               CLRF    PORTA         ; Limpia el registro PORTA.
               CLRF    PORTB         ; Limpia el registro PORTB.
;-------------- Aplicación control de NIVEL ----------------------------
COMP1          BTFSS   PORTA,RA0     ; Si RA0=1 brinca a comparar RA2
               GOTO    COMP2         ; si no, brinca a COMP2.
               BTFSS   PORTA,RA2     ; Si RA2=1 brinca a APAGAR
               GOTO    COMP2         ; si no, brinca a COMP2.
               GOTO    APAGAR        ; Brinca a la subrutina APAGAR.
COMP2          BTFSC   PORTA,RA0     ; Si RA0=0 brinca a comparar RA2
               GOTO    COMP1         ; si no, brinca a COMP1.
               BTFSC   PORTA,RA2     ; Si RA2=0 brinca a ENCENDER
               GOTO    COMP1         ; si no, brinca a COMP1.
               GOTO    ENCENDER      ; Brinca a la subrutina ENCENDER.
;-------------- Subrutina prender bomba de agua ------------------------
ENCENDER       BSF     PORTB,RB0     ; Enciende la bomba.
               GOTO    COMP2         ; Brinca a COMP2.
;-------------- Subrutina apagar bomba de agua -------------------------
APAGAR         BCF     PORTB,RB0     ; Apaga la bomba.
               GOTO    COMP1         ; Brinca a COMP1.
;============== Fin del Programa ========================================
               END                   ; Le indica fin al ensamblador
```

Código 8 – Archivo principal "**main.ASM**".

3) Configurar el microcontrolador PIC16F84A

Seguir los pasos de la práctica 1, para construir el proyecto y realizar la configuración de los fusibles.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 48, seguir los pasos de la práctica 1, para realizar la simulación funcional del circuito:

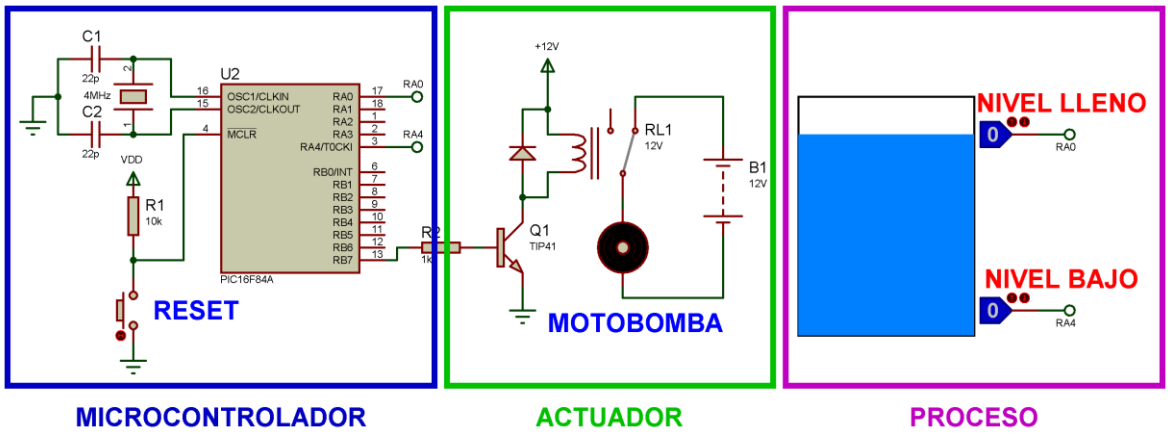

Figura 48 – Diagrama esquemático de práctica 6.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 1, para realizar la programación del **PIC16F84A**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 49.

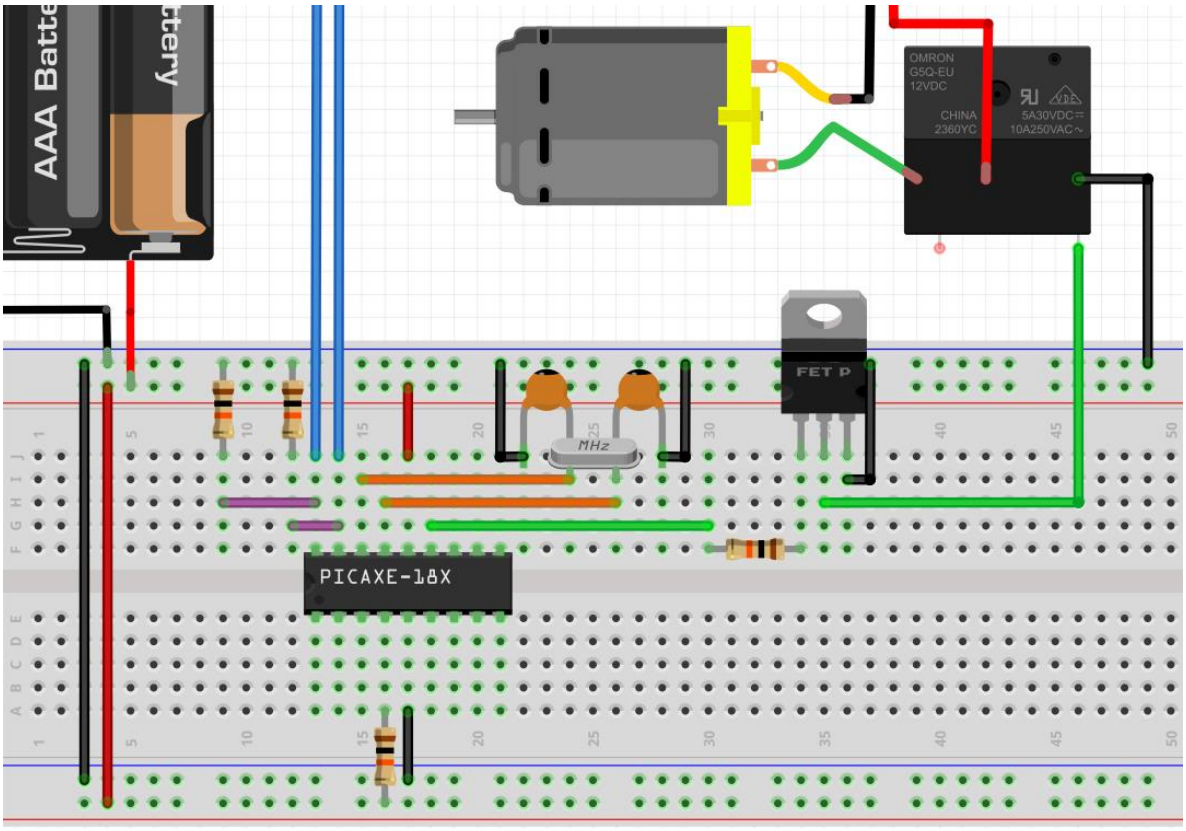

Figura 49 – Conexión del **PIC16F84A** a la bomba de agua.

Figura 50 – Control de nivel de agua con **PIC16F84A**.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje ensamblador en los **microcontroladores**, aprender a usar el software MPLAB y simular su funcionamiento en Proteus Professional, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 7

Display 7 segmentos - PIC16F84A

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a controlar un display de 7 segmentos en **lenguaje ensamblador** usando el software **MPLAB v8.92**.
- Aprender a configurar los fusibles de un microcontrolador **PIC16F84A**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **PIC16F84A** [1].
- Software **MPLAB v8.92** [2].
- Software **Proteus Professional** [3].
- Programador **PICSTART Plus** [4] o **PICKit 3** [5].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 7 resistencias 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 4MHz.
- 1 Display 7 segmentos (en caso de que la tarjeta de entrenamiento no los incluya).

Trabajo previo.

- Conocimientos vistos en clase: **Lenguaje ensamblador**.
- Conocimientos vistos en clase: **Hoja de datos del PIC16F84A**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje ensamblador.

Para realizar la práctica 6, seguir los pasos de la práctica 1 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje ensamblador.

Escribir los siguientes códigos y seguir los pasos de la práctica 1 para la construcción el proyecto y obtener los archivos "**.HEX**" y ".**COF**":

```
; Práctica 7: Contador con display de 7 segmentos                    ;
; Microcontrolador: PIC16F84A      Frecuencia: 4MHz  Voltaje:5V      ;
; Herramienta: MPLAB v8.92 - Microchip MPASM Toolsuite               ;
;-------------- Archivos de inclusión (Librerías) ----------------------
#INCLUDE "p16f84.inc"                ; Librería del PIC16F84
;-------------- Vector de reset ----------------------------------------
                ORG     H'000'       ; Vector de reset del procesador.
                GOTO    CONFIGURACION ; Nos manda a la etiqueta
                ORG     H'005'       ; CONFIGURACION.
;-------------- Configuración del microcontrolador ---------------------
CONFIGURACION   BSF     STATUS,RP0   ; Nos movemos al Banco 1.
                CLRF    TRISB        ; Puerto B como salida.
                BCF     STATUS,RP0   ; Regresamos al Banco 0.
                CLRF    PORTB        ; Limpia el registro PORTB.
;-------------- Loop Infinito ------------------------------------------
CONTADOR        CALL    CERO
                CALL    RETARDO_1seg
                CALL    UNO
                CALL    RETARDO_1seg
                CALL    DOS
                CALL    RETARDO_1seg
                CALL    TRES
                CALL    RETARDO_1seg
                CALL    CUATRO
                CALL    RETARDO_1seg
                CALL    CINCO
                CALL    RETARDO_1seg
                CALL    SEIS
                CALL    RETARDO_1seg
                CALL    SIETE
                CALL    RETARDO_1seg
                CALL    OCHO
                CALL    RETARDO_1seg
                CALL    NUEVE
                CALL    RETARDO_1seg
                GOTO    CONTADOR
;============== Fin del Programa ======================================
#INCLUDE <Display7S.inc>             ; Librería para Display 7 seg.
#INCLUDE <Retardo.inc>               ; Librería para retardo.
                END                  ; Le indica fin al ensamblador
```

Código 9 – Archivo principal "**main.ASM**".

```
;-------------- Variables a usar -------------------------------------
;----------------------- gfedcba -----------------------------------
CERO            MOVLW   B'00111111'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’01000000 para ánodo.
;-------------- Variables a usar -------------------------------------
UNO             MOVLW   B'00000110'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’01111001 para ánodo.
;-------------- Variables a usar -------------------------------------
DOS             MOVLW   B'01011011'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’00100100 para ánodo.
;-------------- Variables a usar -------------------------------------
TRES            MOVLW   B'01001111'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’00110000 para ánodo.
;-------------- Variables a usar -------------------------------------
CUATRO          MOVLW   B'01100110'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’00011001 para ánodo.
;-------------- Variables a usar -------------------------------------
CINCO           MOVLW   B'01101101'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’00010010 para ánodo.
;-------------- Variables a usar -------------------------------------
SEIS            MOVLW   B'01111100'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’00000011 para ánodo.
;-------------- Variables a usar -------------------------------------
SIETE           MOVLW   B'00000111'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’01111000 para ánodo.
;-------------- Variables a usar -------------------------------------
OCHO            MOVLW   B'01111111'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’00000000 para ánodo.
;-------------- Variables a usar -------------------------------------
NUEVE           MOVLW   B'01100111'     ; Valor binario para cátodo común
                MOVWF   PORTB           ; B’00011000 para ánodo.
```

Código 10 – Librería “**Display7S.INC**”.

```
;-------------- Variables a usar -------------------------------------
CONTADOR1       EQU     H'00F'          ; Variable guardada en GPR3
CONTADOR2       EQU     H'010'          ; Variable guardada en GPR4
CONTADOR3       EQU     H'011'          ; Variable guardada en GPR5
;-------------- Subrutina RETARDO_10uS -------------------------------
RETARDO_1seg    MOVLW   D'10'           ; W = 10 decimal
                MOVWF   CONTADOR3       ; CONTADOR3 = W
;-------------- Sección R_EXTERNA ------------------------------------
R_EXTERNA       MOVLW   D'100'          ; W = 100 decimal
                MOVWF   CONTADOR2       ; CONTADOR2 = W
;-------------- Sección R_INTERMEDIA ---------------------------------
R_INTERMEDIA    MOVLW   D'250'          ; W = 250 decimal
                MOVWF   CONTADOR1       ; CONTADOR1 = W
;-------------- Sección R_INTERNA ------------------------------------
R_INTERNA       DECFSZ  CONTADOR1,1     ; CONTADOR1 = CONTADOR1-1
                GOTO    R_INTERNA       ;
                DECFSZ  CONTADOR2,1     ; CONTADOR2 = CONTADOR2-1
                GOTO    R_INTERMEDIA
                DECFSZ  CONTADOR3,1     ; CONTADOR3 = CONTADOR3-1
                GOTO    R_EXTERNA       ;
                RETURN                  ; Salimos del RETARDO_1seg
;============== Fin del Programa ======================================
                END                     ; Le indica fin al ensamblador
```

Código 11 – Librería “**Retardo.INC**”.

Agregar en la carpeta “**Header Files**” del árbol de proyecto la librería “**Display7S.INC**” y “**Retardo.INC**”, presionar el botón “**Build All**”, figura 51.

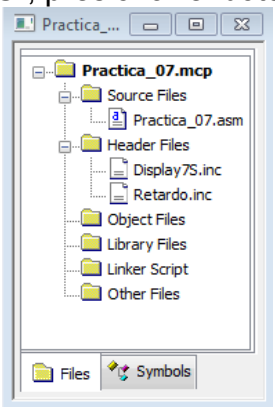

Figura 51 – Árbol de proyectos de práctica 7.

3) Configurar el microcontrolador PIC16F84A

Seguir los pasos de la práctica 1, para construir el proyecto y realizar la configuración de los fusibles.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 52, seguir los pasos de la práctica 1, para realizar la simulación funcional del circuito:

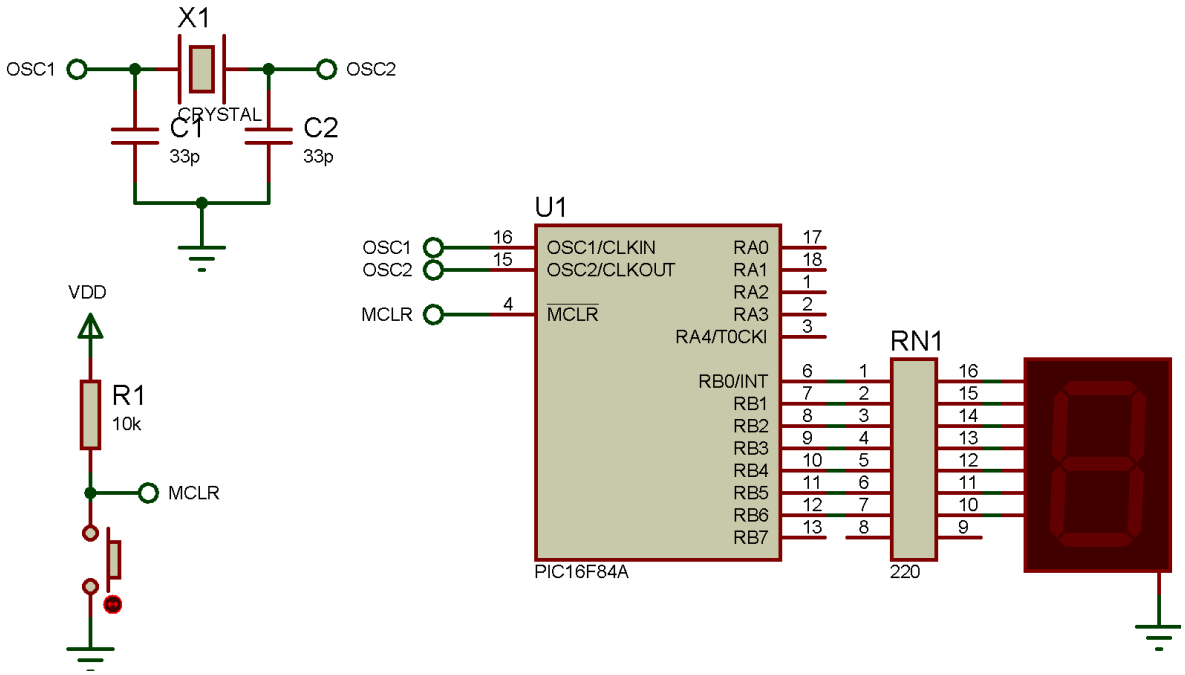

Figura 52 – Diagrama esquemático de práctica 7.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 1, para realizar la programación del **PIC16F84A**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 53.

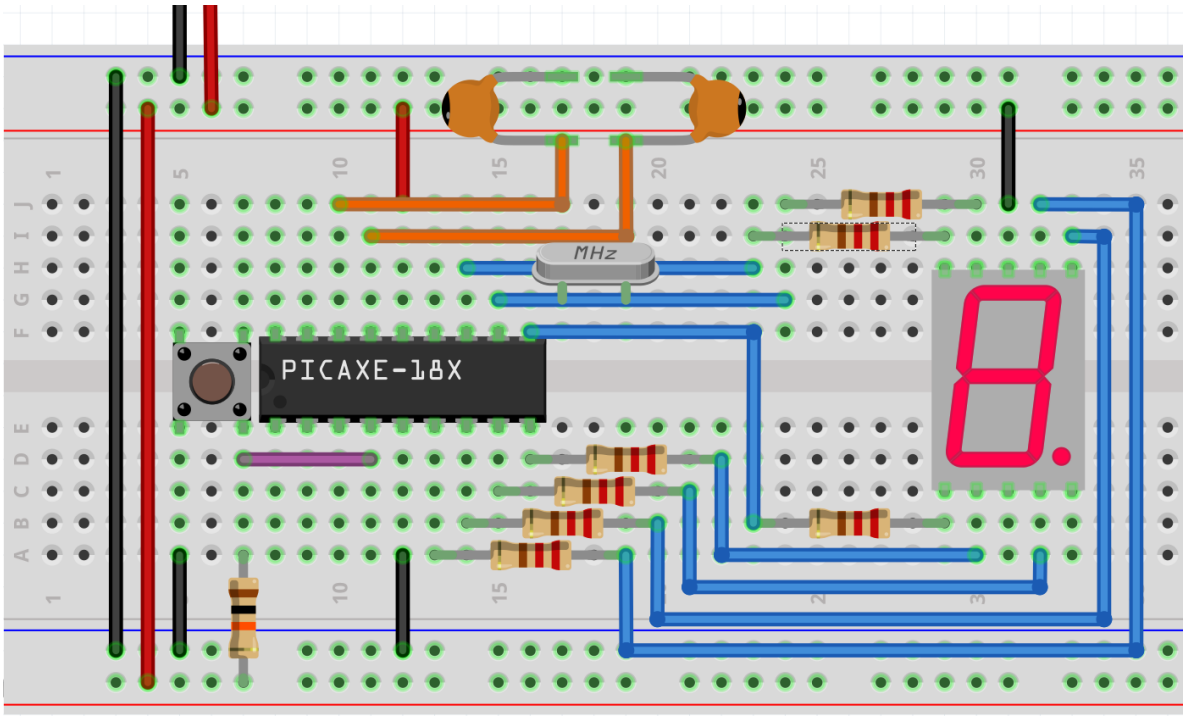

Figura 53 – Display de 7 segmentos con **PIC16F84A**.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje ensamblador en los **microcontroladores**, aprender a usar el software MPLAB y simular su funcionamiento en Proteus Professional, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 8

Programación en lenguaje C – ATMEGA328P

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a usar el software **Atmel Studio 7**.
- Aprender a programar microcontroladores en **lenguaje C**.
- Aprender a configurar los fusibles de un microcontrolador **ATmega328P**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **ATMEGA328P** [6].
- Software **Atmel Studio 7** [7].
- Software **Proteus Professional** [3].
- Programador **AVR Dragon** [8] o tarjeta **Arduino** [9].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 1 resistencia 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 16MHz.
- 1 LED (en caso de que la tarjeta de entrenamiento no los incluya).

Trabajo previo.

- Conocimientos vistos en clase: **Hoja de datos del ATMEGA328P**
- Conocimientos vistos en asignaturas previas: **Lenguaje C**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Identificar el área de trabajo en Atmel Studio v7.

Una vez abierto el software se debe identificar cada uno de los elementos en la GUI como se muestra en la figura 53. De color rojo está la **barra de herramientas**, en color verde el **árbol de proyecto**, en color naranja la **ventana de propiedades de cada archivo**, en color azul el **editor de código**, en morado la **ventana lista de errores** donde nos muestra la cantidad tipo y ubicación de cada error del proyecto al compilar, en color magenta la **consola de salida** donde muestra información el tamaño del proyecto, memoria RAM y memoria Flash consumida en la compilación del proyecto.

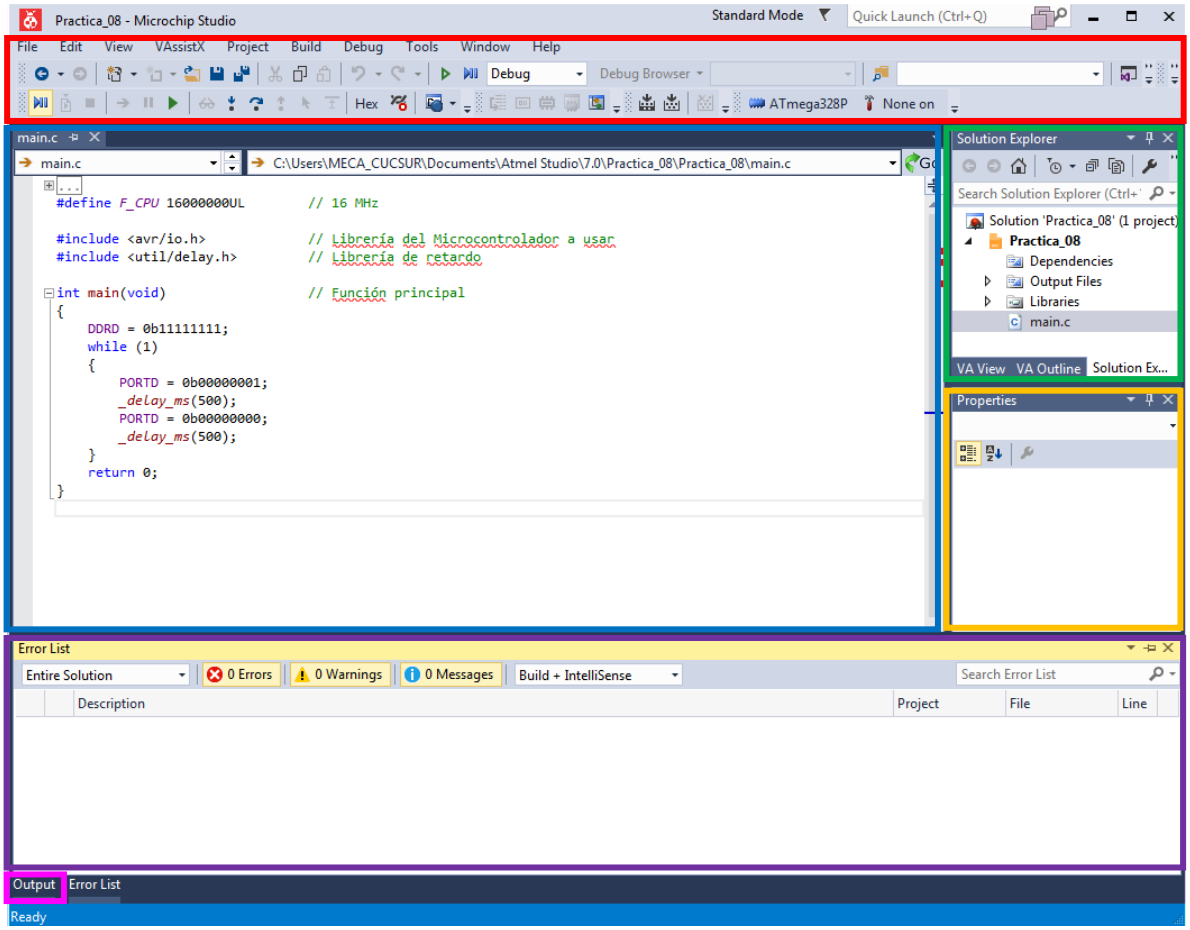

Figura 53 – Interfaz gráfica para Atmel Studio v7.

2) Creación de un proyecto en lenguaje C.

Seleccionar en la barra de herramientas "**File** > **New > Project…**", en figura 54.

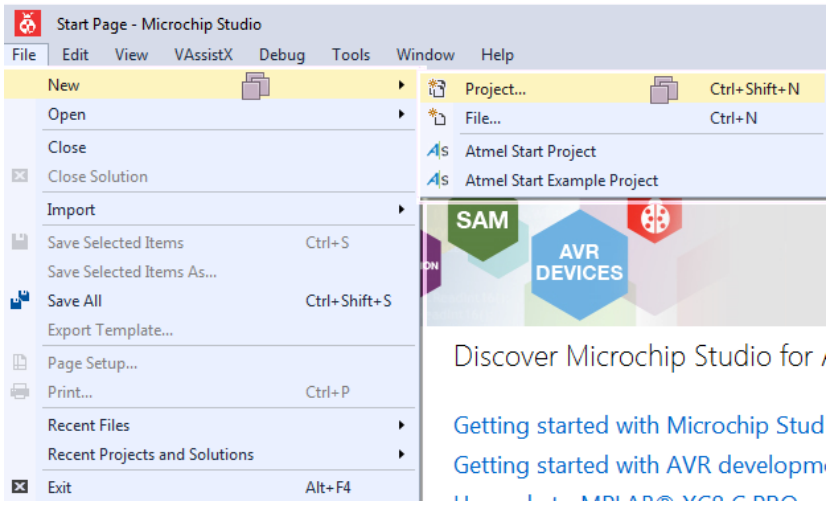

Figura 54 – Menú "**File**" en la barra de herramientas.

Aparece la ventana "**New Project**", seleccionar "**Installed > GCC C Executable Project**", escribir el nombre del proyecto como "**Practica_08**", presionar el botón "**OK**", figura 55.

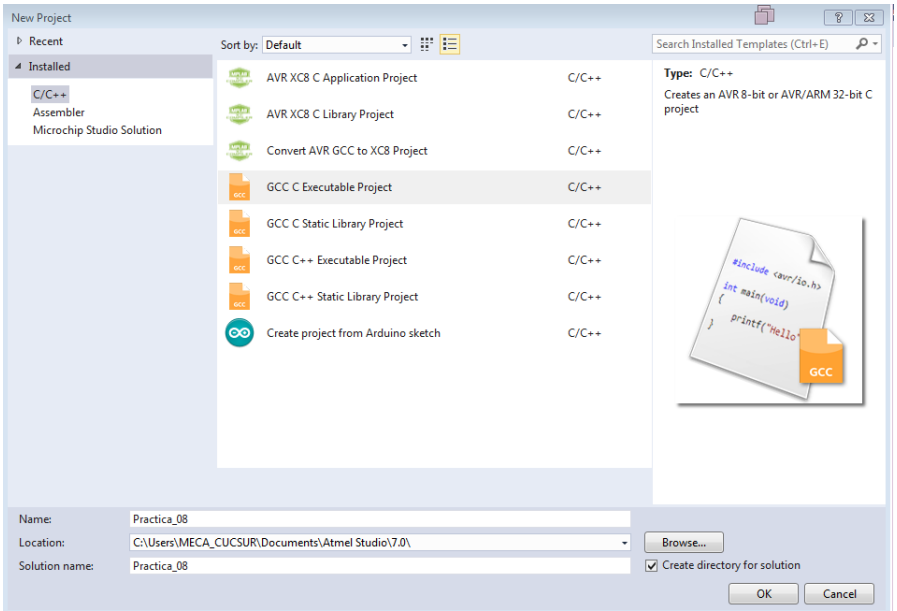

Figura 55 – Ventana de introducción al asistente de proyecto.

En la ventana de selección de dispositivo, elegir el microcontrolador "**ATmega328P"**, presionar el botón "**OK**", figura 56.

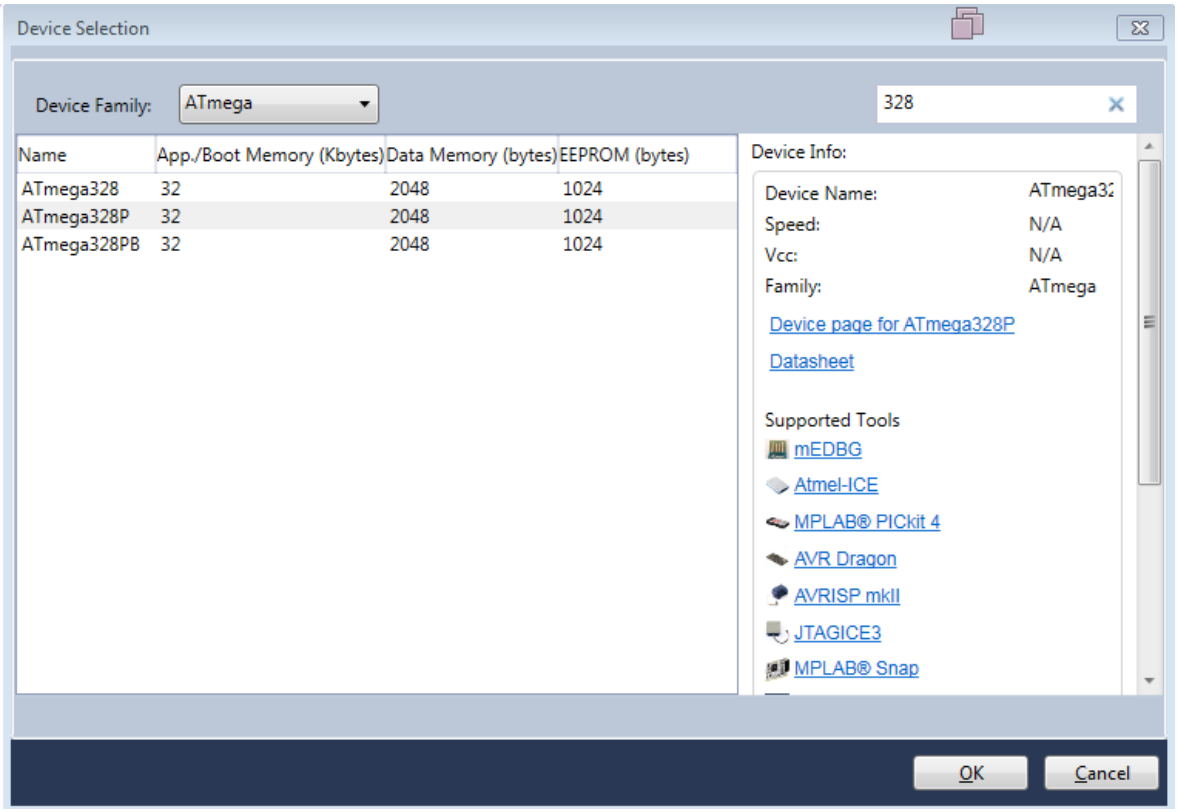

Figura 56 – Ventana de selección de dispositivo.

3) Código en lenguaje C.

Escribir el código como se muestra a continuación:

```
/*
-------------------------------------------------------------------------
 Práctica 8: Programación en lenguaje C
 Microcontrolador: ATMEGA328P   Frecuencia: 16MHz        Voltaje:5V
 Herramienta: Atmel Studio v7 – GCC C
-------------------------------------------------------------------------
*/
//----------------------- Frecuencia del CPU ----------------------------
#define F_CPU 16000000UL                    // Velocidad de Reloj de 16MHz.
#define SALIDA_UNO(port,bit) (port)|= (1<<(bit))
#define SALIDA_CERO(port,bit) (port)&= ~(1<<(bit))
//----------------------- Librerías -------------------------------------
#include <avr/io.h>                         // Librería para AVR.
#include <util/delay.h>                     // Librería para retardo.
//======================= Programa Principal ============================
int main(void)                              // Función Principal.
{
        DDRB = 0b11111111;                  // Configura puerto B como salida.
        PORTB = 0b00000000;                 // Limpia el puerto B.
        while (1)                           // Ciclo while infinito.
        {
                SALIDA_UNO(PORTB,5);        // PB5 = 1
                _delay_ms(500);             // Retardo
                SALIDA_CERO(PORTB,5);       // PB5 = 0
                _delay_ms(500);             // Retardo
        }
        return 0;
}
```

Código 12 – Archivo principal “**main.C**”.

Seleccionar “**Build Solution**” en la barra de herramientas para la construcción del proyecto y obtener los archivos “**.HEX**” y “.**ELF**”, figura 57.

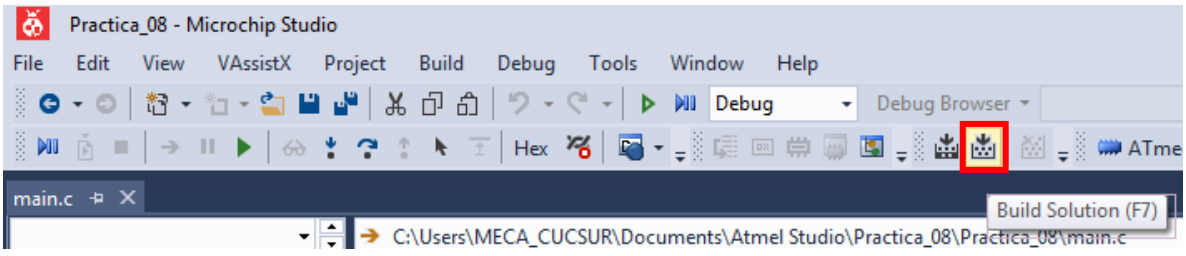

Figura 57 – Botón “**Build Solution**”.

La consola de salida mostrará el mensaje "**BUILD SUCCEEDED**", indicando que no ocurrió ningún error al compilar el proyecto, figura 58.

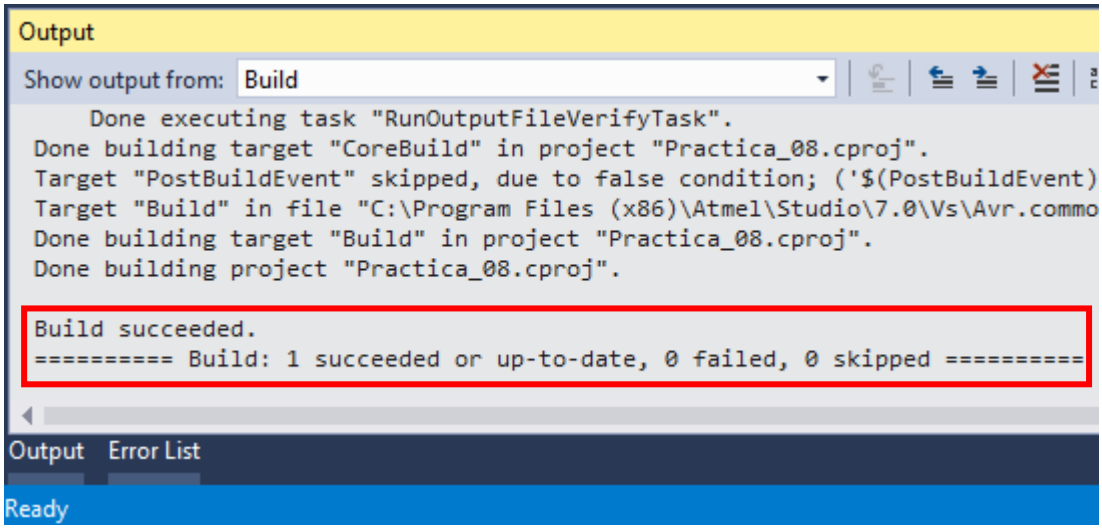

Figura 58 – Ventana de asistente para construir el proyecto.

4) Simulación funcional del circuito diseñado.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 59.

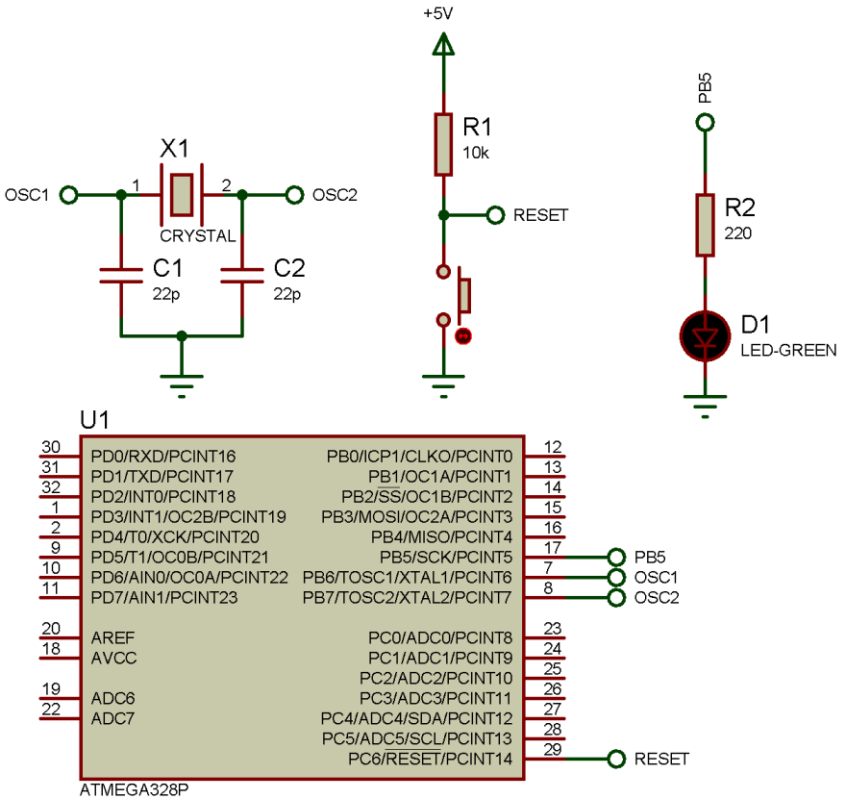

Figura 59 – Diagrama esquemático de práctica 8.

Hacer doble clic en el microcontrolador para ingresar a las propiedades del componente, aparece la ventana "**Edit Component**", presionar el botón "**Program File**", seleccionar el archivo "**Practica_08.HEX**", configurar la frecuencia de reloj a 16MHz, y configurar los fusibles como se muestra en la figura 60.

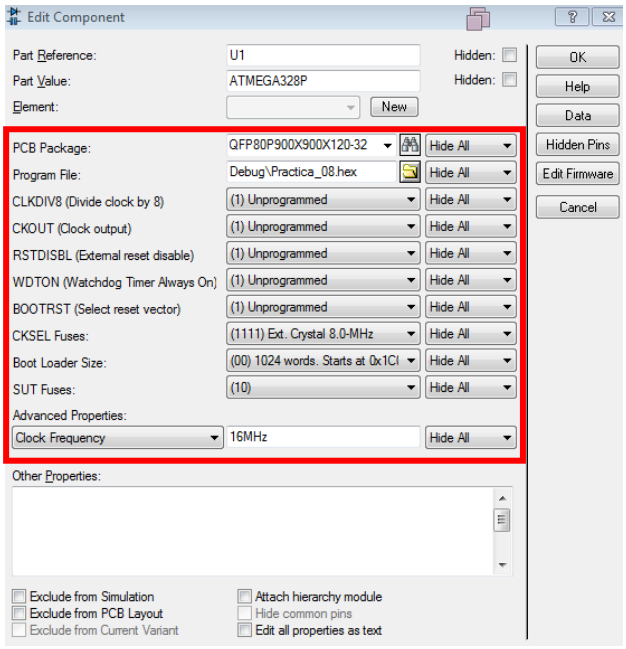

Figura 60 – Ventana "**Edit Component**".

Presionar el botón "**OK**", presionar el botón RUN para ejecutar la simulación.

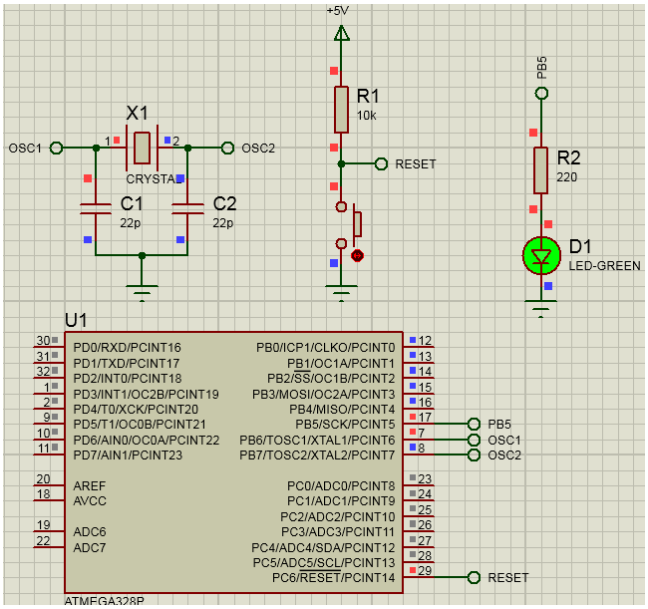

Figura 61 – El **LED** prende y apaga cada 1 segundo.

5) Programación del microcontrolador físico.

Tener a la mano un programador **AVR Dragon**, o una tarjeta **Arduino** con microcontrolador AVR, conectar el programador **AVR Dragon** o la tarjeta **Arduino** a la computadora con el cable USB, figura 62 y 63.

Figura 62 – Programador **AVR Dragon**.

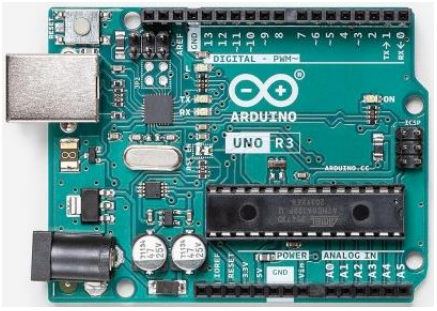

Figura 63 – Tarjeta **Arduino**.

Si se usa el programador **AVR Dragon**, se deberá seguir los pasos del Apéndice C. En caso de usar una tarjeta **Arduino** con microcontrolador AVR, se deberá seguir los pasos del Apéndice D.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 64.

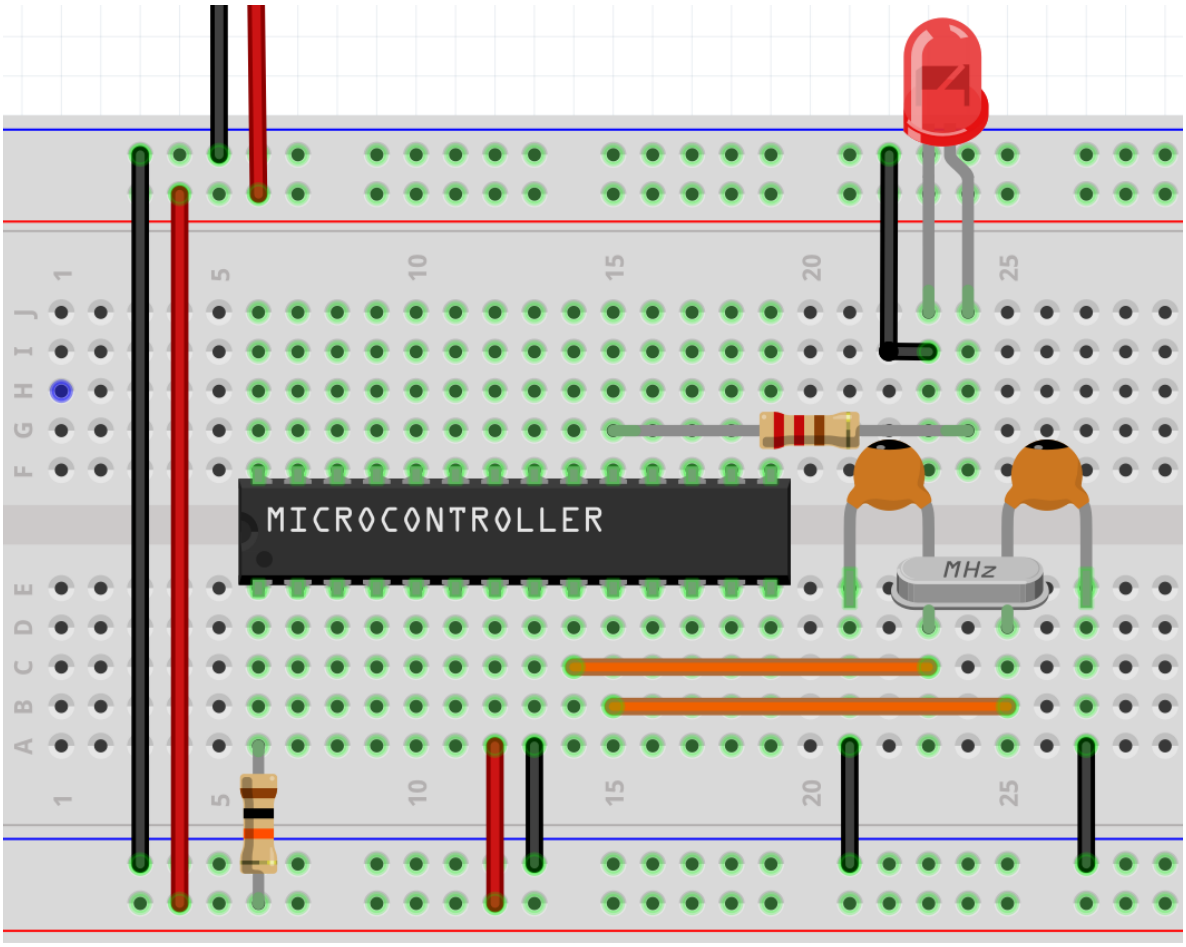

Figura 64 – Parpadeo de un LED con ATMEGA328P.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje C en los **microcontroladores AVR**, aprender a usar el software **Atmel Studio**, y simular su funcionamiento en **Proteus Professional**, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 9

Entradas y salidas digitales en C.

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a usar el software **Atmel Studio 7**.
- Aprender a programar microcontroladores en **lenguaje C**.
- Aprender a configurar los fusibles de un microcontrolador **ATmega328P**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **ATMEGA328P** [6].
- Software **Atmel Studio 7** [7].
- Software **Proteus Professional** [3].
- Programador **AVR Dragon** [8] o tarjeta **Arduino** [9] **.**
- Fuente de alimentación fija de 5V.
- 2 resistencias 10k Ohms.
- 1 resistencia 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 16MHz.
- 1 LED (en caso de que la tarjeta de entrenamiento no los incluya).
- 1 botón pulsador.

Trabajo previo.

- Conocimientos vistos en clase: **Hoja de datos del ATMEGA328P**
- Conocimientos vistos en asignaturas previas: **Lenguaje C**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) **Creación de un proyecto en lenguaje C.**

Para realizar la práctica 9, seguir los pasos de la práctica 8 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) **Código en lenguaje C.**

Escribir el siguiente código y seguir los pasos de la práctica 8 para la construcción el proyecto y obtener los archivos "**.HEX**" y ".**ELF**":

```
/*
-------------------------------------------------------------------------
 Práctica 9: Entrada y salida digital
 Microcontrolador: ATMEGA328P   Frecuencia: 16MHz        Voltaje:5V
 Herramienta: Atmel Studio v7 – GCC C
-------------------------------------------------------------------------
*/
//----------------------- Frecuencia del CPU ----------------------------
#define F_CPU 16000000UL                    // Velocidad de Reloj de 16MHz.
//----------------------- Librerías -------------------------------------
#include <avr/io.h>                         // Librería para AVR.
#include <util/delay.h>                     // Librería para retardo.
#include "MyLib.h"
//======================= Programa Principal ============================
int main(void)                              // Función Principal.
{
        GPIO_Conf();                        // Configura puerto B como salida.
        while (1)                           // Ciclo while infinito.
        {
            if(BOTON_PRESIONADO_PC(0) == 1)
            {
                SALIDA_UNO(PORTD,5);
            }
            if(BOTON_PRESIONADO_PC(0) == 0)
            {
                SALIDA_CERO(PORTD,5);
             }
        }
        return 0;
}
```

Código 13 – Archivo principal "**main.C**".

```
#ifndef MyLib_H_
#define MyLib_H_
//------ Funciones para salida -----------------------------
#define SALIDA_UNO(port,bit) (port) |= (1<<(bit))
#define SALIDA_CERO(port,bit) (port) &= ~(1<<(bit))
//------ Funciones para entradas ---------------------------
#define BOTON_PRESIONADO_PB(bit) (PINB & (1<<bit))
#define BOTON_PRESIONADO_PC(bit) (PINC & (1<<bit))
#define BOTON_PRESIONADO_PD(bit) (PIND & (1<<bit))
//------ Funcion para configurar entradas y salidas -----
void GPIO_Conf()
{
	DDRB = 0b11111111;
	DDRC = 0b00000000;
	DDRD = 0b11111111;
	PORTB = 0b00000000;
	PORTC = 0b00000000;
	PORTD = 0b00000000;
}
#endif
```

Código 14 – Librería “**MyLib.H**”.

3) Configurar el microcontrolador ATMEGA328P

Seguir los pasos de la práctica 8, para construir el proyecto.

4) Simulación del código.

Abrir “**Proteus Professiona**l” y dibujar el diagrama esquemático de la figura 65, seguir los pasos de la práctica 8, para realizar la simulación funcional del circuito:

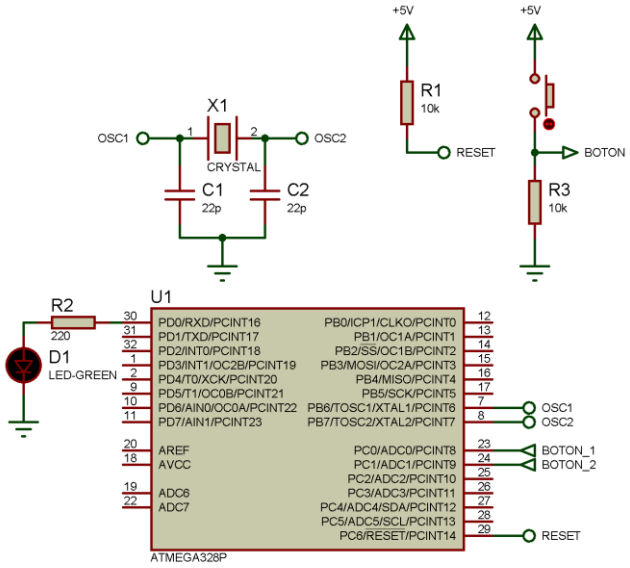

Figura 65 – Diagrama esquemático de práctica 9.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 8, para realizar la programación del **ATMEGA328P**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 66.

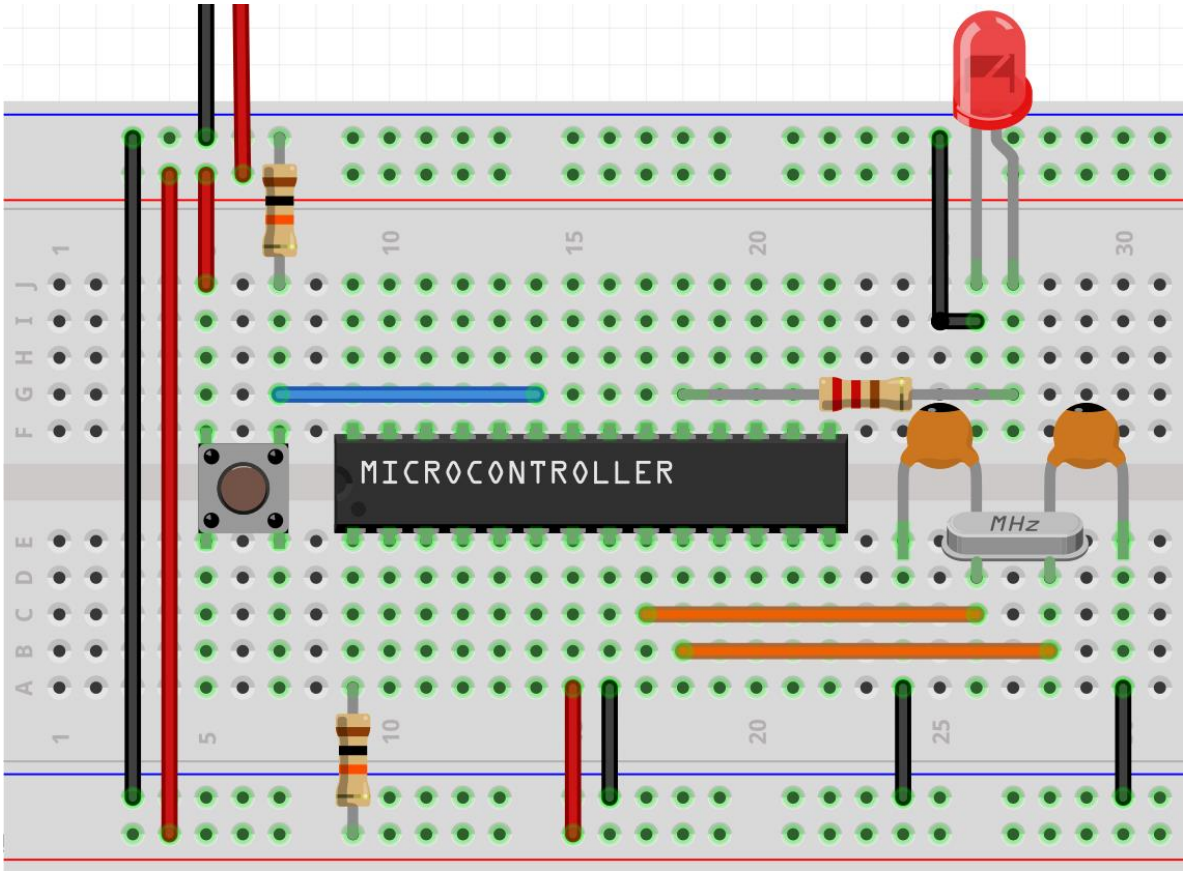

Figura 66 – Entradas y salidas digitales en **ATMEGA328P**.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje C en los **microcontroladores AVR**, aprender a usar el software **Atmel Studio**, y simular su funcionamiento en **Proteus Professional**, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 10

Operadores Bitwise.

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a usar el software **Atmel Studio 7**.
- Aprender a programar microcontroladores en **lenguaje C**.
- Aprender a configurar los fusibles de un microcontrolador **ATmega328P**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **ATMEGA328P** [6].
- Software **Atmel Studio 7** [7].
- Software **Proteus Professional** [3].
- Programador **AVR Dragon** [8] o tarjeta **Arduino** [9].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 1 resistencia 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 16MHz.
- 8 LEDs (en caso de que la tarjeta de entrenamiento no los incluya).

Trabajo Previo.

- Conocimientos vistos en clase: **Hoja de datos del ATMEGA328P**
- Conocimientos vistos en asignaturas previas: **Lenguaje C**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) **Creación de un proyecto en lenguaje C.**

Para realizar la práctica 10, seguir los pasos de la práctica 8 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) **Código en lenguaje C.**

Escribir el siguiente código y seguir los pasos de la práctica 8 para la construcción el proyecto y obtener los archivos "**.HEX**" y ".**ELF**":

```
/*
-------------------------------------------------------------------------
 Práctica 10: Operadores Bitwise
 Microcontrolador: ATMEGA328P   Frecuencia: 16MHz        Voltaje:5V
 Herramienta: Atmel Studio v7 – GCC C
-------------------------------------------------------------------------
*/
//----------------------- Frecuencia del CPU ----------------------------
#define F_CPU 16000000UL                    // Velocidad de Reloj de 16MHz.
//----------------------- Librerías -------------------------------------
#include <avr/io.h>                         // Librería para AVR.
#include <util/delay.h>                     // Librería para retardo.
#include "MyLib.h"
//======================= Programa Principal ============================
int main(void)                              // Función Principal.
{
        GPIO_Conf();                        // Configura puerto B como salida.
        while (1)                           // Ciclo while infinito.
        {
             PORTD = 0b00000001;
             _delay_ms(50);
             while(PORTD<=0b01000000)
             {
               PORTD = PORTD<<1;
               _delay_ms(50);
             }
             while(PORTD>=0b00000100)
             {
               PORTD = PORTD>>1;
               _delay_ms(50);
             }
        }
        return 0;
}
```

Código 15 – Archivo principal "**main.C**".

3) Configurar el microcontrolador ATMEGA328P

Seguir los pasos de la práctica 8, para construir el proyecto.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 67, seguir los pasos de la práctica 8, para realizar la simulación funcional del circuito:

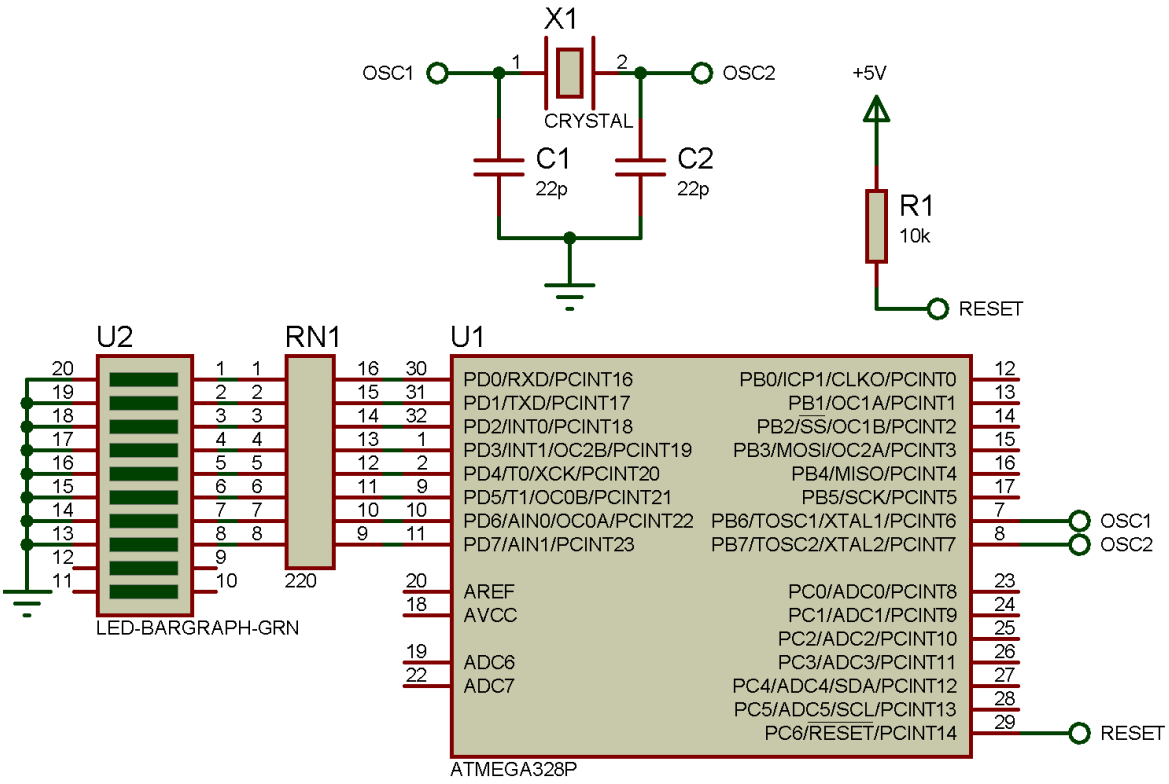

Figura 67 – Diagrama esquemático de práctica 10.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 8, para realizar la programación del **ATMEGA328P**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 68.

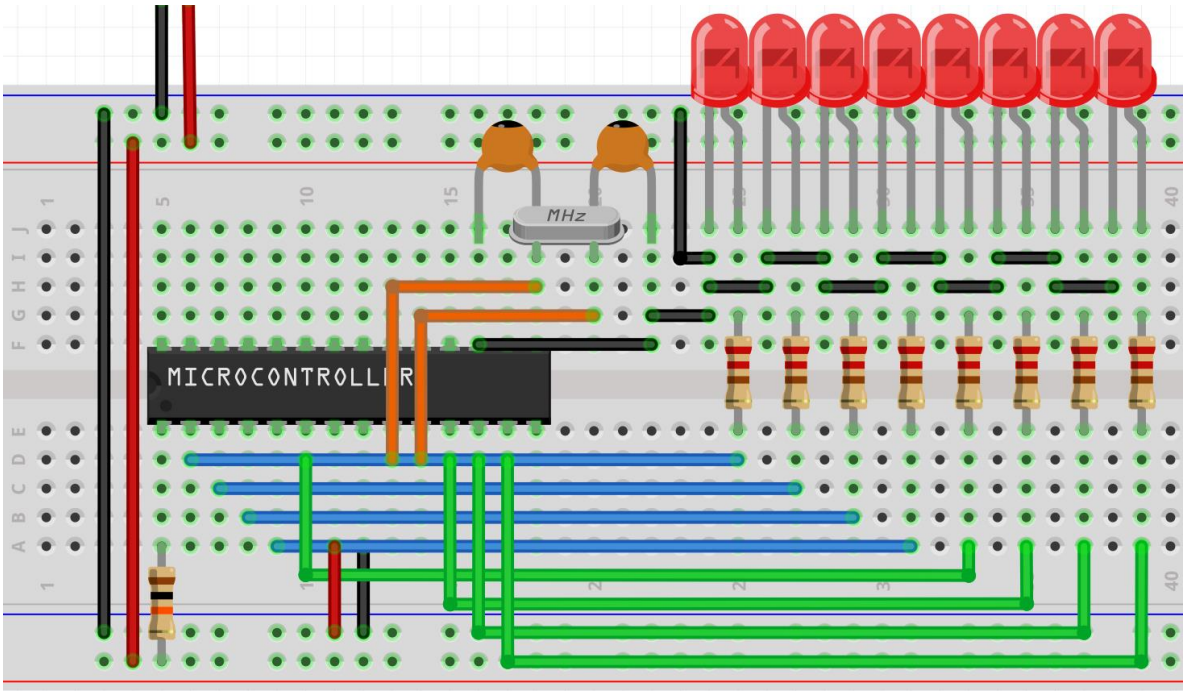

Figura 68 – Corrimiento de LEDs en **ATMEGA328P**.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje C en los **microcontroladores AVR**, aprender a usar el software **Atmel Studio**, y simular su funcionamiento en **Proteus Professional**, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 11

Interrupciones y timers.

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a usar el software **Atmel Studio 7**.
- Aprender a programar microcontroladores en **lenguaje C**.
- Aprender a configurar los fusibles de un microcontrolador **ATmega328P**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **ATMEGA328P** [6].
- Software **Atmel Studio 7** [7].
- Software **Proteus Professional** [3].
- Programador **AVR Dragon** [8] o tarjeta **Arduino** [9].
- Fuente de alimentación fija de 5V.
- 3 resistencias 10k Ohms.
- 8 resistencias 220 Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 16MHz.
- 8 LEDs (en caso de que la tarjeta de entrenamiento no los incluya).

Trabajo Previo.

- Conocimientos vistos en clase: **Hoja de datos del ATMEGA328P**
- Conocimientos vistos en asignaturas previas: **Lenguaje C**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje C.

Para realizar la práctica 11, seguir los pasos de la práctica 8 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje C.

Escribir el siguiente código y seguir los pasos de la práctica 8 para la construcción el proyecto y obtener los archivos "**.HEX**" y ".**ELF**":

```
/*
 Práctica 11: Interrupciones y timers
 Microcontrolador: ATMEGA328P       Frecuencia: 16MHz           Voltaje:5V
 Herramienta: Atmel Studio v7 – GCC C
*/
//----------------------- Frecuencia del CPU ----------------------------
#define F_CPU 16000000UL     // Velocidad de Reloj de 16MHz.
//----------------------- Librerías -------------------------------------
#include <avr/io.h>          // Librería para AVR.
#include <util/delay.h>      // Librería para retardo.
#include <avr/interrupt.h>   // librería necesaria para manejar las interrupciones
#include "GPIO.h"
//====================== Programa Principal ============================
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
void GPIO_Conf();
void IntExt_Conf();
unsigned int cuenta = 0;

ISR(INT0_vect,ISR_NAKED)     // Vector de interrupción externa del INT0
{
        cuenta++;
        PORTC = cuenta;
        reti();
}

ISR(INT1_vect,ISR_NAKED)     // Vector de interrupción externa del INT0
{
        cuenta--;
        PORTC = cuenta;
        reti();
}

int main (void)
{       GPIO_Conf();
        IntExt_Conf();
        sei();
        while(1)
        {
        }
}
```

Código 16 – Archivo principal "**main.C**".

```
#ifndef GPIO_H_
#define GPIO_H_

void GPIO_Conf()
{
  DDRB = 0b00000000;
  DDRC = 0b00000000;
  DDRD = 0b11111111;
  PORTC = 0;
}

void IntExt_Conf()
{
  EICRA = (1<<ISC11)|(1<<ISC10)|(1<<ISC01)|(1<<ISC00);
  EIMSK = (1<<INT1)|(1<<INT0);
  EIFR = (0<<INTF1)|(0<<INTF0);
  PCICR = (0<<PCIE2)|(0<<PCIE1)|(0<<PCIE0);
  PCIFR = (0<<PCIF2)|(0<<PCIF1)|(0<<PCIF0);
  PCMSK2 =
(0<<PCINT23)|(0<<PCINT22)|(0<<PCINT21)|(0<<PCINT20)|(0<<PCINT19)|(0<<PCINT18)|(0<<PCINT17)
|(0<<PCINT16);
  PCMSK1 =
(0<<PCINT14)|(0<<PCINT13)|(0<<PCINT12)|(0<<PCINT11)|(0<<PCINT10)|(0<<PCINT9)|(0<<PCINT8);
  PCMSK0 =
(0<<PCINT7)|(0<<PCINT6)|(0<<PCINT5)|(0<<PCINT4)|(0<<PCINT3)|(0<<PCINT2)|(0<<PCINT1)|(0<<PC
INT0);
}

#endif
```

Código 17 – Librería “**GPIO.H**”.

3) Configurar el microcontrolador ATMEGA328P

Seguir los pasos de la práctica 8, para construir el proyecto.

4) Simulación del código.

Abrir “**Proteus Professiona**l” y dibujar el diagrama esquemático de la figura 69, seguir los pasos de la práctica 8, para realizar la simulación funcional del circuito:

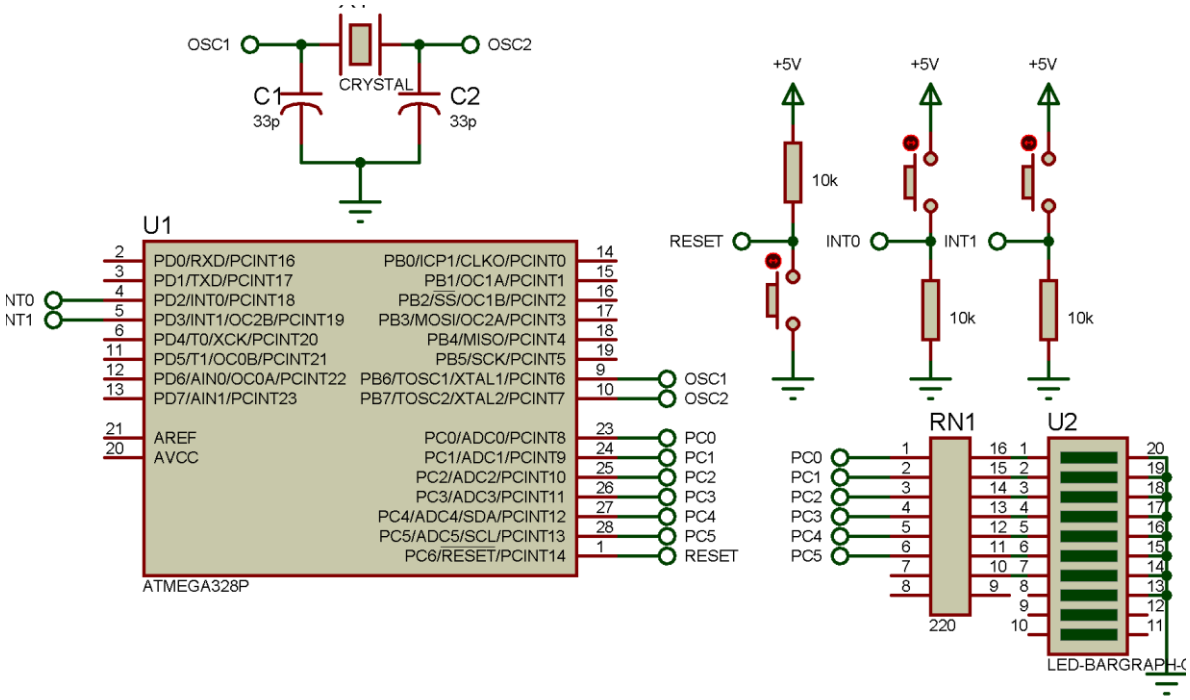

Figura 69 – Diagrama esquemático de práctica 11.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 8, para realizar la programación del **ATMEGA328P**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 70.

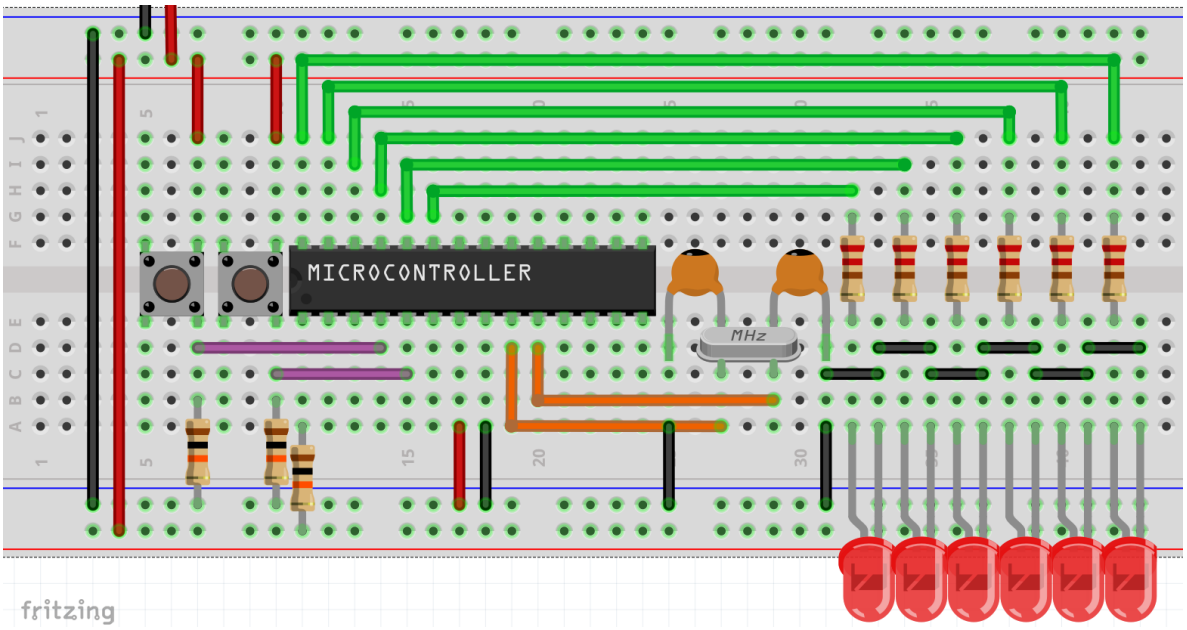

Figura 70 – Entradas y salidas digitales en **ATMEGA328P**.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje C en los **microcontroladores AVR**, aprender a usar el software **Atmel Studio**, y simular su funcionamiento en **Proteus Professional**, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 12

Pantalla LCD.

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a usar el software **Atmel Studio 7**.
- Aprender a programar microcontroladores en **lenguaje C**.
- Aprender a configurar los fusibles de un microcontrolador **ATmega328P**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **ATMEGA328P** [6].
- Software **Atmel Studio 7** [7].
- Software **Proteus Professional** [3].
- Programador **AVR Dragon** [8] o tarjeta **Arduino** [9].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 16MHz.
- 1 pantalla LCD (en caso de que la tarjeta de entrenamiento no la incluya).

Trabajo Previo.

- Conocimientos vistos en clase: **Hoja de datos del ATMEGA328P**
- Conocimientos vistos en asignaturas previas: **Lenguaje C**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje C.

Para realizar la práctica 12, seguir los pasos de la práctica 8 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje C.

Escribir el siguiente código y seguir los pasos de la práctica 8 para la construcción el proyecto y obtener los archivos "**.HEX**" y ".**COF**":

```
/*
-------------------------------------------------------------------------
 Práctica 12: Pantalla LCD
 Microcontrolador: ATMEGA328P   Frecuencia: 16MHz        Voltaje:5V
 Herramienta: Atmel Studio v7 – GCC C
-------------------------------------------------------------------------
*/
//----------------------- Frecuencia del CPU ----------------------------
#define F_CPU 16000000UL                    // Velocidad de Reloj de 16MHz.
//----------------------- Librerías -------------------------------------
#include <avr/io.h>                         // Librería para AVR.
#include <util/delay.h>                     // Librería para retardo.
#include "LCD.h"
//======================= Programa Principal ============================
int main(void)                              // Función Principal.
{
        DDRD = 0b11111111;
        LCD_Init();
        LCD_Gotoxy(0,0);
        LCD_Message("MECATRONICA");
        LCD_Gotoxy(0,1);
        LCD_Message("UdeG CUCSUR");
        while (1)
        {
         }
}
```

Código 18 – Archivo principal "**main.C**".

```
#ifndef LCD_H_
#define LCD_H_
#define ClearBit(x,y) x &= ~(1<<y)
#define SetBit(x,y)   x |= (1<<y)
#include <stdlib.h>                // librería estándar
typedef uint8_t byte;              // declaración de entero para compatibilidad
typedef int8_t sbyte;              // con otros compiladores
// ------------------------------- Definiciones de configuración -----------------
#define LCD_RS      0              // pin para LCD R/S
#define LCD_E 1                    // pin para LCD ENABLE
#define DAT4  2                    // pin para d4
#define DAT5  3                    // pin para d5
#define DAT6  4                    // pin para d6
#define DAT7  5                    // pin para d7
#define port  PORTD                // asignamos el puerto a usar ej. PORTB
// ------------------------------- Comandos de control para HD44780 --------------
#define CLEARDISPLAY 0x01          // Limpia la pantalla
#define SETCURSOR 0x80             // Fija el cursor
void msDelay(int delay)            // Retardo exclusivo para el manejo del LCD
{
      for (int i=0;i<delay;i++)
      _delay_ms(1);
}
void PulseEnableLine ()            // Función para habilitar el LCD
{
      SetBit(port,LCD_E);          // Manda un 1 en el pin ENABLE
      _delay_us(40);               // Retardo de 40 microsegundos
      ClearBit(port,LCD_E);        // Manda un 0 en el pin ENABLE
}
void SendNibble(byte data)         // Función para manejo de cuartetos
{
      port &= 0xC3;                // 11000011 = limpia 4 líneas de datos
      if(data & (1<<4))
      {
            SetBit(port,DAT4);     // Manda un 1 en DAT4
      }
      if(data & (1<<5))
      {
            SetBit(port,DAT5);     // Manda un 1 en DAT5
      }
      if(data & (1<<6))
      {
            SetBit(port,DAT6);     // Manda un 1 en DAT6
      }
      if(data & (1<<7))
      {
            SetBit(port,DAT7);     // Manda un 1 en DAT7
      }
      PulseEnableLine();           // Sincroniza 4 bits en el controlador HITACHI
}
void SendByte (byte data)          // Función para enviar 8 bits (1 byte)
{
      SendNibble(data);            // Envía los 4 bits más significativos
      SendNibble(data<<4);         // Envía los 4 bits menos significativos
      ClearBit(port,5);            // Limpia
}
```

```
void LCD_Cmd (byte cmd)           // Función para comandos del HD44780
{
       ClearBit(port,LCD_RS);     // Pone un 0 en R/S = comandos
       SendByte(cmd);             // Envía los comandos
}
void LCD_Char (byte ch)           // Función para caracteres
{
       SetBit(port,LCD_RS);       // Pone un 1 en R/S = caracteres
       SendByte(ch);              // Envía los caracteres
}
void LCD_Init()
{
       LCD_Cmd(0x33);             // Inicializa el controlador
       LCD_Cmd(0x32);             // Configura en modo de solo 4 bits de entrada
       LCD_Cmd(0x28);             // Configura 2 líneas, y matriz 5x7
       LCD_Cmd(0x0C);             // Apaga el cursor (0x0E para encenderlo)
       LCD_Cmd(0x06);             // Dirección del cursor = derecha
       LCD_Cmd(0x01);             // Inicia con el display limpio
       msDelay(3);                // Retardo para inicializar el LCD
}
void LCD_Clear()                  // Función para limpiar el LCD
{
       LCD_Cmd(CLEARDISPLAY);     // Limpia el LCD
       msDelay(3);                // Retardo para procesar el comando
}
void LCD_Home()            // Función para colocar el cursor en home sin limpiar
{
       LCD_Cmd(SETCURSOR); // Comando para colocar el cursor en posición home
}
void LCD_Gotoxy(byte x, byte y)
{
       byte addr = 0;             // La línea 0 comienza en la dirección 0x00
       switch (y)
       {
       case 1: addr = 0x40; break;// La línea 1 comienza en la dirección 0x40
       case 2: addr = 0x14; break;
       case 3: addr = 0x54; break;
       }
       LCD_Cmd(SETCURSOR+addr+x); // Actualiza la posición del cursor en x,y
}
void LCD_Line(byte row)    // Función para mover el cursor a una línea específica
{
       LCD_Gotoxy(0,row);  // Se mueve el cursor a la columna 0, línea y
}
void LCD_Message(const char *text)// Función para mostrar una cadena en el LCD
{
       while (*text)              //
       {
       LCD_Char(*text++);  // Envía un carácter y se actualiza el puntero *text
       }
}
void LCD_Hex(int data)
{
       char st[8] = "";    // Guarda espacio para los resultados
       itoa(data,st,16);   // convierte a ascii hexadecimal
       LCD_Message(st);    // Manda el resultado al LCD
}
```

```
void LCD_Integer(int data)
{
	char st[8] = "";	// Guarda espacio para los resultados
	itoa(data,st,10);	// Convierte a ascii
	LCD_Message(st);	// Muestra el resultado en el LCD
}
void UpdateCursor (byte count)// Función para FillScreen
{
	switch(count)
	{
		case 0: LCD_Line(0); break;
		case 16: LCD_Line(1); break;
		case 32: LCD_Line(2); break;
		case 48: LCD_Line(3); break;
	}
}
char GetNextChar(char ch)  // Función para FillScreen
{
	if ((ch<0x20) | (ch>=0xFF))
	return 0x20;
	if ((ch>=0x7F) & (ch<0xA0))
	return 0xA0;
	return ++ch;
}
#define NUMCHARS 64		// Número de caracteres por pantalla
void FillScreen ()
{
	char ch = 'A';
	LCD_Clear();
	for (byte count=1;count<100;count++)
	{
		LCD_Gotoxy(18,0);
		LCD_Integer(count);
		for (byte i=0;i<NUMCHARS;i++)
		{
			UpdateCursor(i
			LCD_Char(ch);
			ch = GetNextChar(ch);
			msDelay(60);
		}
	}
}
#endif
```

Código 19 – Librería “**LCD.H**”.

3) Configurar el microcontrolador ATMEGA328P

Seguir los pasos de la práctica 8, para construir el proyecto.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 71, seguir los pasos de la práctica 8, para realizar la simulación funcional del circuito:

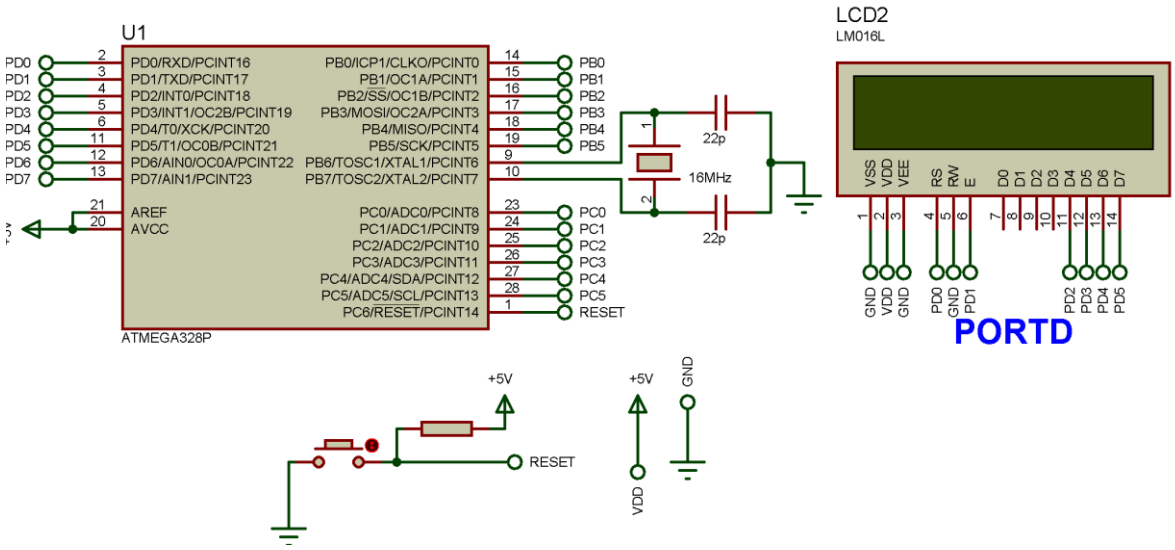

Figura 71 – Diagrama esquemático de práctica 12.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 8, para realizar la programación del **ATMEGA328P**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 72.

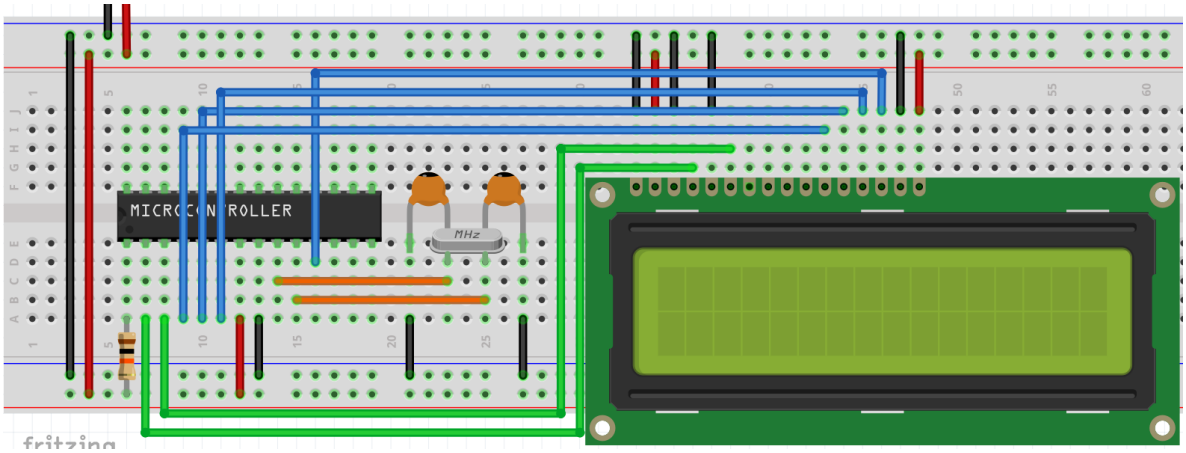

Figura 72 – Pantalla LCD 16x2 en **ATMEGA328P**.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje C en los **microcontroladores AVR**, aprender a usar el software **Atmel Studio**, y simular su funcionamiento en **Proteus Professional**, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 13

Convertidor analógico a digital.

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a usar el software **Atmel Studio 7**.
- Aprender a programar microcontroladores en **lenguaje C**.
- Aprender a configurar los fusibles de un microcontrolador **ATmega328P**.
- Simular el microcontrolador en el software **Proteus Professional**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Microcontrolador **ATMEGA328P** [6].
- Software **Atmel Studio 7** [7].
- Software **Proteus Professional** [3].
- Programador **AVR Dragon** [8] o tarjeta **Arduino** [9].
- Fuente de alimentación fija de 5V.
- 1 resistencia 10k Ohms.
- 2 potenciómetros 10k Ohms.
- 2 capacitores 22pF o 33pF.
- 1 cristal de 16MHz.
- 1 pantalla LCD (en caso de que la tarjeta de entrenamiento no la incluya).

Trabajo Previo.

- Conocimientos vistos en clase: **Hoja de datos del ATMEGA328P**
- Conocimientos vistos en asignaturas previas: **Lenguaje C**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje C.

Para realizar la práctica 6, seguir los pasos de la práctica 8 desde la creación del proyecto hasta la creación de un archivo para diagrama esquemático.

2) Código en lenguaje C.

Escribir el siguiente código y seguir los pasos de la práctica 8 para la construcción el proyecto y obtener los archivos "**.HEX**" y ".**COF**":

```
/*
-------------------------------------------------------------------------
 Práctica 13: Convertidor analógico a digital
 Microcontrolador: ATMEGA328P       Frecuencia: 16MHz           Voltaje:5V
 Herramienta: Atmel Studio v7 - GCC C
-------------------------------------------------------------------------
*/
#define F_CPU 16000000UL     // Velocidad de Reloj de 16MHz.
#include <avr/io.h>          // Librería para AVR.
#include <util/delay.h>      // Librería para retardo.
#include "LCD.h"             // Librería para las pantallas LCD 16x2.
#include "ADC.h"             // Librería para ADC.
//======================== Programa Principal ================================
int main(void)
{
       uint16_t x,y;
       float Vx,Vy;  // Variable flotante para guardar la conversión a voltaje.
       char Mensaje1[16], Mensaje2[16];
       GPIO_Init();  // Función de configuración e inicialización de los GPIO.
       LCD_Init();   // Función de inicialización de la pantalla LCD.
       ADC_Init();   // Función de configuración e inicialización del ADC.
//------------------------------ Ciclo Infinito ----------------------------
    while (1)
    {
              ADC_read(0);
              _delay_us(10);
              x = ADC;
              Vx = x*(5)/1023.0
              ADC_read(1);
              _delay_us(10);
              y = ADC;
              Vy = y*(5)/1023.0;
              sprintf(Mensaje1,"Voltaje = %.2f",Vx);
              LCD_Gotoxy(0,0);
              LCD_Message(Mensaje1);
              _delay_ms(10);
              sprintf(Mensaje2,"Voltaje = %.2f",Vy
              LCD_Gotoxy(0,1);
              LCD_Message(Mensaje2);
              _delay_ms(10);
    }
}
```

Código 20 – Archivo principal "**main.C**".

```
#ifndef ADC_H_
#define ADC_H_

//------------------------------------------------- Funciones -------------------
void ADC_Init()// Función para la configuración de los registros del ADC
{
	ADMUX =  (0<<REFS1)|(1<<REFS0)|(0<<ADLAR);
	ADCSRA = (0<<ADEN) |(0<<ADSC) |(0<<ADATE)|(0<<ADIF) |(0<<ADIE)
|(0<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
	ADCSRB = (0<<ADTS2)|(0<<ADTS1)|(0<<ADTS0);
	DIDR0 =
(1<<ADC5D)|(1<<ADC4D)|(1<<ADC3D)|(1<<ADC2D)|(1<<ADC1D)|(1<<ADC0D);
}

void ADC_read(unsigned int channel)
{
	ADMUX &= 0b11110000; .
	ADMUX |= channel;
	ADCSRA |= (1<<ADEN)|(1<<ADSC);// Enciende el ADC e inicia la conversión.
	while(!(ADCSRA&(1<<ADIF)));
	ADCSRA &= ~(1<<ADIF);
	ADCSRA &= ~(1<<ADEN);	// Apagamos el ADC.
}
#endif
```

Código 21 – Librería "**ADC.H**".

3) Configurar el microcontrolador ATMEGA328P

Seguir los pasos de la práctica 8, para construir el proyecto.

4) Simulación del código.

Abrir "**Proteus Professiona**l" y dibujar el diagrama esquemático de la figura 73, seguir los pasos de la práctica 8, para realizar la simulación funcional del circuito:

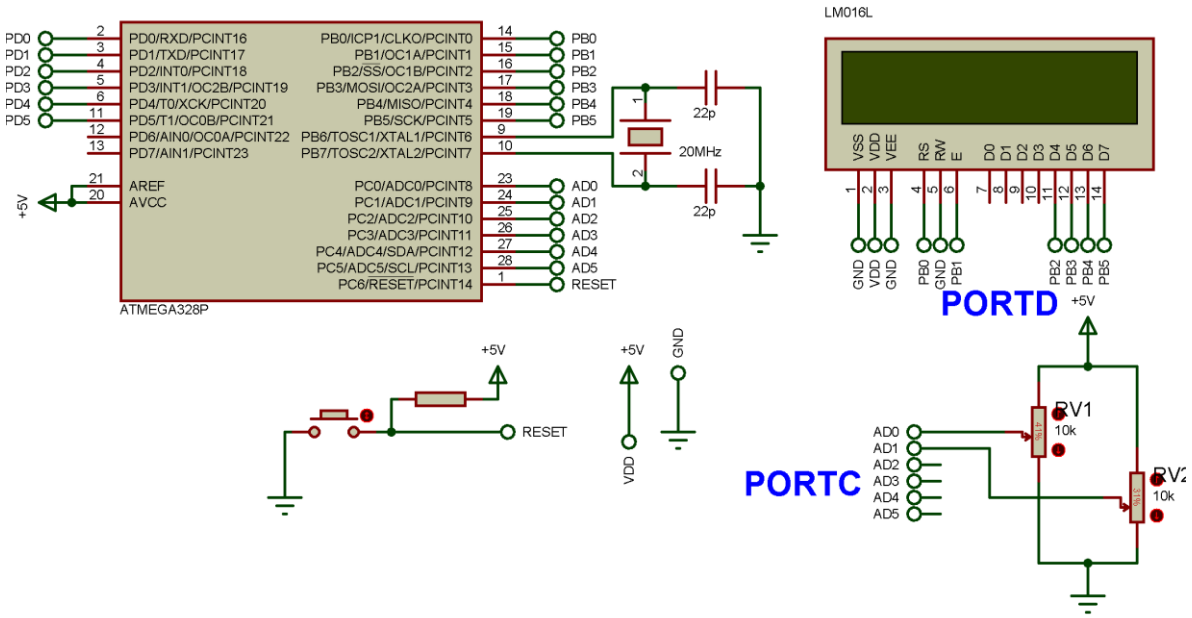

Figura 73 – Diagrama esquemático de práctica 13.

5) Programación del microcontrolador físico.

Seguir los pasos de la práctica 8, para realizar la programación del **ATMEGA328P**.

6) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 74.

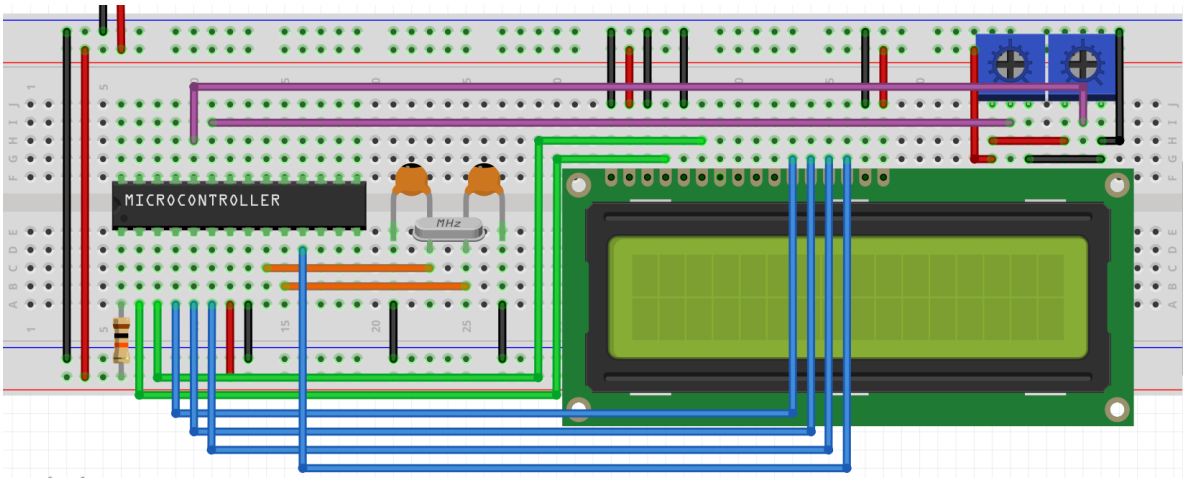

Figura 74 – Convertidor analógico a digital en **ATMEGA328P**.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje C en los **microcontroladores AVR**, aprender a usar el software **Atmel Studio**, y simular su funcionamiento en **Proteus Professional**, comparar los resultados teóricos, simulados y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Práctica 14

Microcontroladores ARM 32bits.

Carrera:	**Ingeniería Mecatrónica**
Nombre de la materia:	**Microcontroladores**

Código	Nombre completo de los alumnos

Fecha:	

Objetivos.

- Aprender a usar el compilador en línea **ARM Mbed**.
- Aprender a programar microcontroladores en **lenguaje C++**.
- Programación en microcontrolador físico.

Material y equipo.

- Computadora.
- Conexión a internet.
- 1 cable USB.
- 1 tarjeta de desarrollo con microcontrolador de la familia **Cortex-M** [10].
- Compilador en línea **ARM Mbed** [11].
- 1 acelerómetro **MMA8452Q** [12].

Trabajo Previo.

- Conocimientos vistos en clase: **Hoja de datos de Cortex-M y un MMA8452Q.**
- Conocimientos vistos en asignaturas previas: **Lenguaje C++**.
- Conocimientos vistos en asignaturas previas: Uso de software tipo **CAD**.
- Conocimientos vistos en asignaturas previas: Interpretación de **diagramas esquemáticos**.
- Conocimientos vistos en asignaturas previas: Circuitos eléctricos y uso de **protoboard** (**breadboard**).

Desarrollo.

1) Creación de un proyecto en lenguaje C++.

Ingresar a la página **https://os.mbed.com/** y elegir entre los tres compiladores disponibles (compilador en línea, compilador en IDE para computadora, o compilador por línea de comando), figura 75.

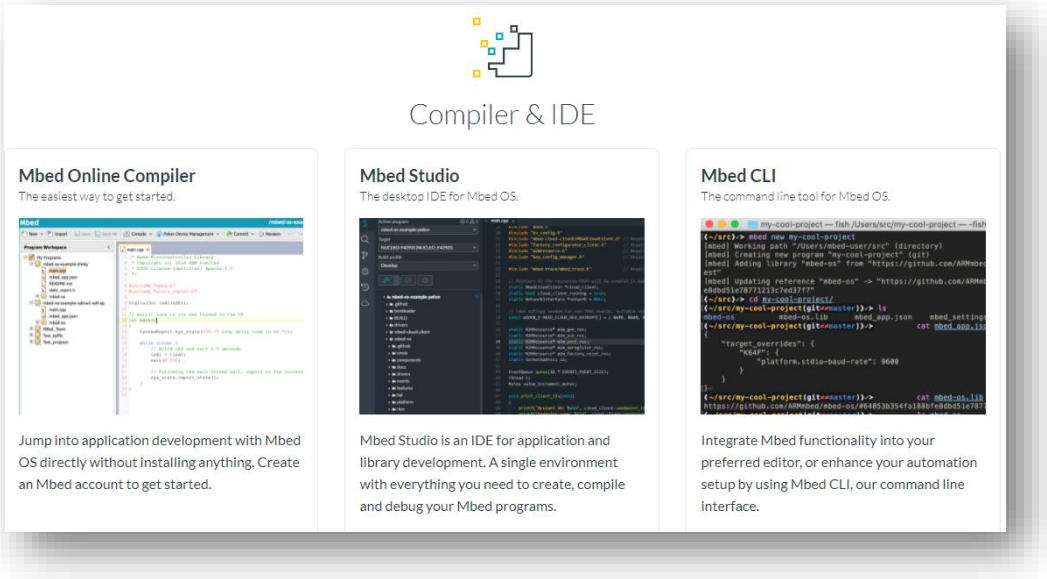

Figura 75 – Compiladores ARM Mbed.

En este ejemplo se usará el compilador Online, el alumno debe crear una cuenta para tener acceso al compilador en la nube, figura 76.

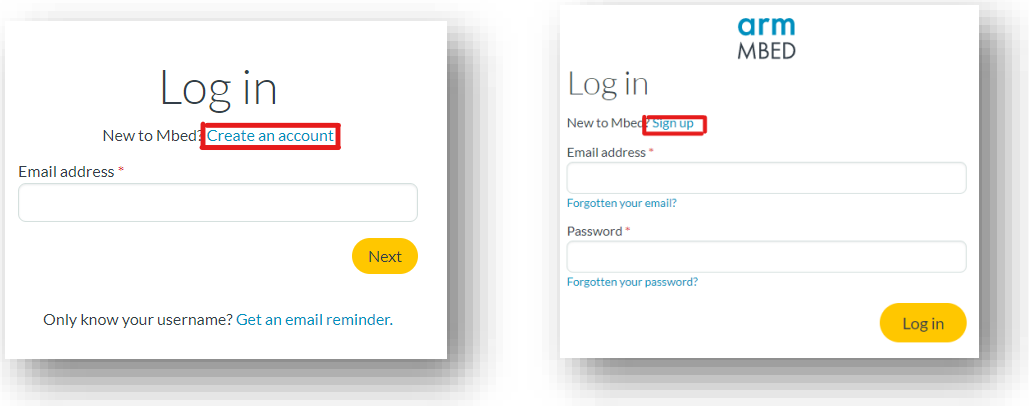

Figura 76 – Pantallas de acceso a registro.

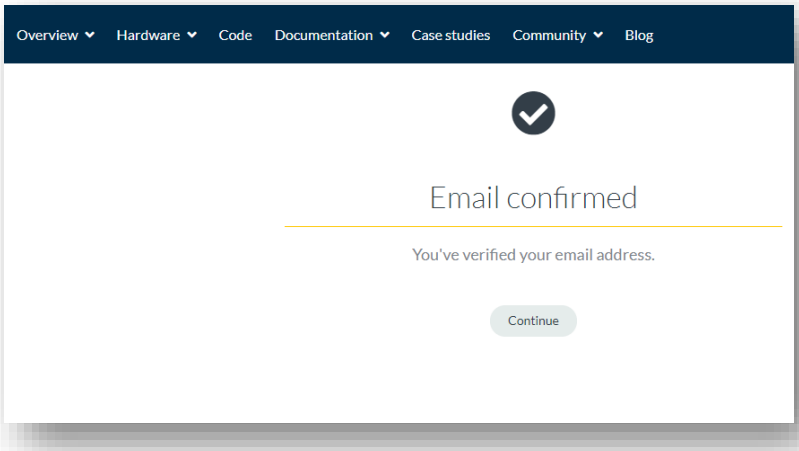

Figura 77 – Pantalla de verificación confirmada.

Una vez llenado el formato, se debe ingresar al correo con el cual se realizó el registro, validar la cuenta con el enlace que será enviado por el equipo de ARM. Después de validar la cuenta, se debe ingresar como usuario al compilador en línea, aparecerá una interfaz de usuario muy simple. Para crear el primer programa, se debe agregar una tarjeta de microcontrolador compatible con Mbed, para esto se debe seleccionar el botón "**No device selected**" , aparecerá la ventana "**Select Platform**", hacer click en el botón "**Add Board**", figuras 78 y 79.

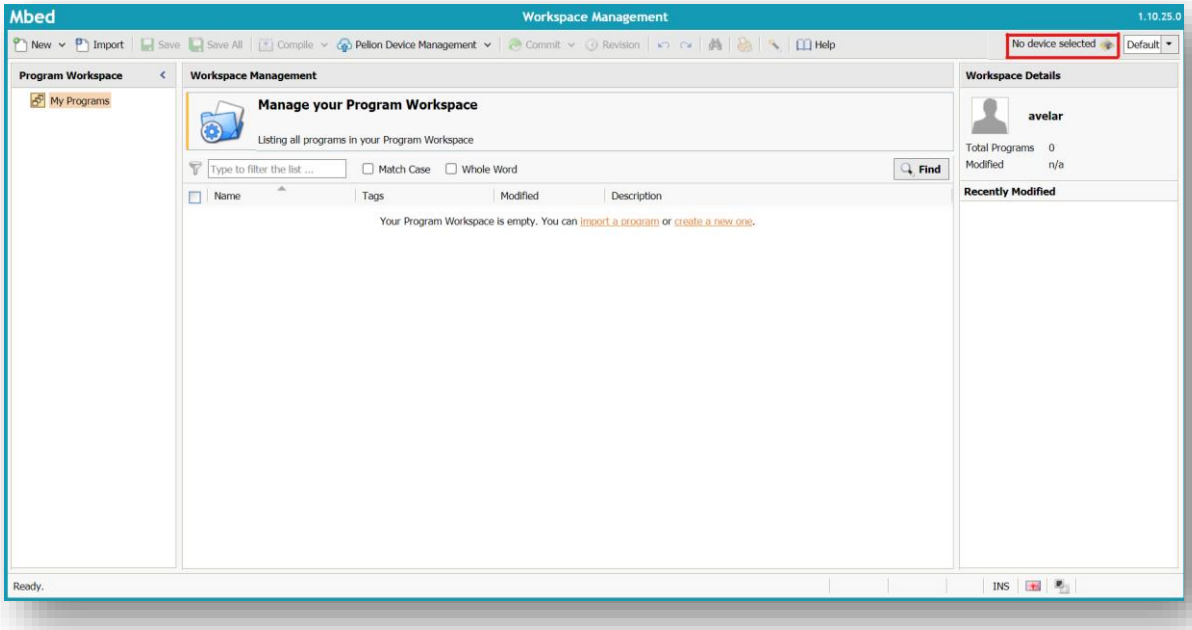

Figura 78 – Interfaz de programación en línea Mbed.

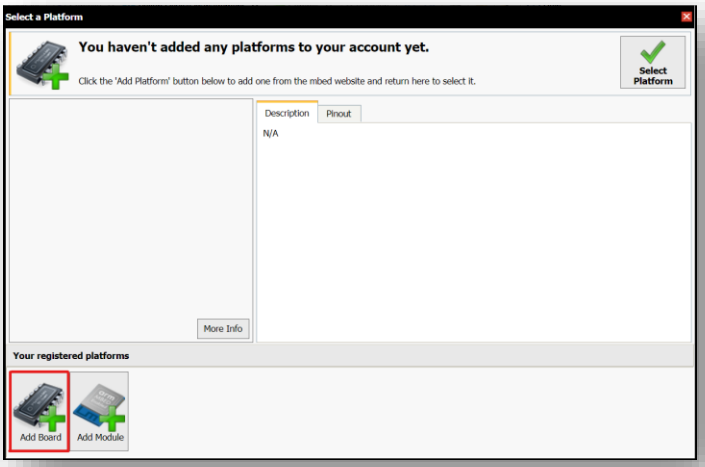

Figura 79 – Ventana “**Select Platform**”.

Con ayuda del asistente de busqueda, se debe escribir el modelo que se desea agregar al compilador, figura 80.

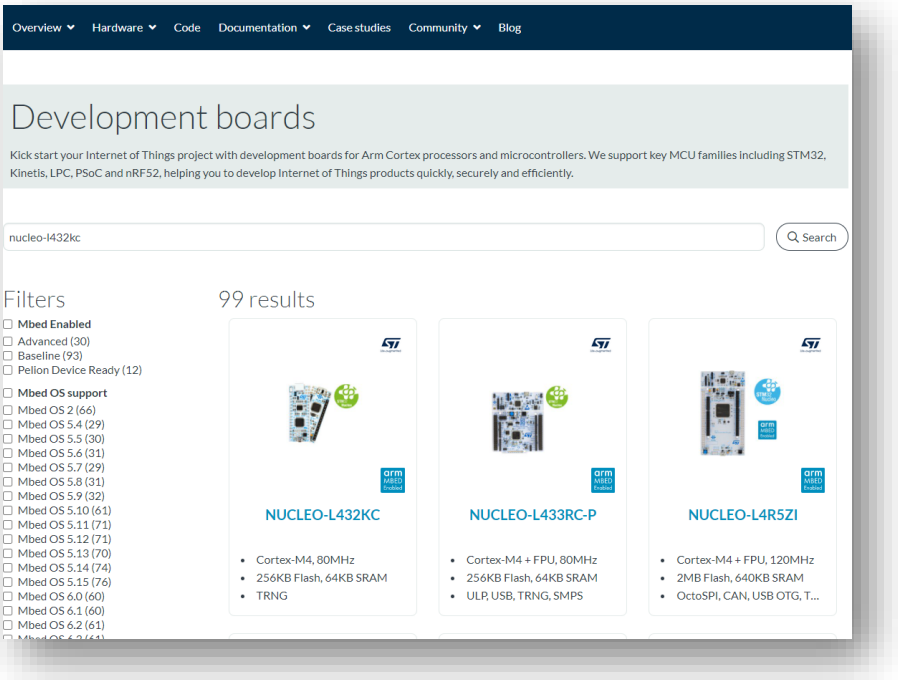

Figura 80 – Asistente de búsqueda para tarjetas de desarrollo en Mbed.

En este ejemplo se empleará la tarjeta NUCLEO-L432KC de la empresa ST, para agregar la tarjeta se debe seleccionar desde el resultado de la búsqueda, se abre la ventana con la información de la tarjeta y el botón para agregarla al compilador, figura 81 y 82.

Figura 81 – Botón para agregar tarjetas de desarrollo al compilador Mbed.

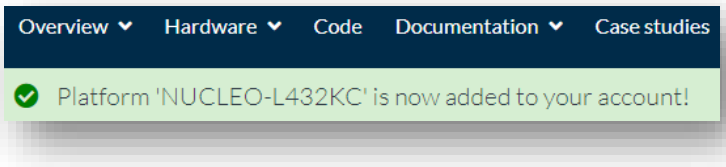

Figura 82 – Letrero informando que se agregó la tarjeta para programar.

2) Código en lenguaje C++.

Se debe crea un proyecto para poder escribir el código, se debe seleccionar el botón "**New**" >> "**New Program…**", aparecerá la ventana "**Create new program**", seleccionar plantilla vacía "**Empty Program**", seleccionar "**OK**", figura 83.

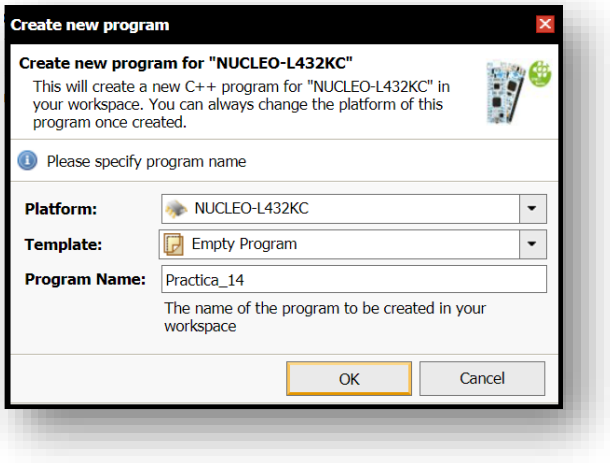

Figura 83 – Ventana para crear un nuevo programa.

Seleccionar el botón “**New**” >> “**New File**”, aparece una ventana para escribir el nombre principal del código, escribir “**Main.Cpp**” y presionar “**OK**”, click derecho al proyecto “**Practica_14**”, seleccionar “**Import Library**” >> “**From Import Wizard…**” figura 84 y 85.

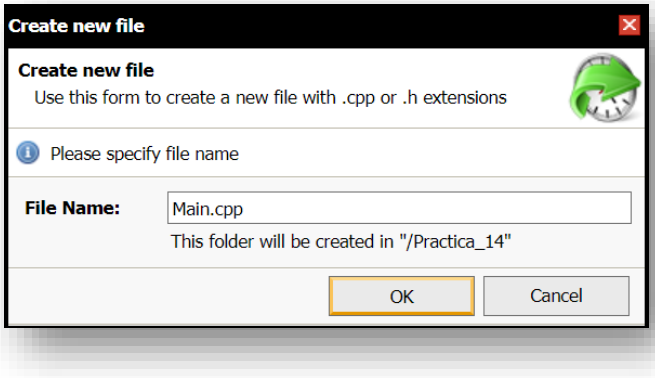

Figura 84 – Ventana para crear un nuevo programa.

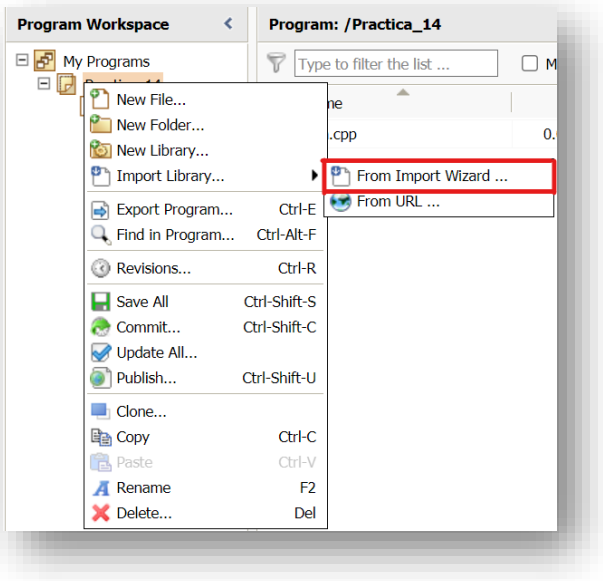

Figura 85 – Menú emergente para importar librerías del repositorio.

Aparece la ventana “**Import Wizard**”, seleccionar la pestaña “**Libraries**”, en la barra de búsqueda buscar la librería “**mbed**”, doble click a la librería creada por “**mbed official**” y la librería “**MMA8452**” figura 86 .

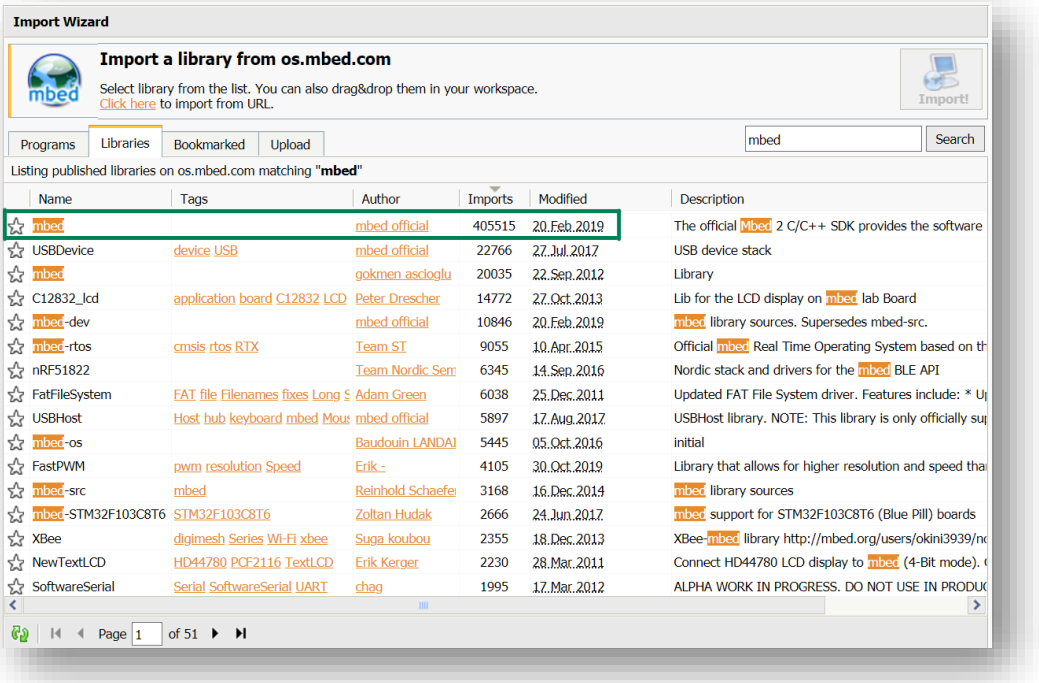

Figura 86 – Asistente para buscar e importar librerías del repositorio.

Escribir el siguiente código dentro del archivo “Main.cpp”:

```
/*
------------------------------------------------------------------------
 Práctica 14: Microcontroladores ARM 32bits
 Microcontrolador: STM32L432KCU6        Frecuencia: 80MHz   Voltaje:3.3V
 Herramienta: Compilador ARM Mbed - GCC C
------------------------------------------------------------------------
*/
#include "mbed.h"
#include "MMA8452.h"
MMA8452 mma8452(PA_10,PA_9);       // SDA, SCL
Serial serial(USBTX,USBRX);

int main()
{
    mma8452.init();
    Acceleration acceleration;
    while(1)
    {
      acceleration = mma8452.readValues();
      serial.printf("x= %.2f y= %.2f
z= %.2f \n",acceleration.x,acceleration.y,acceleration.z);
      wait(0.1);
    }
}
```

Código 22 – Archivo principal “**main.CPP**”.

3) Programación del microcontrolador físico.

Una vez terminado el código, se procede a compilar el proyecto con las herramientas de la figura 87.

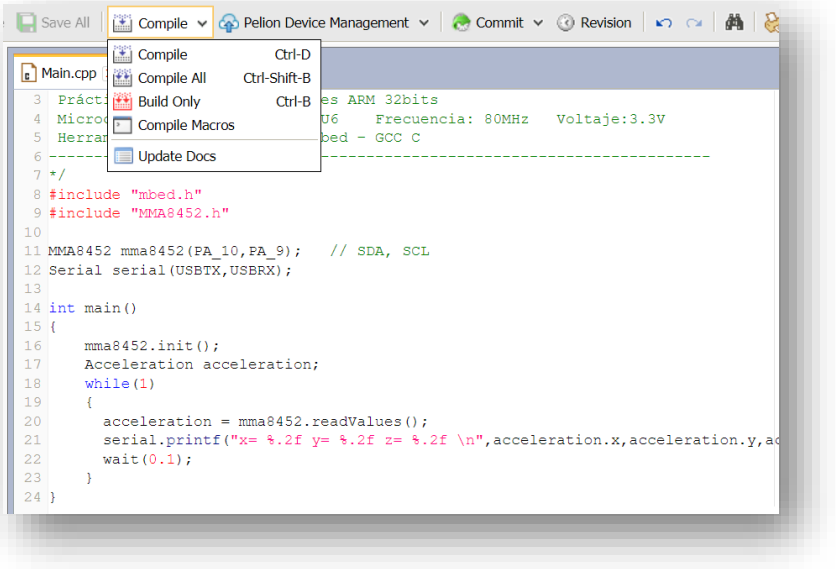

Figura 87 – Herramientas de compilación.

Seleccionar “**Compile**” y esperar a que los servidores terminen de construir el proyecto, una vez finalizada la compilación se descargará el archivo en código maquina con extensión “**.BIN**”, figura 88.

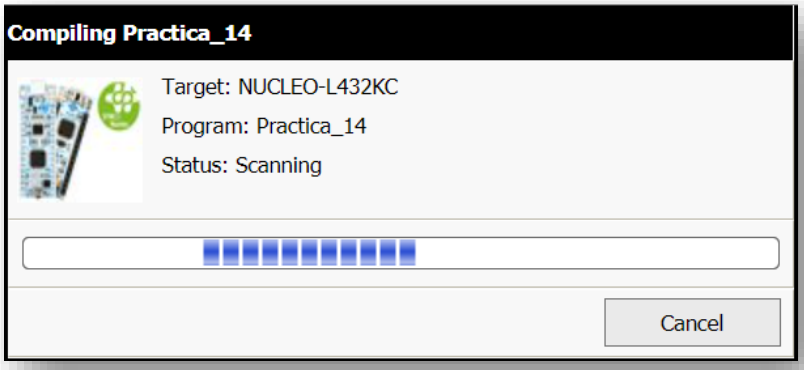

Figura 88 – Barra de progreso de compilación en línea.

Conectar la tarjeta de desarrollo a la computadora, la tarjeta será reconocida como un dispositivo de almacenamiento USB, para programar el microcontrolador solo se debe arrastrar el archivo a la memoria del microcontrolador, figura 89 y 90.

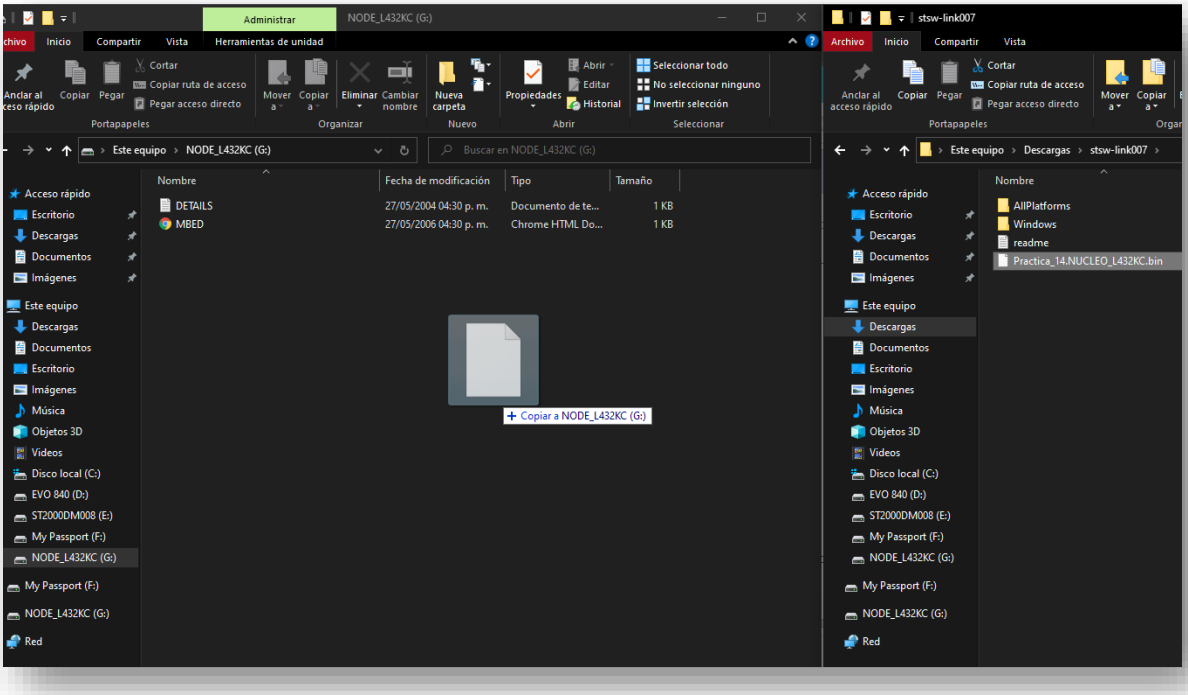

Figura 89 – Se programa igual que un archivo arrastrado a una memoría USB.

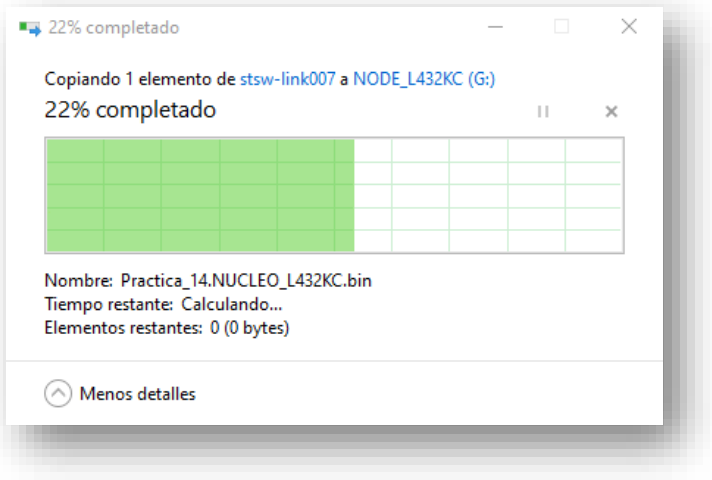

Figura 90 – Progreso de escritura en la memoria del microcontrolador.

4) Prueba de funcionamiento del microcontrolador físico.

Armar el circuito en un protoboard(breadboard), y probar su correcto funcionamiento, figura 91.

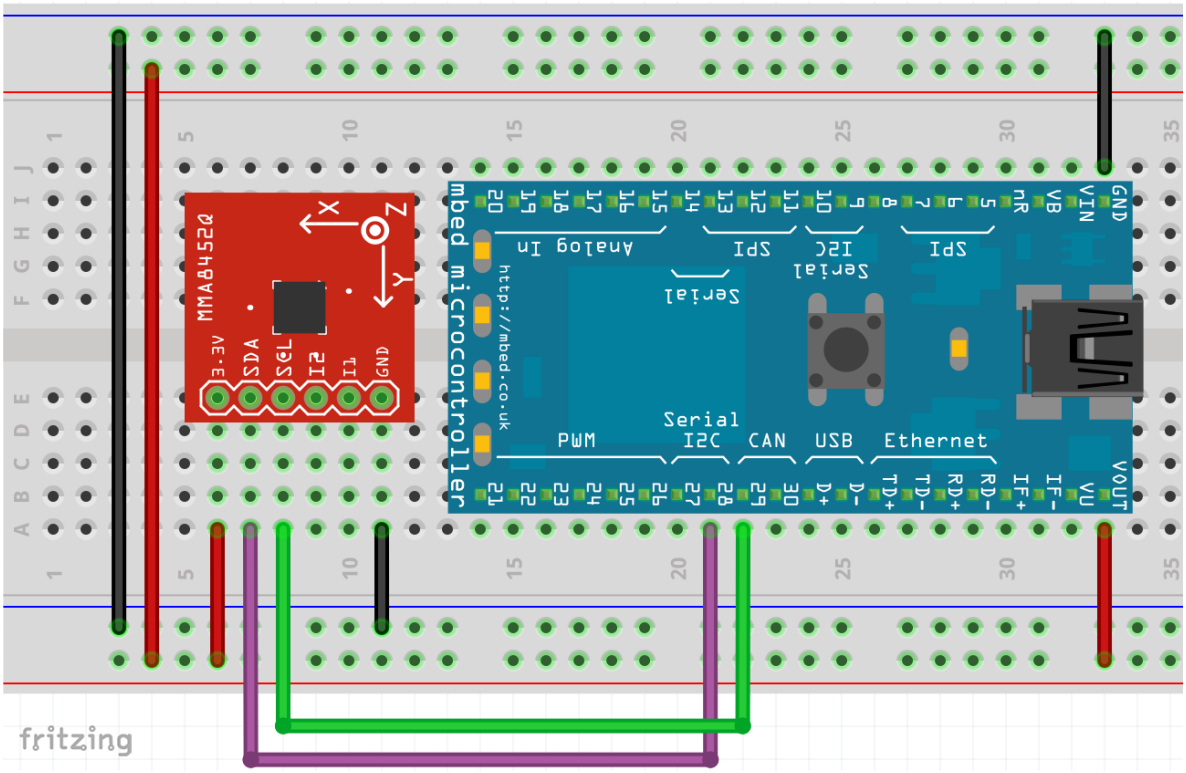

Figura 91 – Acelerómetro **MMA8452Q** conectado a **LPC1768**.

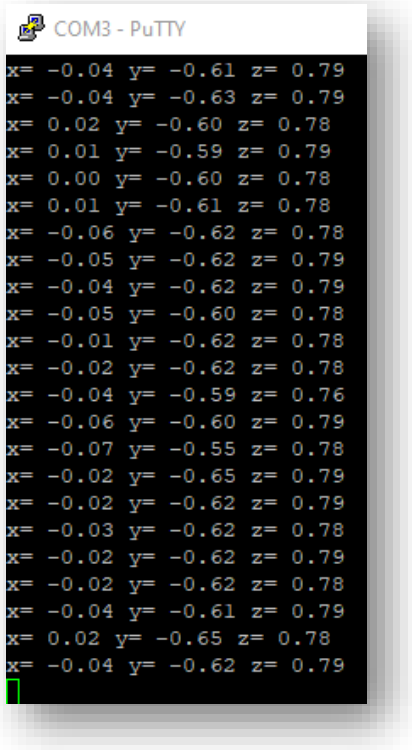

Figura 92 – Lecturas del acelerómetro, enviados por el puerto USART del microcontrolador.

Resultados y conclusiones.

El alumno debe implementar la programación en lenguaje C en los **microcontroladores ARM Cortex-M**, aprender a usar el compilador en línea **ARM Mbed**, comparar los resultados teóricos y experimentales obtenidos, con la finalidad de generar de carácter obligatorio un reporte de práctica con sus propias conclusiones, haciendo énfasis en los objetivos planteados al inicio de la práctica.

Referencias.

[1] https://www.microchip.com/wwwproducts/en/PIC16F84A

[2] https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

[3] https://www.labcenter.com/downloads/

[4] https://www.microchip.com/Developmenttools/ProductDetails/DV003001

[5] https://www.microchip.com/Developmenttools/ProductDetails/PG164130

[6] https://www.microchip.com/wwwproducts/en/ATmega328P

[7] https://www.microchip.com/content/dam/mchp/documents/parked-documents/as-installer-7.0.2542-full.exe

[8]https://www.microchip.com/DevelopmentTools/ProductDetails/PartNO/ATAVRDRAGON

[9] https://www.arduino.cc/en/main/products

[10] https://developer.arm.com/ip-products/processors/cortex-m

[11] https://os.mbed.com/accounts/login/

[12] https://www.sparkfun.com/products/12756

DIRECTORIO

UNIVERSIDAD DE GUADALAJARA

DR. RICARDO VILLANUEVA LOMELÍ
RECTOR GENERAL

DR. HÉCTOR RAÚL SOLÍS GADEA
VICERRECTOR EJECUTIVO

MTRO. GUILLERMO ARTURO GÓMEZ MATA
SECRETARIO GENERAL

CENTRO UNIVERSITARIO DE LA COSTA SUR

DRA. LILIA VICTORIA OLIVER SÁNCHEZ
RECTORA

DR. HIRINEO MARTÍNEZ BARRAGÁN
SECRETARIO ACADÉMICO

DR. LUIS CARLOS GÁMEZ ADAME
SECRETARIO ADMINISTRATIVO

MTRO. ENRIQUE JARDEL PELÁEZ
DIRECTOR DE LA DIVISIÓN DE DESARROLLO REGIONAL

DR. DANIEL EDÉN RAMÍREZ ARREOLA
JEFE DEL DEPARTAMENTO DE INGENIERÍAS

Av. Independencia Nacional No. 151, Autlán de Navarro, Jalisco, C.P. 48900
Tel. (317) 382 5010 www.cucsur.udg.mx

Centro Universitario de la Costa Sur CU Costa Sur UdeG @CUCSur CU Costa Sur @cucostasur