UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ECATEPEC

DESARROLLO DE TARJETA ENTRENADORA PARA PIC16F628

TESIS

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

C. DAVID CORTES BAUTISTA

ASESOR ACADÉMICO:

Dr. en C. RODOLFO ZOLA GARCÍA LOZANO

REVISORES:

M. En C.C. JOSÉ ENRIQUE TINAJERO PÉREZ

Dr. En C. JUAN DE JESÚS AMADOR REYES

ECATEPEC DE MORELOS, ESTADO DE MÉXICO, NOVIEMBRE 2024

Contenido

Introducción

Hoy en día, los dispositivos programables, como los microcontroladores, están presentes en una amplia variedad de aplicaciones. Los encontramos en automóviles, electrodomésticos y, más comúnmente, en dispositivos portátiles como los teléfonos inteligentes. En las últimas décadas, el uso de esta tecnología ha crecido de manera significativa.

Un microcontrolador es un pequeño dispositivo electrónico que puede ser programado para ejecutar un conjunto de instrucciones, conocido como programa fuente, con el objetivo de resolver problemas específicos. Estas instrucciones permiten al microcontrolador realizar tareas diversas, dependiendo de la aplicación para la que se haya diseñado.

Existen múltiples variantes de microcontroladores, adaptadas a diferentes usos: desde la enseñanza escolar hasta la industria, la tecnología militar, e incluso la exploración espacial. Independientemente de la aplicación, todos los microcontroladores tienen en común la capacidad de recibir datos del entorno, procesarlos y tomar decisiones basadas en esos datos para obtener un resultado esperado. Estos dispositivos pueden ser programados en lenguajes de alto nivel, como C, o en lenguajes de bajo nivel, como ensamblador. Aunque los lenguajes de alto nivel facilitan la comprensión y el desarrollo de aplicaciones, la programación en lenguaje de bajo nivel proporciona una visión más profunda sobre cómo los sistemas digitales almacenan, procesan y transmiten información.

Este documento tiene como objetivo reunir la información técnica más relevante sobre la programación del microcontrolador PIC16F628. A lo largo del texto, se abordarán aspectos clave relacionados con la programación en lenguaje ensamblador, el entorno de desarrollo necesario, la metodología para implementar proyectos, y se presentará una propuesta de tarjeta de entrenamiento, diseñada para facilitar un aprendizaje más eficiente en cursos de nivel introductorio.

Objetivo general

Desarrollar un documento de referencia del PIC16F628 que permita a las y los estudiantes entender los principios fundamentales de programación en lenguaje ensamblador e iniciar en el mundo de la programación de estos microcontroladores.

Objetivos particulares

En el documento se presentarán:

- Las bases fundamentales de la arquitectura y prestaciones del PIC16F628.
- Se explicará a detalle el proceso de instalación del entorno de desarrollo y el compilador.
- Se describirán los programas necesarios para introducir a las y los lectores a la programación de PIC's en lenguaje ensamblador.
- Se diseñará y presentarán módulos de prueba básicos y de nivel más avanzado, los cuales podrán ser utilizados en los diferentes proyectos para un análisis más sencillo.
    - Módulo de entrada-salida:
        - Ocho entradas binarias con interruptores deslizables.
        - Ocho entradas binarias con interruptores de presión normalmente abiertos.
        - Ocho salidas conectadas a LED's.
    - Módulo display de 7 segmentos:
        - Ocho entradas binarias
    - Módulo pantalla LCD 16x2
        - Entradas de alimentación con potenciómetro
        - Tres pines de alimentación
        - Cuatro pines de control
        - Ocho pines de bus de datos
    - Módulo teclado matricial
        - Teclado matricial con dieciséis push button.
        - Teclado configurado con cuatro entradas y cuatro salidas hacia el microcontrolador.
    - Módulo de reset para EEPROM
        - Configuración de pin MCLRE en modo pull up.

- La alimentación de los módulos se realizará a través de la conexión a la alimentación a 5V.
- Se describirá como descargar e instalar software de grabado para el PIC, así mismo cómo cargar un programa con la extensión adecuada para el microcontrolador.
- Se propondrá una tarjeta PCB entrenadora para el PIC16F628.

1 Microcontroladores PIC.

Los microcontroladores son circuitos integrados programables, que contiene en su estructura: una CPU (Unidad de Procesamiento Central), memoria RAM (Memoria de Acceso Aleatorio), circuitos de entradas y salidas y otros módulos que permiten el desarrollo y programación de aplicaciones específicas [1, p. 3].

Es importante resaltar que el microprocesador es un dispositivo diferente al microcontrolador. El microprocesador es el CPU en sí, es decir el módulo que se encarga del procesamiento aritmético y lógico de la información. Para que el microprocesador pueda ser utilizado y programado es necesaria la utilización de otros módulos externos, de control y almacenamiento, para que pueda funcionar. En cambio, al interior del microcontrolador ya se cuenta con un microprocesador y los módulos necesarios para que este circuito integrado pueda ser utilizado en algún circuito de aplicación.

Utilización

Los microcontroladores se encuentran en muchas aplicaciones electrónicas de la vida cotidiana. En la Fig. 1.1 se presentan algunos dispositivos donde se utilizan los microcontroladores para el desarrollo de ciertas funciones programadas. Por ejemplo, en la cocina de la mayoría de los hogares se utilizan en los hornos de microondas. En estos aparatos el microcontrolador es el encargado de controlar muchas de las funciones, como realizar el registro del tiempo de calentamiento que el usuario programa, hacer sonar la alarma cuando este llegue a cero o los programas de calentamiento con los que cuenta el dispositivo, por ejemplo, el botón de las palomitas.

Los ordenadores, los celulares, calculadoras, laptops, relojes son algunos otros ejemplos de dispositivos en donde son utilizados los microcontroladores para la implementación de diferentes funciones.

La utilización de estos dispositivos en la industria se presenta en la automatización de procesos. Esta aplicación permite el ahorro de costos excesivos de automatización, con la consecuente optimización de los recursos de la empresa. Los microcontroladores de uso industrial pueden ser utilizados en actividades de vibración como ejemplo o también para el control de temperatura para la longevidad de una máquina, etc. [5]. Una ventaja importante del uso de los microcontroladores en la industria es su alta inmunidad al ruido eléctrico, así como sus reducidas dimensiones.

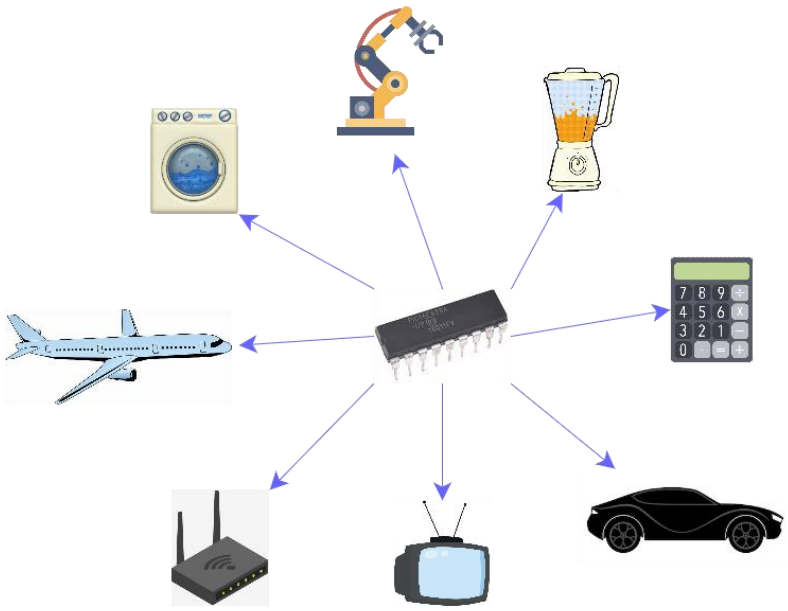


Fig. 1.1 Aplicaciones de los microcontroladores.

Tipos de Microcontroladores

Existen diferentes tipos de microcontroladores, con muy diversas características y capacidades. En función del tamaño del bus de la memoria de datos, los microcontroladores comúnmente se clasifican en microcontroladores de 8 bits, 16 bits y 32 bits. Estos a su vez pueden ser subdivididos en gamas y cada gama tiene una familia diferente. En la Fig. 1.2 se presenta una clasificación general de los microcontroladores PIC. A continuación, se describen las principales características de los microcontroladores PIC, clasificados en función del bus de la memoria de datos.

PIC de 8 Bits.

Microcontroladores simples, flexibles y de baja potencia. Se basan en acumuladores, aunque algunas arquitecturas más modernas tienen conjuntos de registros. Alcanza sus límites cuando se enfrenta a redes y comunicaciones, pero en la industria automovilística son lo mejor por el bajo consumo de energía, es por eso por lo que también se integran en los electrodomésticos, el ahorro de energía, pues alcanza a consumir 100nA cuando entra en modo sleep [9].

PIC de 16 Bits.

Tienen la capacidad para almacenar 3072 palabras lógicas en su memoria, estos PIC tienden a tener buen rendimiento de baja potencia, suelen tener coprocesadores matemáticos para una aplicación avanzada si así se desea. Tienen 16 bits de direccionamiento que solo pueden acceder a 64KB, sin embargo, existen arquitecturas que tienen 24 bits de direccionamiento extendido lo que le permite al PIC almacenar 16Mb de espacio de direcciones [8].

PIC de 32 Bits.

Tienen un bus de direccionamiento de 32 bits que proporciona acceso de hasta 4 GB de memoria. Son muy utilizados en rangos de rendimiento, fácilmente admiten redes y comunicaciones, la velocidad de reloj alcanza los Giga Hertz. Admite operaciones matemáticas avanzadas como el álgebra y punto flotante [8].

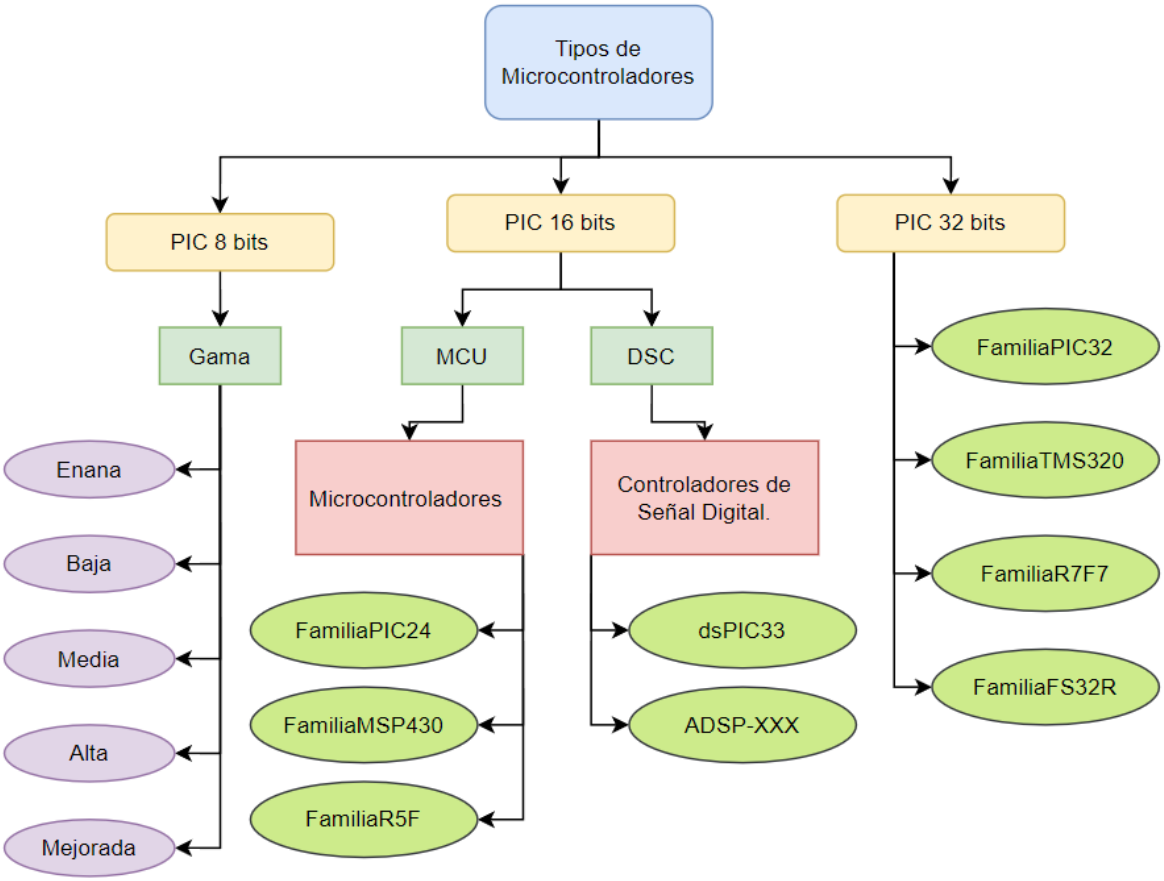


Fig. 1.2 Clasificación general de los microcontroladores.

Fabricantes de microcontroladores

Existen diferentes compañías que se dedican al diseño y fabricación de diferentes microcontroladores, a continuación, se describen algunas de las empresas más importantes.

Microchip Technology Inc.

Los catálogos son grandes y extensos, pero entre los microcontroladores más innovadores están los AVR (MCU) de 8 bits, los controladores de señal digital (DSC) dsPIC de 16 bits, entre otros.

Microchip tiene diferentes gamas y familias de microcontroladores, en la tabla 1.1 se muestran las características principales de las gamas de microcontroladores, así como algunos ejemplos de estas y sus principales características.

Tabla 1.1 Gamas y familias de microcontroladores de Microchip.

Gama	Familia	Ejemplos	Características
Enana	PIC12CXXX	PIC12CE673 PIC12CR509 PIC12C672 PIC12CE518	La principal característica es que son muy pequeños, con encapsulados de 8 pines y con juegos de 33 instrucciones de 12 bits [3].
Baja	PIC16C5XXX	PIC16C56A PIC16C57 PIC16C57C PIC16C54 PIC16C55 PIC16C505	Se compone de microcontroladores con 33 instrucciones de 12 bits. Sus versiones están encapsuladas terminales que van de 18 a 28 pines [2]. La memoria del programa es de 512 palabras. La memoria de datos está comprendida entre 25 y 73 bytes. No permiten interrupciones [3].
Media	PIC16CXXX	PIC16F628 PIC16C662 PIC16C781 PIC16CR77 PIC16F84	Las versiones de esta gama están encapsuladas con terminales que van de 18 a 68 pines. Disponen de interrupciones y una pila de 8 niveles que permite el anidamiento de subrutinas [2]. Tienen un conjunto de 35 instrucciones de 14 bits de ancho [3].
Alta	PIC17CXXX	PIC17C752 PIC17C44 PIC17FR44 PIC17C756	Alcanzan las 58 instrucciones de 16 bits. Cuentan con comunicación serial y paralelo. Lo más destacable de esta serie de PIC´s es la ampliación del microcontrolador con elementos externos [2].

Nomenclatura de los PIC.

Los PIC tienen letras y números como identificadores para saber la familia, el modelo y su alimentación [4, p. 35]. La identificación de los dispositivos se realiza utilizando el siguiente formato:

PIC nnLLLxxx

donde:
nn = Número propio de la gama del PIC
LLL = Código de letras donde la primera indica la tensión de alimentación y las otras dos el tipo de memoria que utiliza.
xxx = Número que indica el modelo del PIC.

Por ejemplo, el PIC16F628, significa:

PIC16 = Gama media
F = Alimentación estándar (Sufijo)
628 = Modelo del PIC.

En la tabla 1.2 se presentan los rangos de alimentación y el correspondiente sufijo de identificación.

Tabla 1.2 Rangos de alimentación del PIC de acuerdo con el sufijo

Letras (Sufijo)	Alimentación
C	Estándar (4.5-6.0V)
CR	Estándar (4.5-6.0V)
F	Estándar (4.5-6.0V)
LC	Extendida (2.5-6.0V)
LCR	Extendida (2.5-6.0V)
LF	Extendida (2.5-6.0V)

Texas Instruments.

Es una empresa dedicada al diseño, fabricación, prueba y venta de semiconductores analógicos e integrados en mercados industriales, automotriz, electrónica personal, equipos de comunicaciones y sistemas empresariales. Actualmente esta empresa ya no diseña microcontroladores de 8 bits. En esta sección solo se describirán las características de los microcontroladores (µM) de 16 y 32 bits.

Los procesadores de señales mixtas (Mixed Signal Processor) MSP430 son una familia de microcontroladores de 16 bits que operan en un rango de voltaje de 1.8 a 3.6V, con velocidades de operación de hasta 25MHz. La memoria del programa varía desde 512 bytes hasta 256 Kb. Están diseñados para aplicaciones embebidas de bajo costo y baja potencia [10].

La identificación de los microcontroladores de esta fábrica se realiza de la siguiente forma:

MSP430F2618ATZQWT-EP

donde:

MSP430 = Familia de microcontroladores

F = memoria flash.
2 = Generación del dispositivo.
6 = Modelo dentro de la generación.
18 = Cantidad de memoria del dispositivo (es un sufijo)
A = Revisión
T = Rango de temperatura
ZQW = Encapsulado, en este caso Ball grid array.
T = Entregado en carrete pequeño
-EP = Características adicionales

En la tabla 1.3 se dará a conocer la cantidad de almacenamiento en RAM y ROM que tiene cada microcontrolador de Texas Instruments, sabiendo el sufijo que lo acompaña.

Tabla 1.3 Identificación de almacenamiento en RAM y ROM de los MCU.

Sufijo	RAM	ROM
0	128	1K
1	128	2K
2	256	4K
3	256	8K
4	512	12K
5	512	16K
6	1K	24K
7	1K	32K
8	2K	48K
9	2K	60K
10	5K	32K
11	10K	48K
12	5K	55/56K
16	4K	92K
17	8K	92K
18	8K	116K
19	4K	120K

Los microcontroladores de 32 bits tienen a la familia C2000 con periféricos integrados de alto rendimiento diseñados para aplicaciones de control en tiempo real. Consta de 5 subfamilias.

- C28x + ARM Cortex M3.
- C28x Delfino de punto flotante.
- C28x Piccolo.
- C28c de punto fijo
- C240x

La serie C2000 se destaca por su conjunto de periféricos de control en chip de alto rendimiento, incluidos, PWM, ADC, módulos codificadores de cuadratura y módulos de captura. Se utilizan para aplicaciones de control y accionamiento de motores, automatización industrial, energías renovables, granjas de servidores, energía digital, comunicaciones por línea eléctrica e iluminación.

Los fabricantes han demostrado que el microcontrolador es capaz de aplicarse en diferentes ramas de la tecnología, son flexibles para cualquier entorno que se desee trabajar, existen infinidad de MCU (Microcontroladores) adaptables ante cualquier necesidad que surja, se pueden modelar y programar de tal forma que cumpla una tarea específica según el usuario.

En el presente documento se seleccionó el uso del del PIC16F628 debido a que cuenta con características que permiten el desarrollo de una gran variedad de aplicaciones. El tamaño de este dispositivo lo hace un candidato muy adecuado para el aprendizaje de la programación de microcontroladores, al grado que actualmente está desplazando al PIC16F84A en el sector académico.

2 Características del PIC16F628

Algunas de las principales características del PIC16F628 se listan a continuación:

- El rango de voltaje de funcionamiento del PIC es entre 3.0 y 5.5V.
- Memoria de programa de 2Kb
- Memoria RAM 224B
- Memoria EEPROM 128B
- Temporizador de vigilancia (WDT).
- Detección de caída de tensión.
- Cuenta con un oscilador interno de dos velocidades, una de 37 KHz y otra de 4 MHz[6].
- Si el PIC se alimenta con 3V y se requiere una salida digital, entonces en la salida se obtendrán los 3V, esto es, en la salida de los pines programados siempre darán como resultado el voltaje que se suministre de entrada.
- CPU RISC de alto rendimiento.
- Cada pin del Microcontrolador que se requiera configurar como entrada debe tener el voltaje igual al que se está suministrando, esto es, por cada pin de entrada, cada pin puede admitir desde los 3 a los 5.5V.
- Pila de 8 niveles de profundidad.
- Dos comparadores analógicos.
- Ahorro de energía en modo SLEEP. La corriente consumida en este estado es de 1µA
- Timer0: Temporizador/contador de 8 bits con pre escalador programable.
- Timer1: Temporizador/contador de 16 bits con capacidad de cristal/reloj externo.
- Timer2: temporizador/contador de 8 bits con registro de período de 8 bits, pre escalador y pos escalador.
- Módulo de captura, comparación, PWM (CCP).
- Cada pin que se configure como salida, es capaz de entregar 25mA (corriente suministrada por terminal).
- Comunicación AUSART [6].
- Cuenta con un set de 35 instrucciones [6].
- Está disponible el Master Clear (MCLR) programable [6].

Diagrama de pines.

En la Fig. 2.1 se muestra el diagrama de pines del PIC16F628, así como el nombre de cada pin y para qué puede ser utilizado.

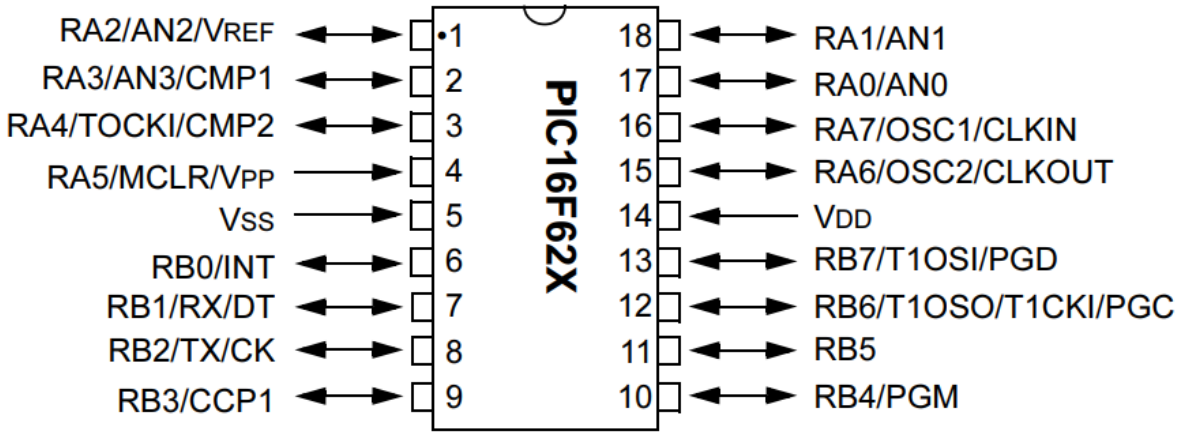


Fig. 2.1 Diagrama de pines del MCU (Microcontrolador) PIC16F628

Hay que recordar que el microcontrolador de forma física no dice los números de pin, pero se pueden identificar con un punto a un lado de la muesca como se puede mostrar en la fig. 2.2. Esto serviría para saber la posición del componente. El Pin uno comenzaría a la izquierda desde el punto identificador y este punto identificador se encuentra a la izquierda de la muesca.

Fig. 2.2 Identificación de pines del MCU.

Es muy importante identificar qué pines solo pueden ser de entrada y que otros nada más de salida, o si pueden ser ambos. Más adelante se explicará de forma detallada.

Descripción de la distribución de pines

El PIC16F628 cuenta con 18 pines y cada pin tiene una función en específico, esto dependerá de cómo se trabaja con el CI (Circuito integrado). En la Tabla 2.1 se describen las funciones que tiene cada pin, así como su tipo de entrada y salida.

Tabla 2.1. Descripción de las funciones de los pines del PIC16F628.

PIN	Nombre	Función	Tipo de entrada	Tipo de salida	Descripción
1	RA2/AN2/VREF	RA2	ST	CMOS	I/O Puerto Bidireccional
		AN2	AN	-	Entrada de Comparador logico
		VREF	-	AN	Salida VREF
2	RA3/AN3/CMP1	RA3	ST	CMOS	I/O Puerto Bidireccional
		AN3	AN	-	Entrada de Comparador logico
		CMP1	-	CMOS	Salida de comparador 1
3	RA4 /TOCKI/ CMP2	RA4	ST	OD	I/O Puerto Bidireccional
		TOCKI	ST	-	Entrada de reloj Timer0
		CMP2	-	OD	Salida de comparador 2
4	RA5 /MCLR/ VPP	RA5	ST	-	Puerto de entrada
		MCLR	ST	-	Master clear
		VPP	-	-	Entrada de tensión de programación. Cuando se configura como master clear, este pin es activado como RESET y es con una señal digital baja. El voltaje en MCLR/VPP no debe exceder el voltaje de VDD durante la operación.
5	VSS	VSS	Energía	-	Referencia a tierra para pines lógicos y de I/O
6	RB0/INT	RB0	TTL	CMOS	I/O Puerto Bidireccional. También se pude programa mediante software para pull-up interno débil.
		INT	ST	-	Interrupción externa
7	RB1/RX/ DT	RB1	TTL	CMOS	I/O Puerto Biddireccional.
		RX	ST	-	Pin de recepción USART
		DT	ST	CMOS	I/O de datos síncronos
8	RB2/TX/ CK	RB2	TTL	CMOS	I/O Puerto Bidireccional
		TX	-	CMOS	Pin de transmición USART
		CK	ST	CMOS	I/O reloj síncrono. Puede ser programado mediante software para pull-up interno débil
9	RB3/CCP1	RB3	TTL	CMOS	I/O Puerto Bidireccional. También se pude programa mediante software para pull-up interno débil.
		CCP1	ST	CMOS	Captura y Compara datos de PWM I/O.

10	RA1/AN1	RA1	ST	CMOS	I/O Puerto Bidireccional.
		AN1	AN	-	Entrada de Comparador logico
11	RA0/AN0	RA0	ST	CMOS	I/O Puerto Bidireccional.
		AN0	AN	-	Entrada de Comparador logico
12	RA7/OSC1/CLKIN	RA7	ST	CMOS	I/O Puerto Bidireccional
		OSC1	XTAL	-	Entrada del oscilador de crystal
		CLKIN	ST	-	Entrada de fuente externa de reloj
13	RA6/OSC2/CLKOUT	RA6	ST	CMOS	I/O Puerto Bidireccional
		OSC2	XTAL	-	Salida del oscilador de crystal. Se conecta a un cristal o resonador en modo crystal oscilador.
		CLKOUT	-	CMOS	En modo ER/INTR el pin OSC2 puede generar CLKOUT, que tiene 1/4 de la frecuencia del OSC1
14	VDD	VDD		Energía	Suministro de energía positivapara pines I/O
15	RB7/ T1OSI/ PGD	RB7	TTL	CMOS	I/O Puerto Bidireccional. También se pude programa mediante software para pull-up interno débil.
		T1OSI	XTAL	-	Entrada del oscilador Timer1. Despertar del modo SLEEP al cambiar el pin. Se puede progrmar mediante software para pull-up interno débil
		PGD	ST	CMOS	I/O puerto de entrada ICSP
16	RB6/ T1OSO/ T1CKI/ PGC	RB6	TTL	CMOS	I/O Puerto Bidireccional. También se pude programa mediante software para pull-up interno débil.
		T1OSO	-	XTAL	Salida del Oscilador Timer 1
		T1CKI	ST	-	Entrada del relor Timer1
		PGC	ST	-	Reloj de prohramación ICSP
17	RB5	RB5	TTL	CMOS	I/O Puerto Bidireccional. También se pude programa mediante software para pull-up interno débil.
18	RB4/PGM	RB4	TTL	CMOS	I/O Puerto Bidireccional. También se pude programa mediante software para pull-up interno débil.
		PGM	ST	-	Pin de entrada de progrmación de bajo voltaje. Interrumpe el cambio de pin. Cuando la programación de bajo voltaje está habilita, el cambio de intrrupción en el pin y la resistencia pull-up débil están deshabilitados.

Arquitectura

El PIC16F628 posee una arquitectura Harvard. Cabe destacar que existió una arquitectura de Von Neumann en microcontroladores pasados, pero ¿Qué diferencia hay entre la arquitectura Neumann y Harvard? Fig. 2.3.

La arquitectura Von Neumann se caracteriza por disponer de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. Para acceder a esta memoria se hace a través de un solo sistema de buses que pueden ser direcciones, datos y control [11].

La arquitectura Harvard accede a la memoria de programa y a la memoria de datos simultáneamente, esto hace una buena organización de la memoria del sistema porque las memorias son independientes y estas a su vez de tamaños y longitudes de palabras diferentes [11].

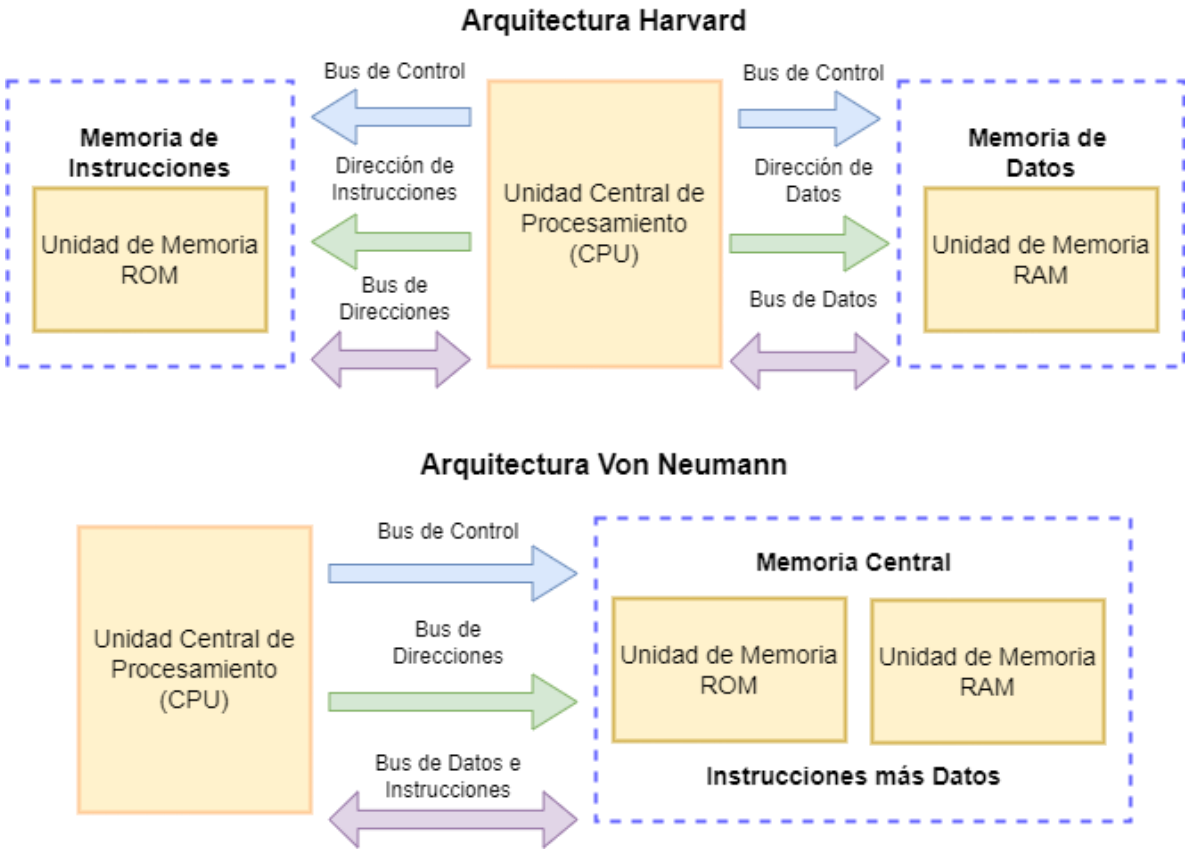


Fig. 2.3 Diferencias entre arquitectura Von Neumann y Harvard.

Diagrama de Bloques

En la Fig. 2.4 se muestran los bloques que contiene el microcontrolador, estos bloques representan la forma en que está consolidado. De manera gráfica se puede entender el comportamiento de los datos o información dentro del PIC ya que se muestra por qué registros o porque tipo de memoria pasa está información y el ancho de bus que tiene cada canal de comunicación entre estos bloques, de igual manera si es direccionamiento directo o indirecto.

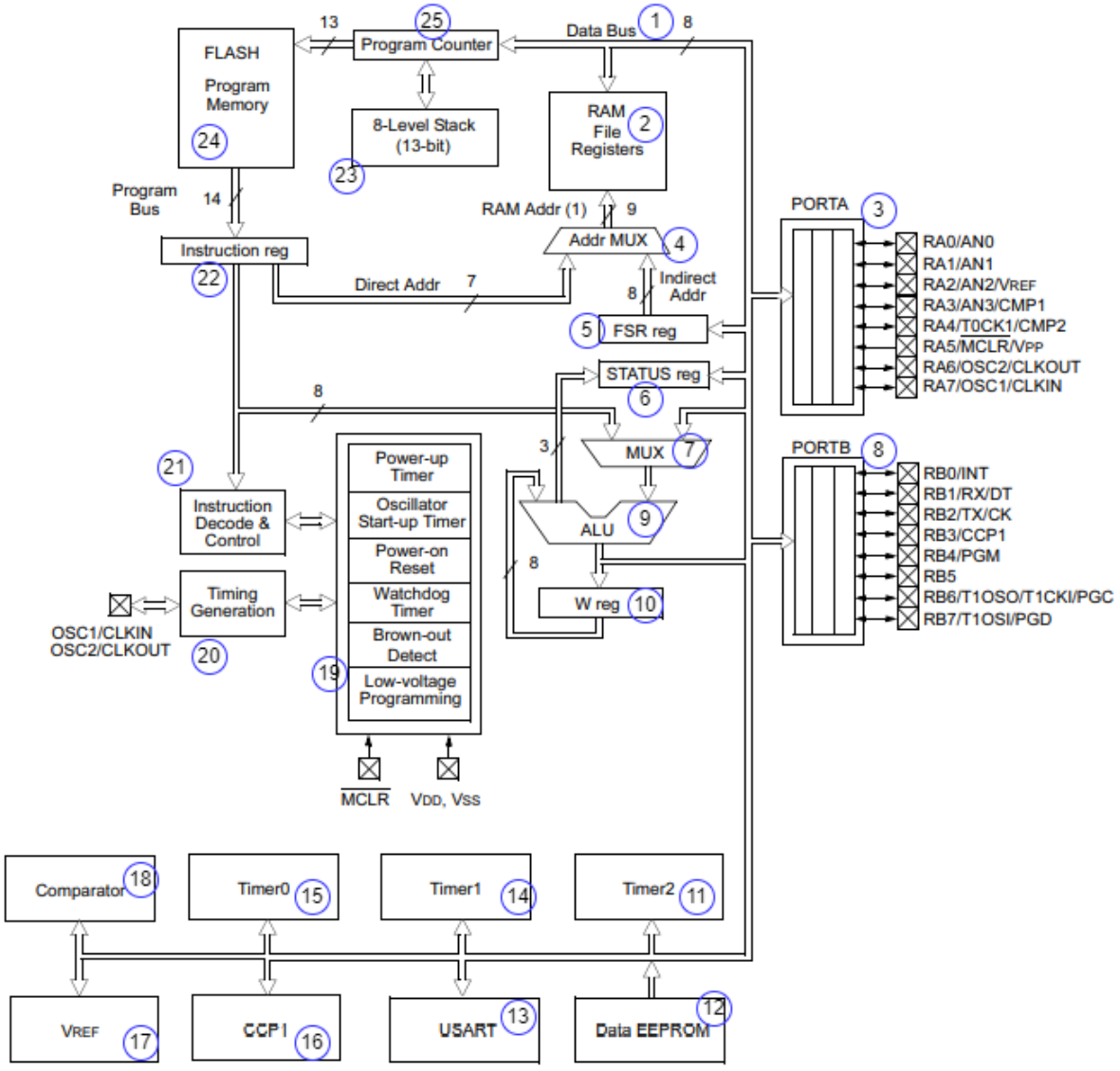


Fig. 2.4 Diagrama de bloques del PIC16F628

1.- Este MCU (microcontrolador) tiene un ancho de 8 bits de comunicación.

2.- RAM, registros de propósito específico y propósito general.

3-. Puerto A, contiene las entradas o salidas al dispositivo, que van desde la RA0 a RA7 contando también con los comparadores analógicos.

4.- Multiplexor de direccionamiento, afectado por el FSR y el registro STATUS.

5.- Registro de selección de archivo.

6.- Registro STATUS, donde están las banderas de Z, DC, RP0, etc. Para la configuración de bancos del PIC.

7.- Multiplexor.

8.- Puerto B, contiene entradas o salidas al dispositivo, que contiene desde RB0 a RB7, contando también con RX y TX para el envío y recepción de datos.

9.- Unidad Aritmética Lógica.

10.- Registro de trabajo.

11.- Timer2.

12.- Memoria de Datos EEPROM.

13.- Recepción Transmisión Universal Serial Asíncrona.

14.- Timer1.

15.- Timer0.

16.- Comparador de modulación por ancho de pulso.

17.- Periférico de voltaje de referencia.

18.- Periférico de Comparador.

19.- Características de funcionamiento del PIC.

20.- Generador de tiempo para oscilador externo.

21.- Instrucciones de control y decodificación.

22.- Registro de instrucción. Afecta directamente a las banderas de registro y de status.

23.- 8 niveles de subrutinas, esto es, el microcontrolador puede tener 8 anidaciones de subrutinas de 13 bits cada una.

24.- Memoria flash, donde residirá el código del programa del PIC.

25.- Contador de programa, este incrementa con 13 bits y va ubicando las posiciones de memoria, en el cual están los registros del set de instrucciones, se ejecutan uno a uno, dando saltos en las banderas de carry etc.

Organización de la memoria

En la presente sección se analizarán las características de las diferentes tecnologías de memoria con las que cuenta el PIC16F628, tales como la memoria RAM (Random Acces Memory – Memoria de Acceso Aleatorio) que posee la capacidad de leer y escribir. Una característica de las memorias RAM es que son volátiles, esto es los datos almacenados se pierden cuando se desconecta la alimentación del microcontrolador [7, p. 607]. También existe la memoria ROM (Read Only Memory – Memoria de Solo Lectura) está memoria mantiene los datos almacenados permanentemente que pueden ser leídos, pero no se pueden cambiar. Se utiliza para almacenar las instrucciones programadas para la inicialización del microcontrolador y funcionamiento del sistema. Por esta razón se denomina memoria no volátil [7, p. 622].

Las memorias PROM son similares a las ROM, en el sentido de que son memorias de una sola programación y una vez programadas ya no es posible modificar la información almacenada. A diferencia de la memoria ROM, las EPROM salen de la fábrica sin estar programadas y son programadas por el usuario para satisfacer sus necesidades [7, p. 629].

La EPROM son memorias no volátiles, similares a las PROM. La diferencia es que las EPROM pueden ser reprogramada. Para esto es necesario previamente borrar el programa que existe en dicho espacio de memoria a través de algún mecanismo, como puede ser la incidencia de luz UV [7, p. 630].

La memoria FLASH es una memoria no volátil, con una alta capacidad de almacenamiento. Dispone de la capacidad de lectura y escritura en el propio sistema, es decir está tecnología permite al usuario grabar y borrar información de la memoria directamente en estas localidades de memoria no volátiles [7, p. 632].

Cada elemento de almacenamiento de una memoria puede almacenar un bit, es decir un 1 o un 0 lógico. A estos elementos se les conoce como celdas de memoria. Las memorias están formadas por matrices de celdas [7, p. 602]. Una unidad de memoria se identifica mediante el número de palabras que puede almacenar, multiplicado por el tamaño de la palabra [7, p. 603]. En la Tabla 2.2 se presentan las características de la memoria de los PIC de la familia 16XX627 y 628. Como se puede observar, el PICXX628 es el que cuenta con mayor capacidad de memoria flash.

Tabla 2.2 Cantidad de almacenamiento de la familia PIC16XX62X

Dispositivo	Memoria Flash	Memoria RAM	Memoria EEPROM
PIC16f628	2048 x 14	224 x 8	128 x 8
PIC16f627	1024 x 14	224 x 8	128 x 8
PIC16LF627	1024 x 14	224 x 8	128 x 8
PIC16LF628	2048 x 14	224 x 8	128 x 8

Memoria del Programa (FLASH)

Antes de abordar el tema de la memoria del programa es conveniente recordar que un byte está formado por ocho bits de información. Un bit es la unidad mínima de información, la cual es almacenada con ceros o unos [7, p. 602]. La memoria de programa es una memoria flash. Para el PIC16F628 tiene una capacidad de 2048 (2KB) palabras de 14 bits, tal y como se puede observar en la Tabla 2.2.

En la memoria del programa se almacenarán las instrucciones que realizará el microcontrolador cuando esté en operación. Una vez que estas instrucciones sean cargadas en el microcontrolador (en la memoria del programa de este), estas instrucciones se deben de mantener almacenadas sin importar si el dispositivo está conectado a la alimentación eléctrica o no. Como se analizó anteriormente, a este tipo de memorias se les conoce como memorias no volátiles. En la arquitectura del programa mostrada en la Fig. 2.4, el bloque número 24 representa la memoria del programa.

Memoria RAM

En la memoria RAM se almacenan los registros de propósito específico (SFR, especial Function Registers – Funciones de propósito específico) y de propósito general. Cada uno de estos registros almacenan información sobre el funcionamiento, configuración del PIC o los valores de las variables que se utilizan al momento de realizar los programas. En secciones posteriores se analizarán más a detalle estos registros.

El PIC16F628 tiene una memoria RAM que puede almacenar 224 bytes (ver tabla 2.2). Está particionada en cuatro bancos, en donde se almacenan tanto los registros de propósito general, como los de propósito específico (ver Fig. 2.5). Un aspecto importante de recordar es que la memoria RAM es volátil. Esto significa que una vez que se quita la alimentación del PIC o se reinicia el programa los datos se borrarán de la memoria.

Los SFR (Special Function Registers – Funciones de Propósito Específico) están localizados en las primeras 32 locaciones del banco 0 y banco 1, para el banco 2 y 3 solo ocupan las primeras 16 locaciones. Como se puede observar en la Fig. 2.5, los SFR inician desde la posición 00h hasta la 1Fh (en sistema decimal sería del registro 0 al 31). En el banco 1 los SFR van desde 80h hasta 9Fh. En el banco 2 inician en 100h y terminan en 10F y el banco 3 desde la dirección 180h hasta la 18Fh.

Existen dos modos de direccionamiento en el microcontrolador para los registros SFR y los registros de propósito general, solo son formas de acceder a datos o a registros de forma diferente, a continuación, se explica cómo funciona cada uno.

Modo de direccionamiento directo

El modo de direccionamiento directo es cuando uno de los operandos de la instrucción hace referencia a una localidad de memoria o a un registro, ejemplo *movwf PORTB*, aquí el registro W pasa al registro PORTB, en este caso el operando W está haciendo referencia al registro PORTB, dicho de otra forma, no existen las instrucciones que permitan que ambos operandos sean direcciones de la memoria de datos [13].

Modo de direccionamiento indirecto

El direccionamiento indirecto es utilizado como apuntador, es decir, se usa la dirección guardada en el registro para indicar la posición de un dato, por ejemplo, se tiene una variable llamada LED y con un valor cargado en binario de '011001', la instrucción incf LED,0 es un modo de direccionamiento indirecto puesto que la variable LED incrementará en uno y el dato se guardará en W. W es la dirección guardada en el registro que indica la posición del dato de la variable LED [13].

Memoria EEPROM

Como se mencionó anteriormente, la memoria de datos es volátil, es decir, cuando se retira la alimentación se pierden los datos que estaban almacenados. En esta sección de la memoria es donde se deben de almacenar los datos que se deben de mantener aún después de retirar la fuente de alimentación del dispositivo, es decir, si en el transcurso de la ejecución del programa hubo variables o datos leídos externamente y que son importantes, está memoria es capaz de volverlos a mostrar o que se queden guardados para su próxima utilización del componente, cuando se restablezca la fuente de alimentación eléctrica. En otras palabras, es una memoria no volátil que se puede programar y borrar rápidamente [7, p. 631]. En el PIC16F628 la capacidad del almacenamiento de la memoria EEPROM es de 128 bytes x 8 bits. En la arquitectura del PIC16F628 (Fig. 2.4) La memoria EEPROM se identifica con el número 12.

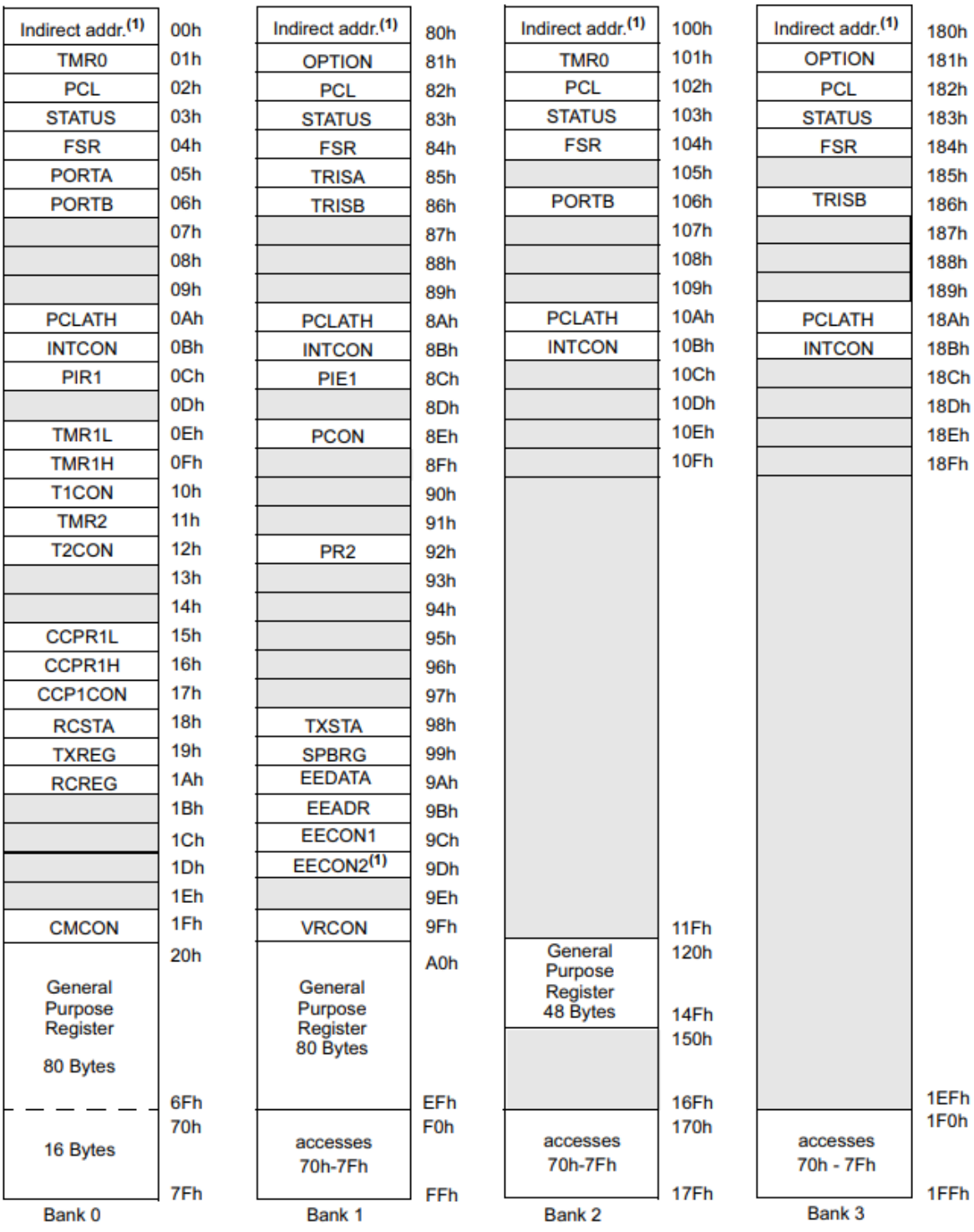


Fig. 2.5 Registros de Propósito Específico, almacenados en bancos dentro de la RAM.

Entradas y salidas de datos. Puertos del PIC16F628.

El PIC16F628 cuenta con dos puertos, el puerto A y puerto B, ambos con ocho bits. Para el puerto A, la terminal RA5 solo puede ser utilizada como entrada. Esta terminal está vinculada con el reset del microcontrolador (MCLR), por lo que la configuración interna del microcontrolador no permite que RA5 actúe como salida [6]. La terminal RA4 puede ser configurada como entrada o salida. Sin embargo, es muy importante destacar que la salida de RA4 es una salida a drenador abierto (Open Drain). Esta condición hace que el circuito de salida tenga una configuración especial. El resto de las terminales de los puertos del PIC16F628 (RA0, RA1, RA2, RA3, RA6 y RA7) pueden ser configuradas como entradas o salidas, según lo requiera el diseño. Todas las salidas cuentan con una configuración CMOS. Posteriormente se analizará la forma en que se realiza la configuración de las terminales de los puertos.

En el caso del puerto B todas sus terminales pueden ser configuradas como entradas o salidas. Cuando se configuran las terminales del puerto B como entradas, estas operan como entradas TTL (Lógica Transistor-Transistor, Transistor-Transistor Logic). Esto significa que las entradas están implementadas con tecnología TTL, por lo que un CERO lógico se representará con un nivel de voltaje que va de 0 a 0.8V, mientras que un UNO lógico se representará con un voltaje comprendido entre 2 y 5V (ver tabla 2.3).

En caso de que las terminales del puerto B se configuren como salidas, debido a que todas ellas son salidas CMOS, los niveles de voltaje que representan los valores lógicos se muestran en la tabla 2.3 [12].

Tabla 2.3 Diferencia en nivel de tensión entre tecnología TTL y CMOS.

Nivel de tensión	TTL	CMOS
Bajo (0)	0V – 0.8V	0V – 1.5V
Alto (1)	2V – 5V	3.5V – 5V

3 Circuitos Periféricos básicos

La importancia de los microcontroladores es que son circuitos que pueden ser programados para realizar ciertas tareas. Estos programas toman en cuenta las señales de entrada que llegan a las terminales de los microcontroladores para realizar acciones a partir de los circuitos que están conectados en las terminales de salida del circuito. Por ejemplo, en un horno de microondas, las diferentes funciones del horno estarán programadas en al menos un microcontrolador. De esta forma, cuando el usuario selecciona el tiempo de cocción o la potencia de calentamiento, lo que hace es introducir valores de voltaje a diferentes terminales del microcontrolador utilizando los interruptores del teclado. Con esta información el microcontrolador ejecuta un programa que manda una señal de salida que activa el dispositivo de calentamiento. Los circuitos de entrada de datos sería el teclado y los actuadores conectados a la salida sería el dispositivo que genera las microondas, son los circuitos periféricos.

Existe una gran cantidad de circuitos periféricos que pueden ser utilizados en diferentes aplicaciones. En esta sección se analizarán los circuitos más básicos que serán utilizados en secciones posteriores como circuitos de entrada o salida. Entre los dispositivos que se analizarán están:

- Dispositivos periféricos de salida: Diodos LED, display de siete segmentos, zumbadores y pantalla LCD.
- Dispositivos de entrada: Interruptores deslizables, pulsadores, teclados matriciales, resistencias variables, optoacopladores, entre otros.
- Circuitos de alimentación: Circuitos periféricos que permitan alimentar eléctricamente los proyectos basados en microcontroladores.

Alimentación

El PIC se alimenta a través de las terminales VSS, terminal 5 y VDD, terminal 14. Como se vio en las características (ver Capítulo 2), debe alimentarse con una tensión entre 3 y 5.5V, siendo el VSS tierra o negativo y el VDD positivo.

Para suministrar energía el microcontrolador se pueden ocupar fuentes regulables de corriente directa (DC) o hacer un circuito extra para brindarle el voltaje correcto al PIC. A continuación, se muestra un esquema de cómo se podría hacer esté circuito extra.

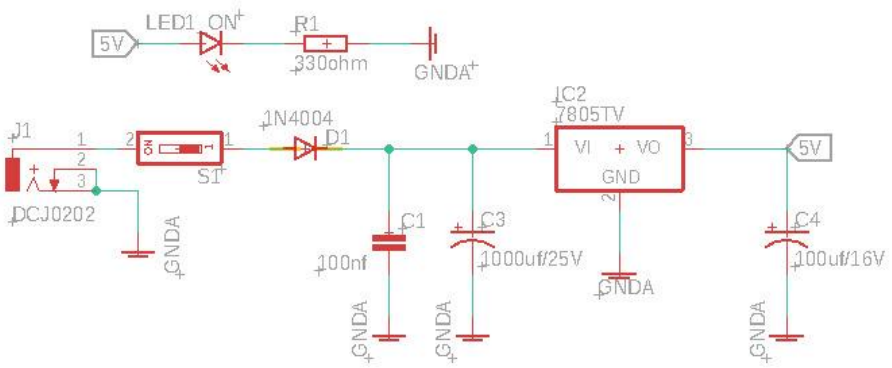


Fig. 3.1 Esquema eléctrico para alimentación desde 9 a 12 V

El regulador 7805 es circuito llamado regulador de voltaje, puesto que se encarga de regular la intensidad de voltaje entre 9V y 12V en su entrada a 5V en su salida, El diodo 1N4004 es un dispositivo de protección por si se polariza a la inversa la entrada. Esta entrada la puede brindar un cargador de baterías que llega al Jack o como está en el dibujo “DCJ0202”.

Circuitos periféricos de entrada de datos

Para lograr decirle al microcontrolador que hacer, éste requiere de la información digital como los ceros y unos, estos datos se introducen a través de interruptores o pulsadores, algunas configuraciones son en forma de pull-up y pull-down.

La configuración de pull-up se muestra En la Fig. 3.2 se muestra la configuración pull-up de un circuito de entrada de datos. Con este circuito el Vout se mantiene a estado alto (HIGH) mientras el interruptor no sea presionado. Cuando se activa el botón, el nodo de salida es conectado directamente a tierra, por lo que Vout se va a un estado bajo (LOW).

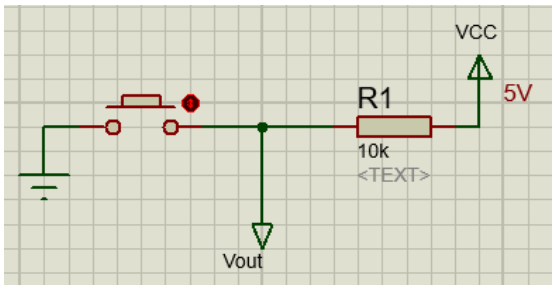


Fig. 3.2 Configuración pull-up.

Para la configuración pull-down es distinto pues Vout ahora siempre es 0V hasta que se presiona el botón, el Vout cambia de LOW a HIGH, puesto que ahora los 5 volts estarán en esta terminal, como lo muestra la Fig. 3.3

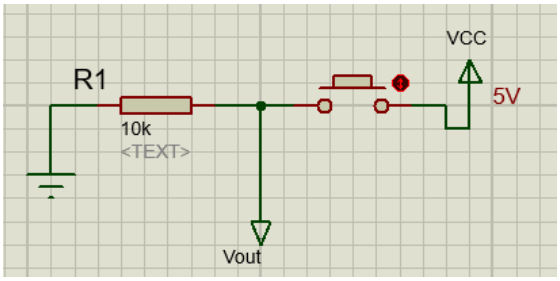


Fig. 3.3 Configuración pull-down

El microcontrolador PIC16F628 acepta entradas de configuración tipo pull-down, para cada pin o patita que se requiera configurar como entrada del MCU (Microcontrolador) se debe realizar este circuito por cada uno como en la Fig. 3.4

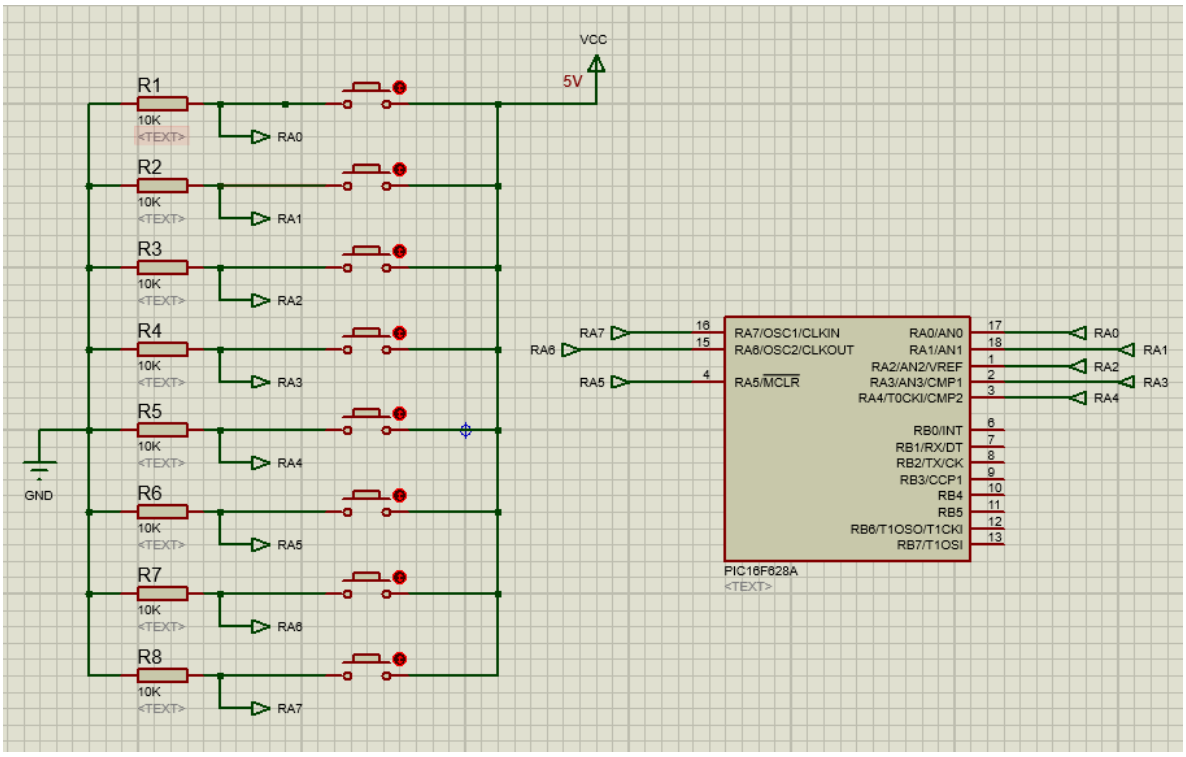


Fig. 3.4 Entradas pull-down para el PIC16F628

Circuitos periféricos de salida de datos

Estos circuitos periféricos mostrarán la información resultante del procesamiento en el microcontrolador, estás salidas de datos pueden activar ciertos procesos o tareas para un objetivo en común, como la visualización de un contador de piezas con display de 7 segmentos o la activación de bombas hidrantes cuando el tanque este lleno, el puente H puede ser capaz de realizar está tarea, así mismo la representación de salida se puede escuchar como alarmas o indicadores de precaución con el zumbador. La utilización de los transistores BJT se pueden utilizar para amplificar una señal o en este caso darle potencia a un motor controlando su velocidad de giro en conjunto con un potenciómetro. También

existen las pantallas LCD para mostrar texto al usuario, se hizo un tema en específico para hablar sobre este periférico, se puede encontrar en el titulo 7 que habla sobre la programación elemental.

LED

Para mostrar los datos, existen los LED como lo más básico para la representación de información digital. En la Fig. 3.5 se puede mostrar un circuito que por RB0 y RB6 del PORTB se mostrarán salidas con este periférico.

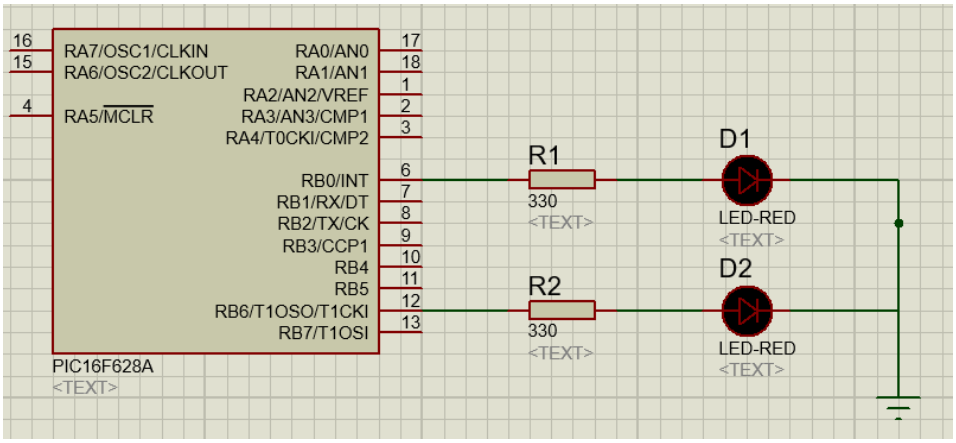


Fig. 3.5 Circuito con salidas LED.

También se pueden mostrar salidas con gráfico de barras LED como en la Fig. 3.6

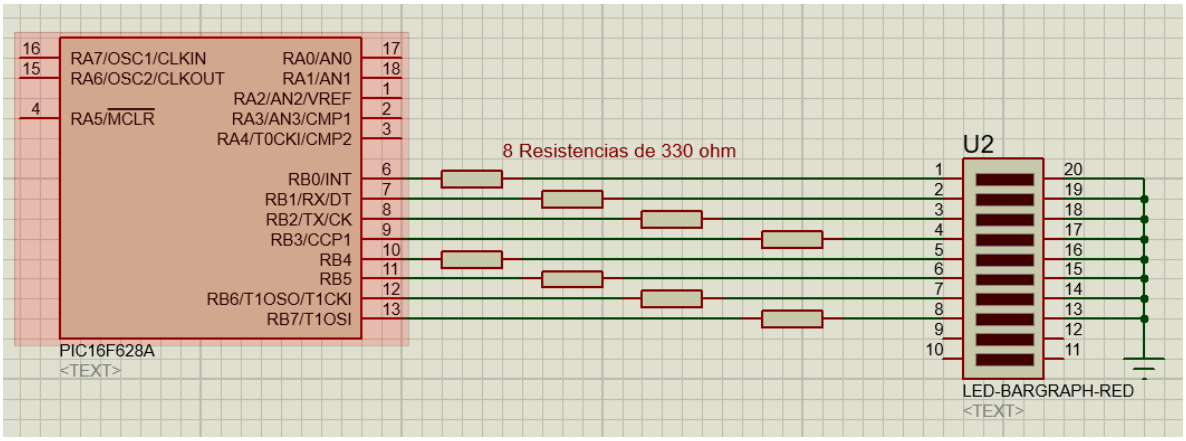


Fig. 3.6 Circuito con salidas a grafico de barras LED

Display de 7 segmentos

Los display de 7 segmentos son dispositivos que permiten mostrar caracteres numéricos formados por líneas luminosas. Como se puede observar en la fig. 3.7 cada segmento está formado por un LED. Los display pueden encontrarse en configuración de cátodo o ánodo comunes. Existen dos tipos, cátodo y ánodo comunes. A continuación, se muestran las diferencias entre estos en la Fig. 3.7.

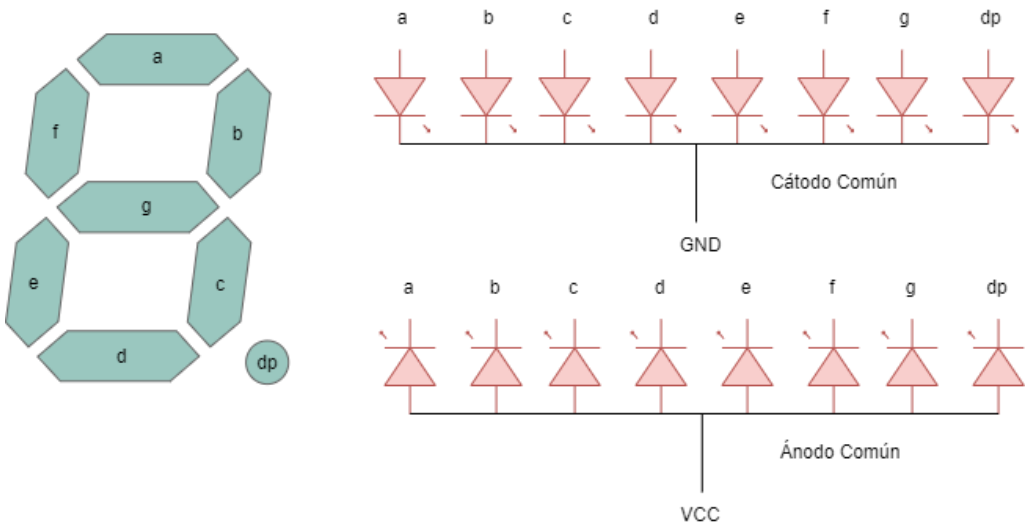


Fig. 3.7 Display de 7 segmentos en configuración de ánodo y cátodo común.

Para la representación física de este display con el PIC se mostrará en la siguiente Fig. 3.8. Donde se utiliza un display de 7 segmentos en su configuración de cátodo común.

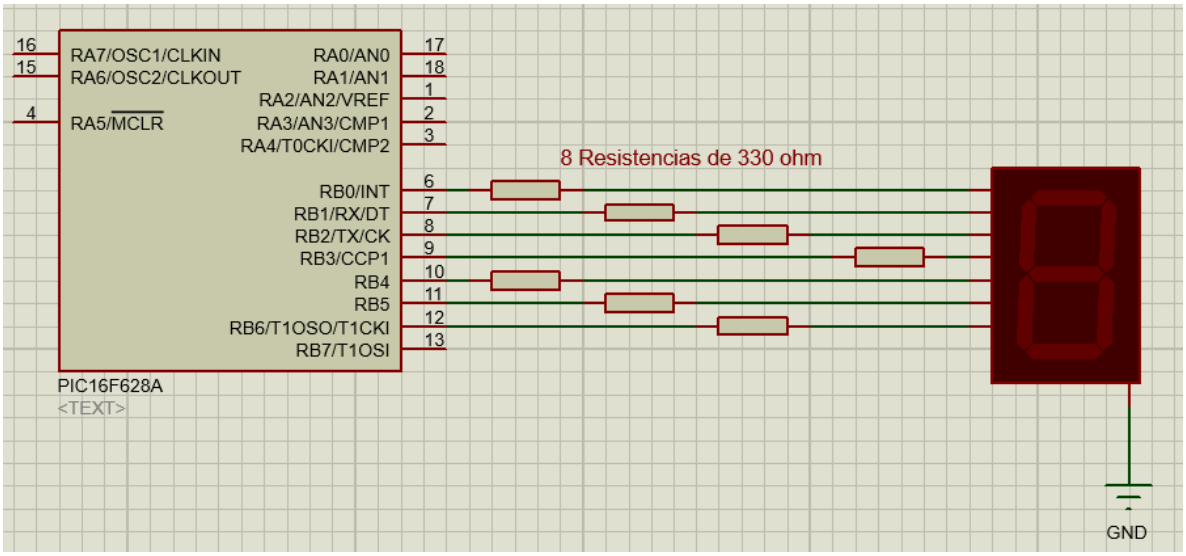


Fig. 3.8 Display de 7 segmentos en configuración cátodo común.

Zumbador

El zumbador o buzzer es un dispositivo que emite un sonido al momento que se le aplica un voltaje. En la Fig. 3.9 se puede observar la forma de conexión de este dispositivo. La resistencia puede variar de acuerdo con el consumo de energía del buzzer. Esto se hace para no dañar el pin del microcontrolador, la resistencia ofrece una entrega de corriente adecuada para el buzzer.

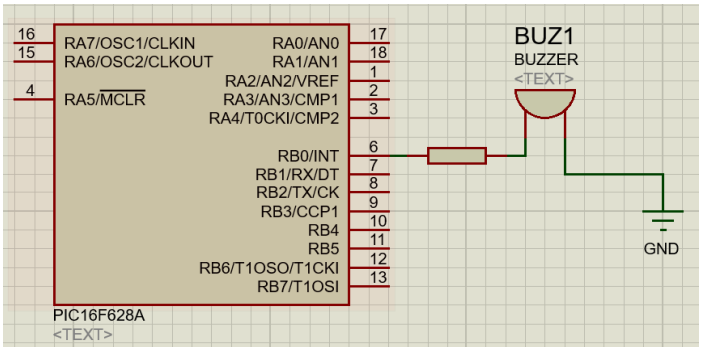


Fig. 3.9 Salida con zumbador.

Ejemplo: Se tiene un buzzer de 8 ohm y el PIC entrega 5 Volts por el pin RB0, la corriente que entregaría el microcontrolador sería 625mA, lo cual indica que es demasiada corriente para entregar y el datasheet del buzzer indica que puede funcionar entre 40 y 60 mA, superando está barrera el zumbador puede quemarse. Al aplicar la ley de ohm se tiene que la resistencia adecuada es 100 ohm.

Transistor BJT para controlar un motor de DC

La siguiente salida es para controlar un motor de corriente continua con un transistor BJT. Como se sabe el transistor BJT (Transistor de Unión Bipolar – Bipolar Junction Transistor) es controlado por corriente a diferencia de los FET que son controlador por voltaje. Estos transistores se dividen en dos grupos, tipo *NPN* y *PNP.* Cuentan con tres terminales, las cuales son colector, base y emisor.

En esta ocasión se mostrará como conectar un motor de 12V a un transistor 2n2222a, es uno de los más comerciales en electrónica. Para esto se necesitará un potenciómetro de 50 Kohm para ir regulando la corriente que pasa por la base del transistor, además de una fuente extra que brinde el voltaje adecuado para el motor DC que se conectará al colector. En la Fig. 3.10 Se mostrará la forma de conexión.

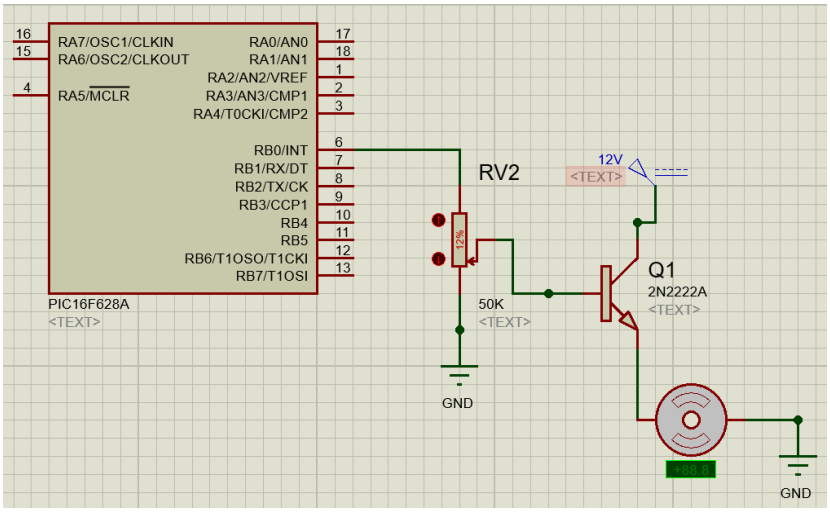


Fig. 3.10 Conexión del transistor 2n2222a hacía un motor de 12V

El datasheet del motor y del transistor se pueden encontrar en internet, para tener cuidado en la corriente que consume el motor y cuanta se le puede suministrar, el transistor es de potencia y soportaría muy bien la carga que ejercería el motor.

Puente H

Algo similar pasa con el puente H puesto que es un circuito integrado que se encarga de controlar de uno a tres a motores, la hoja de datos de este dispositivo se encuentra con el nombre L293D el diagrama de pines se muestra a continuación. Fig. 3.11.

Fig. 3.11 Diagrama de pines puente H (L293D)

El pin 1 y 9 Se encargan de habilitar el sentido de giro del motor, por ejemplo, el pin 2 que es la entrada 1 le da sentido antihorario (Sentido contrario a las manecillas del reloj) y el pin 7 o entrada 2 le da sentido horario. Cuando la entrada 1 este activa la entrada 2 debe estar desactivada para que el flujo de la corriente le dé sentido de giro al motor. El pin 3 y 6 que son salidas, estás van conectadas al motor DC no importa la polarización, al final las entradas determinan el sentido de giro. Vcc1 es una entrada de 5V, es propia del funcionamiento del driver (Puente H) el pin 8 o Vcc2 es una entrada de hasta 36V para brindarle la suficiente energía a motores que necesitan este voltaje. Los pines del 9 al 15 hacen exactamente lo mismo que los pines del 1 a 7 [15].

Para la conexión de este periférico es importante destacar que el diagrama de este circuito consta de tres etapas, la etapa uno la brinda el MCU que es la de control, la etapa dos es de potencia y el puente H se encarga de esto, la etapa tres son los actuadores, en este caso los motores. Para el ejemplo siguiente de la Fig. 3.12. se muestra la conexión solo de un motor.

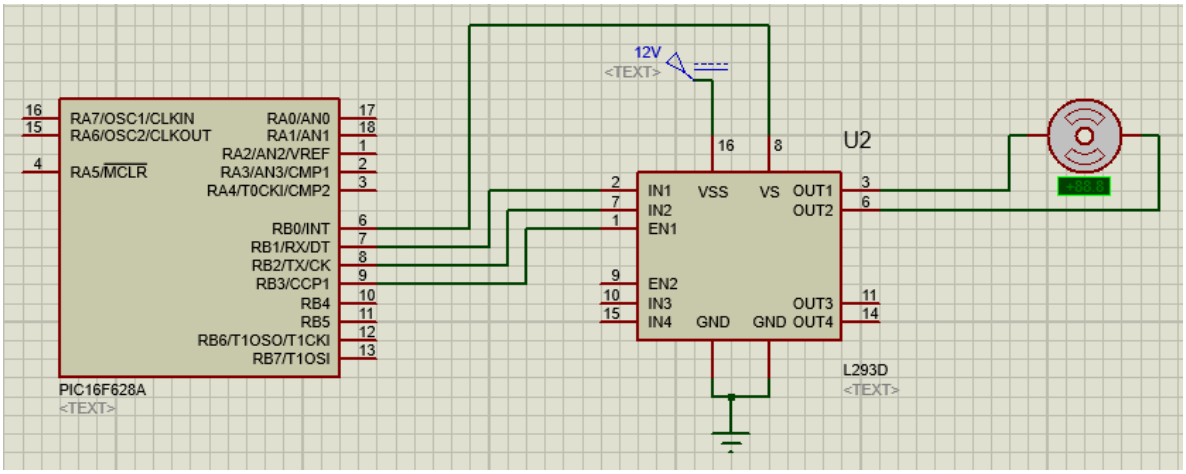


Fig. 3.12 Conexión de periférico puente H

Existe la posibilidad de controlar motores que necesitan más potencia, de hasta 460V, para eso, existen los PLC que son circuitos lógicos programables, estos se ocupan en la industria del CNC, pero la etapa uno y dos, anteriormente explicadas, gobiernan dentro del PLC.

4 Registros de funciones específicas

Los registros de funciones específicas (SRF, Special Registers Functions) son localidades de memoria, alojados en algunos espacios de los cuatro bancos de la RAM. Estos registros son utilizados por la unidad central (CPU) para controlar y monitorear la operación del microcontrolador y del programa en ejecución. Estos registros son RAM estática [7-17]. Cada registro está formado por 8 bits, que son numerados desde el bit 0 (el menos significativo, a la extrema derecha) hasta el bit 7 (el más significativo, colocado a la extrema izquierda) tal y como se muestra en la Fig. 4.1

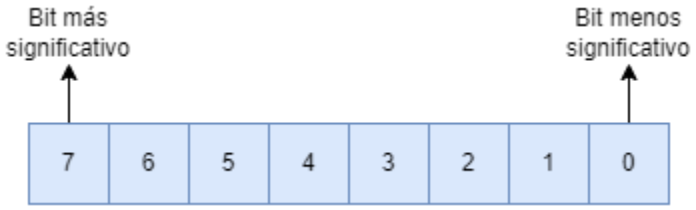


Fig. 4.1 Bit del menos al más significativo.

En las siguientes secciones se explicará brevemente los principales SRF, su función y descripción de sus bits.

Configuración de Palabra

Aunque éste no es un registro para almacenar como un SFR ni como registro de propósito general es importante decir que estos bits de configuración se cargan junto con el programa que se almacenará en la memoria flash del PIC y sirven para configurar ciertos aspectos generales de funcionamiento del microcontrolador. Ciertas funciones se muestran en la tabla 4.1 las banderas o bits en color gris no están implementadas y se leen como un 0.

Tabla 4.1 Configuración de Palabra

Bit	Bandera
13	CP1
12	CP0
11	CP1
10	CP0
9	
8	CPD
7	LVD
6	BODEN
5	MCLRE
4	FOSC2
3	PWRTE
2	WDTE
1	FOSC1
0	FOSC0

Bit 13-10. CP1:CP0: Bits de código de protección.

Código de protección para 2Kb de memoria del programa.
1 1 = Protección apagada.
1 0 = 0400h - 07FFh código protegido.
0 1 = 0200h - 07FFh código protegido.
0 0 = 0000h - 07FFh código protegido.

Código de protección para 1Kb de memoria del programa.
1 1 = Protección apagada.
1 1 = Protección apagada.
0 1 = 0200h – 03FFh código protegido.
0 0 = 0000h – 03FFh código protegido.

Bit 9. No se implementa y se lee como un '0'.

Bit 8. CPD: Bit de protección de código de datos.
1 = Protección desactivada para el código de la memoria de datos.
0 = Protección habilitada para el código de la memoria de datos.

Bit 7. LVP: Habilitación de programación de bajo voltaje.
1 = El pin RB4/PGM tiene función PGM, programación de bajo voltaje habilitada.
0 = RB4/PGM es entrada o salida digital, se debe usar HV (High Voltaje) en MCLR para la programación.

Bit 6. BODEN: Bit de habilitación de restablecimiento de detección de caída de tensión.
1 = Restablecimiento habilitado de BOD.
0 = Restablecimiento deshabilitado de BOD.

Bit 5. MCLRE: Función de selección del pin RA5/MCLR.
1 = RA5/MCLR es configurado como MCLR.
0 = RA5/MCLR es una entrada digital y MCLR es internamente conectada a VDD.

Bit 4,1 y 0. FOSC2:FOSC0: Bits de selección para el Oscilador.
1 1 1 = Oscilador ER. Significa que habrá una resistencia en el pin 16 mientras que en el pin 15 se ocupará la función de CLKOUT.
1 1 0 = Oscilador ER. Se entiende que habrá una resistencia en el pin 16 mientras el pin 15 se comporta como entrada o salida.
1 0 1 = Oscilador INTRC. La función CLKOUT del pin 15 se podrá utilizar y el pin 16 actúa como entrada o salida digital.
1 0 0 = Oscilador INTRC. Tanto en el pin 15 y 16 fungirán como entrada o salida digital.
0 1 1 = EC: Función de Entrada o salida en pin 15 mientras que el pin 16 se utiliza la función CLKIN.
0 1 0 = Oscilador HS. Cristal/resonador de alta velocidad por pin 15 y 16.

0 0 1 = Oscilador XT. Cristal/resonador en pines 15 y 16.
0 0 0 = Cristal de baja potencia, oscilador LP por pin 15 y 16.

Bit 3. PWRTEN: Bit de habilitación de temporizador encendido.
1 = PWRT deshabilitado.
0 = PWRT habilitado.

Bit 2. WDTEN: Bit de habilitación del temporizador del perro guardián.
1 = WDT habilitado.
0 = WDT deshabilitado.

Registro OPTION

En este registro se puede escribir y también leer, contiene vario bits de control para configurar la interrupción externa, los pull-ups débiles en PORTB, pero los que más importan para el desarrollo de este documento son desde el bit cero al bit cinco ya que en el capítulo de programación elemental se hablará sobré el TMR0 y estás banderas estarán configurándose para el control del TMR0.

Tabla 4.2 Registro OPTION.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bandera	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Bit 7. RBPU: bit de habilitación de pull-up en PORTB
1 = los pull-ups de PORTB están deshabilitados.
0 = los pull-ups de PORTB están habilitados por valores de bloqueo de puerto individuales.

Bit 6. INTEDG: bit de selección de interrupción.
1 = interrupción en el flanco ascendente del pin RB0/INT.
0 = interrupción en el flanco descendente del pin RB0/INT.

Bit 5. T0CS: bit de selección de fuente de reloj.
1 = transición en pin RA4/T0CKI.
0 = reloj de ciclo de instrucción interno.

Bit 4. T0SE: bit de selección de incremento en el tipo de transición.
1 = Incremento cuando se lee una transición de alto a bajo en el pin 4
0 = Incremento cuando se lee una transición de bajo a alto en el pin 4

Bit 3. PSA: bit de asignación de pre-escalador.
1 = El preescaler está asignado al WDT (Perro guardián).
0 = El preescaler está asignado el módulo Timer0.

Bit 0-2. PS0:PS2: bits de tasa de pre-escalador.

Tabla 4.3 Pre – escalador.

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

Registro STATUS

Como se puede observar en la Fig. 2.4, el registro STATUS está alojado en los cuatro bancos de la memoria RAM. En el banco cero se encuentra en la dirección 03h, en el banco uno 83h, en el banco dos 103h y en el banco tres 183h. Esto significa que el mismo valor estará almacenado (mapeado) en las cuatro localidades de la memoria RAM indicadas.

El registro STATUS contiene el estado de la unidad aritmética lógica (ALU), el estado del RESET y los bits de selección de banco para la memoria (SRAM).

Como se observa en la tabla 10, los bits 0, 1 y 2 del registro STATUS se conocen como las banderas de Carry (C), Digit Carry (DC) y Zero (Z). Estos bits son conocidos como bits bandera, porque proporcionan información sobre los resultados obtenidos a partir de la realización de operaciones aritméticas o lógicas realizadas por el microcontrolador. Estos bits cambian su valor a un nivel lógico alto o bajo de forma automática, en función del resultado de la operación lógica o aritmética. De esta forma es fácil identificar cuando una operación es igual o si es menor o mayor a un número. La Fig. 4.2 muestra una forma más ilustrativa de cómo se comportan estos 3 bits.

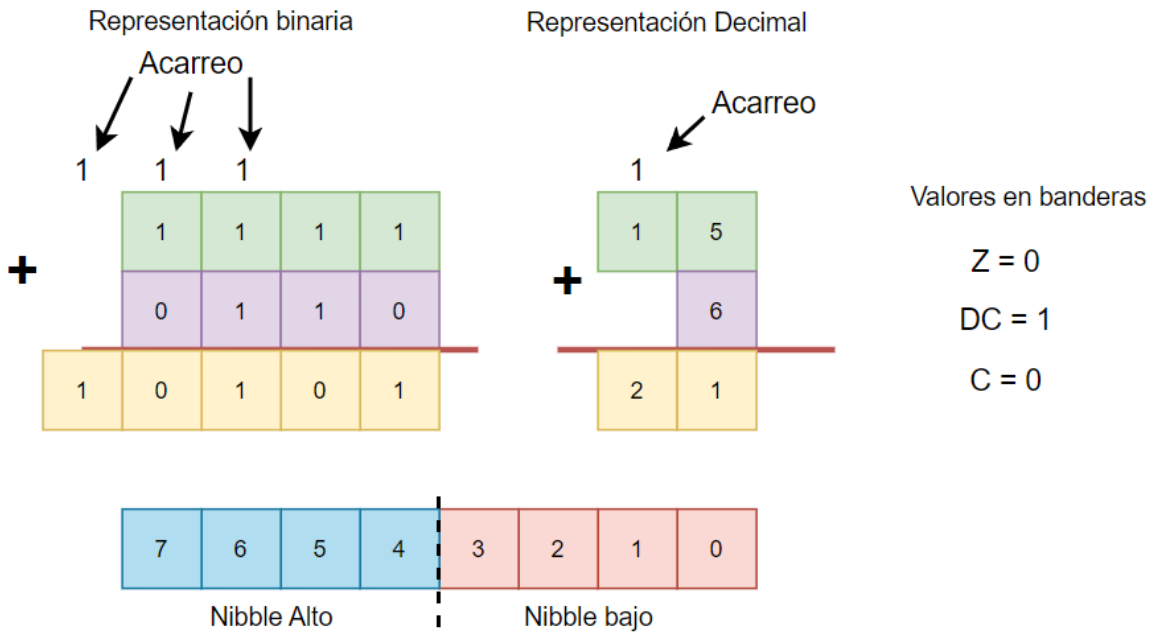


Fig. 4.2 Representación ilustrativa banderas Z, DC y C.

El valor de Z es cero porque el resultado no es cero, es decir, si se restara cinco menos cinco el resultado sería cero y por lo tanto el valor de Z cambiaría a uno. El valor de DC es igual a uno ya que hubo acarreo desde el bit 3 al 4, en cambio si fuera una suma de dos más dos no se ocuparía el nibble alto porque solo hasta el bit 2 es suficiente para representar el cuatro. El Valor de la bandera C es cero porque la suma no provoco un desborde en el byte, es decir, si la suma hubiera dado como resultado más de 255 que tiene cómo máximo el MCU (Microcontrolador) entonces C cambiaría a uno.

Ahora para la selección de bancos, primero se debe configurar el bit 7 o bandera IRP para después pasar a un banco en específico con RP0 y RP1.

Tabla 4.4 Registro STATUS.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bandera	IRP	RP1	RP0	TO	PD	Z	DC	C

Bit 7. IRP: Bit de selección de banco de registro.
1 = Banco 2, 3
0 = Banco 0, 1

Bit 5-6. RP1:RP0: bits de selección de bancos de registros.
0 0 = Banco 0.
0 1 = Banco 1.

1 0 = Banco 2.
1 1 = Banco 3.

Bit 4. TO: Bit de tiempo de espera.
1 = Después del encendido del MCU, instrucción CLRWDT o instrucción SLEEP.
0 = Se agotó el tiempo de espera de WDT (Watch Dog Timer).

Bit 3. PD: Bit de apagado.
1 = Después de encenderse o por la instrucción CLRWDT.
0 = Por la ejecución de la instrucción SLEEP.

Bit 2. Z: Bit Zero.
1 = El resultado de la operación aritmética o lógica es cero.
0 = El resultado de la operación aritmética o lógica no es cero.

Bit 1. DC: Bit de acarreo y préstamo de dígitos. Instrucciones que intervienen en está bandera son: ADDWF, ADDLW, SUBLW y SUBWF.
1 = Un acarreo ha sucedido en el cuarto bit de orden inferior del resultado.
0 = No ha ocurrido un acarreo en el cuarto bit de orden inferior del resultado.

Bit 0. C: Bit de acarreo y préstamo. Instrucciones que intervienen en está bandera son: ADDWF, ADDLW, SUBLW y SUBWF.
1 = Ocurrió un acarreo del bit más significativo del resultado
0 = No se produjo ningún acarreo del bit más significativo del resultado.

Registro PCON

Este registro contiene un bit importante que se debe recalcar, ya que está bandera será importante para el presente documento, se trata del bit3 o bandera OSCF. Está bandera es la encargada de decidir qué oscilador se debe ocupar internamente después de decirle al microcontrolador si ocupara o no el oscilador interno. Los demás bits sirven para diferenciar ante un restablecimiento de encendido, un restablecimiento MCLR externo, restablecimiento de WDT o detección de caída de tensión.

Este registro se aloja dentro del banco uno con la dirección 8Eh. Las banderas o bits en color gris no están implementadas y se leen como un 0.

Tabla 4.5 Registro PCON

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bandera					OSCF		POR	BOD

Bit 3. OSCF: frecuencia del oscilador INTRC/ER.
1 = 4MHz
0 = 37 KHz

Esta bandera establece la frecuencia del oscilador interno.

Bit 1. POR: bit de estado de restablecimiento de encendido.
1 = No se produjo ningún restablecimiento de encendido.
0 = Se produjo un reinicio de encendido.

Bit 0. BOD: bit de estado de detección de caída de tensión.
1 = No se produjo ningún restablecimiento de caída de tensión.
0 = Se produjo un restablecimiento de caída de tensión.

Registros PCL y PCLATH

El contador de programa (PC) tiene 13 bits de ancho como se puede ver en el diagrama de bloques del capítulo dos, estos 13 bits permiten llegar hasta 2048 palabras que puede almacenar la memoria del programa (Flash) pero los 13 bit se dividen en 2 partes, el byte bajo que es del bit 0 al bit 7 proviene del registro PCL y del bit 8 al bit 12 proviene del registro PCLATH.

El registro PCL contiene un byte de información que hace referencia a la dirección de memoria de la instrucción a ejecutar. Cuando se presenta un desbordamiento del PCL el registro se reinicia, por lo que el microcontrolador regresará a la instrucción inicial. Un ejemplo se muestra en la Fig. 4.3 donde se explica el funcionamiento de estos registros [17].

```
8     Org 0                 ; Está línea significa que el programa empezará en la dirección 0 en la memoria de programa
9     Inicio                ; Está Linea se considera una etiqueta
10
11    movlw d'1'          ;Se mueve un "1" al registro de trabajo "W"
12     movwf  PCLATH      ;Ahora W se mueve al registro PCLATH
13     ;**********
14    ;Al mover un uno al registro PCLATH significa que aunque el registro PCL se desborde o supere las 256 líneas de código
15    ;Va a seguir incrementandose en el registro PCLATH, de está forma se pueden ocupar los 5 bits restantes que ocupa el
16    ;Registro PCLATH y se pueden ocupar más de 257 líneas de código, siempre y cuando se le diga a PCLATH en que lugar de
17    ;La memoria se encuentra el programa
18    movlw  d'3'         ;Mover un 3 a W
19    call tabla          ;Se llama a la subrutina tabla, la cual se enuentra en la posición o línea de código 257
20    goto Inicio         ;Regesar desde el principio del programa
21
22    org 100             ;Aquí se llega a la posición 257 de la memoria y PCL vale 0 porque ya se desbordo
23    tabla
24    addwf  PCL,f   ;0   Al momento de sumar lo que tiene W que es un "3", al registro PCL se desbordaría en el 257 y comenzaría
25                         ;desde la primera línea de código, pero gracias a que se incremento el PCLATH se puede ocupar más de 256
26                         ;líneas de código y
27    retlw  'H'     ;1                    Está sería la línea de código 258
28    retlw  'O'     ;2                    Está sería la línea de código 259
29    retlw  'L'     ;3                    Está sería la línea de código 260 y así sucesivamente
30    retlw  'A'     ;4 El resultado de la suma caería hasta está línea de código y retornará con la A o en binario '01000001' el cual
31                   ;se mostrará en W
32    END
```

Fig. 4.3 Ejemplo de PCL y PCLATH

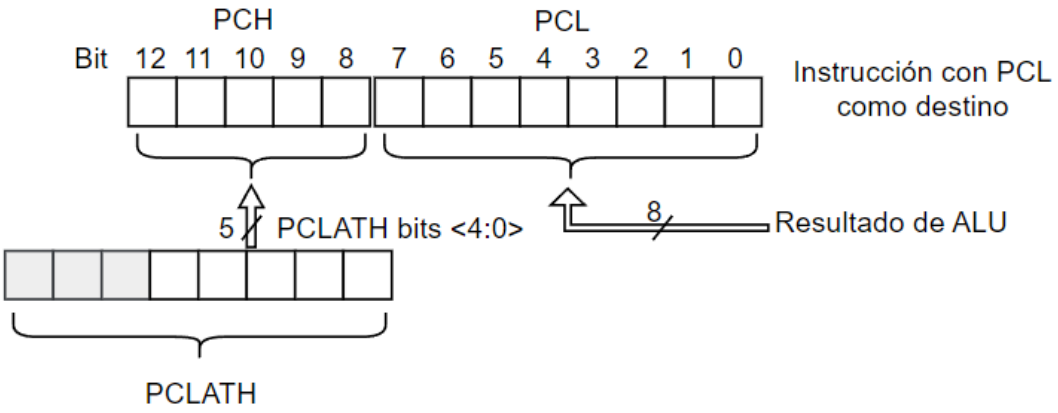


Fig. 4.4. Carga al PC al escribir en PCL

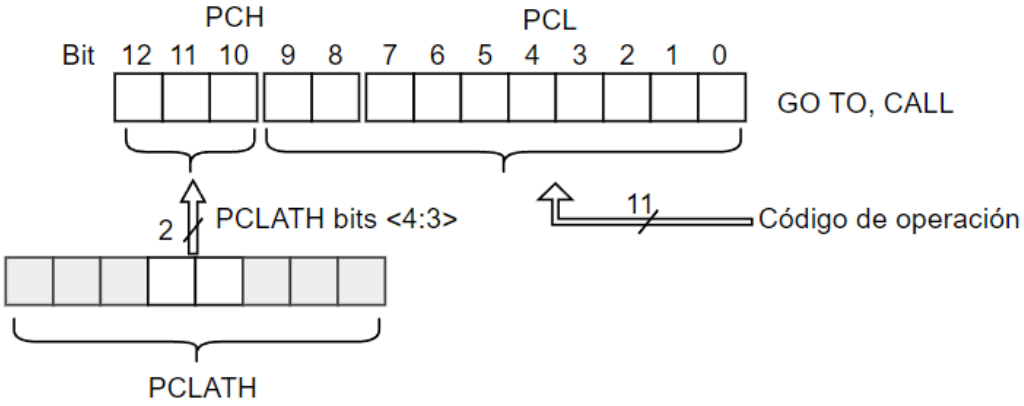


Fig. 4.5 Carga al PC durante una instrucción CALL o GO TO

Registros PORTA y TRISA

El PORTA es un registro dedicado a enviar o recibir datos. De forma que si se envían es porque internamente el PIC ya hizo algún procesamiento y se debe mostrar un resultado, por ejemplo, enviar voltaje de salida por el pin RA2 para que prenda un LED. Ahora, si recibe datos lo hará por medio de pulsadores o switches, de esa forma se introduce información al PIC.

Para la configuración del PORTA, se ocupa el registro TRISA. Este se encarga de decirle de qué forma se comportará (como entrada o salida). Si es de entrada entonces recibe datos y si es salida, envía datos. Es decir, el TRISA configura que bits del PORTA se pueden comportar como salidas o entradas. Así mismo el PORTA es un registro que está multiplexado con otras funciones, esto quiere decir que se pueden compartir funciones en un mismo pin, solo hay que darles la configuración adecuada ya que no se pueden usar al mismo tiempo, por ejemplo, el pin RA4 está multiplexado con la entrada de reloj T0CKI.

Los pines del PORTA están multiplexados con funciones de comparador (AN) y referencia de voltaje (VREF), la operación de estos pines se selecciona mediante bits de control en el registro CMCON (Registro de control del comparador).

El pin RA2 funcionará como salida para referencia de voltaje con este modo activo la salida tendrá una muy alta impedancia y el usuario debe configurar el bit 2 de TRISA como entrada y utilizar cargas de alta impedancia.

Para que los pines RA3 y RA4 que son de comparación analógica, puedan funcionar se deben definir en CMCON como salidas y para que funcione, en TRISA los pines 3 y 4 deben borrarse para permitir que lo utilicen como salida.

Registros PORTB y TRISB

PORTB es un puerto bidireccional de 8 bits de ancho. Para configurar al PORTB como salida, el registro TRISB se encarga de decidir que pines pueden utilizarse como entrada y salida, es decir, si en el registro TRISB hay un 1 todos los bits serán configurados como entrada mientras si hay un cero en ese byte o mejor dicho si hay 8 ceros en este registro serán configurados como salidas todos los pines. Se pueden alternar entre configurar salidas o entradas, de la misma forma que ocurre con PORTA y TRISA.

PORTB tiene multiplexación con la interrupción externa, USART, módulo CCP y entrada/salida de reloj TMR1. Cada pin del PORTB tiene un pull-up interno débil (200 uA).

Cuatro de los pines del PORTB tienen una función de cambio de interrupción los cuales son desde el bit 4 al bit 7. Solo los pines configurados como entrada pueden hacer que ocurra una interrupción. Si el PIC se configura para que entre en estado SLEEP, está interrupción puede despertar el dispositivo.

Registro CMCON

Esté registro contiene dos comparadores analógicos. Las entradas a los comparadores son multiplexadas con los pines RA0 a RA3.

Tabla 4.6 Registro CMCON

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bandera	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0

Bit 7. C2OUT: Salida del comparador 2.
Cuando C2INV = 0;
1 = C2 Vin+ > C2 Vin-
0 = C2 Vin+ < C2 Vin-

Cuando C2INV = 1;
1 = C2 Vin+ < C2 Vin-
0 = C2 Vin+ > C2 Vin-

Bit 6. C1OUT: Salida del comparador 1
Cuando C1INV = 0;
1 = C1 Vin+ > C1 Vin-
0 = C1 Vin+ < C1 Vin-

Cuando C1INV = 1;
1 = C1 Vin+ < C1 Vin-
0 = C1 Vin+ > C1 Vin-

Bit 5. C2INV: Inversión de salida del comparador 2.
1 = C2 salida invertida.
0 = C2 salida no invertida.

Bit 4. C1INV: Inversión de salida del comparador 1.
1 = C1 salida invertida.
0 = C1 salida no invertida.

Bit 3. CIS: Interruptor de entrada del comparador.
Cuando CM2:CM0: = 001
Entonces:
1 = C1 Vin- se conecta a RA3
0 = C1 Vin- se conecta a RA0

Cuando CM2:CM0: = 010
Entonces:
1 = C1 Vin- se conecta a RA3
C2 Vin- se conecta a RA2
0 = C1 Vin- se conecta a RA0
C2 Vin- se conecta a RA1

Bit 2-0: CM2:CM0: Modo comparador, esto es, a partir de estos bits se pueden configurar los demás pines porque aquí se determina la habilitación o des habilitación del modo de comparador, así como también los siete modos restantes de operación. En la Fig. 4.6 se pueden mostrar los modos de funcionamiento.

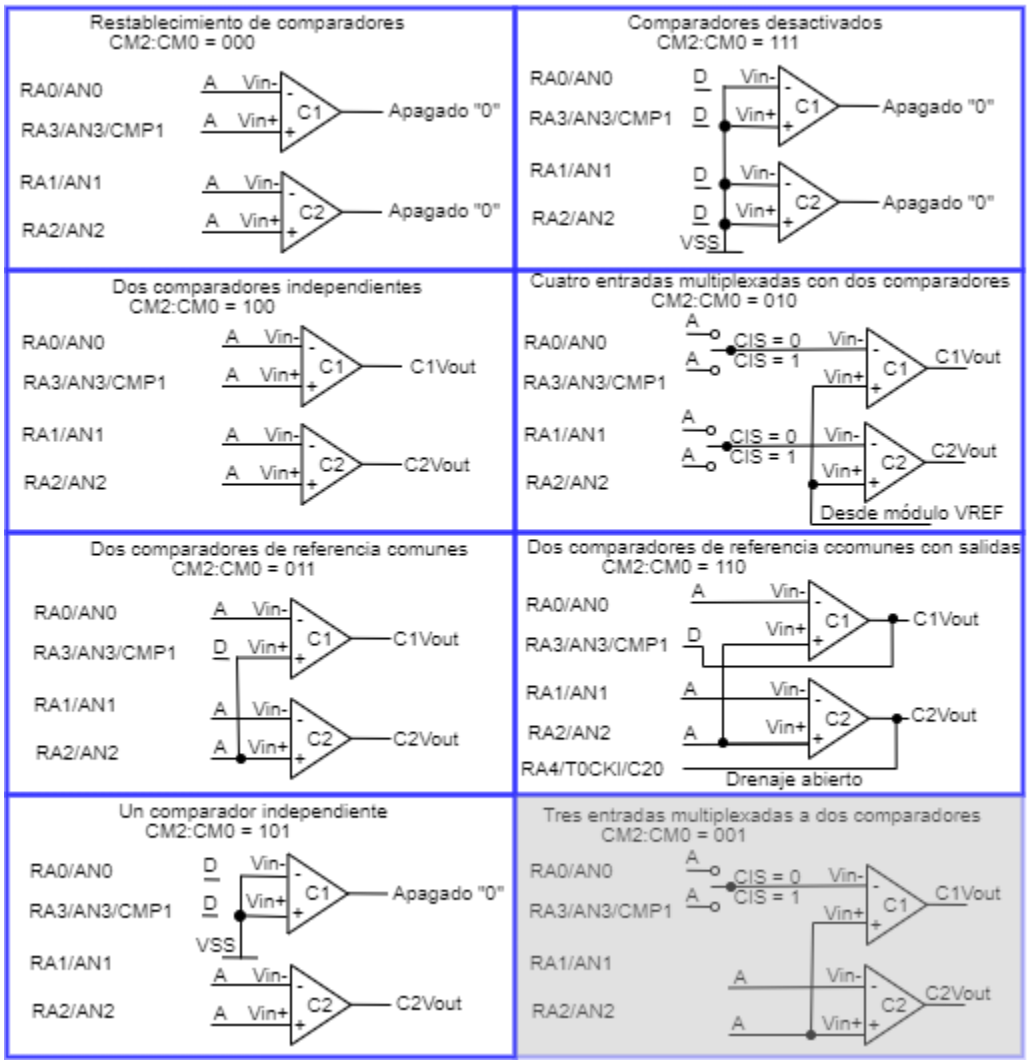


Fig. 4.6. Modos de funcionamiento del comparador.

El último rectángulo que está en color gris es disfuncional por lo cual no se puede ocupar.

La A significa entrada analógica y la D es entrada digital. CIS es el interruptor de entrada del comparador.

Registro TXSTA y RCSTA

El Módulo Receptor/Transmisor Síncrono/Asíncrono Universal (USART), es uno de los módulos de entradas y salidas del serial. Puede ser configurado como un sistema asíncrono dúplex completo que puede comunicarse por periféricos como las computadoras personales o las terminales CRT. También se puede configurar como un sistema síncrono semidúplex para comunicarse con componentes periféricos como A/D o D/A circuitos integrados, así mismo con las EEPROM.

Este módulo USART cuenta con dos registros el TXSTA y RCSTA. Dentro de cada byte o registro se alojan los modos de configuración a continuación en la Tabla 11 se explican las banderas del Registro TXSTA y en la Tabla 4.6 el registro RCSTA.

Registro TXSTA: Registro de Control y Estado de Transmisión.

Este registro se encuentra en la dirección 98h del banco 1

Tabla 4.7 Registro TXSTA

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bandera	CSRC	TX9	TXEN	SYNC		BRGH	TRMT	TX9D

Bit 7. CSRC: Bit de selección de fuente de reloj.
Modo asíncrono.
No importa.

Modo síncrono.
1 = Modo maestro
0 = Modo esclavo

Bit 6. TX9: Bit de habilitación para la transmisión de 9 bits.
1 = Selección de transmisión de 9 bits.
0 = Selección de transmisión de 8 bits.

Bit 5. TXEN: Bit de habilitación para transmisión.
1 = Transmisión habilitada.
0 = Transmisión deshabilitada.

Bit 4. SYNC: Bit de selección para el modo USART.
1 = Modo síncrono.
0 = Modo asíncrono.

Bit 3. No se implementa.

Bit 2. BRGH: Bit de selección de alta velocidad en baudios, los baudios es una unidad de transmisión de cantidad de información, equivalente al número de impulsos por segundo.

Modo asíncrono.
1 = Alta velocidad.
0 = Baja velocidad.

Modo síncrono.
No se usa en este modo.

Bit 1. TRMT: Bit de estado del registro de desplazamiento de transmisión.
1 = TSR vacío.
0 = TSR lleno.

Bit 0. TX9D: Transmisión de datos del noveno bit que puede ser un bit de paridad.

El USART se puede configurar en las siguientes modalidades:

- Asíncrono (dúplex completo).
- Síncrono – Maestro (dúplex semi completo).
- Síncrono – Esclavo (dúplex semi completo).

Registro RCSTA: Registro de Control y Estado de Recepción.

Este registro se encuentra en la dirección 18h del banco 0.

Tabla 4.8 Registro RCSTA

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bandera	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RXD

Bit 7. SPEN: Bit de habilitación del puerto serial.
1 = Puerto serial habilitado.
0 = Puerto serial deshabilitado.

Bit 6. RX9: Bit de habilitación para la recepción de 9 bits.
1 = Selección de recepción de 9 bits.
0 = Selección de recepción de 8 bits.

Bit 5. SREN: Bit de habilitación de recepción única.
El modo asíncrono no importa.

Modo síncrono – maestro:
1 = hablita la recepción única.
0 = deshabilita la recepción única.
El bit de borra una vez que se completa la recepción.

Para el modo síncrono - esclavo no se implementa.

Bit 4. CREN: Bit de habilitación de recepción continua.
Modo asíncrono:
1 = Habilita la recepción continua.
0 = Deshabilita la recepción continua.

Modo síncrono:
1 = habilita la recepción continua hasta que se borra el bit de habilitación CREN

(CREN anula SREN).
0 = deshabilita la recepción continua.

Bit 3. ADEN: Bit de activación de detección de dirección.
Modo asíncrono de 9 bits. RX9 = 1
1 = habilita la detección de direcciones, la interrupción y la carga de búfer de recepción cuando se establece RSR<8>.
0 = deshabilita la detección de direcciones, todos los bytes son recibidos y el nove bit puede usarse como bit de paridad.

No se implementa el modo síncrono de 8 bits.

Bit 2. FEER: Bit de error de trama.
1 = Error de trama (Se puede actualizar leyendo el registro RCREG y recibir el siguiente byte valido).
0 = Sin error de trama.

Bit 1. OERR: Bit de error de desbordamiento.
1 = Error de desbordamiento (Se puede borrar borrando el bot CREN).
0 = Sin error de desbordamiento.

Bit 0. TX9D: Recepción de datos del noveno bit que puede ser un bit de paridad.

Registro EEADR

EEADR se encuentra en la dirección 9B del banco 1 como se muestra en la Fig. 4.9.

La memoria de datos EEPROM es legible y escribible durante el funcionamiento normal. Esta memoria no está mapeada directamente en el espacio del archivo de registro. En cambio, se aborda indirectamente a través de SFR hay cuatro registros de funciones especiales para leer y escribir en esta memoria los cuales son EECON1, EECON2 (No es un registro implementado físicamente), EEDATA y EEADR.

EEDATA contiene los datos de 8 bits para lectura y escritura, EEADR contiene la dirección de la ubicación de EEPROM.

Tabla 4.9 Registro EEADR.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bandera		EADR6	EADR5	EADR4	EADR3	EADR2	EADR1	EADR0

Bit 7. No se implementa.

Bit 6-0. EEAD: Especifica una de las 128 ubicaciones de la operación de lectura y escritura de la EEPROM.

Este Registro puede direccionar hasta un máximo de 256 bytes de datos EEPROM. Sólo los primeros 128 bytes de EEPROM de datos están implementados y sólo siete de ocho bits en el registro (EEADR<6:0>) son requeridos.

El bit superior es la dirección decodificada, esto significa que el bit siempre estará en 0 para asegurarse de que la dirección está en el espacio de memoria de 128 bytes.

Registro EECON1

Es el registro de control con cinco bits de bajo orden implementado físicamente. Los tres bits superiores no se implementan. Este registro se encuentra en el banco 1 con la dirección 9Ch como lo muestra la Fig. 2.4.

Los bits de control RD y WR indican la lectura y escritura. Estos bits no se pueden borrar, solo establecer en el programa, se borran en el hardware al finalizar la operación de lectura o escritura. La imposibilidad de despejar el bit WR en el programa evita la prematura terminación de una operación de escritura.

El bit WREN, cuando se establece, permitirá una operación de escritura. En el encendido del PIC, el bit WREN está limpio. El bit WRERR es establecido cuando una operación de escritura es interrumpida por un MCRL o un tiempo de espera del WDT, en estas situaciones el usuario puede verificar el bit el bit WRERR y reescribir la ubicación, estos datos y la dirección no se modificarán en los registros EEDATA y EEADR.

El bit indicador de interrupción es EEIF en el registro PIR1 y se establece en el registro cuando la escritura este completa. Este bit debe borrarse en el programa.

Tabla 4.10 Registro EECON1

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bandera					WRERR	WREN	WR	RD

Bit 7-4. No se implementa.

Bit 3. WRERR: Bit de indicador de error en EEPROM
1 = Operación de escritura finaliza prematuramente.
0 = La operación de escritura se completó.

Bit 2. WREN: Bit de habilitación para escritura en la EEPROM
1 = Permite ciclos de escritura.
0 = Inhibe la escritura de la EEPROM de datos.

Bit 1. WR: Bit de control de escritura.
1 = Inicia un ciclo de escritura.
0 = El ciclo de escritura está completo.

Bit 0. RD: Bit de control de lectura.
1 = Inicia una lectura de EEPROM.
0 = No se inicia una lectura en la EEPROM.

Registro de trabajo o Registro W

El registro W es un registro que se ocupa durante diferentes operaciones que realiza la ALU. Tiene la característica particular de que no se encuentra físicamente en la memoria de datos, como los registros de función específica o los registros de propósito general. En lugar de esto, como se observa en la Fig. 2.4, W (número 10) se encuentra a la salida de la Unidad Aritmético Lógica (número 9). Esta situación ocasiona que algunas instrucciones que se aplican a los registros, que se encuentran en la memoria de datos, no sean funcionales para el registro W. Mas adelante se analizarán cuáles son estas instrucciones.

Una de las funciones del registro W es como salida de las operaciones realizada por la ALU. Para explicar esta función es importante mencionar que cada vez que una instrucción le indica a la ALU realizar una operación, el resultado puede ser almacenado en dos distintas localidades:

- En el mismo registro que se utilizó como entrada para realizar la operación, o
- En el registro W

cómo se verá en la sección donde se explica el set de instrucciones, el destino donde se almacenará el resultado lo define el programador, a través del operador d. de la siguiente forma:

- Si d=0, entonces el resultado se almacena en el registro W
- Si d=0, entonces el resultado se almacena en el registro que se utilizó como entrada.

Para explicar la otra función que puede tener el registro W es importante comentar que algunas instrucciones utilizan este registro como entrada. Esta situación se puede visualizar en el diagrama de bloques de la arquitectura del PIC (Fig. 2.4). Como se observa, existe una conexión directa entre W y una de las entradas de la ALU.

En esta sección se mencionó superficialmente como trabaja la ALU y W, debido a que ambos van de la mano en la operación de varias de las instrucciones que contiene a W como operando.

5 Set de instrucciones

El PIC 16F628 es un microcontrolador de tipo RISC, esto significa que tiene un juego de instrucciones reducido. En este caso en particular, el set está compuesto por 35 instrucciones, clasificadas en tres categorías:

- Orientadas a byte.
- Orientadas a bit.
- Operaciones con literales y de control.

En la Tabla 5.1 se presenta un resumen de las instrucciones, los operadores que se utilizan en la sintaxis, así como una breve descripción de la instrucción y las banderas que son afectadas. Para un mejor análisis y entendimiento de cada una de las instrucciones se recomienda identificar que el nombre de la instrucción está relacionado con la función de esta. Por ejemplo:

- ADDWF significa ADD Work to File, es decir, suma el registro de trabajo al registro F.
- CLRF significado CLEAR File, es decir, borra el registro F
- DECFSZ significa Decrement File and skip If Zero, es decir, decrementa el registro F y salta la siguiente instrucción si el resultado es CERO.

Hacer explícita esta relación entre el nombre y la función de la instrucción les permite a los programadores recordar fácilmente la aplicación de cada instrucción.

Tabla 5.1 Set de instrucciones.

Instrucción con operandos		Significado	Descripción	Ciclos	Bandera de STATUS Afectada
Operaciones de registro de archivo orientado a byte					
ADDWF	f,d	Suma W a F	Suma el contenido que está en el registro W con el registro 'f'. Si 'd' es 0 el resultada se guardará en W y si es '1' el resultado se alojará en 'f'	1	C, DC, Z
ANDWF	f,d	Operación AND de W con F	Hace una operación AND de W con 'f'. Si 'd' se establece como un 0 el resultado se alojará en W, caso contrario si 'd' es 1 se guardará en 'f'	1	Z
CLRF	f	Limpia F	Se borran los datos del registro 'f'	1	Z
CLRW	–	Limpia W	El byte del registro W se pone a 0	1	Z
COMF	f,d	Complemento de F	El contenido del registro 'f' es complementado. Si 'd' es 0 el resultado residirá en W caso contrario se guardará en 'f'	1	Z
DECF	f,d	Decrementa a F	Decrementa el registro 'f'. Si 'd' es 0 el resultado es guardado en W. Si es 1 el resultado residirá en 'f'	1	Z
DECFSZ	f,d	Decrementa a F y salta si es 0	El contenido del registro 'f' es decrementado. Si 'd' es 0 el resultado se pasará a W y si es un 1 en 'f'. Si el resultado de la operación es 0 la siguiente instrucción es ignorada, ahora si es un 1 la siguiente instrucción se ejecutará	2	
INCF	f,d	Incrementa a F	El contenido del registro 'f' es incrementado. Si 'd' es 0 el resultado es colocado en W, si 'd' es 1 el resultada estará en el registro 'f'	1	Z
INCFSZ	f,d	Incrementa a F y salta si es 0	El contenido del registro 'f' es decrementado. Si 'd' es 0 el resultado se pasará a W y si es un 1 en 'f'. Si el resultado de la operación es 1 la siguiente instrucción es ignorada, ahora si es un 0 la siguiente instrucción se ejecutará	2	
IORWF	f,d	Operación OR inclusiva de W con F	El registro W hace una operación OR inclusiva con el registro 'f'. Si 'd' es 0 el resultado residirá en el registro W. Si 'd' es 1 se guardará en 'f'	1	Z
MOVF	f,d	Mover a F	El contenido del registro 'f' es movido a un destino, despendiendo el estado de 'd'. Si 'd' es 0 se moverá a W y si es 1, el registro 'f' se sobrescribirá	1	Z
MOVWF	f	Mover W a F	Mueve el contenido de W al registro 'f'	1	
NOP	–	No operación	Se considera una operación nula porque no hace nada	1	

RLF	f,d	Rotar a la izquierda a través del acarreo	El contenido del registro 'f' se rota un bit a la izquierda a través de la bandera de acarreo. Si 'd' es 0 el resultado se coloca en W y si es un 1 el resultado se almacena en 'f'	1	C
RRF	f,d	Rotar a la derecha a través del acarreo	El contenido del registro 'f' se rota un bit a la derecha a través de la bandera de acarreo. Si 'd' es 0 el resultado se coloca en W y si es un 1 el resultado se almacena en 'f'	1	C
SUBWF	f,d	Restar W de F	Resta el contenido que está en el registro W con el registro 'f'. Si 'd' es 0 el resultada se guardará en W y si es '1' el resultado se alojará en 'f'	1	C, DC, Z
SWAPF	f,d	Intercambiar nibbles en F	Los nibbles superior e inferior del registro 'f' se intercambian. Si 'd' es 0 Se guardará en W y si es un 1 el resultado se coloca en 'f'	1	
XORWF	f,d	Operación OR exclusiva de W con F	El contenido del registro W hace una operación XOR con el registro 'f'. Si 'd' es 0 el resultado se alojará en w y si es un 1 en el registro 'f'	1	Z

Operaciones de registro de archivo orientado a bits					
BCF	f,b	Limpiar un bit de F	Un bit 'b' en del registro 'f' es limpiado o puesto a 0	1	
BSF	f,b	Poner un 1 en un bit de F	Se pone un 1 en el bit 'b' del registro 'f'	1	
BTFSC	f,b	Monitorea un bit de F y salta si es 0	Si un bit 'b' del registro 'f' es 0 entonces la siguiente instrucción es saltada o ignorada. Si no, entonces la siguiente instrucción es ejecutada.	2	
BTFSS	f,b	Monitorea un bit de F y salta si es 1	Si un bit 'b' del registro 'f' es 1 entonces la siguiente instrucción es saltada o ignorada. Si no, entonces la siguiente instrucción es ejecutada.	2	
Operaciones de control y literal					
ADDLW	K	Suma una literal a W	El contenido del registro W se suma a la literal de ocho bits 'K' y el resultado se guarda en el registro W	1	C, DC, Z
ANDLW	K	Operación AND de una literal a W	El contenido del registro W hace una operación AND con los ocho bits de la literal 'K' y el resultado se guarda en W	1	Z
CALL	K	Llama a una subrutina	Se llama a una subrutina	2	
CLRWDT	–	Limpia el Watch Dog Timer	Es un reset al WDT	1	TO, PD
GOTO	K	Ir hacía una dirección	Ir hacía alguna parte dentro del programa	2	

IORLW	K	Operación OR inclusiva de una literal con W	El contenido del registro W hace una operación OR inclusiva con los ocho bits de la literal 'k' y el resultado estará en W	1	Z
MOVLW	K	Mover una literal a W	El byte de la literal 'k' es cargado dentro del registro de trabajo	1	
RETFIE	–	Regresar desde una interrupción	Retorno desde una interrupción. LA pila se hace POP y la parte superior de la pila (TOS) se carga en la PC	2	
RETLW	K	Regresar con una literal en W	El registro W es cargado con los ocho bits de la literal 'k' y el contador de programa se carga desde la parte superior de la pila (Dirección de retorno)	2	
RETURN	–	Regresar de una subrutina	Retorno de una subrutina desde el lugar que fue llamada la rutina.	2	
SLEEP	–	Entrar al modo de espera	El procesador entra en estado de reposo, pero sigue operando a bajo consumo de energía	1	TO, PD
SUBLW	K	Restar W de la literal	El registro W es restado de los ocho bits de la literal 'k' y el resultado se almacena en W	1	C, DC, Z
XORLW	K	Operación OR exclusiva de una literal con W	El contenido del registro W hace una operación XOR con la literal 'k' y el resultado se alojará en W	1	Z

6 MPLAB

El entorno de desarrollo para escribir el código o tareas que realizará el MCU se eligió MPLAB IDE puesto que el PIC fue desarrollado por la misma empresa, es decir, Microchip desarrolló el IDE para programar sus microcontroladores y de esa misma forma el compilador.

Existen más entornos de desarrollo pero que mejor que el mismo fabricante tanto de hardware como de software. Así que en este capítulo se verá la instalación de MPLAB IDE V5.35. Así como también el compilador XC Compiler 8.

Instalación

1. Primero hay que entrar al sitio oficial de MPLAB. Es el sitio de Microchip.

[MPLAB® Ecosystem Downloads Archive | Microchip Technology]

En la sección de MPLAB X IDE Archives. Aparecerán tres sistemas operativos, Windows, macOS y Linux. En esta ocasión se optará por la versión 5.35 para Windows porque es la que se ha probado y se considera estable.

MPLAB X IDE Archives

1 ... 5 6 7 8 9

Windows® (x86/x64)	macOS® (10.X)	Linux® (32/64 bit)
MPLAB X v5.15	MPLAB X v5.15	MPLAB X v5.15
MPLAB X v5.20	MPLAB X v5.20	MPLAB X v5.20
MPLAB X v5.30	MPLAB X v5.30	MPLAB X v5.30
MPLAB X v5.35	MPLAB X v5.35	MPLAB X v5.35

Fig. 6.1 Proceso de instalación, selección de versión del programa

2. Al dar clic sobre la versión 5.35 y comenzará la descarga del programa.

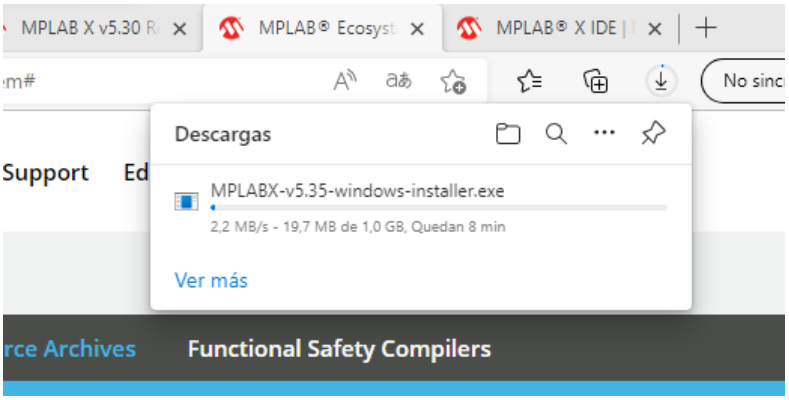


Fig. 6.2 Proceso de instalación, descarga del programa.

3. Lo que sigue es ubicar en que sitio de la PC se descargó el programa y se mostrará un archivo como el siguiente.

Fig. 6.3 Proceso de instalación, archivo de instalación descargado

4. Al dar doble clic sobre él y se abrirá una pantalla, donde pedirá permisos de administrador y se dará clic en **Si**. Empezará a cargar el instalador y saldrá lo siguiente.

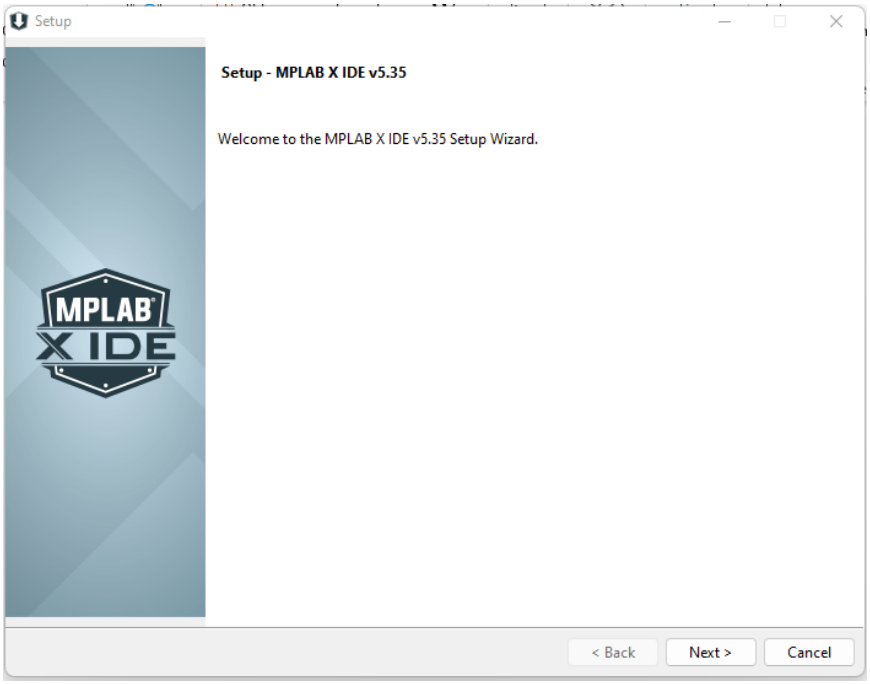


Fig. 6.4 Proceso de instalación, ventana de inicio

5. Dar clic en **Next.** Enseguida saldrá una interfaz como la que se muestra a continuación.

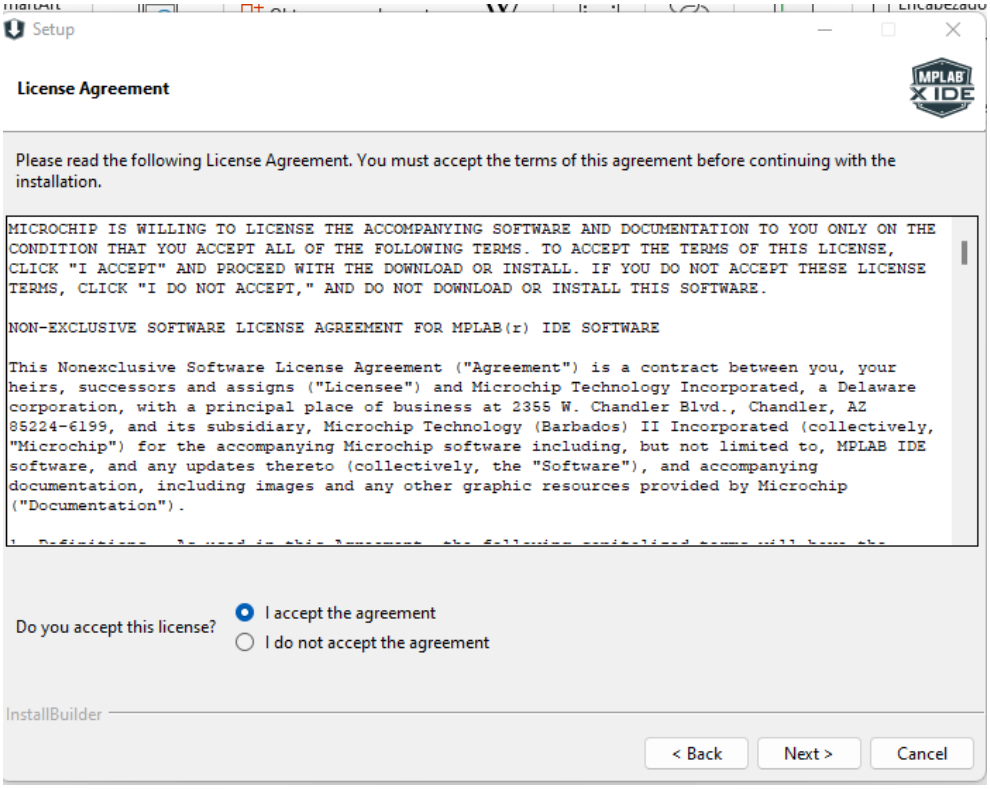


Fig. 6.5 Proceso de instalación, licencia de uso

Solo es activar la casilla de aceptar los términos de licencia y privacidad y enseguida dar clic en **Next.**

6. Después aparece una interfaz como está.

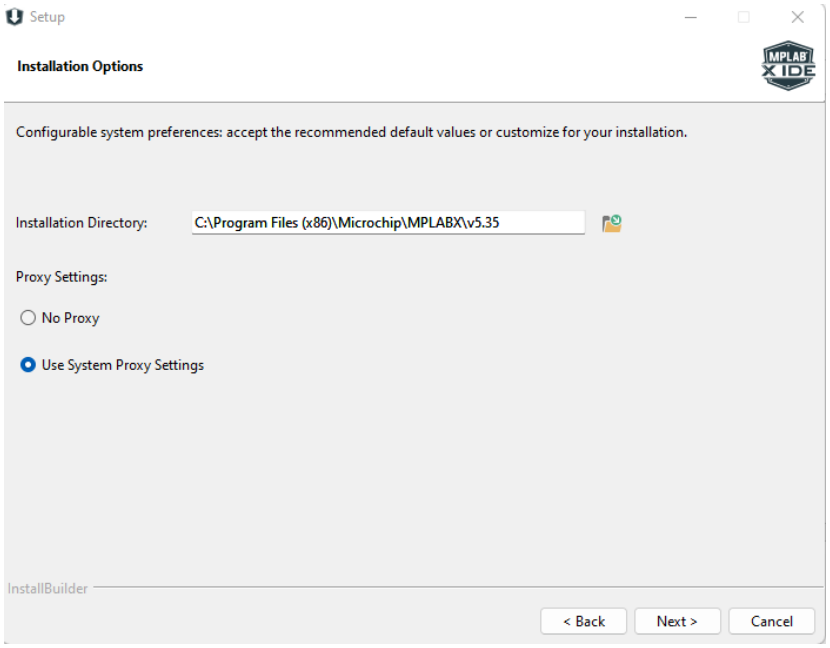


Fig. 6.6 Proceso de instalación, directorio de instalación.

Solo son las opciones de instalación. Es donde se ubicarán los archivos del programa. Es recomendable dejar la configuración que da por defecto. Enseguida clic en **Next.**

7. Luego saldrá la pantalla como se muestra en la imagen siguiente.

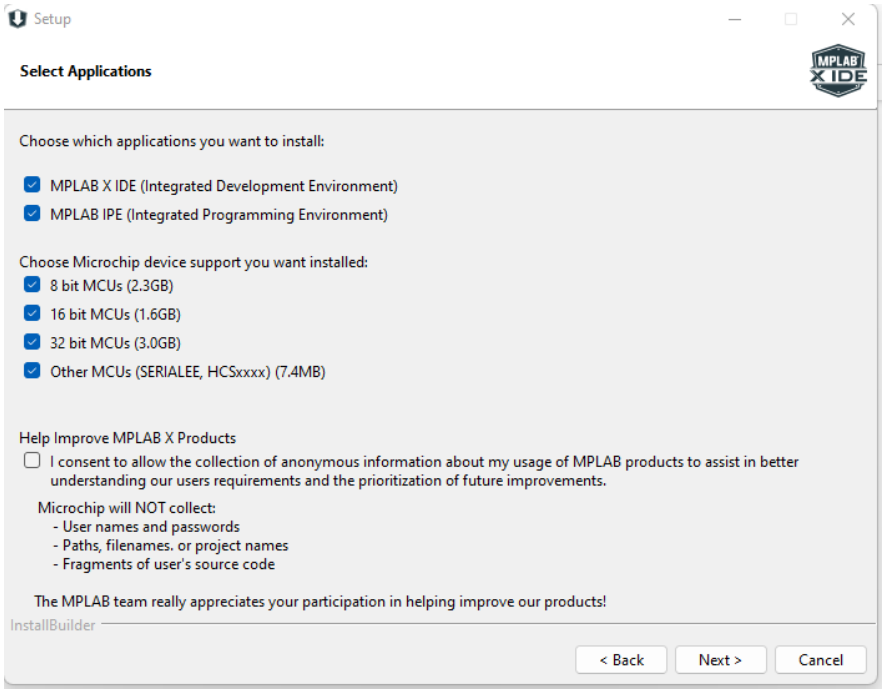


Fig. 6.7 Proceso de instalación, selección de aplicaciones.

Está ventana es importante ya que se establece la configuración de cómo se usará el programa, es por eso por lo que se deja activado el IDE y el IPE. El IDE es el entorno de desarrollo donde se escribirá el programa del microcontrolador. El IPE significa el entorno para programar o quemar el PIC a través del PICKit.

Las opciones de 8, 16, 32 y others MCU, es para establecer los microcontroladores que se van a ocupar para programar. No está demás dejar los de 16 y 32 bits, aunque solo se ocupe el de 8 bits. Pero serviría para futuros proyectos. Después deshabilitar de brindar ayuda a microchip para mejoras, solo es para apoyar a los desarrolladores para mejorar el software o si detectan algo inusual. Ahora clic en **Next.**

8. Solo dirá que empezará la instalación del programa y que si damos Next comenzará la descarga del programa. Dar clic en **Next** si ya está todo listo para la instalación.

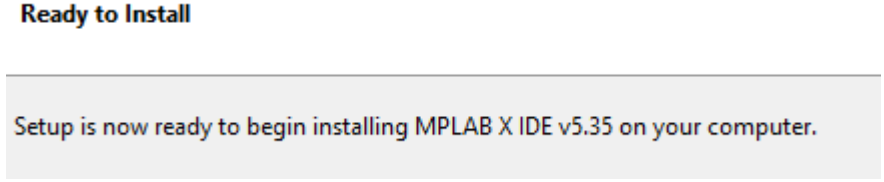


Fig. 6.8 Proceso de instalación, comenzará la instalación.

Comenzará la descarga de los archivos de MPLAB.

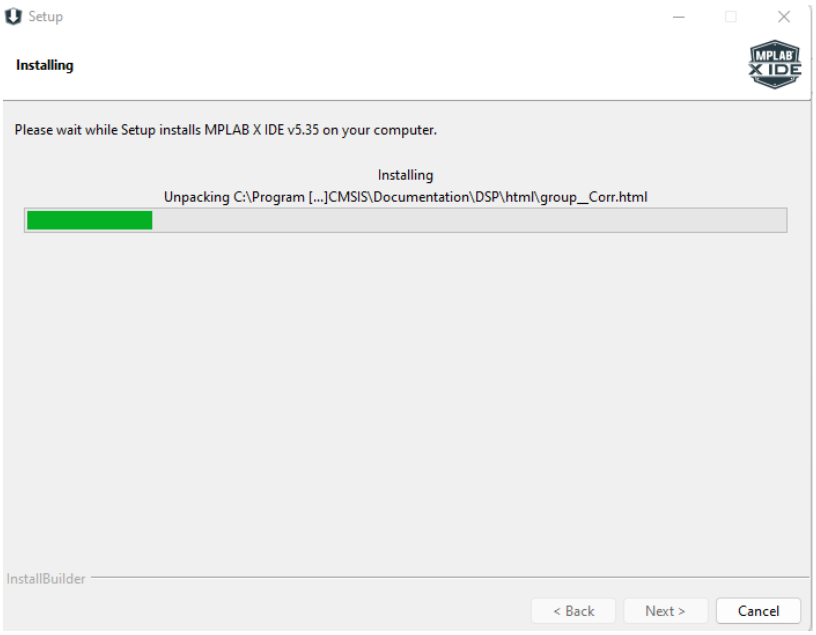


Fig. 6.9 Proceso de instalación, instalación de archivos y paquetes.

9. Cuando ya está listo tal vez aparecerá una ventana de la seguridad de Windows avisando que si de verdad se desea instalar el software. Simplemente dar clic en **Instalar.**

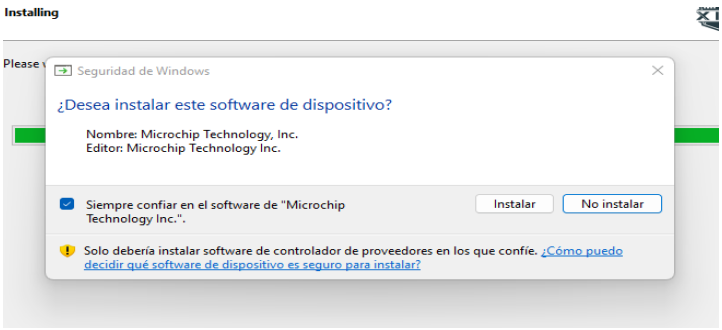


Fig. 6.10 Proceso de instalación, confirmación de instalación.

Esto quizá salga un par de veces, si no aparece no hay de que preocuparse.

10. Cuando acabé la instalación aparece una última ventana habrá que poner las opciones como se muestra a continuación.

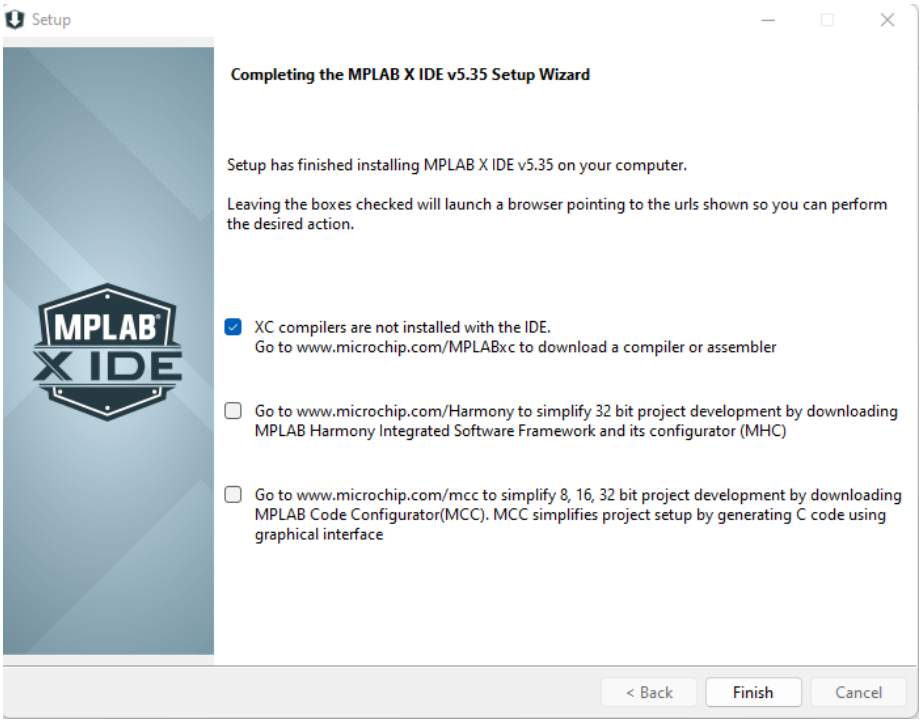


Fig. 6.11 Proceso de instalación, herramientas extra.

Aún no se instala el compilador, por lo tanto, dejar activa la primera casilla, ya que redirige a la página de donde instalarlo.

11. Clic en **Finish,** aparece la página de microchip en el navegador que se tenga predeterminado, el software para el compilador es el que aparecerá.

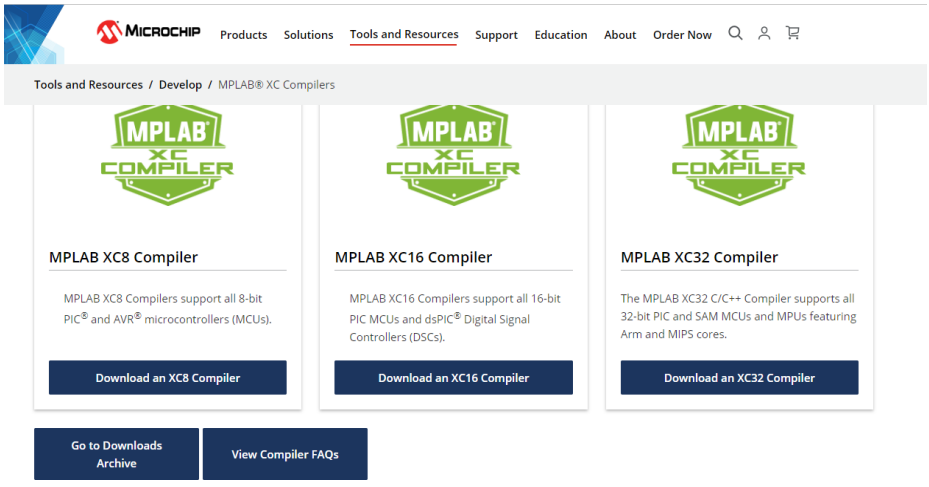


Fig. 6.12 Proceso de instalación, Selección del compilador.

¿Qué significa el Compilador? Recordar que es el encargado de traducir el código que se esté programando para el PIC en hexadecimal, y crea la extensión **.hex** así mismo detecta si el código en el IDE está bien escrito. El archivo **.hex** es el que se carga (quema) al microcontrolador por medio del PICKit.

MPLAB® XC Compilers | Microchip Technology

12. En esta postura se descargará solo el compilador para los PIC de 8 bits. Si se desea instalar los demás por futuras tareas no hay problema. Entonces dar clic en **MPLAB XC8 Compiler.**

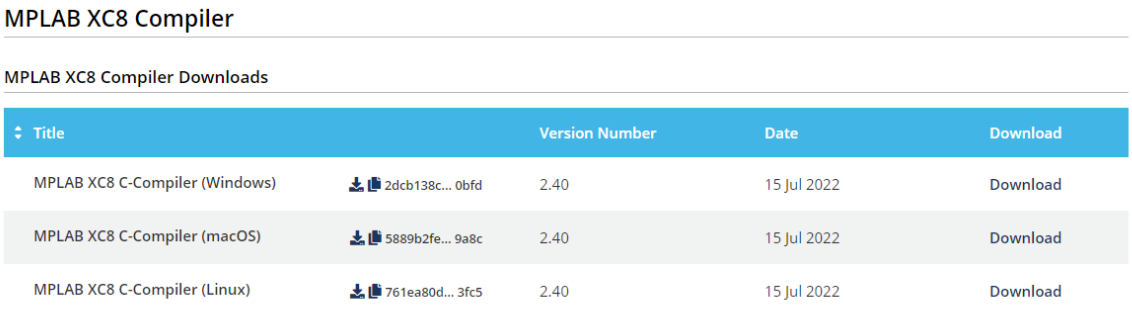


Fig. 6.13. Proceso de instalación, Selección del compilador para Windows.

Se abre una nueva interfaz donde solo hay que ubicarse en la versión de Windows y clic en **Download.**

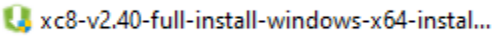


Fig. 6.14 Proceso de instalación, archivo instalador del compilador.

13. Al dar doble clic en el instalador se abrirá una ventana pidiendo permisos de administrador, la cual solo hay que dar en **Si.**

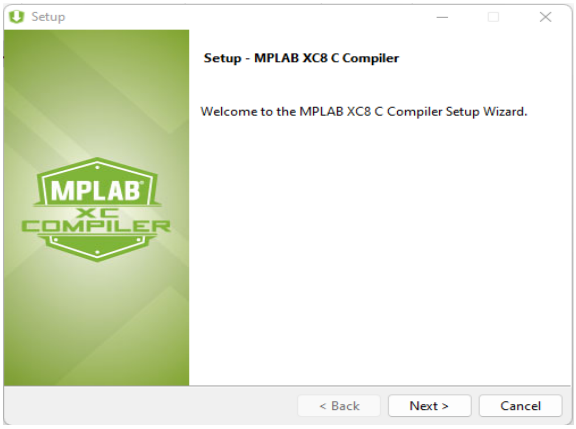


Fig. 6.15 Proceso de instalación, ventana de inicio.

14. Clic en **Next.**

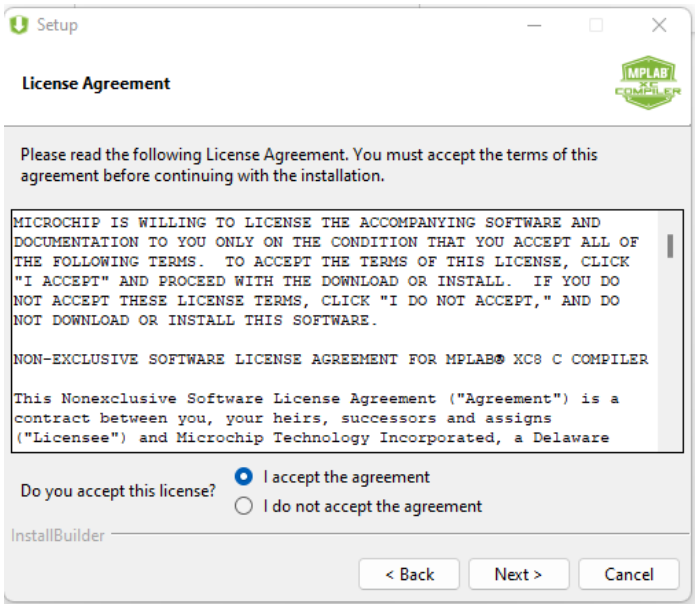


Fig. 6.16 Proceso de instalación, licencia de uso.

En esta ventana hay que aceptar los términos de licencia y la política de privacidad, enseguida clic en **Next.**

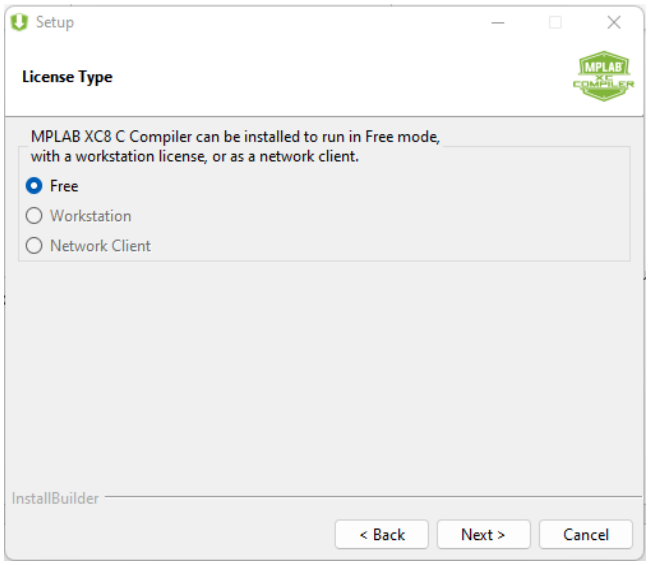


Fig. 6.17 Proceso de instalación, tipo de licencia

La instalación va a hacer para proyectos de escuela o por hobbies, se pueden hacer aplicaciones robustas sin necesidad de tener una cuenta empresarial o cliente especifico, esas cuentas existen para la atención al cliente o usuario que necesitan alguna capacitación o licencias para varias estaciones de trabajo etc. Entonces solo activar el modo **Free.** Y clic en **Next.**

15. Lo que sigue será la ruta de instalación del programa. Clic en **Next.**

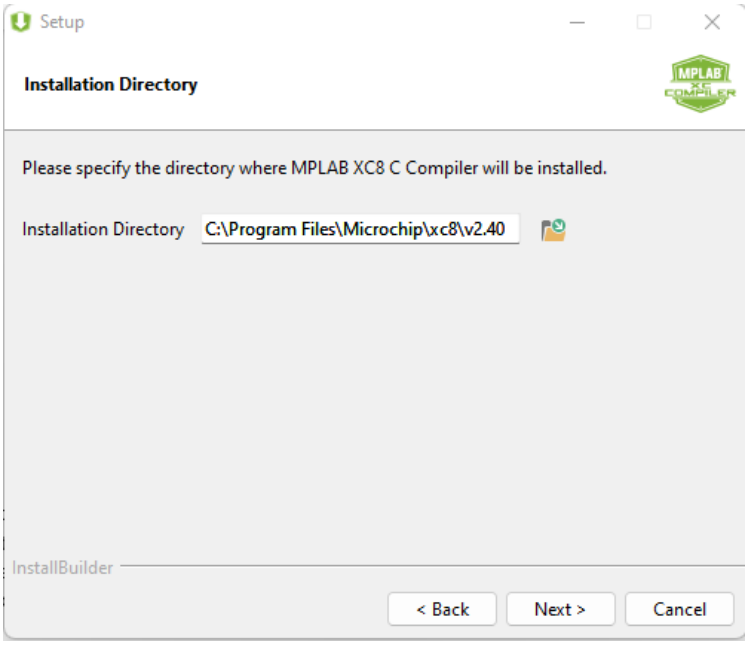


Fig. 6.18 Proceso de instalación, ubicación de instalación.

16. Si se activa el primer check significa que el programa estará disponible para todos los usuarios de la PC, esto si se tienen varias cuentas en una misma PC, si no se activa solo aparecerá en la cuenta del administrador del PC.

El segundo Check sirve para crear una variable path. Hay que recordar que este tipo de variables utilizadas por Windows, permiten acceder a archivos del programa por medio de la ventana de consola, la cual no se es necesario en esta ocasión, dejarla desactivada. Y clic en **Next.**

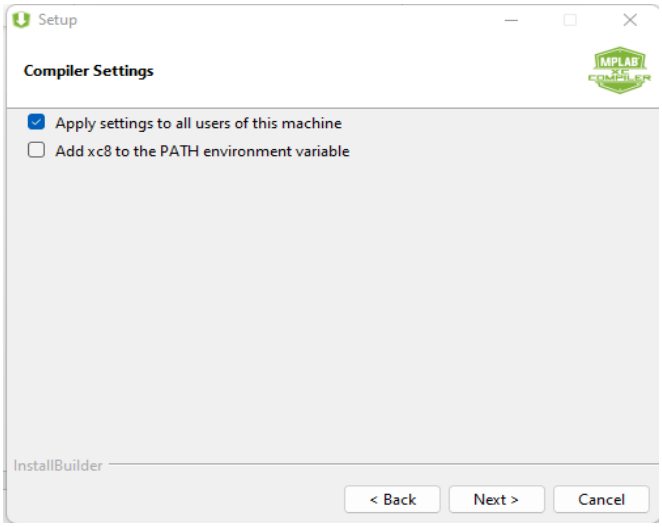


Fig. 6.19 Proceso de instalación, configuración para los usuarios de la PC.

17. Dirá la ventana nueva que si se da clic en Next comenzará la instalación, clic en **Next** para proceder con la instalación.

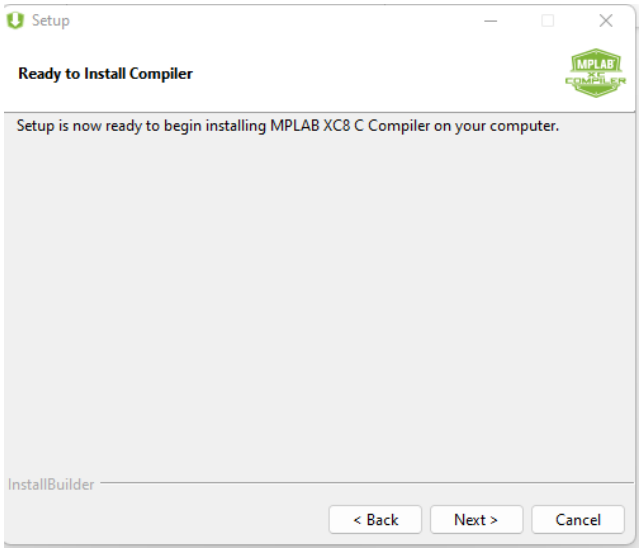


Fig. 6.20 Proceso de instalación, comenzará la instalación del compilador.

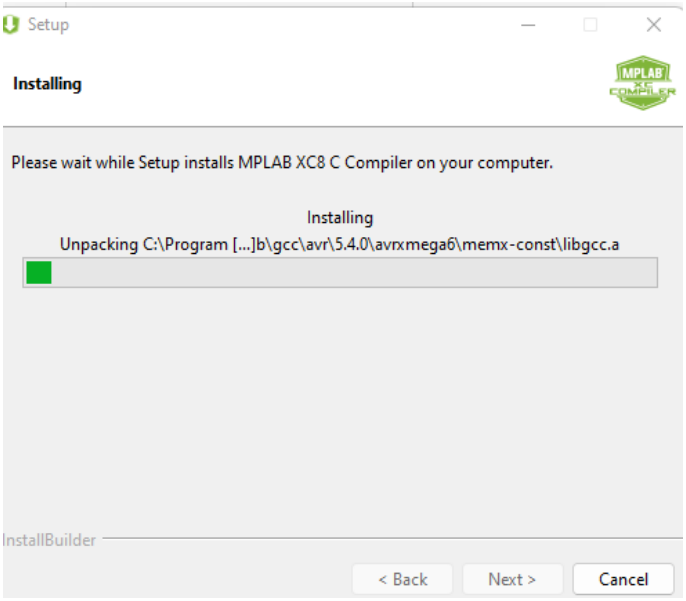


Fig. 6.21 Proceso de instalación, descarga de paquetes.

18. Saldrá una interfaz diciendo sobre adquirir la licencia o probar la licencia durante 60 días para el uso que se desee utilizar el programa, mientras dar clic en **Next** para una instalación gratuita.

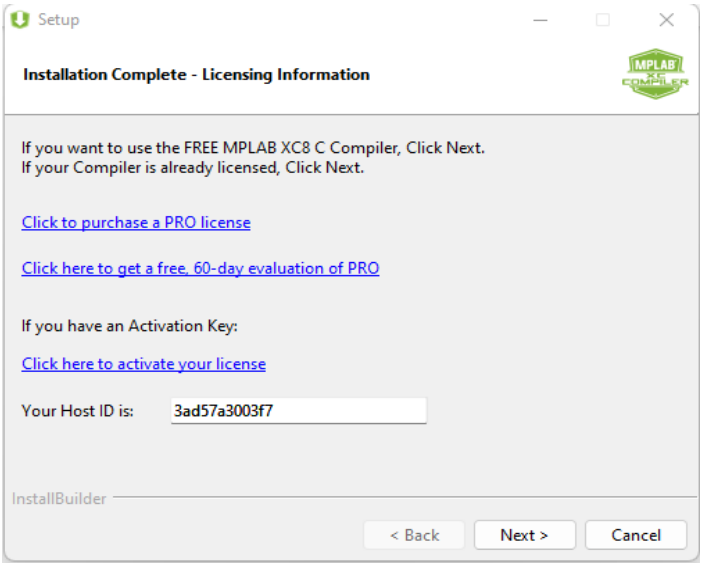


Fig. 6.22 Proceso de instalación, tipo de licencia a adquirir.

19. El último paso es dar clic en **Finish.** De esa forma ya se tendrá el programa MPLAB listo para usarse.

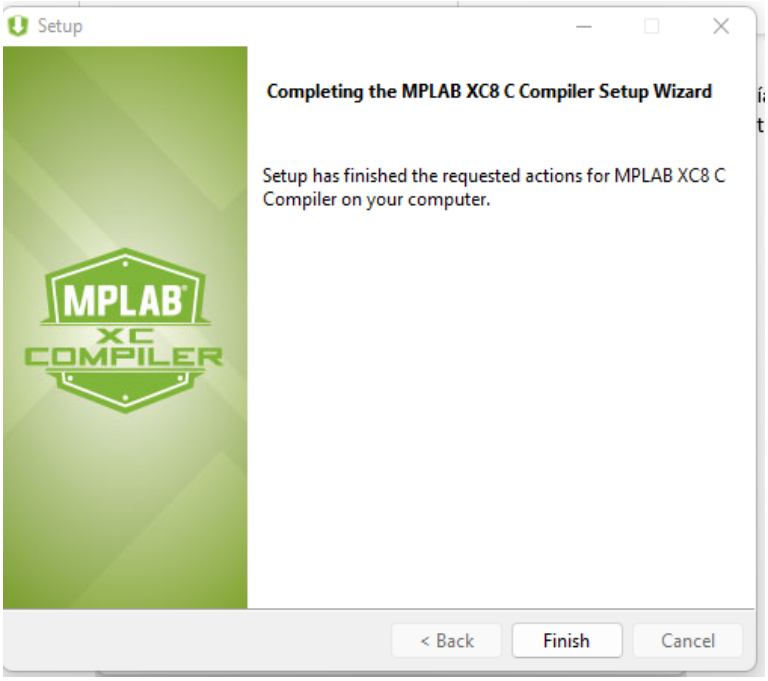


Fig. 6.23 Proceso de instalación, ventana final.

Ya que se instaló el entorno de desarrollo hay que saber cómo es la interfaz de usuario y las características de las opciones que brinda el software, en breve se pasará la explicación de cómo se pueden utilizar las funciones principales del programa.

Descripción de MPLAB

Dentro del programa se encontrarán diferentes funciones y opciones que serán de gran utilidad, cabe destacar que este software permite simular entradas y salidas de datos, quizá no tan gráfico como en PROTEUS pero al menos se verá la información digital, también ayuda bastante la identificación de las instrucciones, variables, literales y registros, con colores amigables para la vista humana.

Al abrir el programa se mostrará la siguiente ventana de inicio.

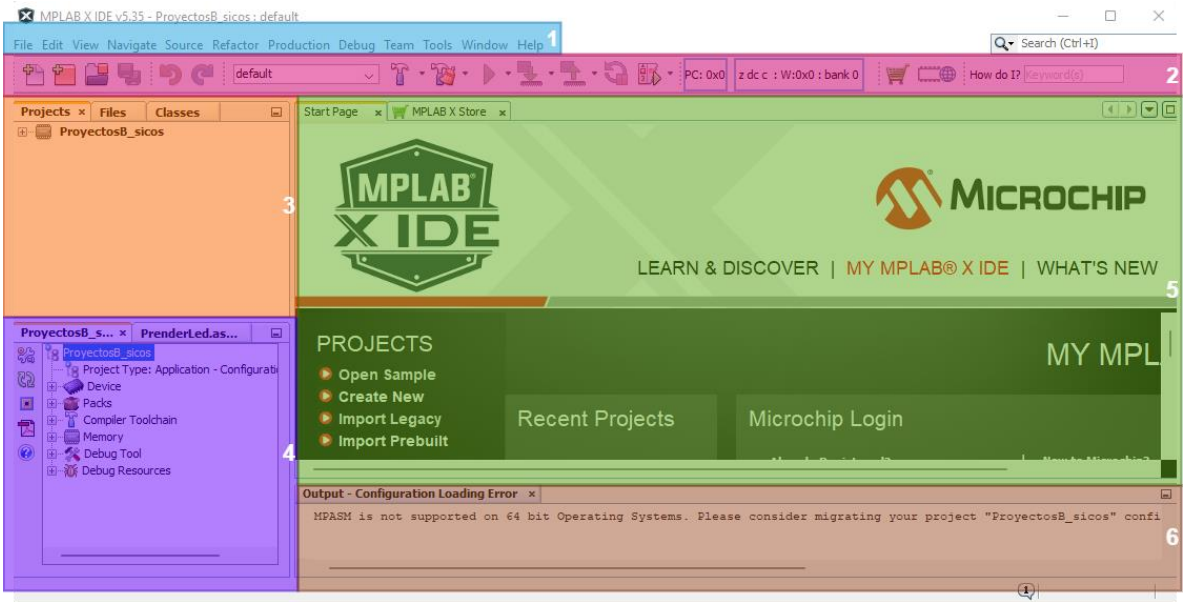


Fig. 6.24 Entorno de desarrollo del MPLAB.

1.- Cinta de opciones. Aquí se encontrarán funciones de compilación del código escrito, también se encuentra la información para crear, importar y guardar el proyecto, así mismo se encuentra la opción para agregar ventanas como la ventana stimulus para simular entradas con un clic y la ventana watch para ver las variables locales y globales.

2.- Son funciones rápidas de la cinta de opciones, como un atajo, también se encuentran funciones de deshacer y rehacer la operación, como un ctrl+z, de igual manera se encuentran los iconos como el atajo de compilar y correr el programa por líneas.

3.- Esta pequeña ventana es la encargada de decir en que proyecto se está trabajando y específicamente en que extensión de archivo, es decir si se está codificando en un archivo .asm o .c

4.- Aquí se encuentran las propiedades del proyecto, se puede mostrar el estado de la memoria RAM y FLASH, también se puede observar el tipo de PIC en el que se está trabajando, incluso se puede redirigir al datasheet del PIC en formato PDF.

5.- El entorno de desarrollo o codificación del programa, dentro de este espacio residirá la configuración, las instrucciones o tareas que se le cargarán al microcontrolador.

6.- Es una ventana de estado para saber el comportamiento de una variable, ya sea local o global, también muestra el estado de compilación e información sobre algún error y en qué línea del código se localiza.

Creación de un nuevo proyecto

Una vez abierto el programa MPLAB v5.35 en la pestaña o cinta de opciones al dar clic en "File", se desplegarán varias funciones, "New proyect" ayudará a crear un nuevo proyecto.

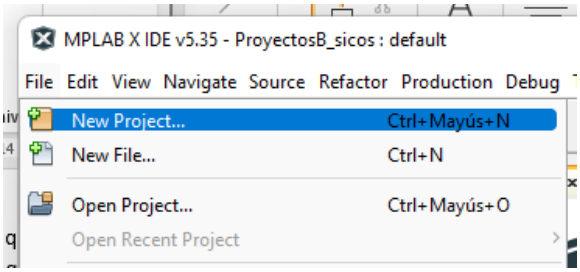


Fig. 6.25. Creación de nuevo proyecto.

Se abrirá una nueva interfaz y se seleccionará la carpeta "Microchip embedded". Solo hay que dar clic en "Next".

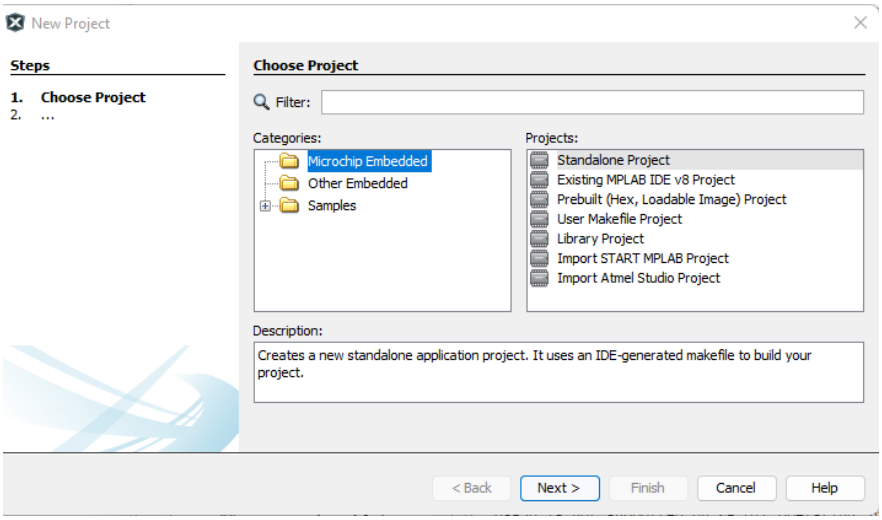


Fig. 6.26 Elección de tipo de proyecto.

Después hay que seleccionar una familia y un dispositivo, en este caso la familia es de los PIC de 8 bits y en específico el PIC16F628. Clic en "Next".

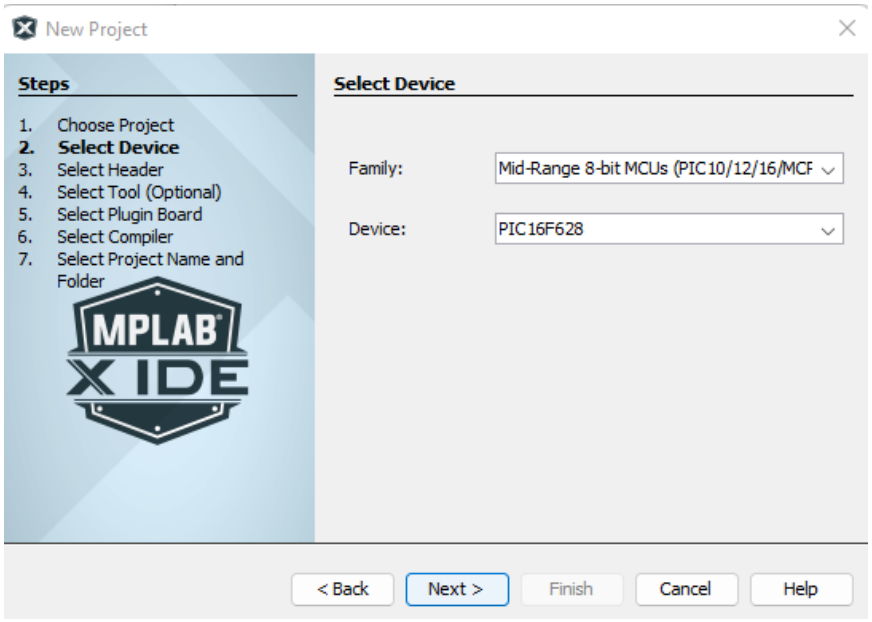


Fig. 6.27 Selección del microcontrolador.

Luego hay que seleccionar una herramienta de hardware, en otras palabras, para grabar el PIC que equipo se usará entre ese listado, por el momento se simulará, en el capítulo 8 se habla de cómo grabar el CI (Circuito integrado). Clic en “Next”.

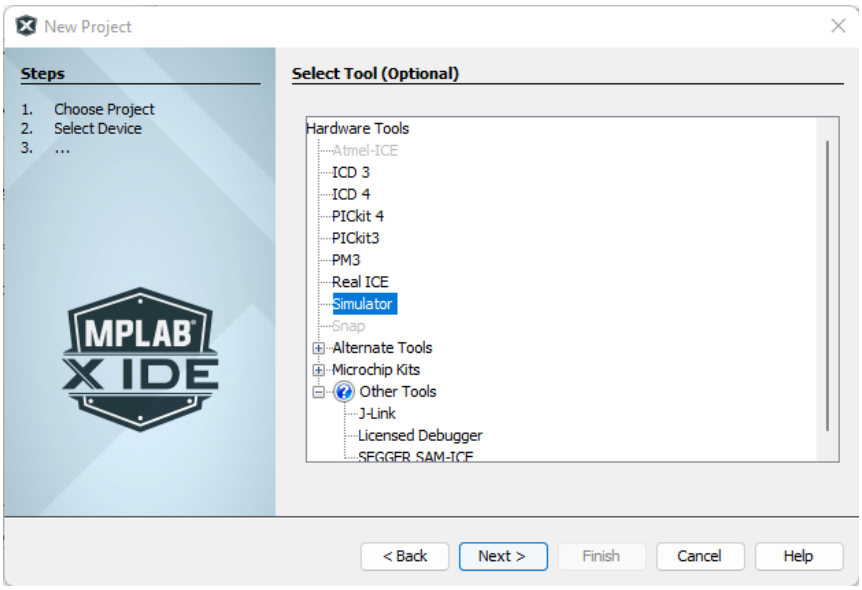


Fig. 6.28 Selección de herramienta (hardware).

Seleccionar el compilador “mpasm” y clic en “Next”.

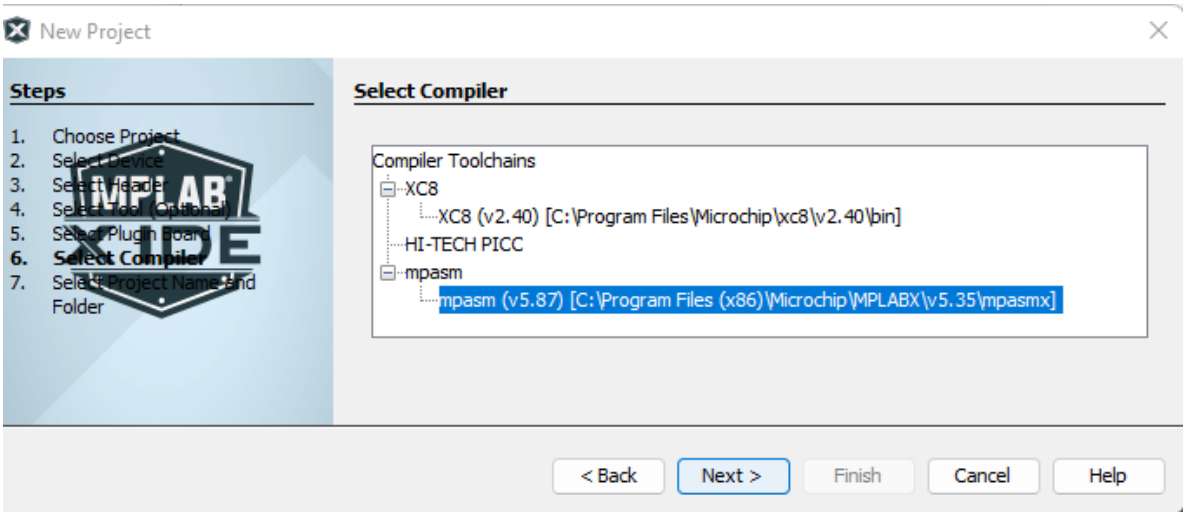


Fig. 6.29 Selección del compilador

Se pondrá el nombre del proyecto, así como la localización donde se guardará, junto con los archivos que se seguirán creando. Clic en “Finish”.

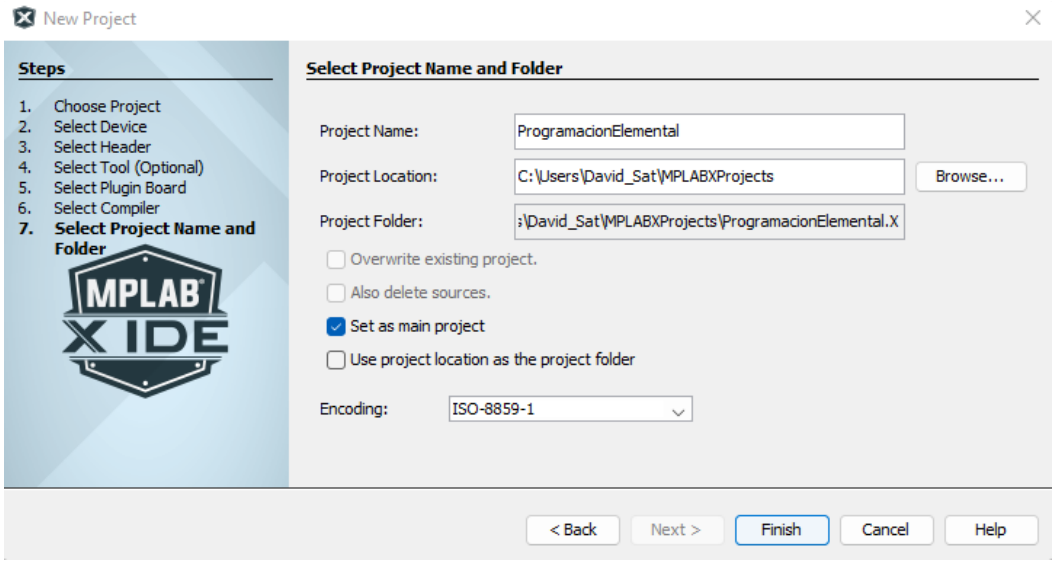


Fig. 6.30 Nombre del proyecto y ubicación.

Con esto ya se habrá creado el proyecto.

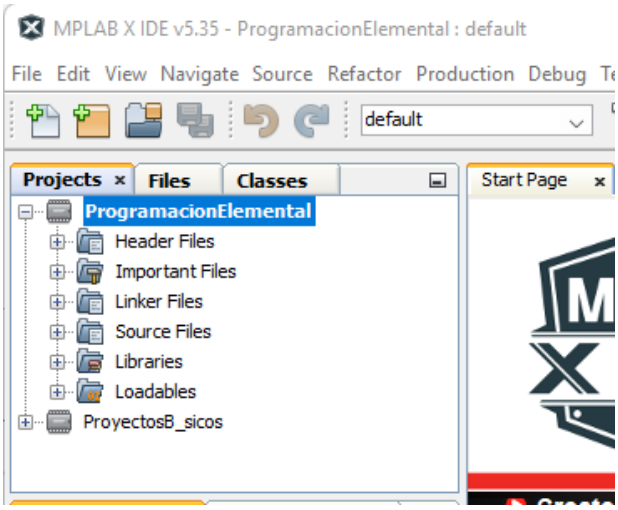


Fig. 6.31 Visualización de archivos al crear un nuevo proyecto.

Creación archivo .asm

Ya que se creó el proyecto, al desplegarlo se encontrarán subcarpetas, específicamente en "Source Files" residirán los archivos .asm. Solo hay que seleccionar esa carpeta, clic derecho y en las opciones desplegables seleccionar "New" y después "Assembly file.asm"

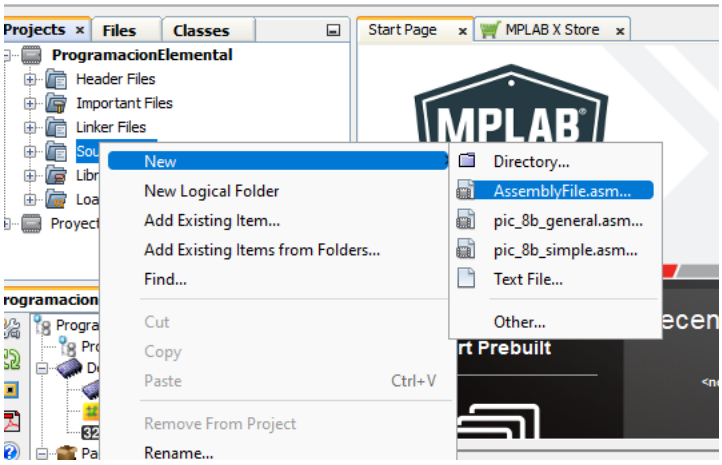


Fig. 6.32 Nuevo archivo ensamblador.

Se abrirá una ventana pidiendo el nombre del archivo y la ubicación donde se guardará. Clic en "Finish"

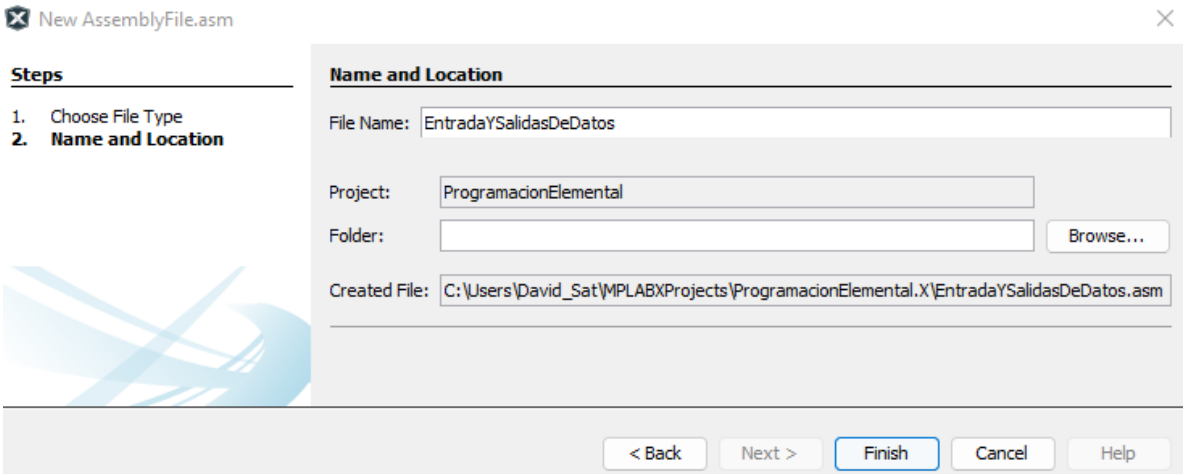


Fig. 6.33 Nombre y localización del archivo ensamblador.

Ya ha quedado el entorno, listo para empezar a codificar.

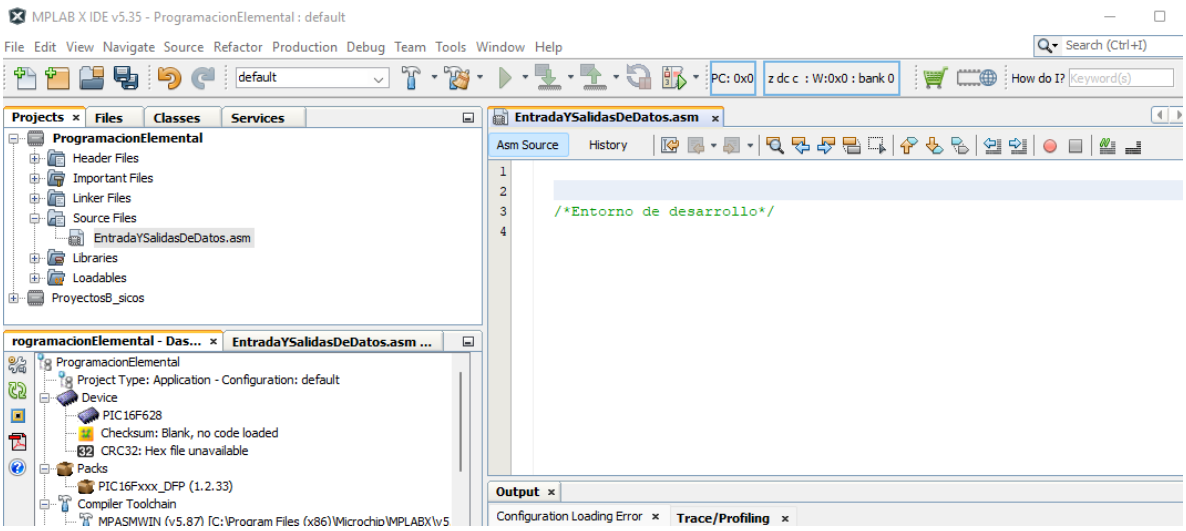


Fig. 6.34 Entorno listo para programar.

Dentro de este proyecto se pueden tener varios programas .asm, algo importante que se debe tomar en cuenta es que, si se desea crear más archivos, se deberá ser precavido cuando se depure un programa porque MPLAB no sabrá cual de todos es el que deberá ejecutar y por default siempre será el primero en la lista. Para depurar el que se requiera solo hay que dar clic derecho sobre el programa que no se requiera depurar y excluirlo de dicha depuración, como se muestra en el ejemplo siguiente.

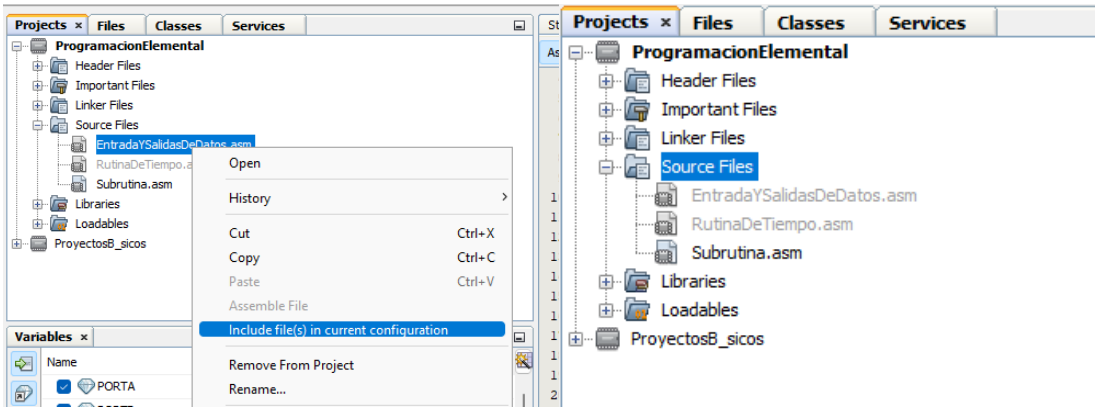


Fig. 6.35 Exclusión de un archivo ensamblador.

Los archivos excluidos quedarán como de fondo o color gris, así MPLAB solo depurará "Subrutina.asm" en este ejemplo.

Ventana de variables

Dentro de la cinta de opciones "Window" tiene al submenú que se ocupará, será "Debuggin" y dentro de esté se encuentra "Variables" y se seleccionará está interfaz.

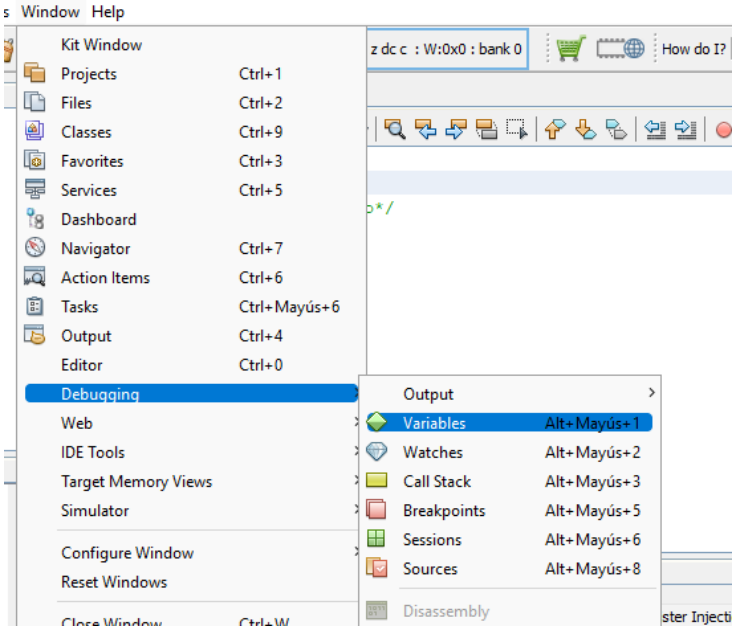


Fig. 6.36 Agregar ventana de variables al entorno.

Una vez abierta al dar doble clic izquierdo sobre la celda "<Enter new watch>" aparece una ventana mostrando los SRF (Registros de Función Especial), así mismo los símbolos globales que se representan como las variables que se crean dentro del entorno de desarrollo.

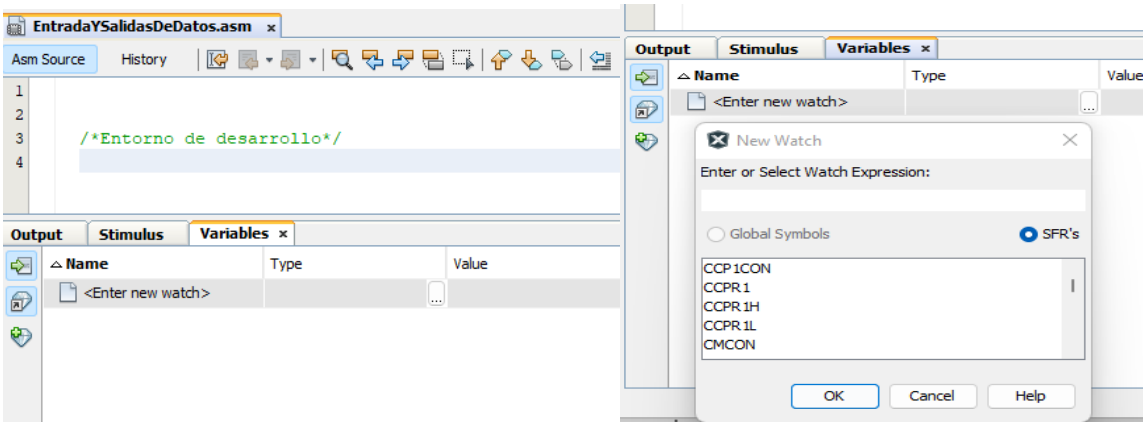


Fig. 6.37 Añadir variables locales o globales.

Añadir el registro PORTA, PORTB, etc. Los que se muestran en la imagen siguiente, si se desea agregar más no hay ningún problema.

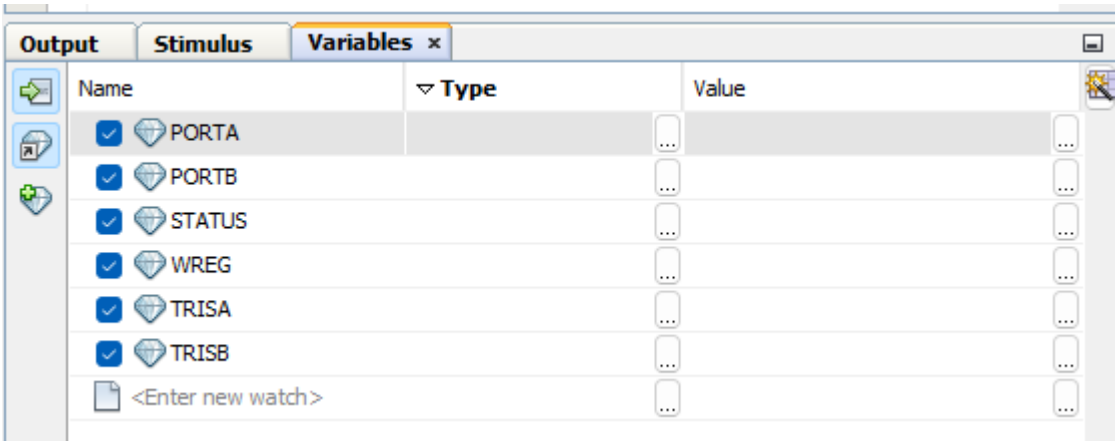


Fig. 6.38 Ventana de variables.

Para separar las ventanas o ponerlas en otra parte de la interfaz principal, basta con mantener seleccionada la ventana y arrastrarla a uno de los extremos de la pantalla.

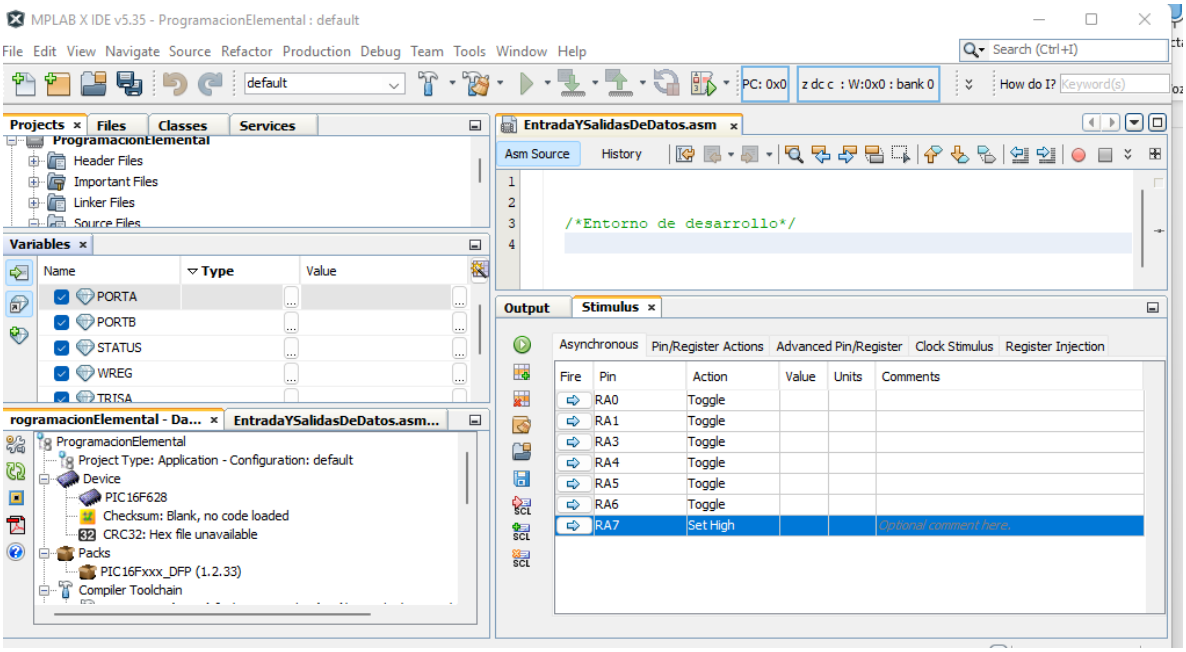


Fig. 6.39 Ventanas separadas.

Ventana Stimulus

Dentro de la cinta de opciones “Window” al dar clic se desplegará la opción que se está buscando, “Simulator” es la indicada y dentro de este submenú está la ventana “Stimulus”. Al dar clic en “Stimulus” se añadirá la ventana en la parte inferior del programa.

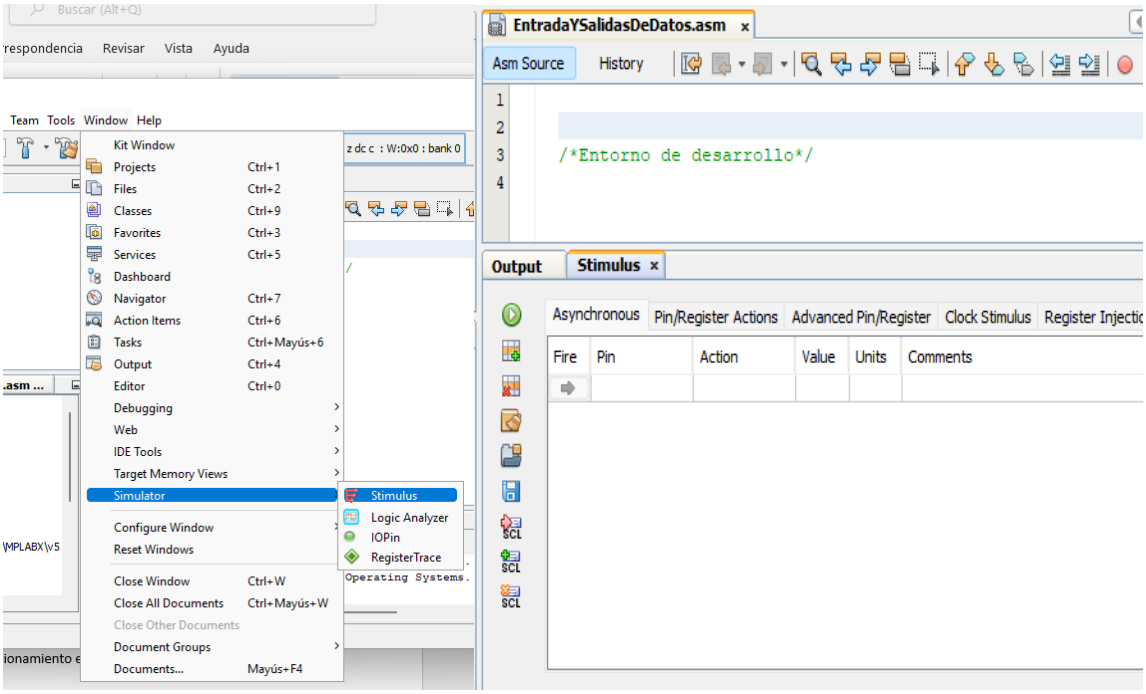


Fig. 6.40. Añadir ventana Stimulus.

Ahora bien, para agregar una entrada solo hay que dar clic en la celda debajo de “pin” y saldrá la cinta de que pines están disponibles para la entrada de datos.

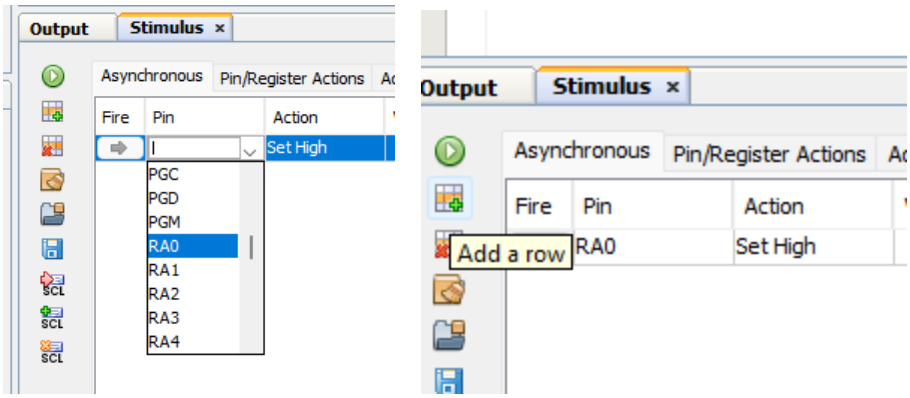


Fig. 6.41 Añadir pin del microcontrolador que se requiera interactuar.

Se seleccionará desde RA0 hasta RA7 y en la columna “Action” la opción “Toggle” servirá para que cuando se dé un clic en la columna “Fire” este funja como un switch. Es decir, cada vez que se presione, este cambiara de estado high a low o viceversa.

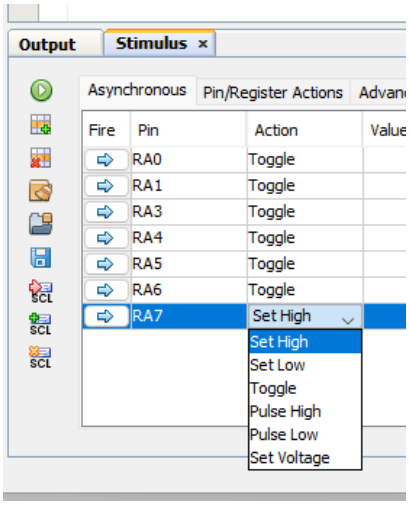


Fig. 6.42 Acción de cada pin para el microcontrolador

En futuros proyectos quizá no solo el Puerto A servirá como entrada, también se pueden seleccionar otros pines.

Configuración de bits en PC (Contador de programa)

Esta es una parte importante al iniciar con la programación sobre el PIC16F628 pues a partir de aquí se establece la configuración principal. Tal y como lo describe el registro PC del capítulo 4 hay que prestar atención sobre que se habilita y que no. Por consiguiente, se procede a abrir la venta de configuración de bits.

Clic en la cinta de opción “Production” al desplegarse la función “Set Configuration Bits” habrá que seleccionarla para que salga la ventana correspondiente.

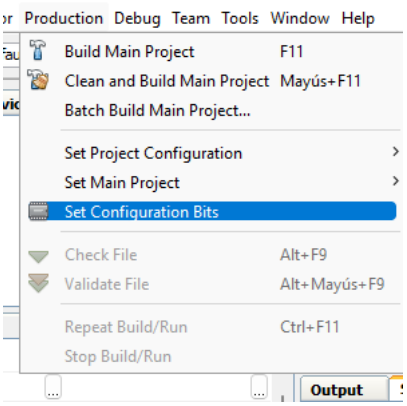


Fig. 6.43 Añadir ventana de configuración de bits.

La función del oscilador se trabajará internamente, esto quiere decir que para configurarlo de esta manera solo hay que decirle al microcontrolador que los pines 15 y 16 funcionarán como entradas y salidas digitales. Él internamente hará los ajustes para interpretar que debe ocupar un oscilador interno. En el Capítulo 4 en el registro PCON se especifica que banderas activan el oscilador de 4Mhz y el de 37Khz.

El WDT se dejará activo, Recordar que el perro guardián sirve para vigilar la ejecución del programa así evita que se quede pasmado o bloqueado. Mientras, las demás funciones no se ocuparán, si en algún punto se necesitan habilitar en algún programa se hará saber.

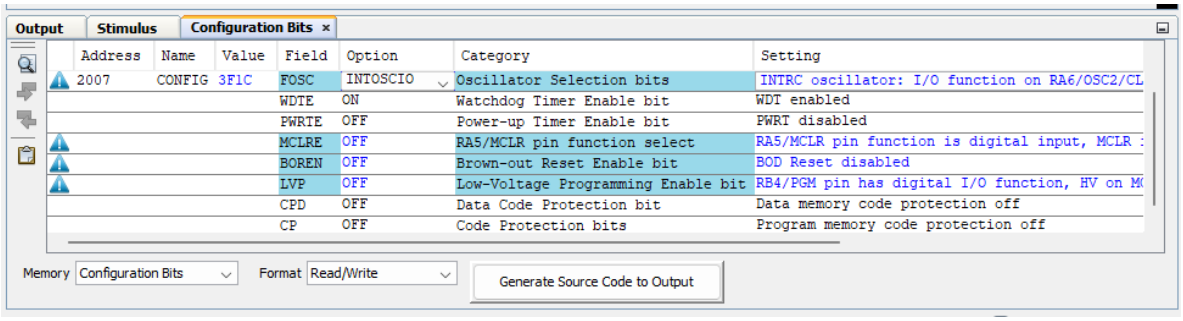


Fig. 6.44 Ventana de configuración de bits.

Al dar clic en “Generate Source Code to Output” (este botón se encuentra en la parte inferior de la interfaz) automáticamente el programa se encargará de lanzar un código qué, al momento de quemar o cargar el programa al PIC sabrá que configuración debe hacer. Este código se copiará y se pegará en el entorno de desarrollo del programa, tal y como lo muestra la siguiente imagen.

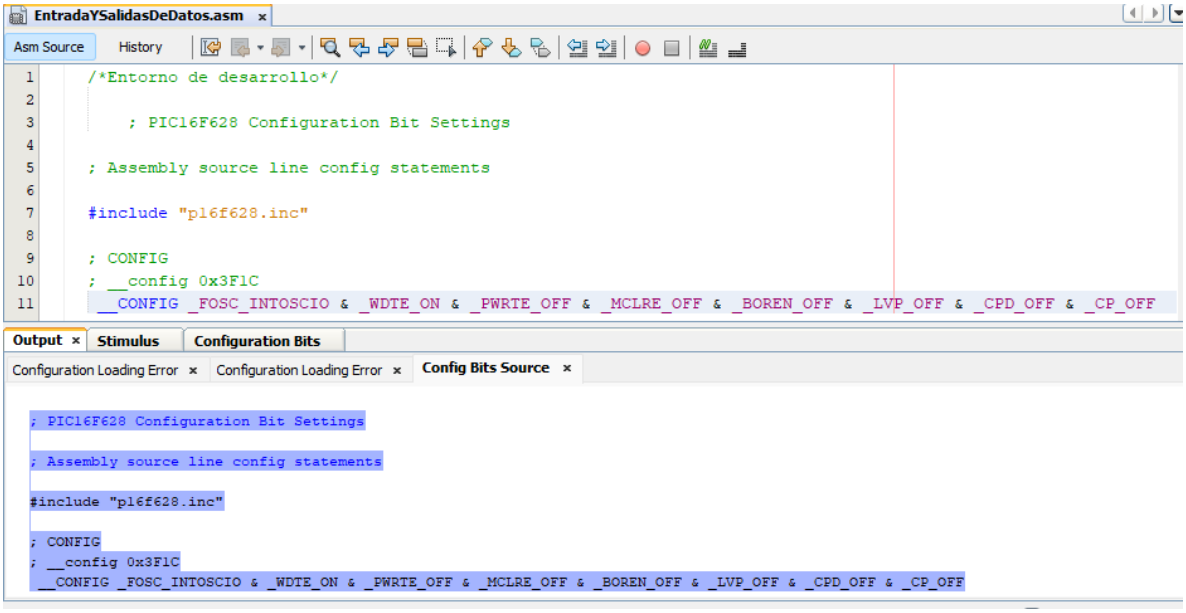


Fig. 6.45 Generación de código para configuración de bits.

Debug o Depuración de un programa.

Cuando se tenga un programa terminado hay que verificar que este bien escrita su sintaxis, es decir, si hay algún error, el compilador dirá en que línea está el problema. Para compilar hay que dirigirse al siguiente icono, que se encuentra en las funciones rápidas.

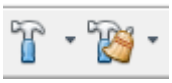


Fig. 6.46 Icono de compilación de programa.

El primero solo es compilar sea como este el programa, es decir no reinicia los registros SFR pero que sucede si se requiere hacerlo, por esa razón está la segunda opción donde está el martillo y la escobita, es compilar y limpiar. En el ejemplo siguiente se verá que en la ventana "Output" se muestran los resultados de compilación. Cuando aparezca "BUILD SUCCESSFUL" quiere decir que el código escrito no tiene errores.

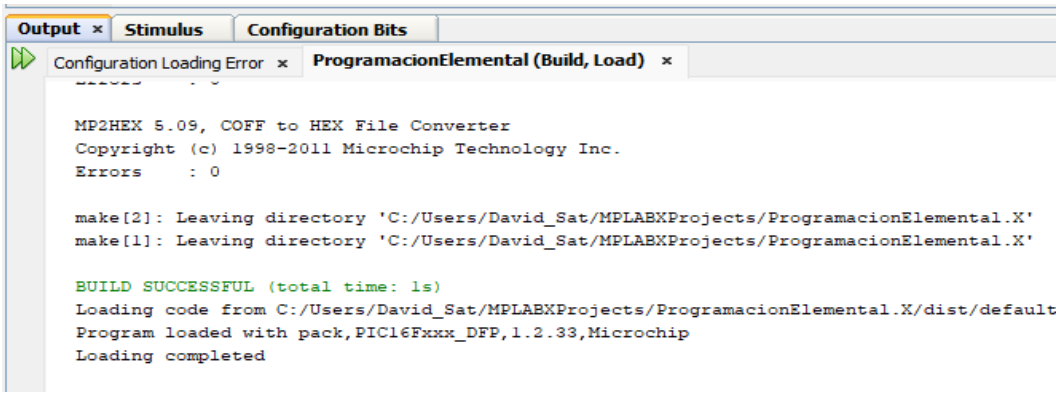


Fig. 6.47 Resultado de un programa compilado satisfactoriamente.

¿Cómo se vería un error? En el siguiente ejemplo se muestra.

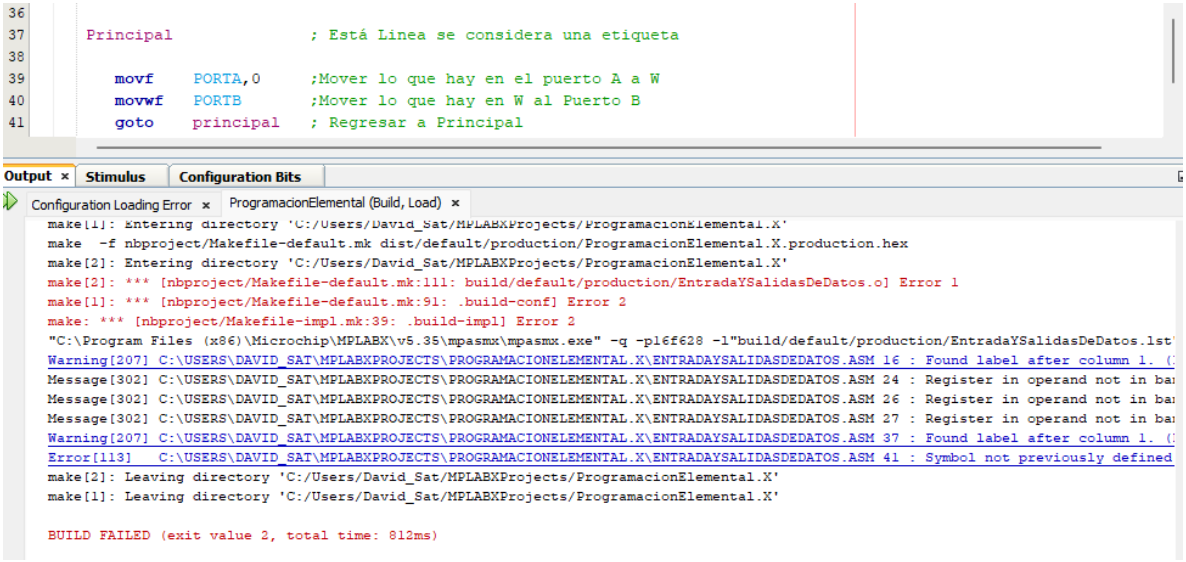


Fig. 6.48 Resultado de un programa compilado erróneamente.

Se aprecia que en la línea 41 está el error ya que la etiqueta “Principal” se escribe con la P en mayúscula y cuando la instrucción goto lee una minúscula, no encuentra lo que está buscando y lanza un error. Si no se conoce el error, se puede dar clic en el hipervínculo que se muestra a continuación.

Esto servirá para encontrar en que línea se encuentra el error. Automáticamente MPLAB dirá la ubicación del dato erróneo.

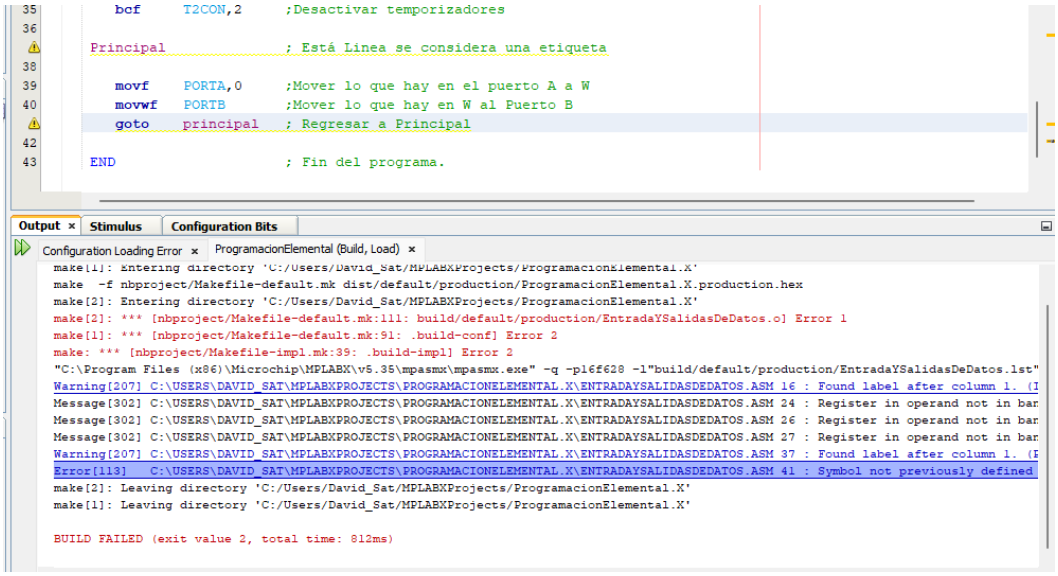


Fig. 6.49 Localizar la línea de código donde se encuentra el error.

El siguiente icono permitirá depurar el programa y se podrá percatar que se puede jugar entre líneas del código, es decir se puede correr el código y ponerlo en pausa para que con la tecla F7 se den saltos de línea manualmente, de esa forma se verá cómo se comporta un registro o una variable en el momento.

Fig. 6.50 Icono de depuración de programa.

Cuando se da clic se aparecen los botones de stop, pause, reset, play, entre otros.

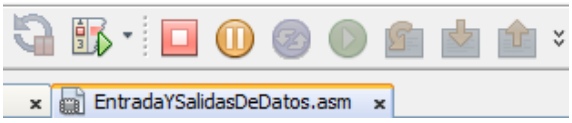


Fig. 6.51 Iconos de control de flujo de programa en ejecución.

Ahora cuando se dé la pausa en el programa se podrá notar en que línea se quedó pausado. Con una flecha verde en la numeración de línea de código.

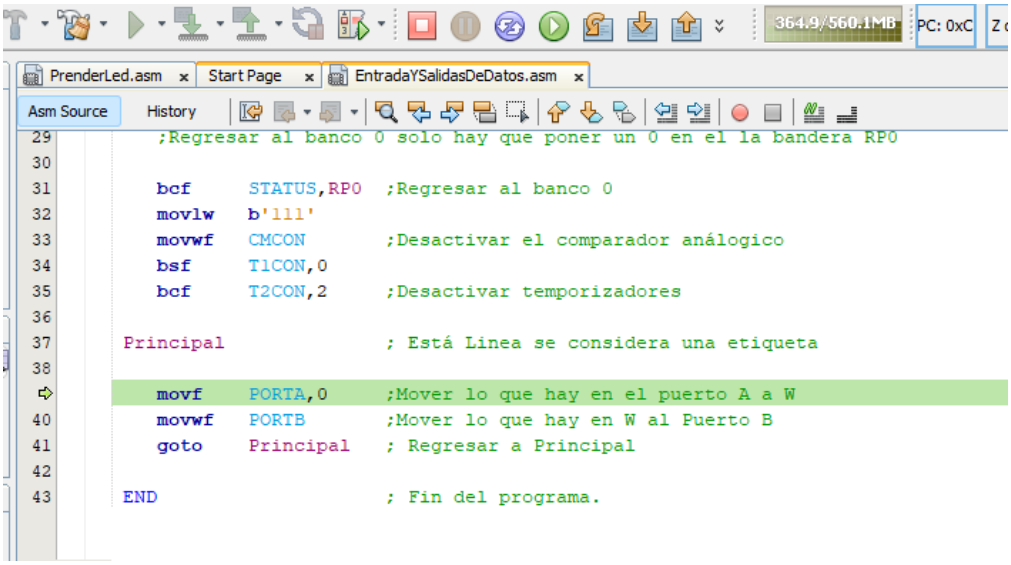


Fig. 6.52 Simulación del programa en ejecución línea por línea.

De esta forma se podrá simular paso a paso, línea a línea el comportamiento del código o programa que se escribió.

Declaración de variables

Las variables son muy importantes dentro del programa en desarrollo, puesto que dentro de ellas se alojan valores de diferentes tipos, por ejemplo, un número, un carácter o una letra. De esta forma cuando se requiera un valor que cambie durante la ejecución del

programa y se vaya almacenando al mismo tiempo para utilizarlo después, las variables entran en papel como contenedores de estos valores cambiantes.

En la Fig. 7 se aprecia el espacio de los registros de propósito general o la memoria de datos que se ocupará para el almacenaje de variables, este espacio comienza en 21h puesto que el 20h lo ocupa un SFR, sabiendo esto se creará una variable LED, en el ejemplo siguiente se muestra el lugar correcto de la declaración.

```
3    ; Assembly source line config statements
4
5    #include "p16f628.inc"
6
7    ; CONFIG
8    ; __config 0x3F1C
9     __CONFIG _FOSC_INTOSCIO & _WDTE_ON & _PWRTE_OFF
10
11   LED     EQU h'21'
12
13    Org 0                  ; Está línea significa q
14
15   Inicio                  ; Está Linea se considera
```

Fig. 6.53 Lugar para declarar variables.

¿Qué pasaría si tienen más variables? Hay una forma para que el mismo programa determine un lugar para las variables, como se muestra a continuación.

```
5    #include "p16f628.inc"
6
7    ; CONFIG
8    ; __config 0x3F1C
9     __CONFIG _FOSC_INTOSCIO & _W
10
11    CBLOCK h'21'
12    LED
13    ENDC
14
15   ;LED    EQU h'21'
16
17    Org 0                  ; Est
```

Fig. 6.54 Código para declarar más de una variable.

De esta manera debajo de LED pueden ir más variables, la ventaja de esto es que ya no se determinarán un lugar en el espacio de memoria RAM manualmente, ya se hará de manera automática.

Simulación de variables

Al depurar el programa quizá las variables no se noten en la ventana, esto porque se debe habilitar una opción del proyecto. Clic derecho en el nombre del proyecto y elegir “Propiertes”.

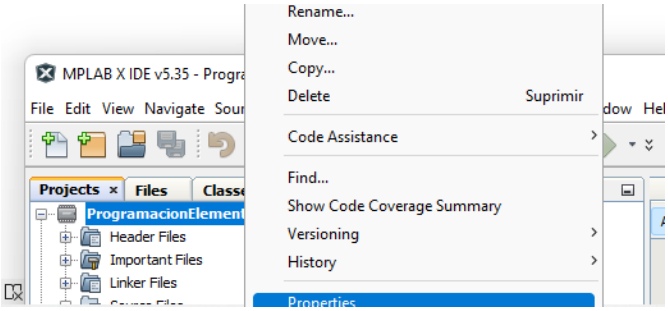


Fig. 6.55 Abrir propiedades del proyecto.

Se abrirá una nueva ventana donde hay opciones de configuración del proyecto. Dirigirse a "mpasm (Global Options)" en el espacio izquierdo de la ventana y después se habilitará la casilla "Build in absolute mode" que está en la tabla de la derecha y clic en el botón "Apply" y luego "Aceptar" enseguida cerrar la ventana.

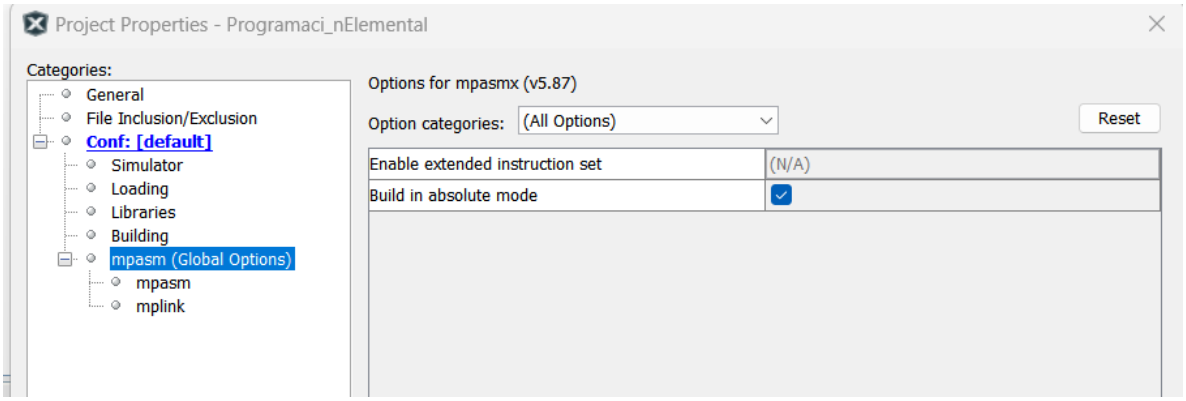


Fig. 6.56 Opciones de configuración de proyecto.

Si en la ventana de añadir variable no se encuentra la que se desea, entonces escribir tal y como se declaró en el programa, a continuación, clic en "OK" y de esta forma se apreciarán las variables que se vayan añadiendo.

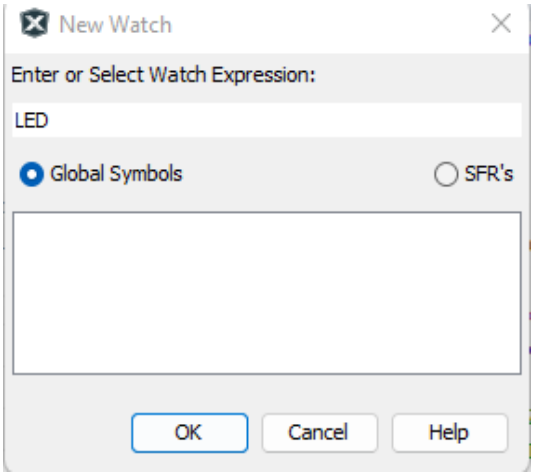


Fig. 6.57 Añadir variables declaradas en el programa.

7 Programación elemental y simulación con proteus

Se abordarán conceptos y temas de nivel básico hasta llegar a un nivel avanzado. Pero ¿En qué lenguaje se programará?

En la actualidad hay lenguajes de alto nivel como Python, Java, C# entre otros. El microcontrolador puede admitir archivos con extensión .asm y .c. El archivo .asm es un formato de un programa compilado de bajo nivel, puesto que se habla con el MCU en lenguaje ensamblador, el .c es un formato del lenguaje C de alto nivel y es más amigable para el entendimiento humano como su programación que el ensamblador. A continuación, se hablarán de las ventajas y desventajas de usar leguaje C y lenguaje ensamblador para la codificación del Circuito Integrado.

Ventajas de usar lenguaje ensamblador.

- Permite mayor control sobre las operaciones que realiza el programador.
- Permite escribir un código más sucinto y por tanto más veloz.
- Para la educación es bastante importante, ya que ensamblador permite conocer a detalle la arquitectura interna del PIC y su funcionamiento entre registros y set de instrucciones.

Desventajas de usar lenguaje ensamblador.

- El mantenimiento del código es más complicado.
- Control sobre los tipos de datos.
- No es portable, debido que depende de la estructura interna del microcontrolador, es decir el código de un PIC no puede correr en otro con diferente arquitectura.

Ventajas de usar lenguaje C.

- Los programas son más fáciles de mantener.
- Proporciona control sobre tipos de datos, así como también la abstracción.
- Es portable. Generalmente un programa escrito en un PIC puede correr en otro PIC de diferente arquitectura con las mínimas modificaciones.
- Entre Python, Java, C++, Basic, el lenguaje C es más rápido y eficiente que otros lenguajes de alto nivel usados también para la programación de microcontroladores.

Desventajas de usar lenguaje C.

- Es menos veloz que ensamblador.
- El código escrito aquí, ocupa más espacio que en ensamblador, sin embargo, los compiladores modernos poseen algoritmos de optimización que logran reducir está diferencia.

La programación en esta ocasión se optará por el lenguaje ensamblador y solo porque así se comprenderá mejor el funcionamiento interno del PIC. Teniendo presente esto, se continuará con el primer programa.

Entrada de datos – salida de datos

Para la entrada de datos se utilizará una ventana de estimulación que fungirán como switch o push button de la vida real. Para la salida de información digital se ocupará una ventana que permite monitorear el estado de los bytes de cada variable local o global, incluso del comportamiento de algunos registros.

En el capítulo anterior se mostró como iniciar un nuevo proyecto, un nuevo archivo .asm y adaptar el programa con algunas ventanas extra, así como la configuración de bits del microcontrolador. En el siguiente ejemplo se muestra la forma de configurar los bits de STATUS. Algunas líneas están documentadas y está de más explicarlo de forma detallada, pero como la línea 16 no se había explicado anteriormente. Las **etiquetas** son indicadores de partes de programa, de esa forma cuando se utilice una instrucción CALL o GOTO la instrucción sabrá en que parte del programa dirigirse.

En las líneas siguientes se trata de acceder al banco 1 ¿Por qué? En los Registros de PORTA y TRISB del capítulo 4 explica que, para configurar el PORTA como entrada, el TRISA debe tener los ocho bits en 1. Por ese motivo se mueve una literal de 8 bits con puros unos a W y del registro W se mueve a TRISA, diciéndole de esa forma al PIC que el PORTA será entrada.

En cambio, la instrucción "*clrf TRISB"* limpia el archivo TRISB, de esta manera el byte tendrá un 0 en cada bit, interpretando que el microcontrolador deberá configurar el PORTB como salida. Esa es la principal razón por la que se cambia de bancos. Se recomienda saber sobre que banco se está trabajando y cuando se deje de utilizar regresar al banco de operación, generalmente el banco de operación es el banco 0. Por eso la línea 32 se encarga de regresar a ese banco.

En la línea 28 se muestra como decirle al PIC en que frecuencia debe trabajar.

PrenderLed.asm | EntradaYSalidasDeDatos.asm

Asm Source | History

```
10    ; CONFIG
11    ; __config 0x3F1C
12    __CONFIG _FOSC_INTOSCIO & _WDTE_ON & _PWRTE_OFF & _MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _CPD_OFF & _CP_OFF
13
14    Org 0                 ; Está línea significa que el programa empezará en la dirección 0 en la memoria de programa
15
16    Inicio                ; Está Linea se considera una etiqueta
17       bcf     STATUS,IRP  ; Bit Clear File f,d    Poner a 0 el Bit IRP del Registro STATUS    Acceder al banco 0 y 1
18
19       ;Ahora para acceder especificamente a un banco
20       ;hay que poner un 0 en RP1 y un 1 en RP0
21
22       bcf     STATUS,RP1
23       bsf     STATUS,RP0
24       clrf    TRISB       ;El PORTB se configura como salida
25       movlw   b'11111111' ;Se mueve una literal de 8 bits a W
26       movwf   TRISA       ;Se configura el PORTA como entrada
27
28       bsf    PCON,OSCF    ;Se establece el oscilador de 4Mhz
29
30       ;Regresar al banco 0 solo hay que poner un 0 en el la bandera RP0
31
32       bcf     STATUS,RP0  ;Regresar al banco 0
33
```

Fig. 7.1 Configuración inicial del microcontrolador.

Bien. La configuración no queda ahí, ya se le ha dicho al Circuito Integrado como se comportará, pero que funciones de pines se ocuparán. En el siguiente ejemplo se desactivarán las entradas analógicas para eso en la Fig. 4.6 del capítulo 4 del Registro CMCON muestra como desactivar los comparadores, las banderas de CM2 CM1 y CM0 deben tener un 1. En la línea 34 se mueve una literal a W y enseguida en la línea 35 se moverá W al registro CMCON de esta manera se desactivan los comparadores.

```
27
28       bsf    PCON,OSCF    ;Se establece el oscilador de 4Mhz
29
30       ;Regresar al banco 0 solo hay que poner un 0 en el la bandera RP0
31
32       bcf     STATUS,RP0  ;Regresar al banco 0
33
34       movlw   b'111'
35       movwf   CMCON       ;Desactivar el comparador análogico
36
37    Principal              ; Está Linea se considera una etiqueta
38
39    END
```

Fig. 7.2 Desactivar comparadores analógicos.

El voltaje de referencia (VREF) también está involucrado en una de las patitas del microcontrolador. Normalmente esa bandera está desactivada, pero si se desea estar seguro hay dos pruebas que se pueden hacer, una es escribirlo directamente en el código y desactivarla u otra cuando se esté simulando el código visualizar el registro VRCON. El bit encargado de habilitar o deshabilitar el VREF es el bit 6, en el ejemplo siguiente se muestra como el registro VRCON se encuentra en 0 y quiere decir que no habrá problema.

Fig. 7.3 Ventana Variables: Registro VRCON.

En la pata 5 del PIC se encuentra VPP (Voltaje de programación) no hay de que preocuparse por este pin pues este pin sabe cuándo se debe utilizar ya que para que funcione necesita más de 10V y se verá más adelante la razón. Ya que su principal función es cuando se esté quemando (Grabando) el MCU (microcontrolador).

CMP1 y CMP2 se involucran con los comparadores analógicos, pero como ya están desactivados no hay de que preocuparse.

T0CKI está en el pin 3 y hay que desactivar los temporizadores para que esta función quede inhabilitada. Es por eso por lo que en la línea 37 y 38 se cumple la desactivación.

```
36
37      bsf     T1CON,0
38      bcf     T2CON,2     ;Desactivar temporizadores
39
40  Principal               ; Está Linea se considera una etique
41
42  END
```

Fig. 7.4 Desactivar Temporizadores.

De esta forma ya está configurado el PIC para que reciba entradas digitales por el PORTA y poder ver el resultado por el PORTB.

El diagrama de flujo es una representación gráfica de cómo se puede comportar el código en el transcurso de su ejecución. Este diagrama es una herramienta muy útil antes de empezar a programar ya que el programador se da una idea más detallada de que sentencias ocupar, que lógica debe llevar, que variables ocupar, etc.

Ahora se representará un diagrama de flujo del programa en cuestión.

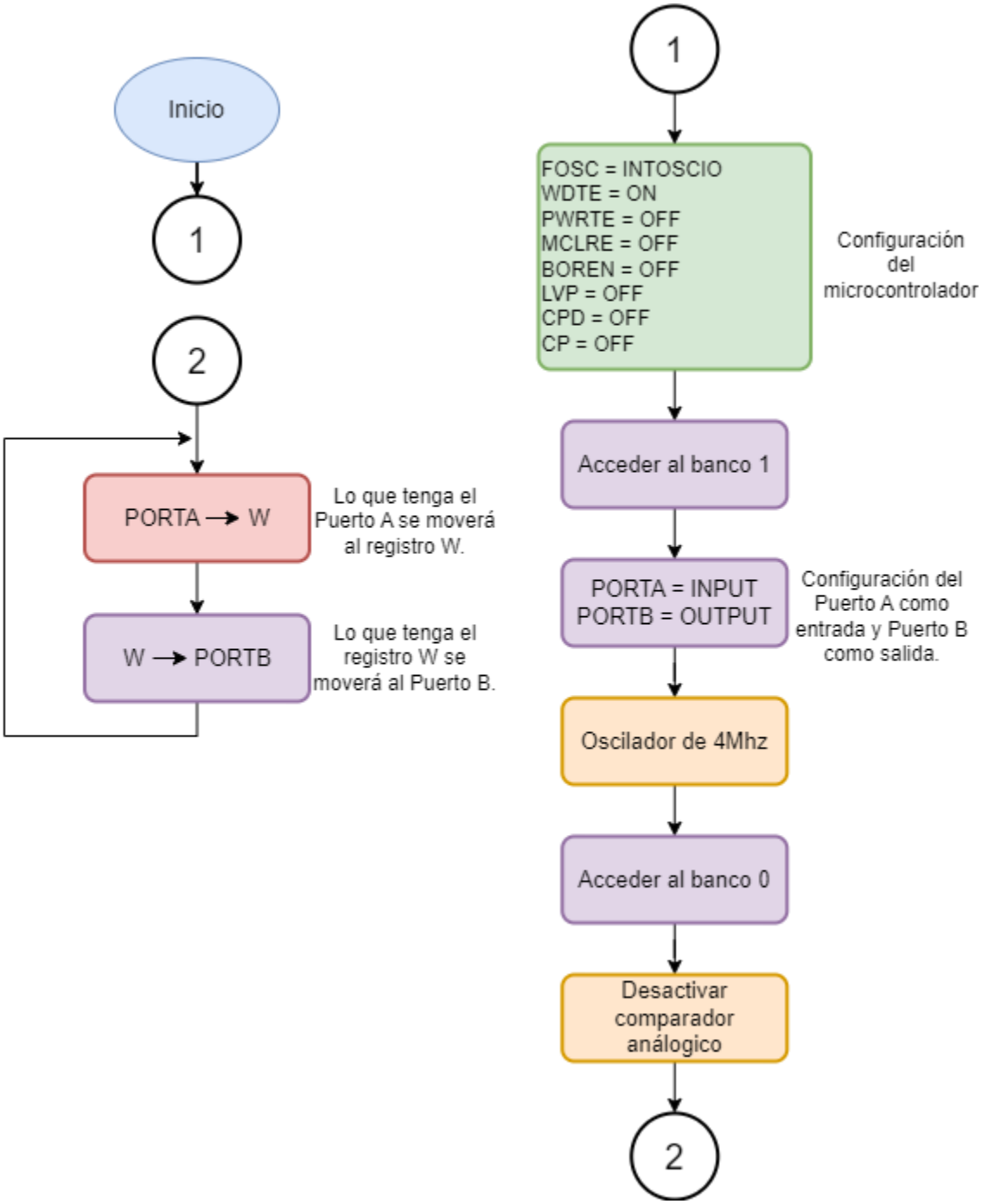


Fig. 7.5 Diagrama de flujo, entrada y salida de datos.

El programa para mostrar la entrada de datos consta de mover lo que hay en el puerto A a W y después para mostrar la salida de datos se reflejarán por el puerto B.

```
36
37      Principal               ; Está Linea se considera una etiqueta
38
39          movf    PORTA,0     ;Mover lo que hay en el puerto A a W
40          movwf   PORTB       ;Mover lo que hay en W al Puerto B
41          goto    Principal   ; Regresar a Principal
42
43      END                     ; Fin del programa.
```

Fig. 7.6 Programa principal: Entrada y salida de datos.

Listo ahora hay que seguir los puntos siguientes.

- Compilar el código (Corregir errores de escritura en el programa, si está mal).
- Debug.
- Pausa.
- Reset (Para comenzar desde la etiqueta INICIO).

Se podrá ver que el PORTA está vacío en la ventana de variables y que el Reset ayuda a colocarse en la línea 17 del código.

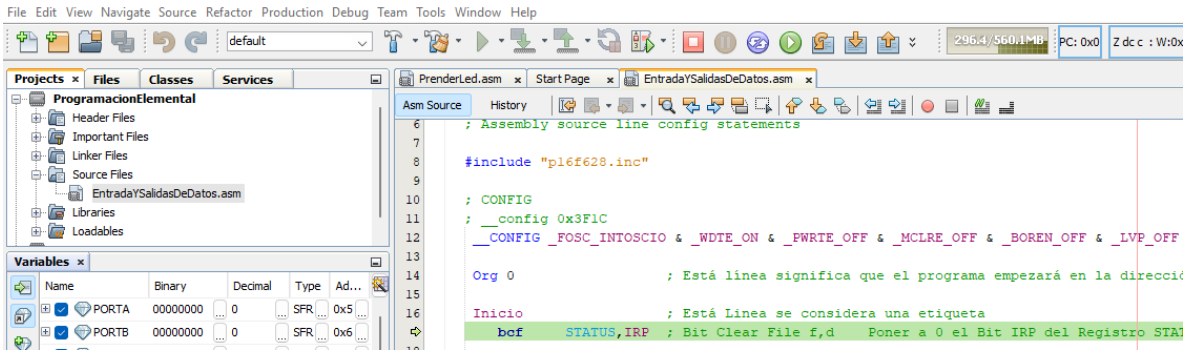


Fig. 7.7 Iniciar ejecución de programa línea por línea.

Cuando se esté pulsando la tecla F7 o también los siguientes iconos ayudan a desplazarse por el código.

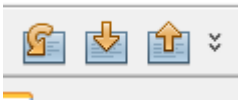


Fig. 7.8 Iconos de desplazamiento por el programa en ejecución.

La línea verde que identifica la posición actual saltará hasta la siguiente instrucción. Los comentarios no cuentan entonces no hay de que preocuparse. Seguir pulsando F7 hasta lograr llegar al primer retorno del *goto*, es decir cuando se esté nuevamente en la etiqueta "**Principal**".

Ahora en la venta "Stimulus" clic en el bit RA1, RA3, RA5 y RA7. Y seguir pulsando F7 hasta notar una diferencia en los registros. Debe mostrarse un resultado como el siguiente.

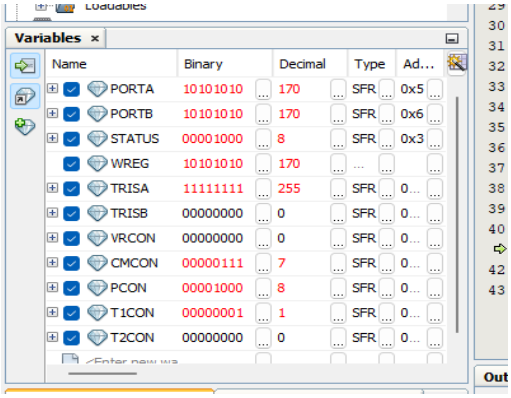


Fig. 7.9 Resultados en registros del programa en ejecución.

Como se puede observar. El Puerto A tiene el mismo valor que el puerto B. También se puede notar que ***WREG*** o el registro W tiene cargado el mismo valor. Con esto se puede comprobar que el programa funciona. Otra forma de saberlo es tener el circuito eléctrico como se explicará a continuación.

Ya que se compilo y ejecuto el programa hay que cargarlo dentro del microcontrolador en PROTEUS primero hay que tener un circuito como el siguiente.

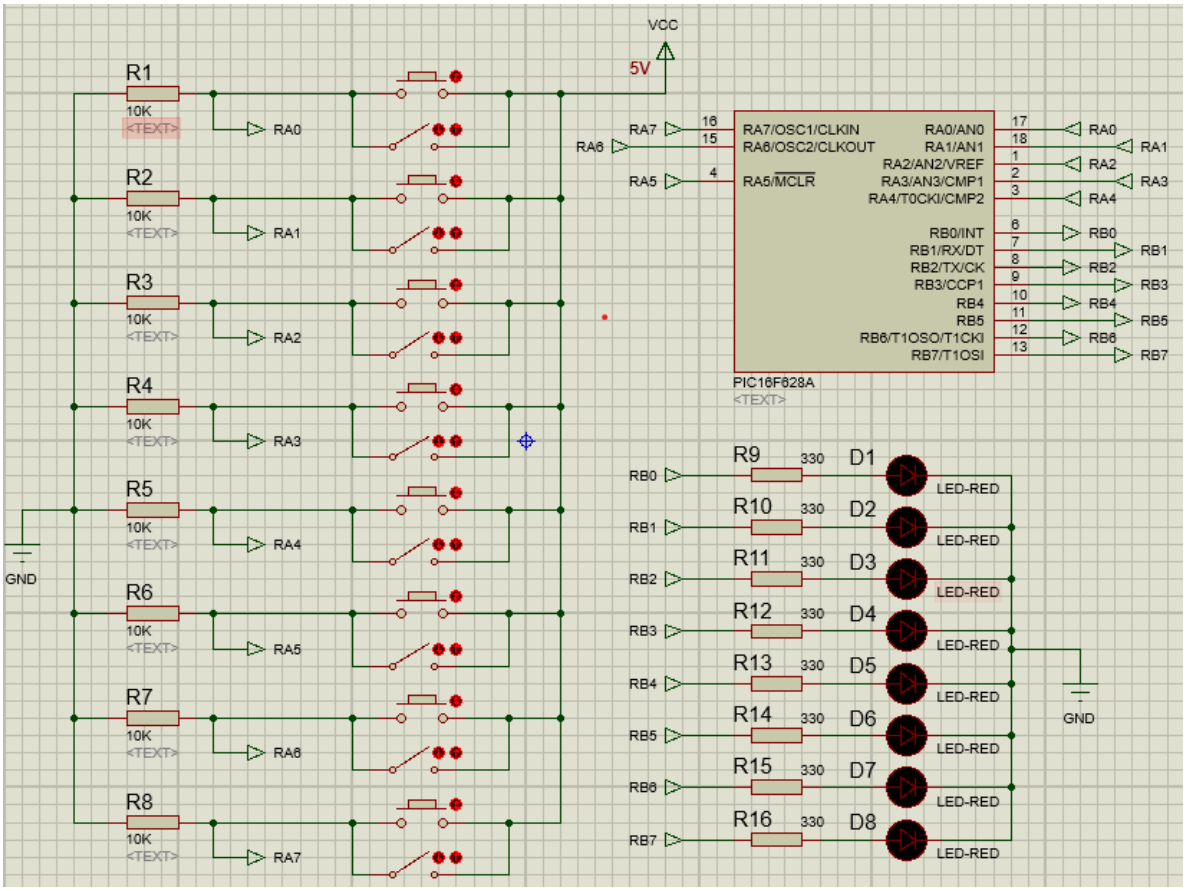


Fig. 7.10 Simulación con proteus, circuito básico de entradas y salidas de datos.

Al dar doble clic izquierdo sobre el PIC se abrirá una ventana, que serán las propiedades del Circuito Integrado y en la opción de “Program File” en el icono de la carpeta que está justo a un lado, se buscará la ruta o dirección donde se aloja el archivo .hex, recordar que MPLAB al compilar el código en la ventana de estado dirá la ruta del archivo.

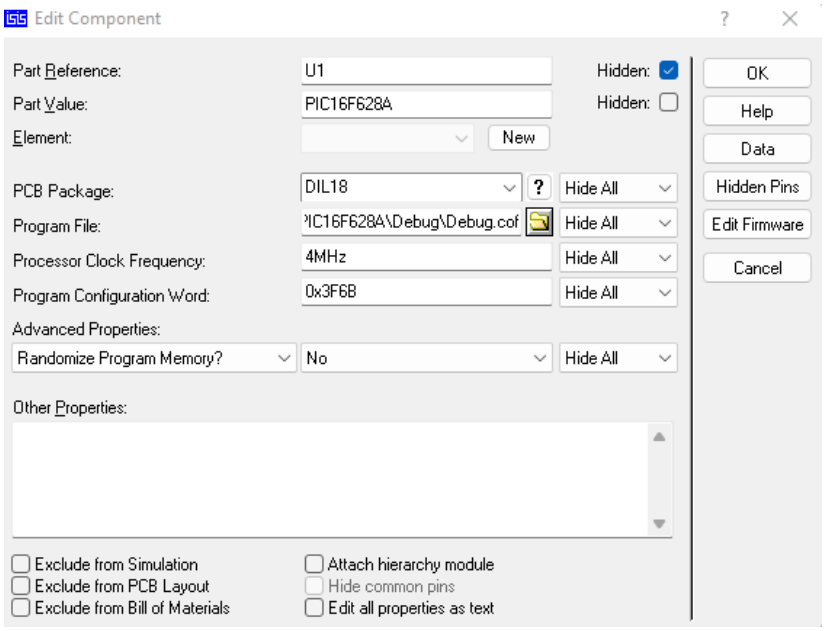


Fig. 7.11 Simulación con proteus, propiedades del circuito integrado.

En la opción “Processor Clock Frequency” determinar el cristal con el que trabajará el PIC. En este caso será de 4MHz y después clic en Ok. En la parte inferior izquierda del software PROTEUS se encuentran botones como en el ejemplo siguiente, para ejecutar la simulación.

Fig. 7.12 Simulación con proteus, botones de ejecución del circuito.

Como resultado se obtiene lo siguiente.

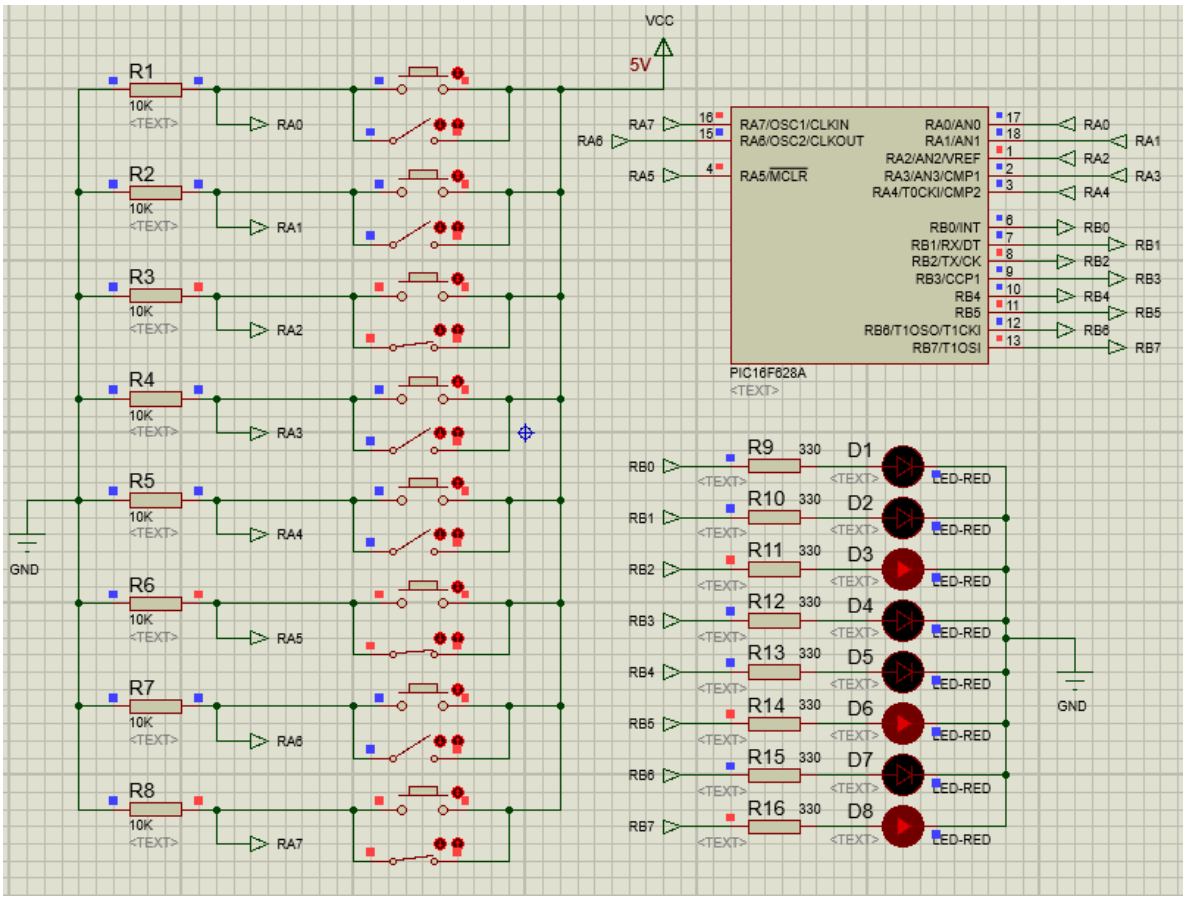


Fig. 7.13 Simulación con proteus, visualización de los resultados obtenidos.

Se puede notar que por RA2, RA5 y RA7 hay entrada de información pues lo switch están cerrados y el microcontrolador indica con un cuadrito de color rojo el estado en 1, que quiere decir que entra voltaje por ese pin y lo mismo pasa con el puerto B, los LED muestran el resultado del procesamiento de datos del MCU.

De esta forma se concluye el primer programa en entradas y salidas de datos.

Subrutina

La subrutina es un conjunto de instrucciones al que se tiene acceso desde cualquier punto del programa principal. Una subrutina es un subprograma que se ejecuta cada vez que el programa principal lo necesite [14].

Este subprograma a su vez puede contener otro subprograma y esté puede contener otro subprograma, en el capítulo 2, una característica del MCU es que tiene una pila de 8 niveles, significa que un programa puede contener 8 subrutinas.

Es interesante saber que un bloque del programa se pueda estar repitiendo infinidad de veces para resolver una tarea especifica. La instrucción *call* es la encargada de llamar a estás

subrutinas, cada instrucción *call* debe tener su *return* para que pueda salir del subprograma y continuar en el programa principal la Fig. 7.14 ilustra cómo funciona está instrucción.

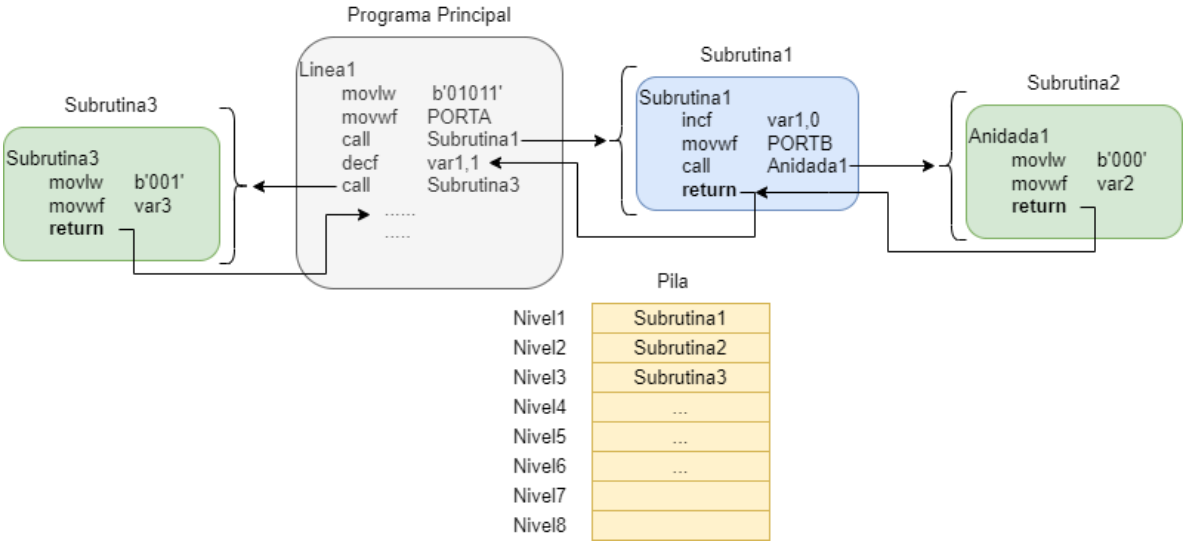


Fig. 7.14 Funcionamiento instrucción *call*.

El programa que se realizará como ejemplo tratará de la compuerta lógica OR, en la Tabla 7.1 se apreciará cómo se comporta está compuerta.

Tabla 7.1 Compuerta OR.

Entrada A	**Entrada B**	***Salida***
0	0	*0*
0	1	*1*
1	0	*1*
1	1	*1*

El programa constará de 2 entradas, una por el bit 0 y otra por el bit 1 del puerto A, El bit 0 fungirá como la entrada A de la compuerta y el bit 1 la entrada de la compuerta B, a la salida del puerto B se mostrará un conteo de 7 hasta 0 si se cumple la condición de salida, es decir, este conteo se realizará únicamente cuando el resultado de la salida de la compuerta sea un uno. Cuando sea un cero entrará en bucle para que siempre este esperando la entrada correcta.

Es recomendable realizar diagramas de flujo, para tener una mayor idea del comportamiento del programa que se codificara. Los diagramas de flujo son representaciones gráficas de un flujo de datos o la secuencia que debe tener la información para cumplir el objetivo de una tarea, dentro de estos diagramas se encuentran condiciones que establecen una decisión limitada por un uno o por un cero, es decir que pasa si “si” o que pasa si “no”. Esto ayuda a saber que instrucción se puede utilizar en el programa y que resultado se obtendría.

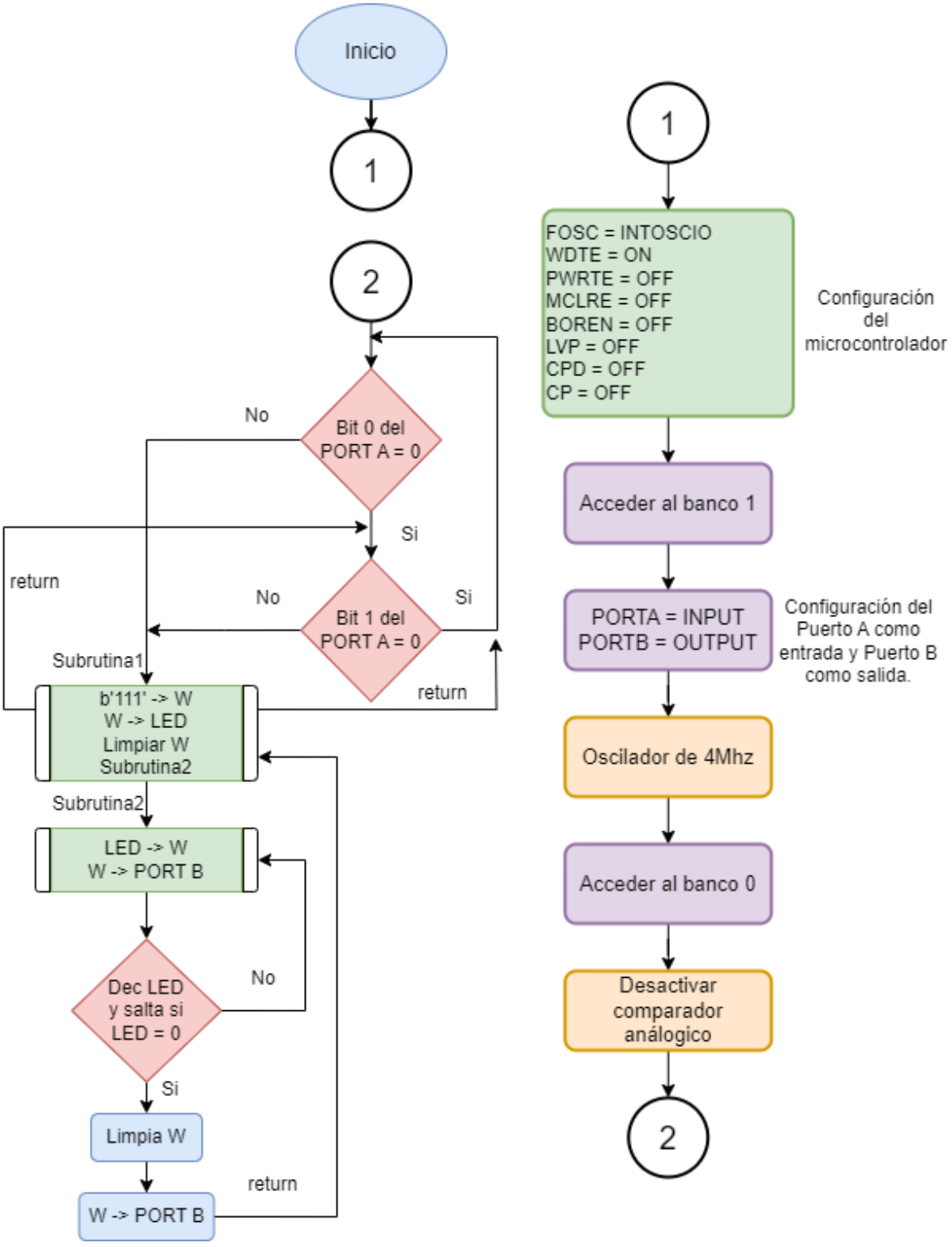


Fig. 7.15 Diagrama de flujo, subrutina.

Copiar el principio del programa anterior desde la línea 0 hasta la etiqueta “Principal”. El Programa quedaría de la siguiente forma.

```
35
36      Principal
37          btfsc   PORTA,0     ; Monitorear el bit 0 del PORTA.
38          call    Subrutinal  ; Llamar a la Subrutinal
39          btfsc   PORTA,1     ; Monitorear el bit 1 del PORTA
40          call    Subrutinal  ; Llamar a la subrutinal
41          goto    Principal   ; Ir a la etiqueta Principal
42
43      Subrutinal              ; Subrutinal
44          movlw   b'111'      ; Mover un 7 a W
45          movwf   LED         ; Mover el 7 a la variable LED
46          clrw                ; Limpiar W
47          call    Subrutina2  ; Llamar a la subrutina2
48          return              ; Retorna a la línea 41 del programa Principal
49
50          Subrutina2          ; Subrutina 2
51          movf    LED,0       ; Mover lo que hay en la variable LED a W
52          movwf   PORTB       ; Mostrar por el PORTB lo que hay en W
53          decfsz  LED,1       ; Decrementa LED y guardarlo en la misma variable
54          goto    Subrutina2  ; ir a Subrutina2
55          clrw                ; Limpiar W
56          movwf   PORTB       ; Mover lo que hay en W al PORTB
57          return              ; Retorna a la línea 48 del bloque de programa Subrutinal
58
59      END                     ; Fin del Programa
```

Fig. 7.16 programa de subrutina.

En el Subtítulo “Declaración de variables” del capítulo 6 se puede ver cómo se crea una variable, dicho esto, se continua con la explicación del programa. La instrucción *btfsc PORTA,0* se interpreta como un monitoreo del bit cero del puerto A, esto es, monitorea el bit cero del puerto A y salta hasta la instrucción *btfsc PORTA,1* (línea 39) si este bit (el bit cero) es igual a cero, si el bit cero del puerto A es uno continua con la instrucción *call Subrutina1* (línea 38).

La instrucción *decfsz LED,1* dice que decrementará a la variable LED hasta cero y cada decremento el valor lo ira guardando en la misma variable LED. El salto a la línea 55 con la instrucción *clrw* sucederá hasta que LED valga cero, mientras no se cumpla la condición seguirá en el bucle entre la línea 50 y 54, al transcurso del bucle el valor nuevo que vaya teniendo la variable LED se irá mostrando en el puerto B, se mostrará un conteo de 7 hasta 0.

Con “Stimulus” se pueden dar los valores del bit 0 y 1 del PORTA puesto que ya está configurado como entrada. Al compilar y depurar el programa en la ventana de variables se apreciará el movimiento de esta instrucción. Así mismo se podrán ver las direcciones de memoria de las variables.

El funcionamiento en PROTEUS debe mostrarse de la misa manera. El circuito que se ve en el subtítulo anterior funcionará también para la prueba de este código. Cargar el archivo

hexadecimal y ejecutar la simulación. Se apreciará que va muy rápido el conteo puesto que se habla de microsegundos. En el subtítulo siguiente se tendrá un mejor control para determinar una pausa y así la vista humana lo pueda apreciar mejor.

Rutinas de tiempo

Las rutinas de tiempo se pueden utilizar para hacer pausas en algún programa que se requiera. Estos lapsos de tiempo se encuentran en microsegundos (µs), en milisegundos (ms) o segundos (s). Un ejemplo puede ser prender y apagar un led cada segundo o cada mitad de segundo.

Antes de entrar en el programa es necesario saber los ciclos de máquina que tiene internamente el microcontrolador. El PIC16F628 hace un ciclo máquina mientras su reloj hace cuatro ciclos, como lo muestra la Fig. 7.17.

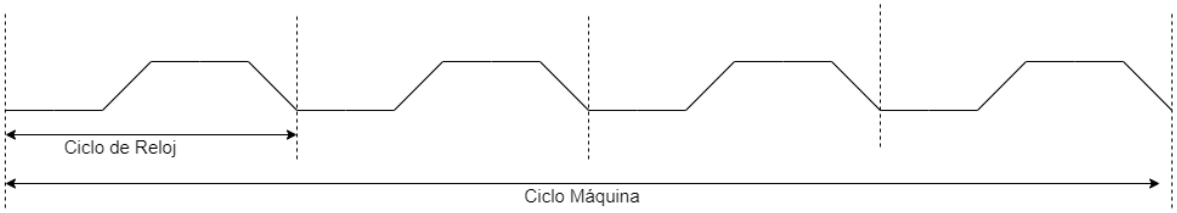


Fig. 7.17 Ciclo máquina para el PIC16F628.

Ahora la columna de ciclos de la Tabla 15 del capítulo 5 tiene sentido, puesto que hay instrucciones que se tardan 2 ciclos máquina, pero ¿Cuánto tiempo toma un ciclo máquina de una instrucción?

El MCU al ejecutar una tarea, está fijada a la formula siguiente:

$$Tiempo = 4\,\frac{1}{f}\,cm$$

Donde:

- ƒ es la frecuencia del oscilador
- cm el número de ciclos máquina que tarda en ejecutar la tarea.

El microcontrolador 16F628 trabajará con una frecuencia de oscilador de 4Mhz y tareas entre un ciclo máquina o dos. Entonces para calcular la duración de la instrucción CALL, por ejemplo, queda de la siguiente manera [14].

$$Tiempo = 4\,\frac{1}{4MHz}\,2 \qquad Tiempo = 2\mu s$$

Sabiendo esto se continua con el programa.

Enrique palacios hizo una gran aportación en su libro que se menciona en este documento, puesto que el archivo “Retardos.inc” se ocupa en la mayoría de las aplicaciones para los

PIC's de diferente arquitectura. Este archivo se puede encontrar en internet y descargarlo, la locación para dejar el archivo se mostrará en la siguiente Fig. 7.18.

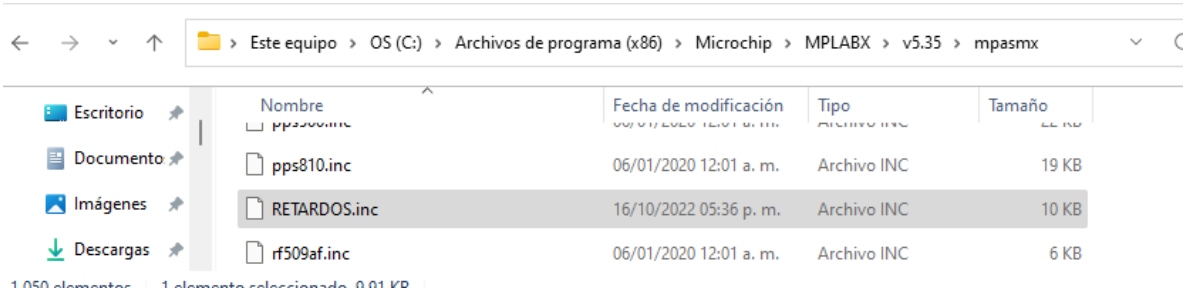


Fig. 7.18 Ubicación de archivo "RETARDOS.inc"

Para utilizar esta librería, se creará un nuevo archivo .asm dentro del proyecto Programación elemental. Se copiará la misma configuración inicial del programa anterior, es decir desde la línea de código 0 hasta la línea de la etiqueta "Principal". Este archivo .asm se llamará "Rutina de tiempo" quedando de la siguiente manera.

```
30      movwf   CMCON       ;Desactivar el comparador análogico
31      bsf     T1CON,0
32      bcf     T2CON,2     ;Desactivar temporizadores
33
34      Principal
35
36      #include "RETARDOS.inc" //Libreria de retardos
37      END
38
```

Fig. 7.19 Posición de las librerías.

Se aprecia que la librería se coloca casi al final del programa y esto es debido a que la sintaxis de MPLAB no permite dejarlo debajo de la librería del microcontrolador porque si no marca error al compilar.

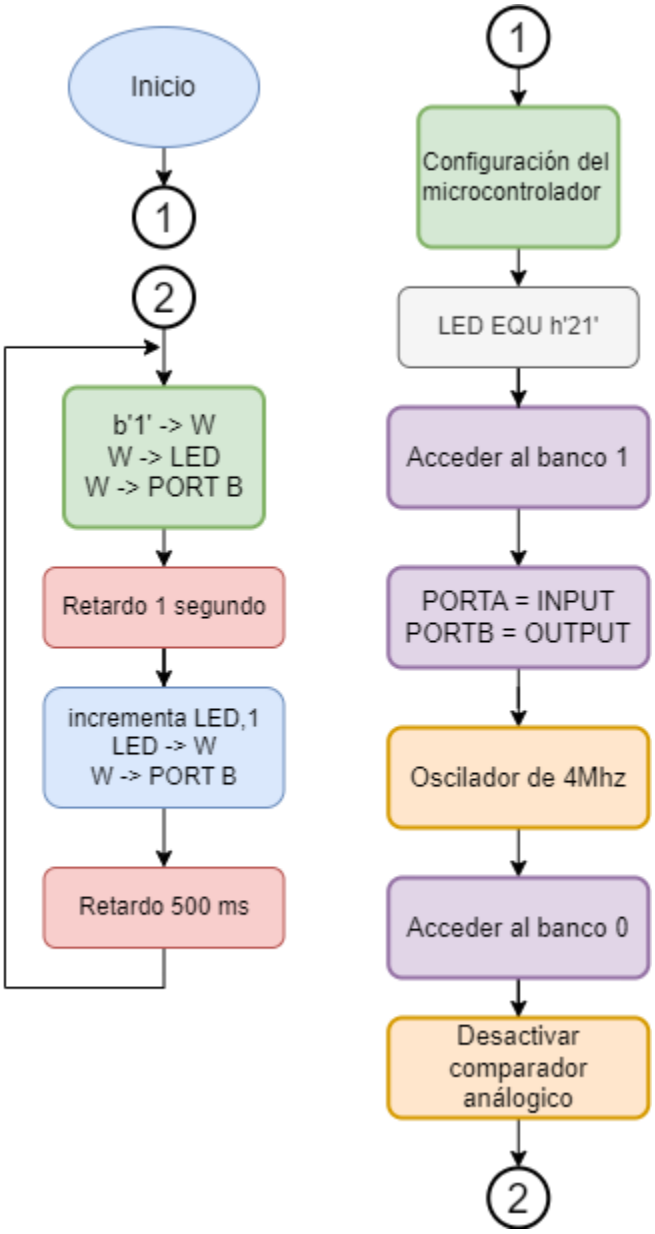


Fig. 7.20 Diagrama de flujo, rutinas de tiempo.

Se creará un programa que prenda un led durante un segundo y se apague durante 500 ms.

```
39  Principal                   ; Está Linea se considera una etiqueta
40      movlw   b'1'            ; Mover un 1 a W
41      movwf   LED             ; Mover lo que hay en W a LED
42      movwf   PORTB           ; Aquí se muestra por el PORTB lo que hay en W
43      call    Retardo_1s      ; Llamar a la Subrutina de Retardo de un segundo
44      incf    LED,1           ; Incrementar en 1 la variable LED y guardar el dato en LED
45      movf    LED,0           ; Mover lo que hay en la variable LED a W
46      movwf   PORTB           ; Mover lo que hay en W al PORTB
47      call    Retardo_500ms   ; Subrutina de Retardo para 500ms
48      goto    Principal       ; Regresar a Principal
49
50
51  #include "RETARDOS.inc"
52   END                        ; Fin del programa.
```

Fig. 7.21 Programa de rutinas de tiempo.

Por la ventana variables se apreciará el movimiento de bits cuando esté programa se haya compilado y depurado correctamente.

En PROTEUS se muestra el tiempo de ejecución (ANIMATING) que se encuentra a un lado de los botones de pausa y stop. Ahí se tendrá una idea de los lapsos de tiempo en que prende y se apaga el led.

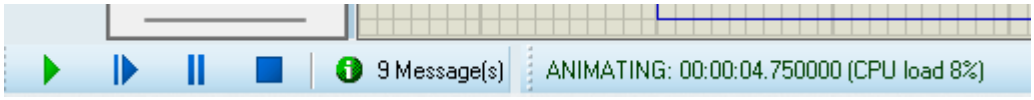


Fig. 7.22 Tiempo de ejecución de un circuito.

Manejo de tablas

Cuando se tiene demasiada información que almacenar y sea necesario requerir de esa información en algún punto especifico dentro de la ejecución del programa, entonces las tablas solucionan el problema, gracias a las filas y columnas que la componen, puesto que los datos residirán ahí.

Suponer que se requieran activar diferentes procesos dependiendo la entrada que se esté solicitando como el ejemplo siguiente.

Datos			Procesos				
D1	D2	D3	P1	P2	P3	P4	P5
0	0	0	1	1	1	0	1
0	0	1	1	1	0	0	1
0	1	0	1	0	1	1	0
0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	0
1	0	1	0	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	0	1	1	0	1

Fig. 7.23 Información para el ejercicio.

Por el puerto A se ingresarán los datos con interruptores quedando la configuración de los bits: bit 0 (D3), bit 1 (D2) y bit 2 (D1). Los procesos que serán la salida se mostrarán por el puerto B quedando la configuración de los bits: bit 0 (P5), bit 1 (P4), bit 2 (P3), bit 3 (P2) y bit 4 (P1).

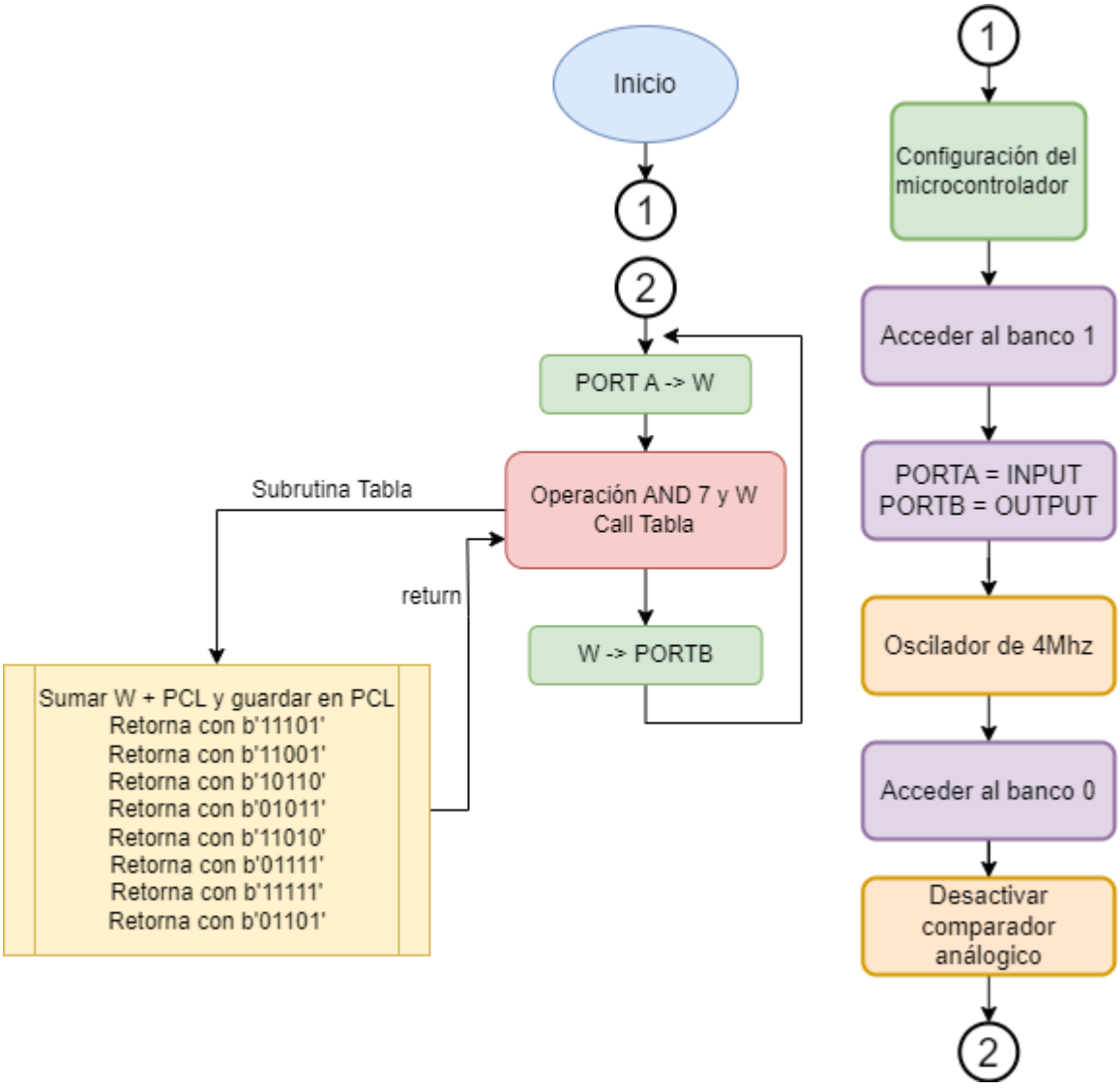


Fig. 7.24 Diagrama de flujo, manejo de tablas.

Se añadirá un nuevo archivo dentro del proyecto “ProgramaciónElemental” el cual se llamará “Tablas.asm”, la configuración inicial de los programas anteriores será la misma hasta la etiqueta “Principal”.

```
37
38  Principal
39      movf    PORTA,W     ; Mover lo que hay en el PORTA a W
40      andlw   b'00000111' ; Operación AND entre una literal y W
41      call    Tabla       ; Llamada a la subrutina Tabla
42      movwf   PORTB       ; Mover lo que hay en W a PORTB
43      goto    Principal   ; Ir a Principal
44
45  Tabla
46      addwf   PCL,1
47      retlw   b'11101'    ; Retorna al programa Principal con una literal en W
48      retlw   b'11001'    ; Retorna al programa Principal con una literal en W
49      retlw   b'10110'    ; Retorna al programa Principal con una literal en W
50      retlw   b'01011'    ; Retorna al programa Principal con una literal en W
51      retlw   b'11010'    ; Retorna al programa Principal con una literal en W
52      retlw   b'01111'    ; Retorna al programa Principal con una literal en W
53      retlw   b'11111'    ; Retorna al programa Principal con una literal en W
54      retlw   b'01101'    ; Retorna al programa Principal con una literal en W
55
56      END                 ; Final de Programa
57
58
```

Fig. 7.24 Programa de manejo de tablas.

La instrucción *andlw b'00000111'* explica una operación lógica AND, la tabla de verdad para esta operación se muestra a continuación. Tabla 7.2

Tabla 7.2 Compuerta lógica AND

Entrada A	**Entrada B**	***Salida***
0	0	*0*
0	1	*0*
1	0	*0*
1	1	*1*

Está instrucción verifica que, si el microcontrolador tiene más entradas, las ignorará porque solo tomara las primeras tres, es decir el bit 0, 1 y 2. Después de esta instrucción se llama a la subrutina tabla, la instrucción *addwf PCL,1* es la siguiente en ejecutarse, se encarga de sumar lo que hay en W (lo que se ingresó por el puerto A) al registro PCL y el resultado que se guarde en el mismo registro PCL. El resultado de la suma determinará a que línea de programa se desplazará puesto que el contador de programa (PCL) va incrementándose cada salto de línea se esté haciendo, dependiendo también los ciclos de la instrucción que se tarde. Dicho de otra forma, cuando el programa este en la línea 46 el registro PCL y tenga 13 saltos o simplemente el número 13 en decimal, al sumarle lo que el usuario ingreso por el puerto A al contador de programa, este se situará en la línea del programa correcta.

La instrucción *retlw b'####'* es una instrucción que retorna al programa principal, pero con un valor, en este caso es la literal que se le agrego a W, y cuando esta, regresa a la línea 42, pues se mostrará por el puerto B la configuración que se le indico.

Lo que sigue es compilar y depurar el programa, ingresar las posibles combinaciones por el puerto A y verificar que por el Puerto B se están mostrando los procesos que deberían, al mismo tiempo se comprenderá cómo se comporta el registro PCL.

En PROTEUS se puede observar que las condiciones con los interruptores se cumplen en las salidas. Qué pasa si no se presiona ningún botón o switch. La salida por el puerto B quedaría de la siguiente forma.

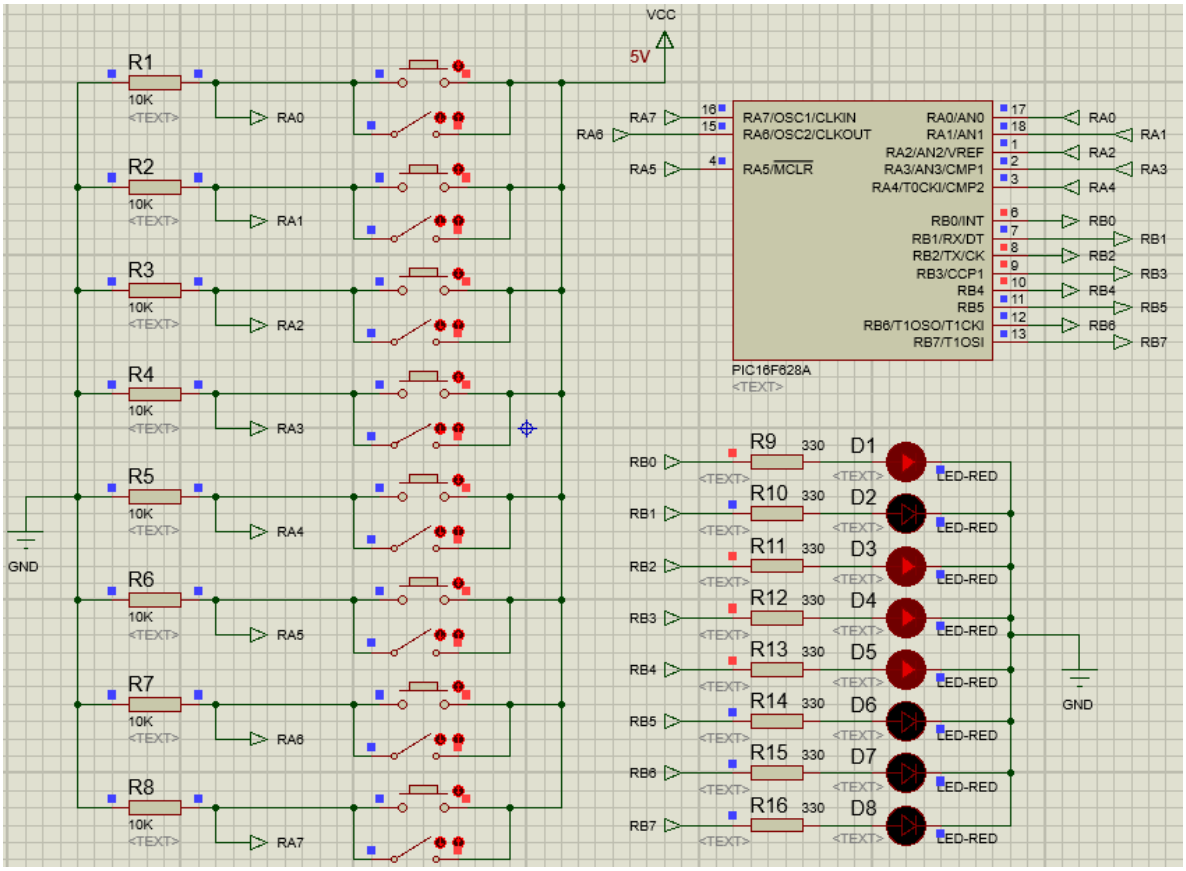


Fig. 7.25 Simulación con proteus, circuito para la comprobación de tablas.

Al momento de interactuar con los switch el estado de la salida cambiará. Probar con los switches del cuarto hacía adelante. Qué ocurre. La compuerta AND (línea de código 40) cumple su función de solo dejar pasar los tres primeros datos.

Pantalla de cristal líquido LCD

Esta pantalla ayuda a ver más claro un mensaje o información. Es un periférico que se puede encontrar en varios dispositivos, como la pantalla de las calculadoras, la pantalla de las terminales de tarjetas de crédito o alguno que otro juguete, incluso los despertadores o relojes tienen estas pantallas LCD.

En la siguiente Fig. 7.26 se puede mostrar de que pantalla se está hablando, de igual manera es la que se usará para este tema, PROTEUS cuenta con este dispositivo también para que se pueda observar el texto o información que se requiera plasmar.

Fig. 7.26 Pantalla de Cristal Líquido (LCD – Liquid Crystal Display) 2 x 16

Existen diferentes modelos de pantallas en este caso se usará el modelo LM016L, el diagrama de pines se mostrará en la Fig. 7.27 y en la Tabla 7.3 se explicará la función de cada pin.

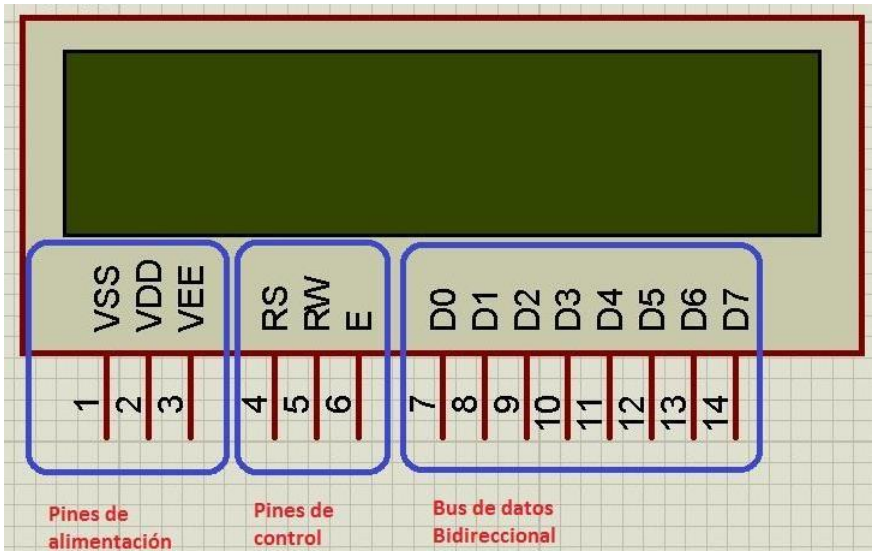


Fig. 7.27 Diagrama de pines pantalla LCD modelo LM016L

Tabla 7.3 Función de pines para la pantalla LCD modelo LM016L

Pines	Señal	Definición	Función
1	VSS	Ground	Tierra o Masa
2	VDD	Power Supply Voltage	Tensión de alimentación +5V
3	VLC o VEE	Liquid Crystal driving Voltage	Tensión para ajustar el contraste
4	RS	Register Select	R/S = 0, Modo comando. R/S = 1 Modo carácter
5	R/W	Read / Write	R/W = 0, Escribe en LCD. R/W = 1 Lee del LCD
6	E	Enable	E = 0, LCD no habilitado. E = 1, LCD habilitado
7-14	DB0-DB7	Data Bus	Bus de datos

Este display tiene una zona de memoria donde los caracteres son almacenados, está memoria es la DDRAM (Data Display RAM) la cual tiene una capacidad de 80 bytes, 40 bytes por cada línea, de los cuales sólo 32 se pueden visualizar a la vez (16 bytes por línea). En la Fig. 7.28 se puede mostrar cómo está seccionada la LCD.

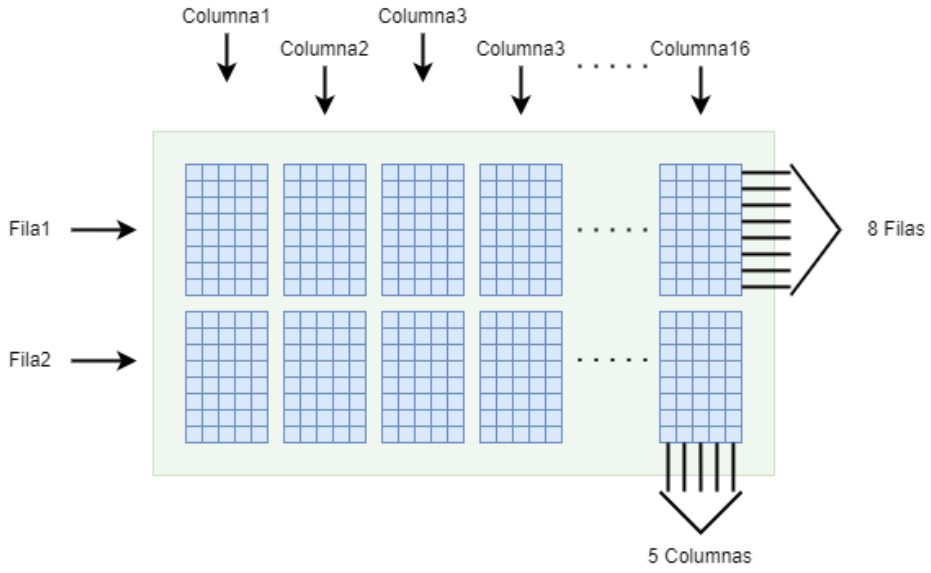


Fig. 7.28 Filas y columnas de la LCD

La CGROM es un tipo de memoria no volátil, dentro de ella se alojan 192 caracteres que se pueden visualizar llamándolos por el bus de datos. Cada carácter tiene su representación binaria de 8 bits. En la Fig. 7.29 se muestran los caracteres que tiene almacenados la CGROM.

Fig. 7.29 Caracteres almacenados en la CGROM.

Para controlar este display hay una serie de comandos, estos se envían a través del bus de datos. En la Tabla 7.4 se pueden apreciar los diferentes comandos para esté módulo.

Tabla 7.4 Comandos de control para el LCD LM06L

Comando	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Limpiar pantalla	0	0	0	0	0	0	0	0	0	1
Cursor a dirección origen	0	0	0	0	0	0	0	0	1	*
Modo entrada	0	0	0	0	0	0	0	1	I/D	S
Control de pantalla	0	0	0	0	0	0	1	D	C	B
Control de desplazamiento del cursor y pantalla	0	0	0	0	0	1	S/C	R/L	*	*
Características del control de hardware	0	0	1	0	1	DL	N	F	*	*
Escribir sobre la dirección CGRAM señalada	0	0	0	1	Dirección CGRAM					
Modificación del puntero DDRAM	0	0	1	Dirección DDRAM						
Lectura de bandera ocupada	0	1	BF	Dirección DDRAM						
Escribir en RAM	1	0	Escribir datos							
Leer RAM	1	1	Leer datos							

Limpiar pantalla: Borra pantalla y devuelve el cursor a la posición inicial.

Cursor a dirección origen: Devuelve el cursor a la posición original pero su contenido sigue intacto.

Modo entrada: Establece las características de escritura de los datos.

- S = 0 La información visualizada en pantalla no se desplaza al escribir un nuevo carácter.
- S = 1 La información visualizada se desplaza al escribir un nuevo carácter. La pantalla se desplaza en el sentido indicado por el bit I/D cuando el cursor llega al filo de la pantalla.
- I/D = 1 Incremento automático de la posición del cursor. La posición de la DDRAM se incrementa automáticamente tras cada lectura o escritura de esta.
- I/D = 0 Decremento de la posición del cursor. Se decrementa el puntero de la DDRAM.

Control de pantalla: Estos bits pueden ser configurados de la siguiente manera.

- B = 0 Parpadeo apagado, no hay efecto de parpadeo del cursor.
- B = 1 Parpadeo encendido, efecto de parpadeo con cursor rectangular.
- C = 0 Cursos apagado, el cursor no se visualiza.
- C = 1 Cursor encendido, el cursor es visualizado.
- D = 0 Pantalla apagada.
- D = 1 Pantalla encendida.

Control de desplazamiento y cursor de pantalla: Estos bits pueden ser configurados de la siguiente manera.

- R/L = 0 El desplazamiento ocurre a la izquierda.
- R/L = 1 El desplazamiento ocurre a la derecha.
- S/C = 0 El efecto de desplazamiento se aplica sólo sobre el cursor sin alterar el contenido de la DDRAM
- S/C = 1 El efecto de desplazamiento se aplica sobre toda la pantalla.

Características de control de hardware: Estos bits pueden ser configurados de la siguiente manera.

- F = 0 Fuente de caracteres de 5 x 7 puntos.
- F = 1 Fuente de caracteres de 5 x 10 puntos.
- N = 0 El número de línea es pantalla de una línea.
- N = 1 El número de línea es pantalla de dos líneas.
- DL = 0 Ancho de datos de comunicación con 4 bits. Indica a la pantalla LCD que solamente se va a utilizar las líneas de DB4 a DB7 para enviarle los datos y que se hará enviando primero el nibble alto, y a continuación el nibble bajo del alto.
- DL = 1 El ancho de datos será comunicación con 8 bits.

Escribir sobre la CGRAM señalada: Indica que se va a escribir sobre esta memoria. Este LCD permite definir ocho nuevos caracteres de usuario, no incluidos en la tabla interna. Estos caracteres se guardan en está memoria.

Modificación del puntero DDRAM: Se utiliza para modificar el puntero de esta memoria, por ejemplo, si la dirección es la 08h se escribirá en el centro de la primera línea. La Fig. 7.30 muestra los posicionamientos en hexadecimal de la pantalla LCD.

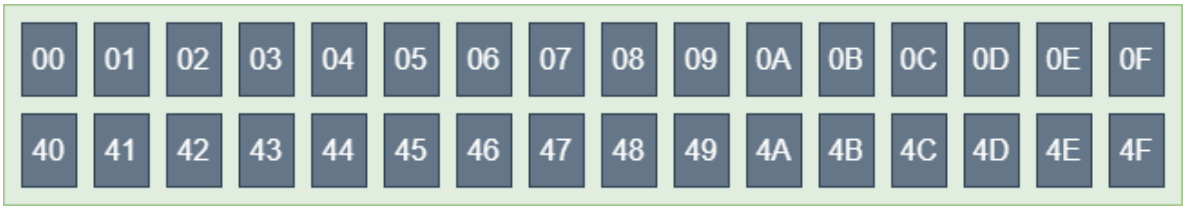


Fig. 7.30 Celdas de posicionamiento en hexadecimal de la pantalla LCD LM016L.

Lectura de bandera ocupada: Lee el BF indicando si hay una operación interna en curso y además lee el contenido de la dirección DDRAM apuntada.

Para mostrar mensajes en la pantalla LCD se ocupará una librería que se puede descargar desde internet, se llama LCD_4BIT.INC, ayudará para enviar los mensajes por el bus de datos de 4 bits y así mismo para ocupar el bus completo de 8 bits. Si se requiere mostrar mensajes más largos o que tengan movimiento la librería LCD_MENS.INC contiene estás subrutinas.

Dentro del proyecto “Programación elemental” se creará un nuevo archivo, el cual se llamará “LCD.asm”. La parte inicial del programa en la configuración de entradas y salidas cambiará un poco, quedará de la siguiente forma.

```
; PIC16F628 Configuration Bit Settings
; Assembly source line config statements

#include "p16f628.inc"

; CONFIG
; __config 0x3F1C
 __CONFIG _FOSC_INTOSCIO & _WDTE_OFF & _PWRTE_OFF & _MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _CPD_OFF & _CP_OFF

 Org 0                  ; Está línea significa que el programa empezará en la dirección 0 en la memoria de programa

Inicio                  ; Está Linea se considera una etiqueta
   bcf      STATUS,IRP  ; Bit Clear File f,d    Poner a 0 el Bit IRP del Registro STATUS    Acceder al banco 0 y 1

    ;Ahora para acceder especificamente a un banco
    ;hay que poner un 0 en RP1 y un 1 en RP0

    bcf     STATUS,RP1
    bsf     STATUS,RP0
    clrf    TRISB       ;El PORTB se configura como salida
    clrf    TRISA       ;El PORTA se configura como salida
    bsf     PCON,OSCF   ;Se establece el oscilador de 4Mhz
    ;Regresar al banco 0 solo hay que poner un 0 en el la bandera RP0
    bcf     STATUS,RP0  ;Regresar al banco 0
    movlw   b'111'
    movwf   CMCON       ;Desactivar el comparador análogico
    bsf     T1CON,0
    bcf     T2CON,2     ;Desactivar temporizadores

 Principal
```

Fig. 7.31 Configuración del microcontrolador para programa de LCD.

Se puede apreciar que el WDTE (Perro guardián) se desactivará, ya que el MCU al acabar de mostrar el mensaje, entrará en modo de bajo consumo (sleep) y el WDTE despertaría al microcontrolador cada cierto tiempo, ese es el trabajo del WDTE entonces ocurriría un desbordamiento de pila, por lo que el MCU mostraría el mensaje por unos segundos, pero se colapsaría después.

Dicho lo anterior, se apaga el WDTE. El puerto A también tiene que ser salida ya que por el bit RA0, RA1 y RA2 son los que controlarán la pantalla LCD. Entonces habrá que hacer el diagrama de flujo.

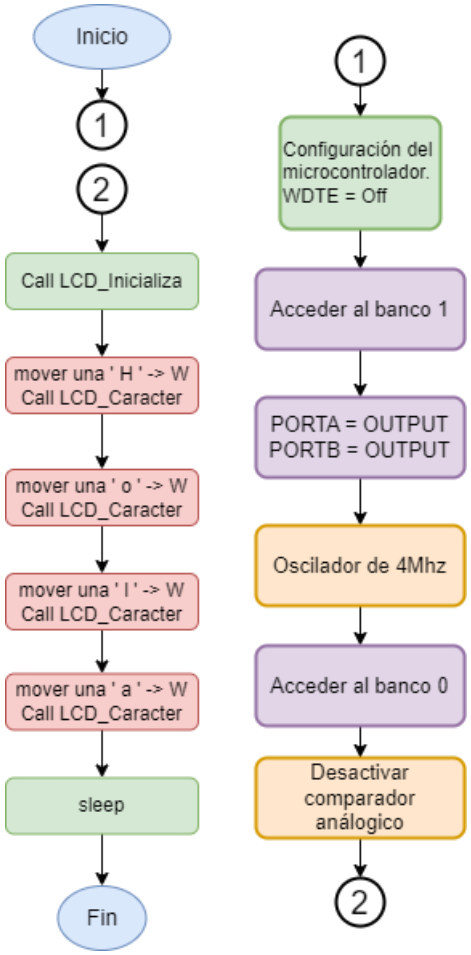


Fig. 7.32 Diagrama de flujo, Mensaje 'Hola' en pantalla LCD.

Lo que viene después de la etiqueta Principal será la llamada a la subrutina "LCD_Inicializa" que se encuentra dentro de la librería "LCD_4BIT.INC", la cual se encarga de la configuración de funciones de la pantalla como producir un reset por software, borrar la memoria DDRAM y encender la pantalla. Siempre se ejecutará esta subrutina al principio de un programa que se requiera utilizar el LCD.

La subrutina "LCD_Caracter" tiene a otras subrutinas anidadas, que en conjunto pueden activar el modo dato, obtener el código de la CGROM para una correcta visualización, guardar el carácter en una variable, comprobación si es o no un carácter especial y también la librería de "RETARDOS.inc" entra en el juego pues se encarga de dar las pausas correctas para hacer el cambio de modos en el LCD así como el tiempo necesario que se necesita para la carga y visualización de los datos en la pantalla LCD.

La instrucción *sleep* como lo dice una de las características del capítulo dos, el MCU es capaz de consumir tan solo un microamperio, de esta forma el ahorro de energía cuando no se ocupe el microcontrolador ayudará mucho en un proyecto robusto.

Enseguida vienen las librerías y el respectivo *END* para finalizar el programa.

```
29
30      Principal
31          call   LCD_Inicializa  ;Inicializa el modulo LCD para su correcto funcionamiento.
32          movlw  'H'             ;Se mueve un caracter "H" a W
33          call   LCD_Caracter    ;Visualiza en la posición actual del cursor el código ASCII del dato contenido en W.
34          movlw  'o'             ;Se mueve un caracter "o" a W
35          call   LCD_Caracter    ;Visualiza en la posición actual del cursor el código ASCII del dato contenido en W.
36          movlw  'l'             ;Se mueve un caracter "l" a W
37          call   LCD_Caracter    ;Visualiza en la posición actual del cursor el código ASCII del dato contenido en W.
38          movlw  'a'             ;Se mueve un caracter "a" a W
39          call   LCD_Caracter    ;Visualiza en la posición actual del cursor el código ASCII del dato contenido en W.
40          sleep                  ; El microcontrolador entra en modo de bajo consumo de energía.
41
42      #include "LCD_4BIT.INC"    ;Librería encargada de la configuración del LCD.
43      #include "RETARDOS.inc"    ;Libreria de retardos
44      END
45
```

Fig. 7.33 Programa para mandar mensaje “Hola” en pantalla LCD.

Para la simulación del programa se creará un circuito como el siguiente.

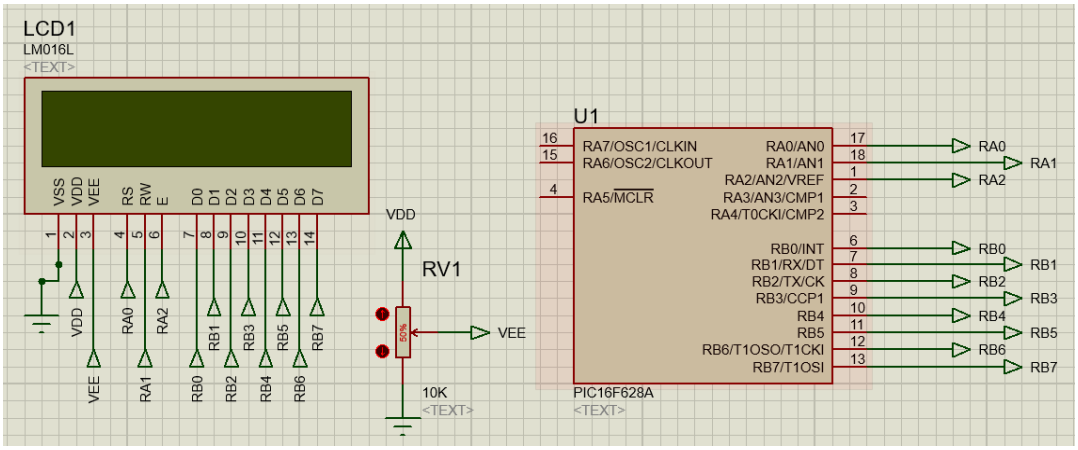


Fig. 7.34 Simulación con proteus, circuito para pantalla LCD.

La pantalla LCD también hay que configurarla, debe trabajar a la misma frecuencia que el microcontrolador. Dar doble clic sobre la LCD y saldrá una pantalla como la siguiente.

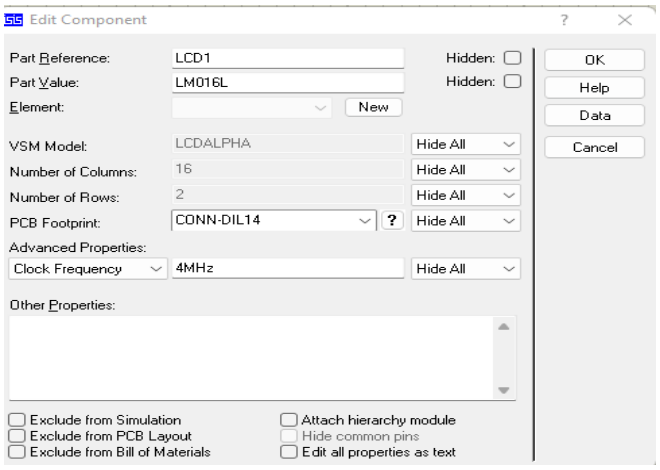


Fig. 7.35 Simulación en proteus, configuración de la pantalla LCD.

En la opción “Advanced Propierties” Escribir los 4MHz en la cinta de opción “Clock Frequency”. De esta forma el LCD y el MCU estarán trabajando a la par. Por último, clic en “Ok”. Ejecutar la simulación para que se muestre el resultado correcto.

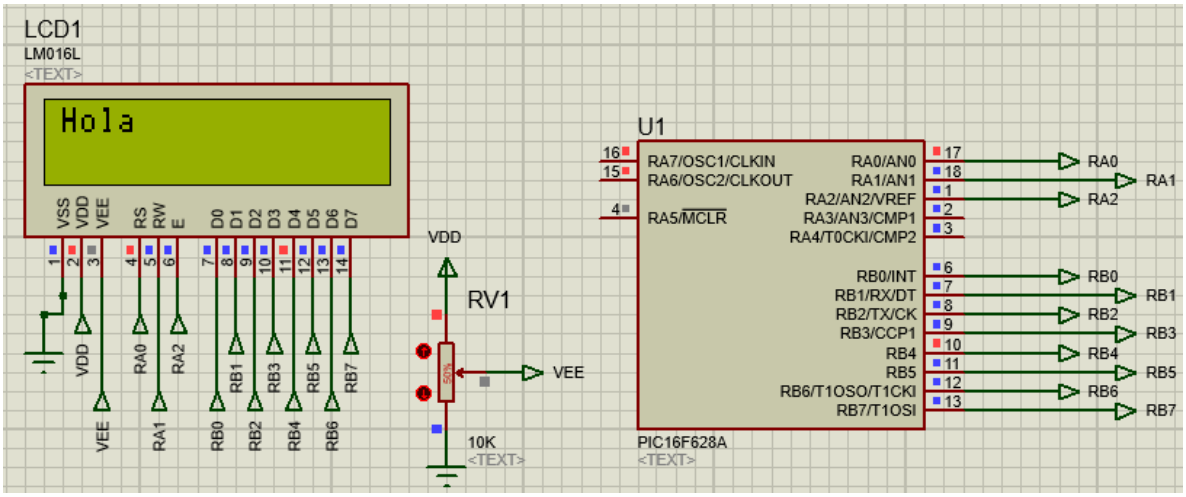


Fig. 7.36 Simulación con proteus, demostración de mensaje “Hola” en pantalla LCD.

Mensajes en una sola línea de código.

Ahora para no escribir una letra por línea y volver a escribir la subrutina “call *LCD_Caracter*”, se puede hacer un programa donde se incluya toda la palabra en una sola línea de código. De esta forma se ahorra espacio y es más eficiente la forma de programación. A continuación, se creará un nuevo código, el nombre del archivo se llamará “LCD2.asm” dentro del mismo proyecto que se ha estado trabajando.

```
; Assembly source line config statements

#include "p16f628.inc"

; CONFIG
; __config 0x3F1C
 __CONFIG _FOSC_INTOSCIO & _WDTE_OFF & _PWRTE_OFF & _MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _CPD_OFF & _CP_OFF

 Org 0                  ; Está línea significa que el programa empezará en la dirección 0 en la memoria de programa

Inicio                  ; Está Linea se considera una etiqueta
    bcf     STATUS,IRP  ; Bit Clear File f,d    Poner a 0 el Bit IRP del Registro STATUS    Acceder al banco 0 y 1
    bcf     STATUS,RP1  ;Ahora para acceder especificamente a un banco
    bsf     STATUS,RP0  ;hay que poner un 0 en RP1 y un 1 en RP0
    clrf    TRISB       ;El PORTB se configura como salida
    clrf    TRISA
    bsf     PCON,OSCF   ;Se establece el oscilador de 4Mhz

    ;Regresar al banco 0 solo hay que poner un 0 en el la bandera RP0

    bcf     STATUS,RP0  ;Regresar al banco 0
    movlw   b'111'
    movwf   CMCON       ;Desactivar el comparador análogico
    bsf     T1CON,0
    bcf     T2CON,2     ;Desactivar temporizadores

    call    LCD_Inicializa  ;Se inicia el LCD para hacer los ajustes pertinentes.
    movlw   Mensaje0        ;Despues lo que se tenga en "Mensaje0" se cargará en W.
    call    LCD_Mensaje     ;Visualiza el mensaje
    sleep                   ;El microcontrolador está consumiendo menos energía.
```

Fig. 7.37 Programa de inicio para nueva instrucción de la subrutina “LCD_4BITS.INC”

Aquí hay un código nuevo, en la línea 31 "*call LCD_Mensaje*", esta subrutina visualiza mensajes fijos, es decir, con la ayuda de otras rutinas anidadas se encargan de que la palabra se tome desde una sola línea de código, enseguida la instrucción "*sleep*" ayuda al microcontrolador a consumir menos energía.

```
33
34      Mensajes
35          addwf   PCL,1   ;Sumar lo que tenga W al contador de programa(PCL) y guardarlo en el mismo PCL
36
37      Mensaje0
38          DT      "Hola!, Que tal?",0     ;Mensaje que se visialuzará en LCD.
39
40         ;Librerias
41      #include "LCD_4BIT.INC"
42      #include "LCD_MENS.INC"
43      #include "RETARDOS.inc"
44
45      END             ;Fin del Programa.
```

Fig. 7.38 Programa para mostrar más caracteres en una sola línea de código.

Lo que sigue en la línea 34 va de la mano con las subrutinas que tiene la librería "*LCD_MENS.INC*" ya que está se encarga de identificar la etiqueta "Mensajes" fuera de la librería para poder continuar con el proceso de obtención y visualización de los mensajes. Lo mismo sucede con la etiqueta "Mensaje0". Estas dos etiquetas son obligatorias si se requiere mostrar este tipo de mensajes por el LCD.

Mensajes en movimiento.

Si se requieren mostrar mensajes que sean más largos que 16 caracteres que es lo que permite la pantalla LCD, se pueden visualizar con movimiento, es decir, el mensaje saldrá completo, desplazándose de derecha del LCD a la izquierda de este. La subrutina "*LCD_MensajeMovimiento*" que se encuentra dentro de la librería "LCD_MENS.INC", se encargará de hacer este proceso más sencillo. Se creará un nuevo programa el cual se llamará "LCD3.asm" dentro del proyecto programación elemental.

```
1   ; PIC16F628 Configuration Bit Settings
2
3   ; Assembly source line config statements
4
5   #include "p16f628.inc"
6
7   ; CONFIG
8   ; __config 0x3F1C
9    __CONFIG _FOSC_INTOSCIO & _WDTE_OFF & _PWRTE_ON & _MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _CPD_OFF & _CP_OFF
10
11   Org 0                  ; Está línea significa que el programa empezará en la dirección 0 en la memoria de programa
12
13  Inicio                  ; Está Linea se considera una etiqueta
14      bcf     STATUS,IRP  ; Bit Clear File f,d    Poner a 0 el Bit IRP del Registro STATUS    Acceder al banco 0 y 1
15      bcf     STATUS,RP1  ;Ahora para acceder especificamente a un banco
16      bsf     STATUS,RP0  ;hay que poner un 0 en RP1 y un 1 en RP0
17      clrf    TRISB       ;El PORTB se configura como salida
18      clrf    TRISA       ;El PORTA se configura como salida
19      bsf     PCON,OSCF   ;Se establece el oscilador de 4Mhz
20
21      ;Regresar al banco 0 solo hay que poner un 0 en el la bandera RP0
22
23      bcf     STATUS,RP0  ;Regresar al banco 0
24      movlw   b'111'
25      movwf   CMCON       ;Desactivar el comparador análogico
26      bsf     T1CON,0
27      bcf     T2CON,2     ;Desactivar temporizadores
28
29      call    LCD_Inicializa  ;Se inicia el LCD para hacer los ajustes pertinentes.
```

Fig. 7.39 Configuración del PIC para la ejecución del programa de mensaje en movimiento.

Al configurar el microcontrolador con la ayuda del registro "Configuración de palabra (PC)" se puede notar que el PWRTE en la línea 9 del programa significa que el temporizador *Power up* está encendido, esto ayuda para que cuando se inicie el microcontrolador alcance el voltaje deseado de operación gracias al retardo que tiene configurado de fábrica, funciona con un pequeño oscilador interno del propio PIC. Esto varía dependiendo de cada microcontrolador [16].

Hasta el momento el programa se comporta de la misma manera que el anterior.

```
30
31  Principal
32      movlw   Mensaje0        ;Despues lo que se tenga en "Mensaje0" se cargará en W.
33      call    LCD_MensajeMovimiento           ;Visualiza el mensaje
34      goto    Principal       ;Repetición de la visualización.
35
36  Mensajes
37      addwf   PCL,1   ;Sumar lo que tenga W al contador de programa(PCL) y guardarlo en el mismo PCL
38
39  Mensaje0
40      DT      "                "      ;Mensaje que se visialuzará en LCD.
41      DT      "Asi es un mensaje en"
42      DT      " Movimiento"
43      DT      "                ",0
44
45     ;Librerias
46  #include "LCD_4BIT.INC"
47  #include "LCD_MENS.INC"
48  #include "RETARDOS.inc"
49
50  END             ;Fin del Programa.
```

Fig. 7.40 Programa para mostrar mensaje en movimiento por pantalla LCD.

La etiqueta “Principal” aparece para que la instrucción *goto* en la línea 34 haga el ciclo repetitivo de visualización del mensaje. Desde la línea 39 a 43 se aprecia la forma de colocar un mensaje para tener una mejor visualización en el LCD. Los espacios en blanco son 16, esto para que el mensaje al acabar e iniciar se pueda leer bien.

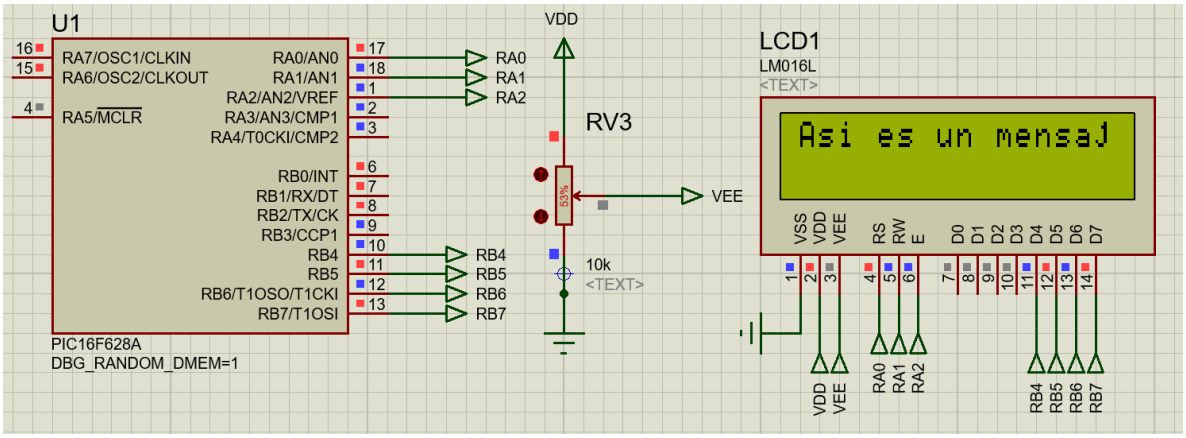


Fig. 7.41 Simulación con proteus, resultados de un mensaje en movimiento.

TIMERO

Es un registro de propósito especifico, aunque no cuente con flags o bits para configuración de este registro. Más bien este registro puede ser utilizado como contador y también pude funcionar como temporizador. Este registro existe para tener un mejor control de tiempos ya que es más preciso que el llamado a la subrutina de tiempos (Retardos).

TMR0 como Contador.

En esta forma el microcontrolador empieza a incrementar un registro de un byte, es decir el conteo llegará hasta 255 si empieza a incrementar desde 0 o también se puede hacer de manera descendente. El pin 4 es el encargado de hacer el conteo, la función T0CKI se debe activar ya que por esta patita se introducirán señales o pulsos digitales de forma externa y se iniciará un conteo ascendente según se configure la bandera T0SE del registro OPTION, para saber de qué manera actuará, en el Capítulo 4 se habla sobre este registro y la función de la bandera.

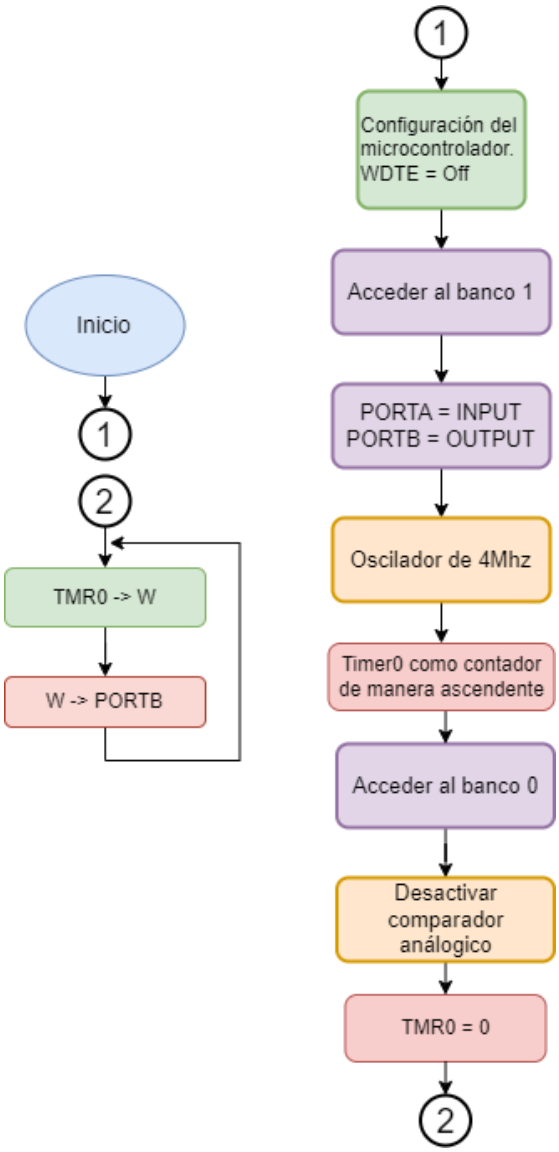


Fig. 7.42 Diagrama de flujo. Timer 0 como contador ascendente.

Está función del registro TMR0 es bastante útil si se requiere incrementar sin ocupar la instrucción de incremento (*incf f,d*) , ahorrando así espacio en la memoria de programa y un código más limpio, además de que el control del conteo es fácil de obtener y así poder

procesarlo a otra variable como mejor convenga, ya que se puede incrementar cuando lee un cero o un uno. La Fig. 7.43 muestra cómo se configura el registro OPTION y cómo actúa el registro TMR0 de forma en configuración de conteo.

```
6       __CONFIG _FOSC_INTOSCIO & _WDTE_OFF & _PWRTE_OFF & _MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _CPD_OFF & _CP_OFF
7       Org 0                   ; Está línea significa que el programa empezará en la dirección 0 en la memoria de
8       Inicio                  ; Está Linea se considera una etiqueta
9         bcf     STATUS,IRP    ; Bit Clear File f,d    Poner a 0 el Bit IRP del Registro STATUS    Acceder al banc
10        ;Ahora para acceder especificamente a un banco
11        ;hay que poner un 0 en RP1 y un 1 en RP0 para acceder a banco 1
12        bcf     STATUS,RP1
13        bsf     STATUS,RP0
14        clrf    TRISB         ;El PORTB se configura como salida
15        movlw   b'11111111'   ;Se mueve la literal 255 a W
16        movwf   TRISA         ;El PORTA se configura como Entrada
17        bsf     PCON,OSCF     ;Se establece el oscilador de 4Mhz
18        movlw   b'00111000'   ;Configuración del Timer0 como contador
19        movwf   OPTION_REG    ;De manera ascendente
20        ;Regresar al banco 0 solo hay que poner un 0 en el la bandera RP0
21        bcf     STATUS,RP0    ;Regresar al banco 0
22        movlw   b'111'
23        movwf   CMCON         ;Desactivar el comparador análogico
24        bsf     T1CON,0
25        bcf     T2CON,2       ;Desactivar temporizadores
26        clrf    TMR0          ;Limpiar el Registro Timer0
27    Principal
28        movf    TMR0,0
29        movwf   PORTB
30        goto    Principal
31        END
```

Fig. 7.43 TMR0 como contador ascendente.

En la línea 18 del programa se alcanza a ver la configuración para que el TMR0 se active (T0CS = 1) y, que cuando detecte un uno (T0SE = 1) en la entrada del pin 4 (T0CKI) el registro TMR0 se incremente automáticamente. Lo que sigue después es el nombre del registro (OPTION_REG) para pasarle la literal de configuración ¿Por qué OPTION_REG y no solo OPTION? OPTION es una palabra reservada del entorno de desarrollo y marcará error al compilar, es por eso por lo que se debe nombrar OPTION_REG. Lo que sigue después de la etiqueta Principal es solo mostrar por el puerto B como se va incrementando el conteo.

El circuito quedaría de la siguiente manera. Fig. 7.44 mostrando que cada vez que se dé un pulso con el botón en la línea RA4 pasará un 1(HIGH) como información y el registro TMR0 se incrementará y en los LED's que están en el PORTB se visualizará el conteo de forma binaria. Si se desea que el conteo se incremente cuando el RA4 lea un 0(DOWN) entonces la entrada debe ser de tipo pull-up y la configuración del registro OPTION_REG en la bandera T0SE debe ser igual a 0.

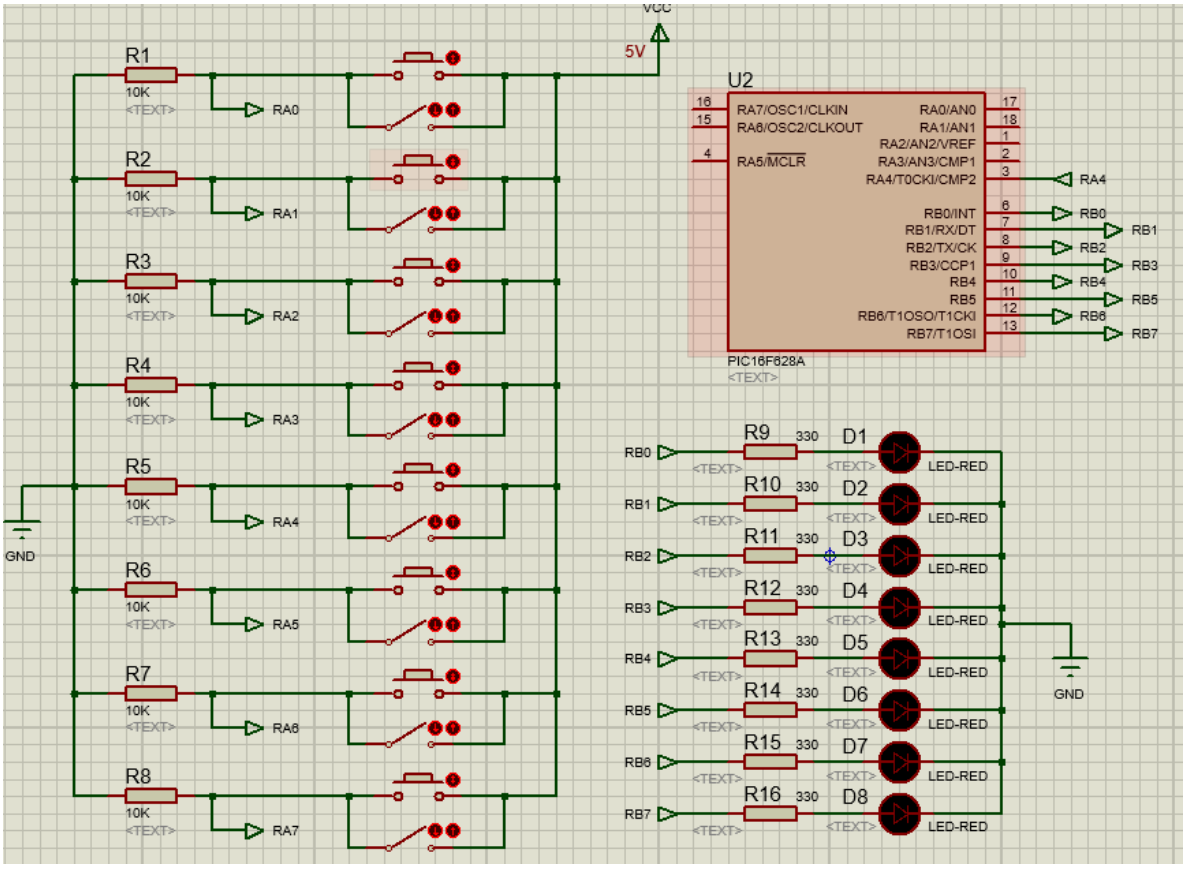


Fig. 7.44 Circuito para mostrar el comportamiento del TMR0 como contador.

Temporizador

Cuando el TMR0 trabaja como temporizador cuenta los ciclos máquina del microcontrolador, recordar que en esta ocasión se está trabajando con el oscilador interno de 4 MHz y se sabe que una instrucción de un ciclo máquina se tarda un microsegundo, en el subtema de rutinas de tiempo de este capítulo se explica el tiempo en micro segundos de instrucciones de uno o dos ciclos máquina, dicho esto, como se trata de un contador ascendente, el TMR0 debe ser cargado con el valor de los impulsos que se desean contar, restados de 256 que es el valor de desbordamiento [14, p. 246]. Por ejemplo, para contar cuatro microsegundos, el timer se carga con el valor de 252 ya que 256 – 4 = 252. Cuando el TMR0 se desborda el flag T0IF del registro INTCON se activa.

Otra cosa que se debe tener en cuenta es el prescaler (Tabla 4.2), los bits que van desde el bit 0 al bit 2 del OPTION_REG son para determinar un tiempo más largo de pausa con un numero de pulsos menores, esto es muy útil ya que si solo se hicieran con los 8 bits del MCU entonces solo se podría contar 255 microsegundos, pero existe una fórmula para determinar tiempos de espera más largos.

$$Temporización = T_{CM} * Prescaler * (256 - Carga\ TMR0)$$

Donde:

Temporización, es el tiempo deseado de pausa.
TCM, es el periodo de un ciclo máquina (1us) que ya se calculó en el subtema, rutinas de tiempo.
Prescaler, es el rango de divisor de frecuencia elegido.
(256 – Carga TMR0), es el número total de impulsos a contar por el TMR0 antes de desbordarse en la cuenta ascendente.

Ya que se entiende está parte, el programa ejemplo constará de encender y apagar un motor de una bomba de aire para que llene el cofre de burbujas durante 1 segundo y así los peces se puedan divertir. Primero hay que calcular la carga en TMR0. Un millón de microsegundos es igual a un segundo, solo como dato.

$$1{,}000{,}000 = 1 * 256 * (256 - CargaTMR0)$$

$$CargaTMR0 = -3{,}650.25$$

Hay un problema aquí. El registro TMR0 tiene espacio para un número máximo como 255, no puede almacenar un número mayor a 255 y mucho menos un valor negativo, entonces ¿Qué se puede hacer? Se calcula que solo 65,536 microsegundos se pueden contar, en la siguiente formula se demuestran los resultados obtenidos.

$$x = 1 * 256\ (256 - 0)$$

$$x = 65{,}536$$

El cero en *CargaTMR0* representa un conteo desde cero del registro TMR0 hasta el número total de impulsos a contar, de esa forma quedan 256 microsegundos libres antes de que se desborde el TMR0 y se si multiplica por el preescaler mayor que puede procesar el PIC, de esta manera el límite en microsegundos a contar es de 65,536. Para saber cuántas veces se debe repetir *X,* basta con dividir el segundo entre los 65,536 microsegundos. Como resultado se obtiene el 15. El número 15 se deberá guardar en una variable e ir decrementándola por cada desborde de TMR0 hasta que llegue a 0 está variable y de esa forma se cumplirá el tiempo pedido.

Para tener una forma más clara se presentará en la Fig. 7.45 el diagrama de flujo.

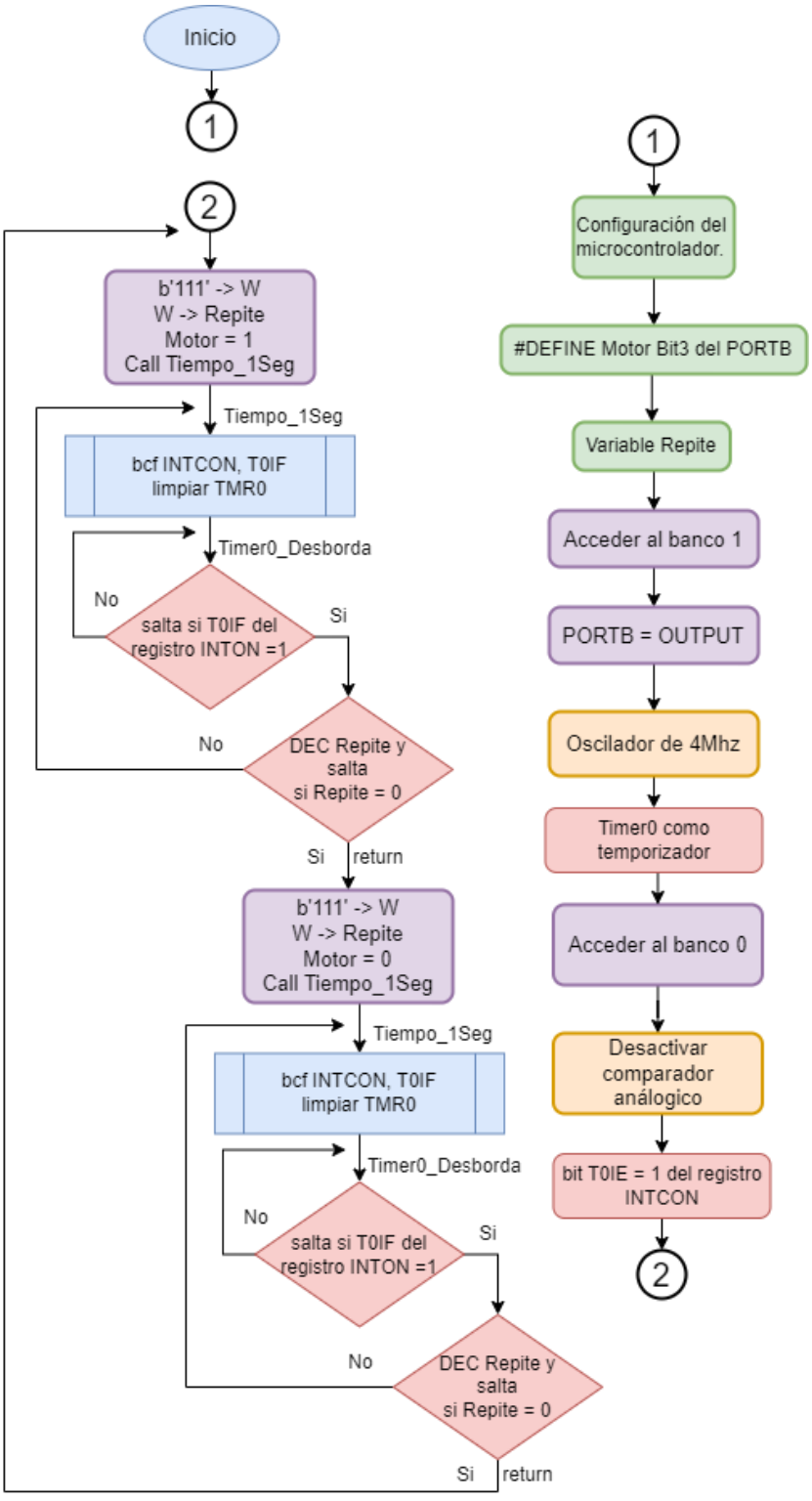


Fig. 7.45 Diagrama de flujo, TMR0 como temporizador.

Ahora se creará un nuevo archivo, el cual se llamará TemporizadorTMR0.asm dentro del proyecto de programación elemental.

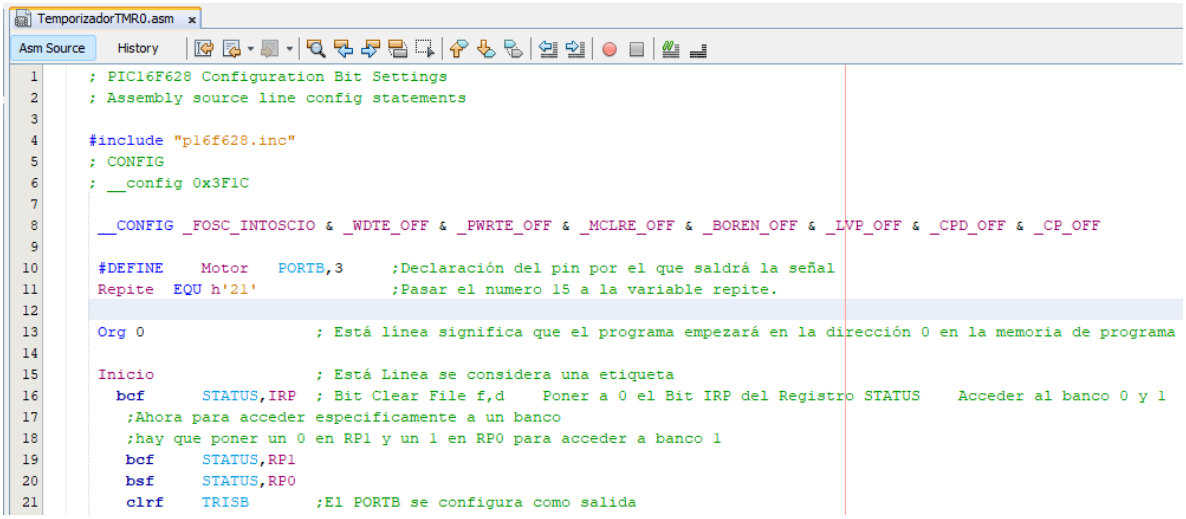


Fig. 7.46 Programa TIMER0 como temporizador, declaración de variables.

En la línea 10 de define que el pin4 o bit3 del puerto B será asignada a una variable, la cual se llamará Motor. En la línea 11 se declara una variable en el espacio de memoria 21 en hexadecimal. Lo que sigue después ya se conoce.

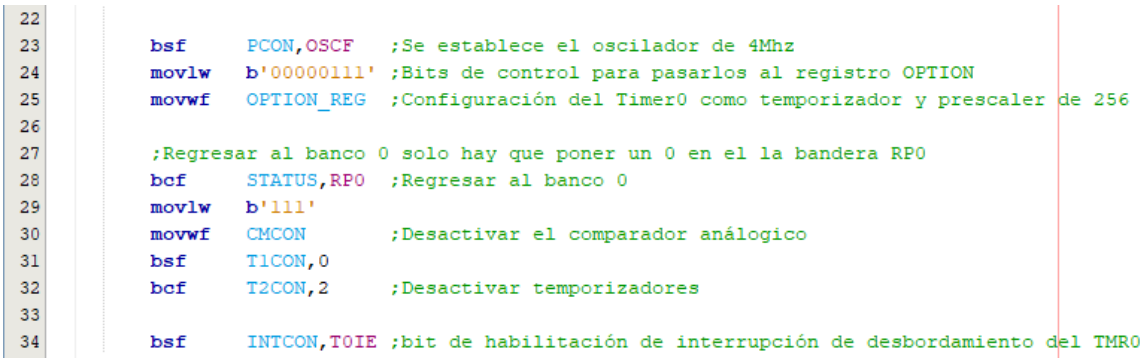


Fig. 7.47 Programa TIMER0 como temporizador, configuración del registro “OPTION_REG”.

En la línea de código 24 se especifica el preescaler y de igual forma los bits de control para el TMR0.

En la línea 34 el registro INTCON se involucra con el TMR0 ya que la bandera T0IE será la encargada de decir la función de desbordamiento, o no, del TMR0, obviamente se requiere que esté habilitado para revisar cuando se desborde el TMR0.

```
35  Principal
36      movlw   b'1111'     ;Mover el 15 a W
37      movwf   Repite      ;Y pasarlo a la variable Repite
38      bsf     Motor       ;Prender o poner a uno el pin 4 del PIC
39      call    Tiempo_1Seg ;Llamado a la subrutina
40      movlw   b'1111'     ;Mover el 15 a W
41      movwf   Repite      ;Y pasarlo a la variable Repite
42      bcf     Motor       ;Apagar o poner a cero el pin 4 del PIC
43      call    Tiempo_1Seg ;Llamado a la subrutina
44      goto    Principal   ;Regreso a la etiqueta Principal
45
46  Tiempo_1Seg             ;Subrutina
47      bcf     INTCON,T0IF ;Aún no se desborda el TMR0
48      clrf    TMR0        ;Limpiar el Registro Timer0
49  Timer0_Desborda         ;Subrutina de revisión de desbordamiento en TMR0
50      btfss   INTCON,T0IF     ;Constante revisión del desbordamiento
51      goto    Timer0_Desborda ;Si no se ha desbordado regresará a verificar de nuevo
52      decfsz  Repite,1        ;Si ya se desbordo el TMR0 se decrementará la variable Repite
53      goto    Tiempo_1Seg     ;Si la variable no se decremntó hasta llegar a cero entonces se empieza de nuevo
54      return                  ;Hasta llegar a cero y de esa forma retornará a la línea del programa principal
55
56      END
```

Fig. 7.48 Programa TIMER0 como temporizador, programa principal.

Aquí es donde sucede la magia, se mueve el numero 15 a una variable porque es el número de veces que se tiene que repetir de acuerdo con el resultado obtenido anteriormente, enseguida el bit 4 del PIC se enciende y pasa a la subrutina “Tiempo_1Seg” que es la encargada de dar el tiempo correcto.

En la línea 47 se empieza con limpiar el bit de revisión de desbordamiento y se limpia el TMR0 por si se tenía algún valor cargado, de la línea 49 a la 53 se entra a un bucle donde el TMR0 como temporizador empieza a trabajar, cuando se terminan las subrutinas ya habrá pasado un segundo y de nuevo se carga la variable “Repite” en la línea 40 con el valor de 15 ya que, si no se hace, la variable seguiría decrementándose y después de 0 está comienza desde 255 y habría una fuga de tiempo.

Teclado matricial

Este hardware se puede ver en muchos dispositivos, como en las calculadoras, los teclados de contraseñas en cajas fuertes, incluso en el teclado de la computadora. Son de mucha utilidad puesto que los push button están ocultos debajo de esas teclas de plástico, las cuales facilitan la identificación de números, letras o caracteres especiales, que se forman como una matriz de ahí su nombre de teclado matricial y así cuando se presiona una tecla, el botón que se encuentra en esa fila y columna determina que símbolo le pertenece. En la Fig. 7.49 se muestra de qué forma se puede hacer una matriz de un teclado hexadecimal.

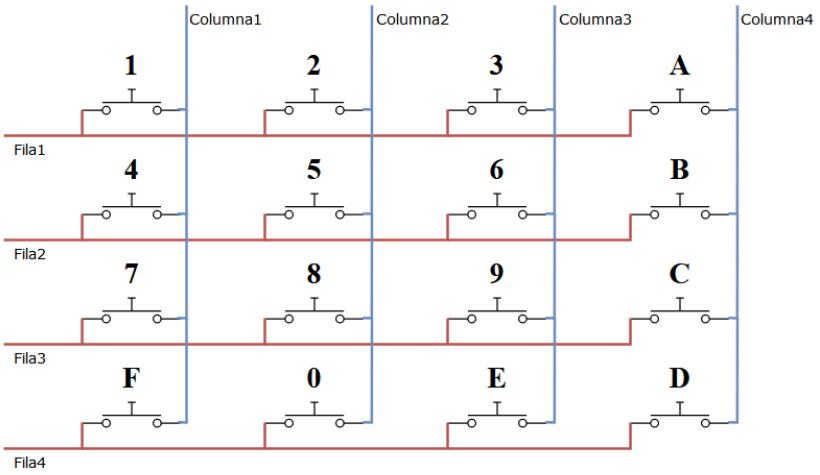


Fig. 7.49 Teclado matricial hexadecimal

La principal ventaja de usar este tipo de arreglo para los teclados es que, si se tienen las 16 teclas, por ejemplo, se deberían ocupar 16 líneas del PIC, es decir 16 pines y con está matriz se reduce a 8. De esta forma se aprovechan los demás pines del microcontrolador.

Para el siguiente ejemplo se ocupará un circuito como en la Figura siguiente. Fig. 7.50

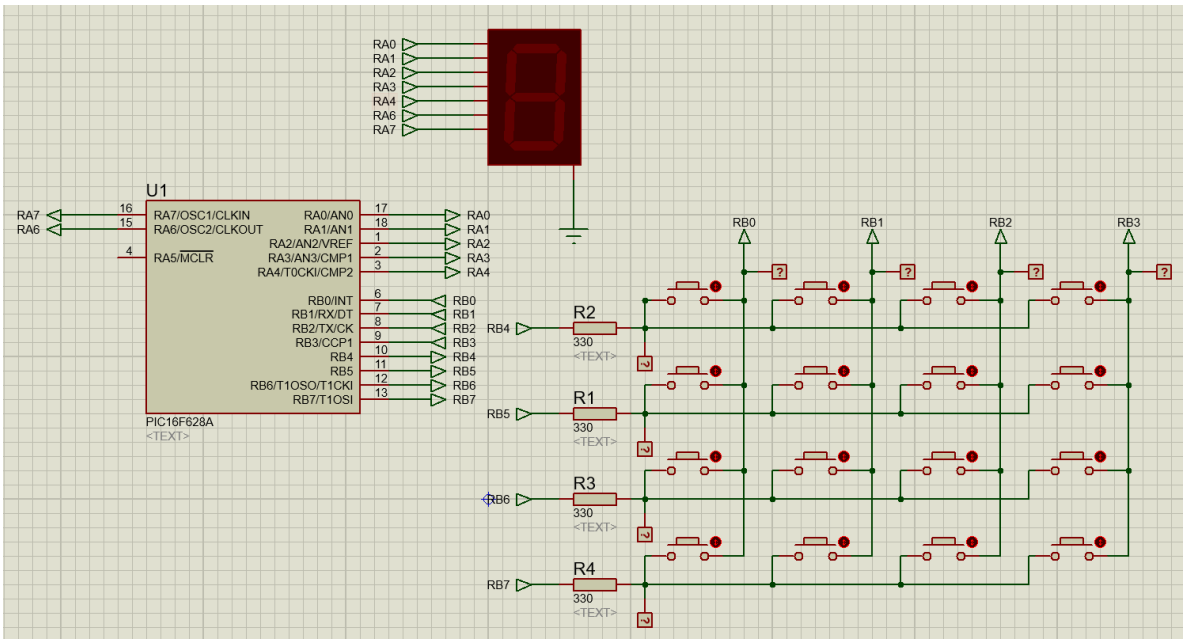


Fig. 7.50 Teclado Matricial con Display de siete segmentos.

Dentro del proyecto “Programación elemental” se creará un nuevo archivo ensamblador con el nombre “TecladoMatricial.asm” para tener el programa prueba.

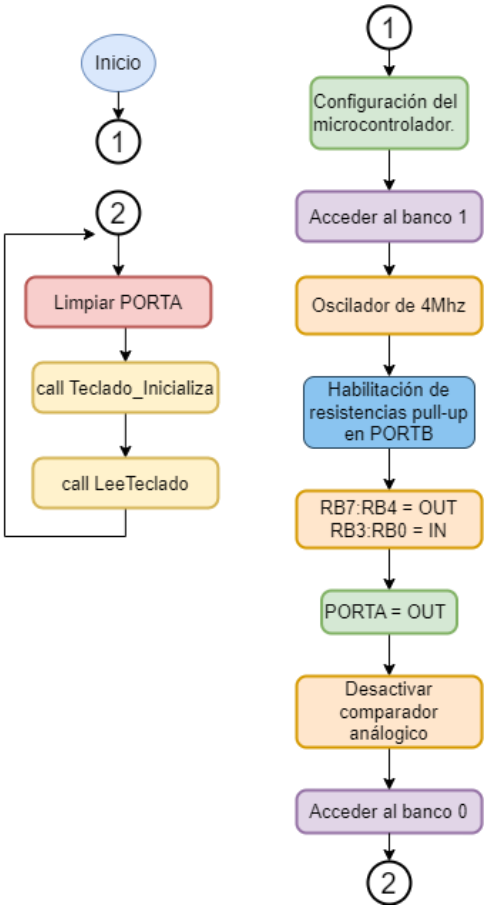


Fig. 7.51 Diagrama de flujo, programa principal teclado matricial.

```
; PIC16F628 Configuration Bit Settings

; Assembly source line config statements

#include "p16f628.inc"

; CONFIG
; __config 0x3F1C
 __CONFIG _FOSC_INTOSCIO & _WDTE_OFF & _PWRTE_ON & _MCLRE_OFF & _BOREN_OFF & _LVP_OFF & _CPD_OFF & _CP_OFF

 Org 0                  ; Está línea significa que el programa empezará en la dirección 0 en la memoria de programa
;========================================
 ;Inicio del Programa
;========================================
Inicio
 ;====Inicializa el microcontrolador y activa las resistencias Pull-up Internas del microcontrolador

 ;=================================================
    ; Habrá que configurar primero el microcontrolador.
;=================================================

        bcf     STATUS,IRP      ; Bit Clear File f,d    Poner a 0 el Bit IRP del Registro STATUS    Acceder al banco 0 y 1
        ;====Ahora para acceder especificamente a un banco hay que poner un 0 en RP1 y un 1 en RP0
        bcf     STATUS,RP1
        bsf     STATUS,RP0
        bsf     PCON,OSCF       ;Se establece el oscilador de 4Mhz
```

Fig. 7.52 Programa teclado matricial, configuración inicial.

Hasta este punto la configuración del microcontrolador ha sido la misma. Habrá un cambio después de la línea 26.

```
21
22  bcf     STATUS,IRP      ; Bit Clear File f,d    Poner a 0 el Bit IRP del Registro STATUS    Acceder al banco 0 y 1
23  ;====Ahora para acceder especificamente a un banco hay que poner un 0 en RP1 y un 1 en RP0
24  bcf     STATUS,RP1
25  bsf     STATUS,RP0
26  bsf     PCON,OSCF       ;Se establece el oscilador de 4Mhz
27  movlw   b'00001111'     ; <RB7:RB4> salidas, <RB3:RB0> entradas
28  movwf   TRISB           ; Mover lo que hay en W al PuertoB
29  bcf     OPTION_REG,NOT_RBPU     ; Habilita resistencia de Pull-Up del Puerto B.
30  movlw   b'00100000'             ;El puerto A sera salida excepto el bit 5
31  movwf   TRISA                   ;por motivos de configuración de fabrica del pic
32  bcf     STATUS,RP0      ;Regresar al banco 0
33  movlw   b'111'
34  movwf   CMCON   ;Desactivar el comparador análogico
35  bsf     T1CON,0
36  bcf     T2CON,2 ;Desactivar temporizadores
37
```

Fig. 7.53 Programa teclado matricial, habilitación de resistencia pull-up.

En la línea 27 se percata que hay un valor que se cargará a W el cual tratará de decirle a los bits RB7, RB6, RB5 y RB4 que actuarán como salida y los restantes como entrada de datos. En la línea 29 se pone a cero el octavo bit del registro “OPTION_REG” para habilitar resistencias pull-up internas que tiene el PIC16F628. Hay que recordar que este tipo de resistencias se vieron en el capítulo 3 en los circuitos periféricos de entrada de datos.

¿De qué se trata habilitar estas resistencias?

Para que el PIC sepa que tecla se está oprimiendo se aplica a las filas un nivel bajo (0V) y a las columnas un nivel alto (5V) ¿Cómo es posible que tengan voltaje si las columnas fueron configuradas como entrada de datos? La magia está en las resistencias pull-up, internamente el microcontrolador manda ese voltaje y cambiara a cero cuando el botón este aterrizado porque las filas le dan el nivel bajo de voltaje, así cambia de estado. De esta forma es muy eficaz y eficiente en cuestión de que no se utilizará otra línea de voltaje y resistencias extra para no dañar el microcontrolador físicamente. Es decir, si no so se hubiera conocido está función del PIC el circuito quedaría de la siguiente manera Fig. 7.54

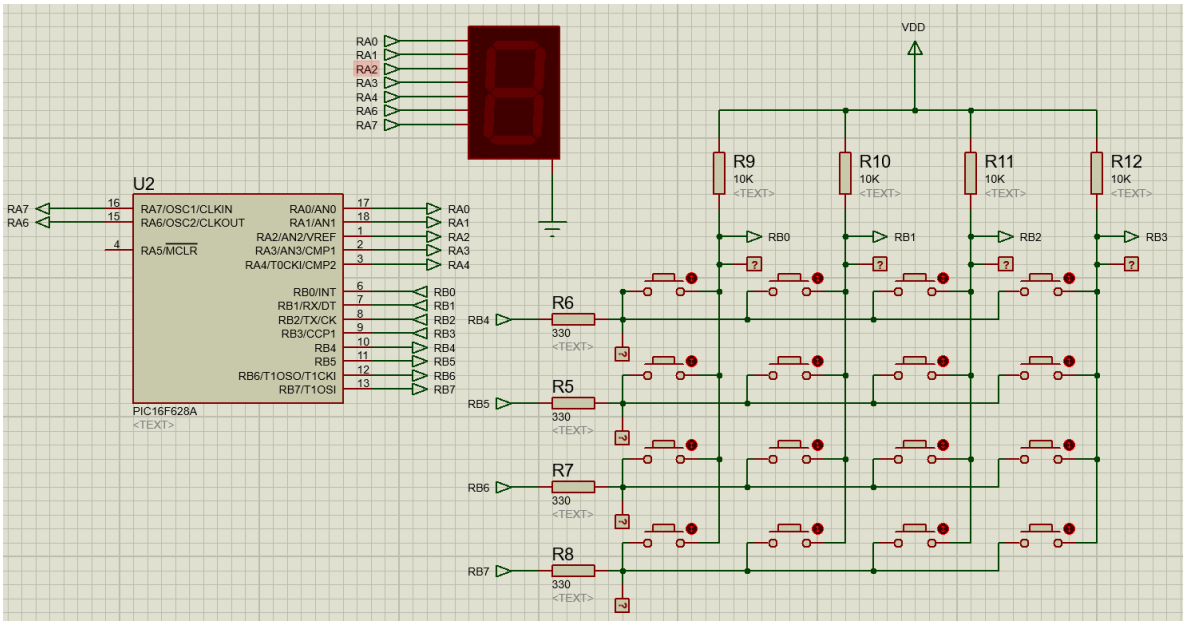


Fig. 7.54 Circuito teclado matricial con resistencias pull-up externas al PIC.

Una vez activadas las resistencias pull-up en la línea de código 30 y 31 se configura el puerto A como salida de información, esto porque el display de siete segmentos mostrará el resultado de la tecla oprimida. El código que sigue ya se ha explicado en capítulos anteriores.

```
37
38  ;=====================================================
39      ;Programa Principal
40          ;Desde está parte se llamarán a subrutinas o subprogramas ubicados en las librerías.
41  ;=====================================================
42
43   Principal
44      clrf        PORTA
45      call        Teclado_Inicializa  ;====Así mismo está subrutina detecta si ya se pulso o no una tecla.
46      call        LeeTeclado          ;Se empieza a leer que tecla se está pulsando
47      goto        Inicio
48
49      #include "RETARDOS.inc"    ;Libreria de retardos
50      #include "TECLADO2.INC"     ;Libreria de Teclado.
51
52      END
```

Fig. 7.55 Programa principal teclado matricial.

Ahora la instrucción "*clrf PORTA*" limpiará el puerto A por si hay algún ruido eléctrico, así no mostrará nada. Después se llama a una subrutina "Teclado_Inicializa", se abrirá una nueva ventana. Antes de mostrar la subrutina se creará el diagrama de flujo de esta la cual se encuentra dentro de la librería "TECLADO2.INC".

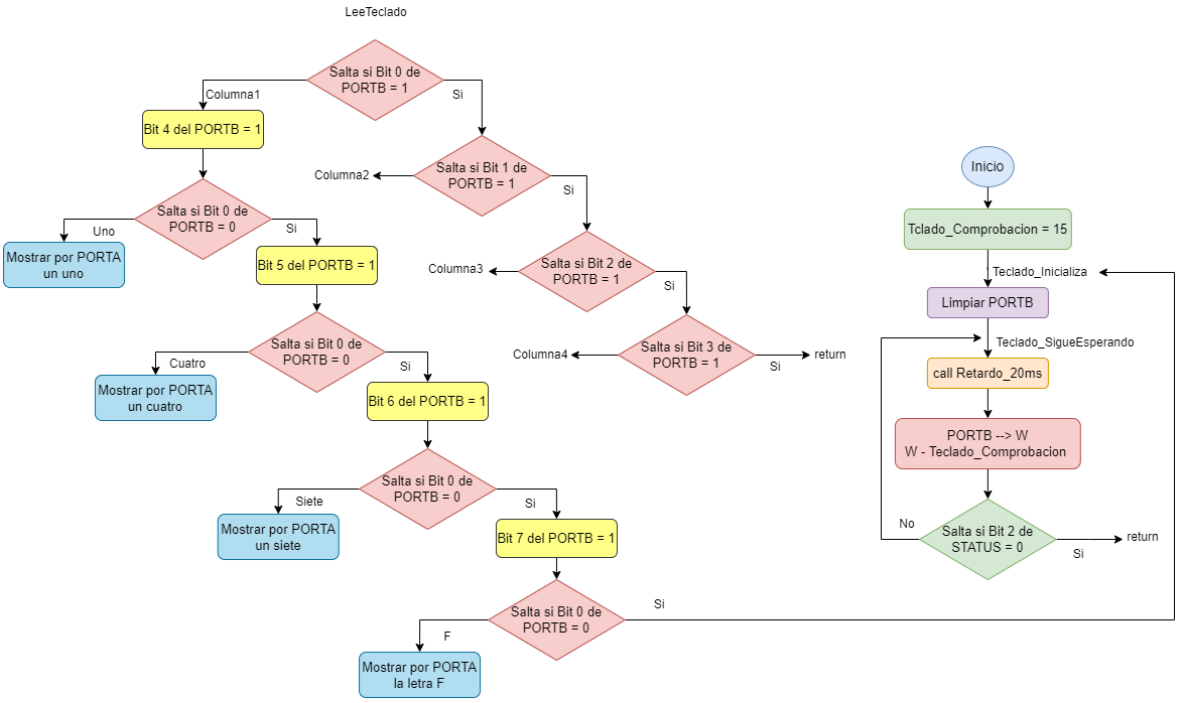


Fig. 7.56 Diagrama de flujo, librería “TECLADO2.inc”

Explicando el diagrama de flujo, lo que sigue en “Columna2” es algo parecido a lo que sucede en la “Columna1” Solo que la diferencia en “Columna2” se está monitoreando el bit1 del puerto B he ira saltando para seguir monitoreando la fila, es decir, desde el bit 4 hasta el bit 7 del puerto B y así también se hará con el bit 2, y el bit 3 del puerto B, se tendrá en constante monitoreo las filas del puerto B para saber si el usuario ya oprimió otra tecla.

La “Columna2” mostrará por el puerto A los números 2,5,8 y 0. Al final de la última iteración regresa a “Teclado_Inicializa” para tener un bucle de constante monitorización del teclado. La “Columna3” mostrara por el puerto A los números 3,6,9 y la letra E. La “Columna4” mostrara por el puerto A las letras A, B, C y D.

TecladoMatricial.asm | TECLADO2.INC

Asm Inc Source | History

```
;============= Libreria TECLADO2.INC ==========

    ;==== Zona de datos ====

    CBLOCK

    ENDC
    Teclado_Comprobacion    EQU b'00001111' ;Variable para comprobacion

;=====Inicia el Programa==========

    Teclado_Inicializa
        clrf    PORTB                   ;Se limpia el PORTB

    ;======== Bucle para estar en constante monitorizacion de que se haya pulsado una tecla o no
    Teclado_SigueEsperando
        call    Retardo_20ms            ;Esperar la estabilizacion de niveles de tension en los botones
        movf    PORTB,W                 ;Si se presiono algun boton, se guardara en W
        sublw   Teclado_Comprobacion    ;Para saber si se presiono alguno, se restara lo que tiene la variable
        btfsc   STATUS,Z                ;menos lo que tiene el puerto B y revisa si el resultado es cero. Si no retorna
        goto    Teclado_SigueEsperando  ;Si es cero entonces regresa a Teclado_SigueEsperando
        return
```

Fig. 7.57 Programa teclado matricial, inicio de la librería “TECLADO2.INC” y de la subrutina “Teclado_Inicializa”.

Esta nueva ventana trata de la librería “TECLADO2.INC”. Aquí se desarrolla el procesamiento de información. “Teclado_Comprobacion” es una variable que se ocupa en la línea 19 para saber si se está pulsando una tecla o no. Funciona de la siguiente manera; se llama a la subrutina “Retardo_20ms” para que se estabilicen los nieles de tensión cuando se presione un botón, ya que físicamente en la vida real, al presionar el botón rebota y manta picos de voltaje inestables, el retardo de 20 milisegundos da la oportunidad de que el botón se estabilice para mandar un voltaje estable. Sabiendo esa parte, la instrucción “*movf PORTB,W*” ayuda a que W tenga el valor que le correspondía al puerto B, y después “*sublw Teclado_Comprobacion*” restará lo que tenga está variable guardada con lo que se cargó a W, después se revisa el tercer bit o mejor dicho la bandera “*Z*” del registro STATUS y este bit o bandera cambiara de estado si el resultado de la resta fue o no fue cero, en dado caso de que el resultado haya dado cero significa que no se ha presionado tecla alguna y sigue en el bucle desde la línea 16 hasta 21 para que este en constante monitorización. Si el resultado fue diferente de cero entonces sale del bucle y por tanto de la subrutina y retorna hasta donde fue llamada.

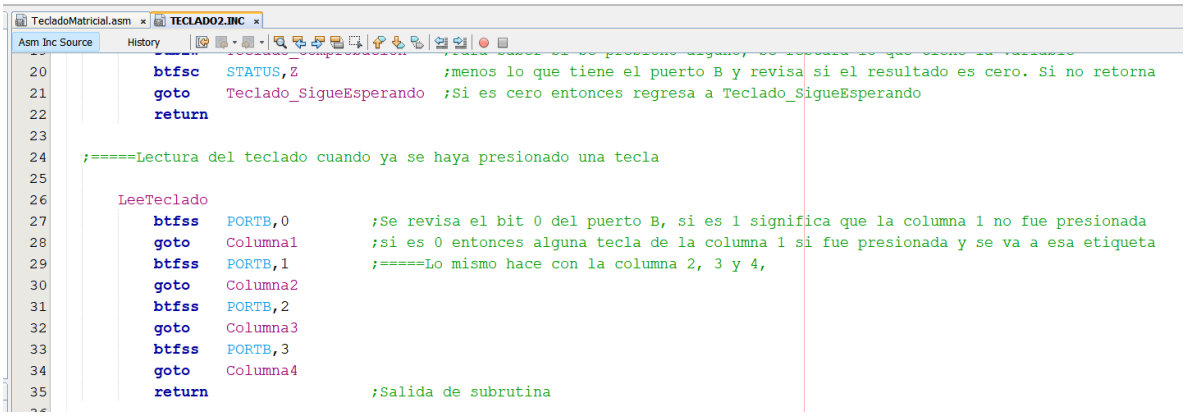


Fig. 7.58 Programa teclado matricial, programa inicial de la subrutina “LeeTeclado”

Después de salir de la subrutina “Teclado_Inicializa” entra en otra subrutina llamada “LeeTeclado” la cual cumple la función de buscar que tecla de alguna columna fue presionada y cuando se detecte que columna fue, el programa se dirige a esa etiqueta correspondiente. Es decir, cuando alguna de las columnas pase de un estado lógico alto a un estado lógico bajo ahí para de seguir buscando la columna, ya que no tendría sentido seguir buscando y se apunta a la respectiva etiqueta.

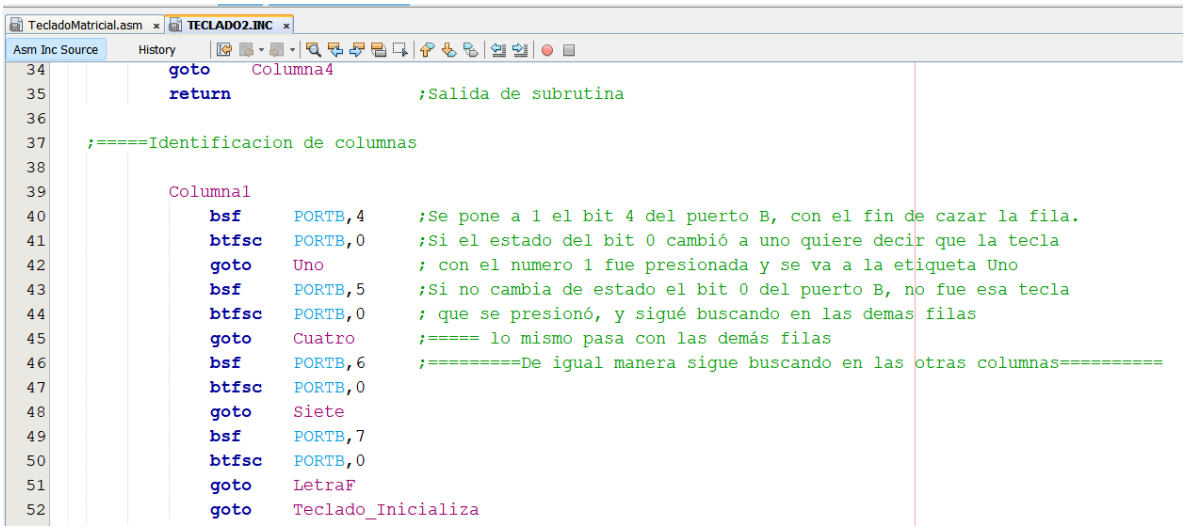


Fig. 7.59 Programa teclado matricial, etiqueta “Columna1”.

Ya que se sabe a qué columna dirigirse el programa, ahora habrá que identificar cual tecla de alguna fila fue presionada. “*bsf PORTB,4*” El bit 4 del puerto B se pone a un nivel lógico alto con el fin de ver si el estado de la columna cambia de un estado de nivel bajo a un estado de nivel alto, si es así entonces el programa sabría a qué tecla pertenece el botón que se ha presionado y así sucesivamente en caso de que no cambie de estado, hasta encontrar la fila correspondiente.

Cuando se haya acabado el proceso de búsqueda y localización de tecla presionada se regresará a “Teclado_Inicializa” para seguir monitoreando si se presiona otra tecla.

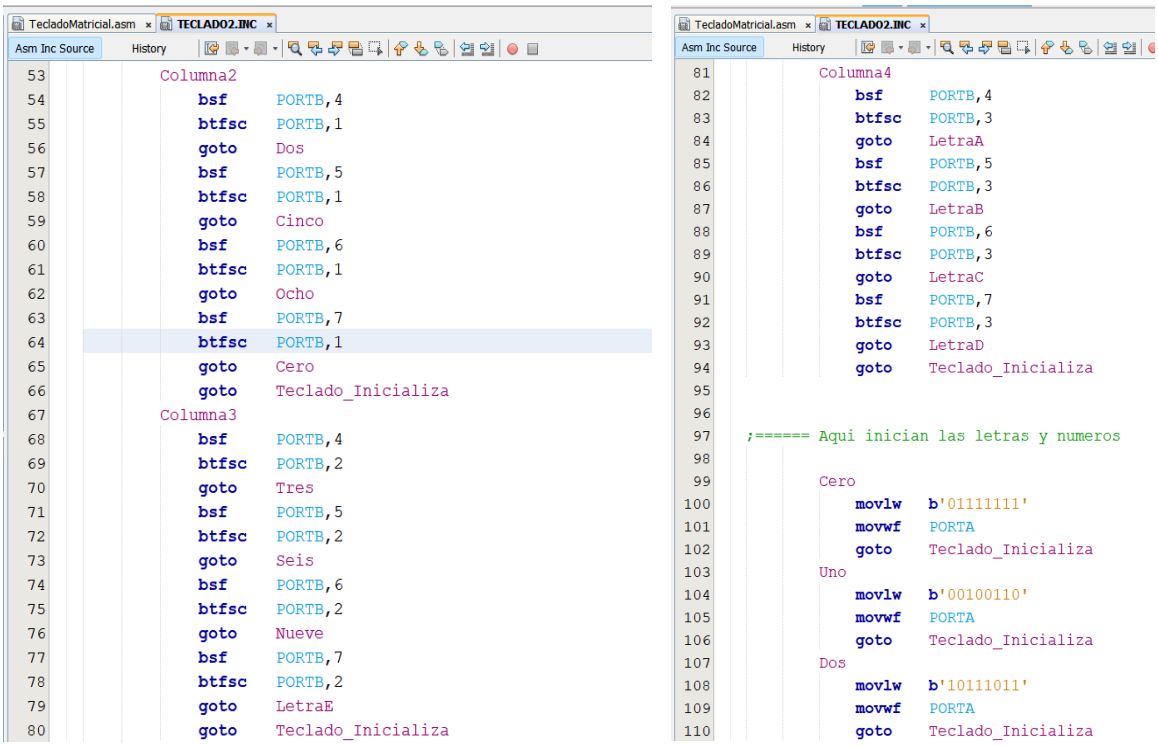


Fig. 7.60 Programa teclado matricial, etiquetas de las columnas, números y letras que se mostrarán en el puerto A.

Lo que sigue es muy sencillo, pues se muestra por el puerto A el numero o letra correspondiente a la tecla y botón presionado. Para saber que bits serán puestos a uno del puerto A en el capítulo 3 se habla del display de 7 segmentos.

EEPROM de datos

En algunos proyectos será necesario guardar la información que se genera durante el proceso de ejecución o interacción con el programa, es decir, esos datos deben permanecer aun cuando el sistema se desconecta de la alimentación. Para realizar esta función la mayoría de los PIC disponen de un área de datos llamado EEPROM no volátil que se describe en este capítulo [14, p. 235].

El PIC16F628 dispone de una zona de 128 bytes de memoria EEPROM, en la tabla 2.2 del capítulo 2 se puede observar las características de esta memoria. Solo se pueden realizar dos operaciones como en cualquier otra memoria EEPROM, una es operación de lectura y la otra de escritura o grabación.

Algunos de los registros involucrados son: EECON1, EECON2, EEDATA y EEADR. Algunos de estos registros se especifican en el capítulo 4.

EEDATA (EEPROM Data Register). Contiene los bytes que se van a escribir o que se han leído de la EEPROM de datos.

EEADR (EEPROM Address Register). Contiene la dirección de la EPROM de datos a la que acceder para leer o escribir.

EECON1 (EEPROM Control Register 1). Los bits de este registro definen el modo de funcionamiento de esta memoria. Los bits RD y WR indican respectivamente lectura y escritura. NO hay que ponerlos a “0” sólo a “1”, ya que se borran automáticamente cuando la operación de lectura o escritura ha sido completada.

EECON2 (EEPROM Control Register 2). Este registro no está implementado físicamente, por lo que es imposible leerlo (si se intenta leer, todos sus bits se leen como ceros) [14, p. 236]. Se emplea como dispositivo de seguridad durante el proceso de escritura de la EEPROM, para evitar las interferencias o ruido.

Una vez explicado esto, El circuito que se necesitará para esta prueba será el siguiente.

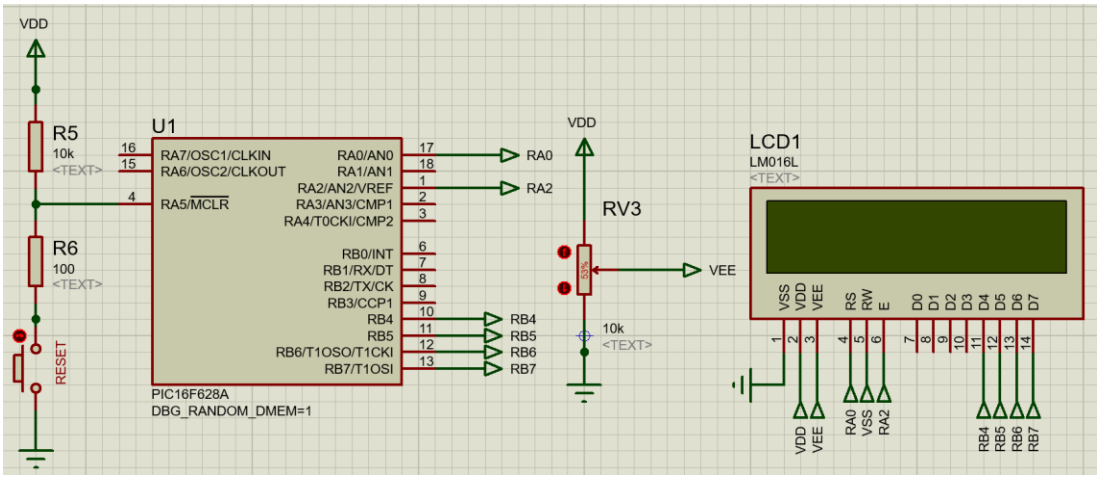


Fig. 7.61 Simulación en proteus, circuito principal para demostración de la EEPROM.

El programa tratará de cada vez que el sistema es reseteado se incrementa un contador que se guarda en la primera posición de la memoria EEPROM de datos del PIC y se visualizará en la LCD cada incremento.

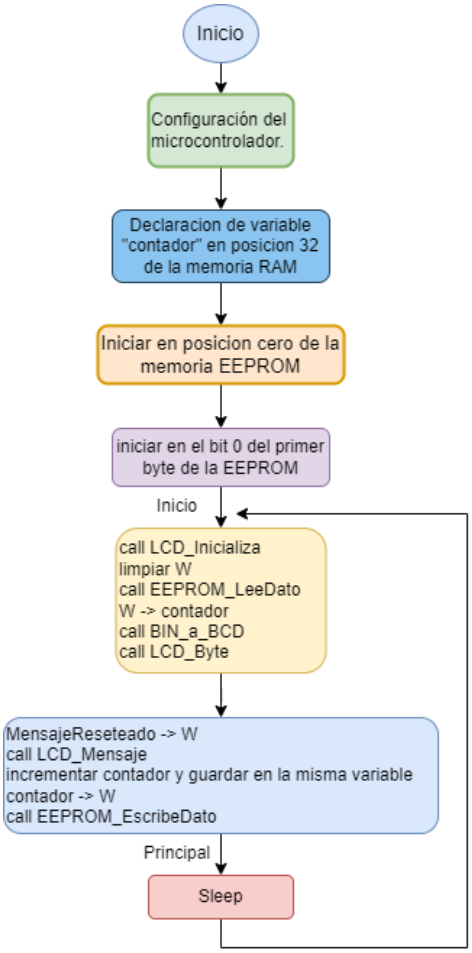


Fig. 7.62 Diagrama de flujo, programa EEPROM.

Se creará un nuevo archivo llamado “Memoria_EEPROM.asm” dentro del proyecto *Programación elemental* Se empezará de la siguiente manera.

```
1   ;Este programa comprueba el funcionamiento de la lectura y escritura en la memoria EEPROM de
2   ;datos. Cada vez que el sistema es reseteado se incrementa un contador que se guarda en la
3   ;primera posición de la memoria EEPROM de datos del PIC y es visualizado en LCD
4
5   include "p16f628.inc"
6
7   ;CONFIG
8   ;__config 0x3F1C
9   __CONFIG _FOSC_INTOSCIO & _WDTE_OFF & _PWRTE_ON & _MCLRE_ON & _BOREN_OFF & _LVP_OFF & _CPD_OFF & _CP_OFF
10
11  CBLOCK 0x32
12  Contador          ;Variable para guardar el incremento cada vez que se resetea el PIC
13  ENDC
14
15  ORG 0x2100              ;Está dirección indica la posición cero de la memoria EEPROM
16                          ;es decir, se empezará a grabar en la EEPROM desde su primer byte
17  DE 0x00                 ;La variable contador empieza en el bit cero dentro del primer byte
18                          ;de la EEPROM.
19  ;Inicio el Programa
20
21  ORG 0
```

Fig. 7.63 Programa EEPROM, configuración inicial del microcontrolador y memoria EEPROM.

En la línea 9 del código se habilito el "*MCLRE_ON*", recordar que esta configuración es para activar el Reset del PIC. Lo que sigue es declarar las variables automáticamente con CBLOCK a partir de la dirección de memoria 0x32, aquí se encontrará el valor de la variable contador.

"ORG 0x2100" es la dirección en la que inicia la memoria EEPROM, si se requiere saber con seguridad en qué dirección inicia la memoria, hay un archivo llamado "**16f628_g.lkr**" que se encuentra en la siguiente ruta "*C:\ProgramFiles(x86)\Microchip\MPLABX\v5.35\mpasmx\LKR*" el cual muestra el inicio y final de cada posición de memoria flash, RAM y EEPROM.

"DE 0x00" indica que la variable "contador" empezará en la dirección cero de la EEPROM.

```
21    ORG 0
22
23    Inicio
24        call    LCD_Inicializa      ;Inicializa la subrutina
25        clrw                        ;Se limpia W y por tanto cuando se llame a la subrutina
26        call    EEPROM_LeeDato      ;"EEPROM_LeeDato" pues vera un 0 y
27        movwf   Contador            ;se guardará en contador de la misma forma
28        call    BIN_a_BCD           ;despues la librería "BIN_a_BCD" es la encargada de poner el numero binario a decimal
29        call    LCD_Byte            ;Se llama a la suburina del display con nibble alto
30        movlw   MensajeReseteado    ;Se mostrara el mensaje que esta en la línea 46
31        call    LCD_Mensaje         ;a traves de la subrutina "LCD_Mensaje"
32        incf    Contador,F          ; se incrementará contador y se gurdará el valor en el mismo
33        movf    Contador,W          ;despues se moverá lo que tenga la variable contador a W
34        call    EEPROM_EscribeDato  ;En este punto se grabará el valor en la EEPROM de datos
35
36    Principal
37       sleep                        ;El microcontrolador entrará en modo reposo
38        goto    Inicio              ;Comienza de nuevo el programa.
39
40    ;Mensajes
```

Fig. 7.64 Programa EEPROM, programa principal.

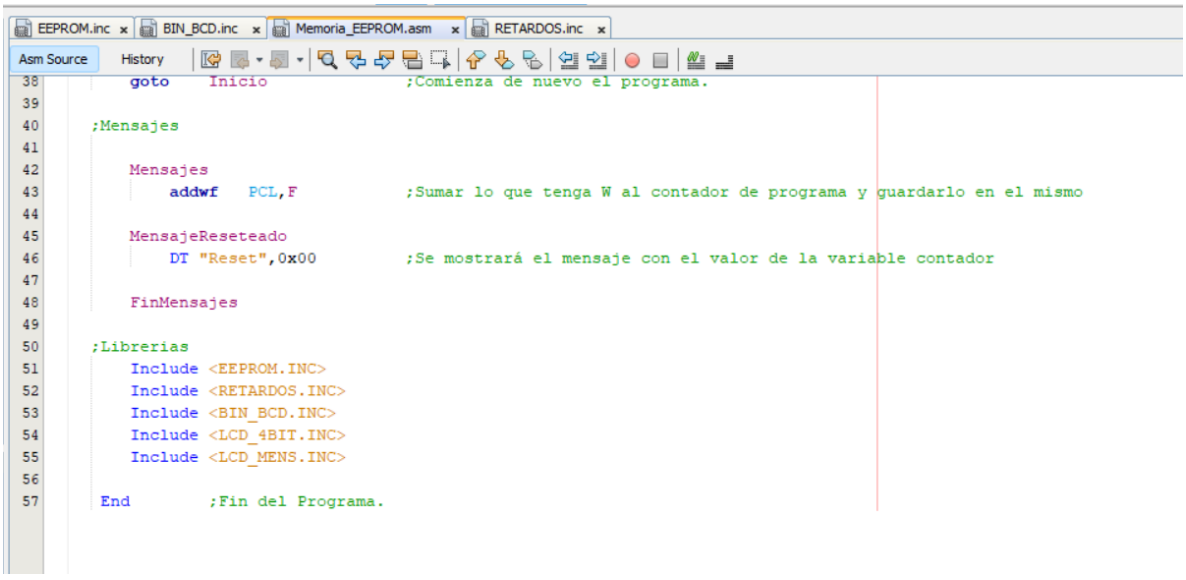
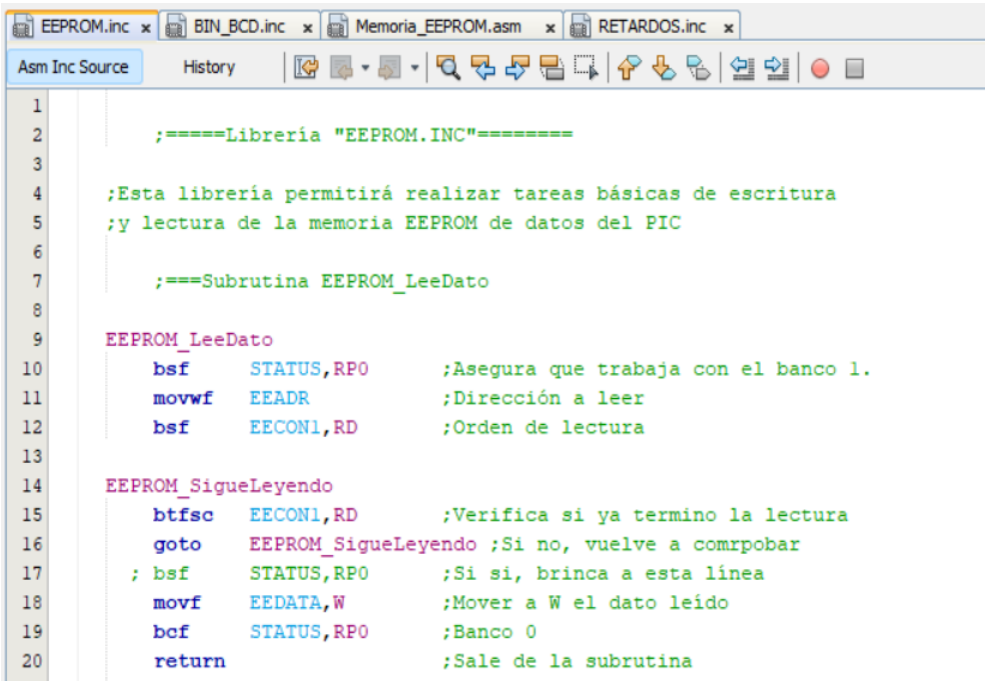


Fig. 7.65 Programa EEPROM, mensaje que se mostrará en la pantalla LCD.

"*call LCD_Inicializa* " empieza a preparar la pantalla para comenzar a visualizar los datos. Lo interesante viene cuando se llama a la subrutina "*EEPROM_LeeDato*"




```
1
2       ;=====Librería "EEPROM.INC"========
3
4   ;Esta librería permitirá realizar tareas básicas de escritura
5   ;y lectura de la memoria EEPROM de datos del PIC
6
7       ;===Subrutina EEPROM_LeeDato
8
9   EEPROM_LeeDato
10      bsf     STATUS,RP0      ;Asegura que trabaja con el banco 1.
11      movwf   EEADR           ;Dirección a leer
12      bsf     EECON1,RD       ;Orden de lectura
13
14  EEPROM_SigueLeyendo
15      btfsc   EECON1,RD       ;Verifica si ya termino la lectura
16      goto    EEPROM_SigueLeyendo ;Si no, vuelve a comrpobar
17    ; bsf     STATUS,RP0      ;Si si, brinca a esta línea
18      movf    EEDATA,W        ;Mover a W el dato leído
19      bcf     STATUS,RP0      ;Banco 0
20      return                  ;Sale de la subrutina
```

Fig. 7.66 Programa EEPROM, librería EEPROM.INC

Se llega a una librería encargada de hacer el proceso de lectura y escritura de datos en la EEPROM. En la línea 10 del código se asegura que se debe trabajar con el banco 1 ya que en ese banco se aloja el registro EEADR, en el capítulo 2 en organización de la memoria se puede apreciar en la figura 2.5 los registros que se ocuparan. Enseguida la instrucción "*movwf EEADR*" moverá lo que hay en W a EEADR, es decir a una posición de la EEPROM que será cero. Ahora con la instrucción "*bsf EECON1,RD*" damos la orden de que se leerá un dato. Después se verifica si ya se terminó de leer el dato, en caso de que no, el bucle se encontrará entre la línea 14 y 16. Si ya acabó, se regresa al banco cero y se continua en el programa principal pasando lo que hay en W a Contador "*movwf Contador*".

Lo que está en la línea 28 a 33 son instrucciones que se encargan de convertir el numero binario a decimal y a mostrar el mensaje "Reset" en la pantalla del LCD y pasará a W el nuevo valor de contador. La subrutina EEPROM_EscribeDato se encargará de guardar el valor de la variable contador en la EPROM.

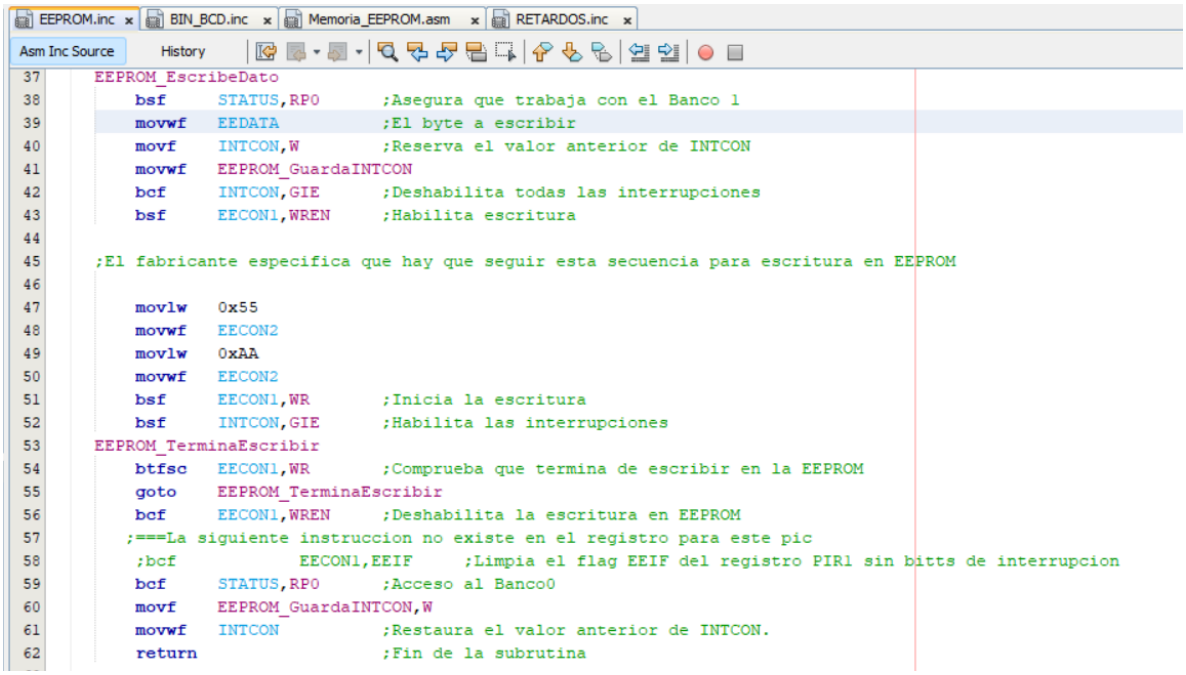


Fig. 7.67 Programa EEPROM, subrutinas de la librería EEPROM.INC.

Una vez que se haya movido el valor de contador a EEDATA hay que deshabilitar las interrupciones por seguridad y se habilita la escritura. Lo que sigue en la línea 47 a 52 son instrucciones que se deben poner para que se escriba un dato bien en la EPROM, así lo especifica el fabricante del MCU. Para terminar de escribir habrá que verificar si en realidad ya hizo el proceso de guardar el dato, automáticamente este bit o bandera pasara de estado cuando acabe de escribir el valor de contador a la EEPROM. Cuando acabe saldrá de la subrutina y llegará al programa principal para entrar en modo reposo (sleep). Hasta que se dé un reset se vuelve a ejecutar el programa de inicio y ahora contador valdrá su siguiente incremento.

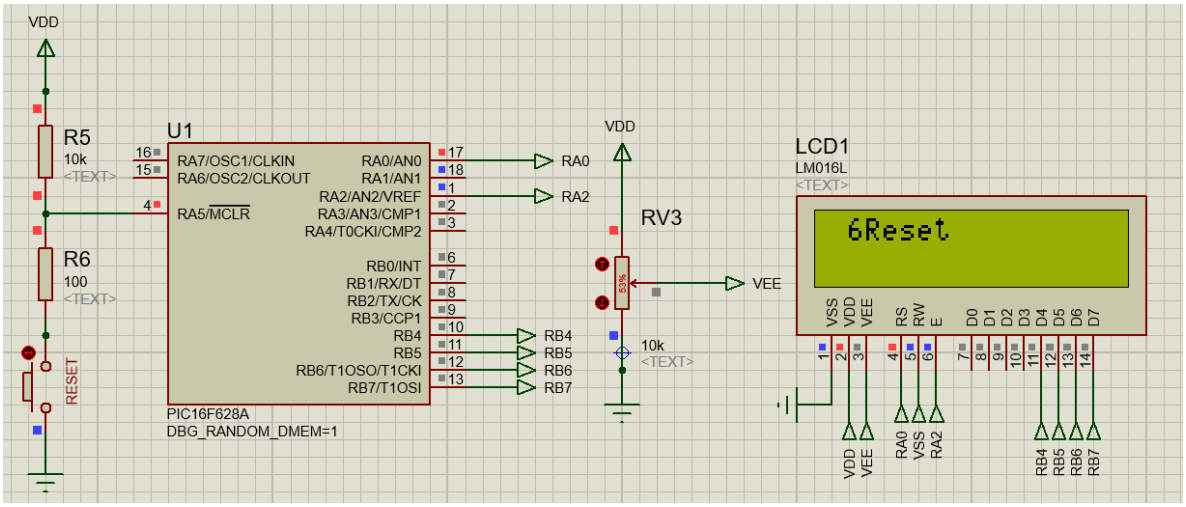


Fig. 7.68 Simulación en proteus, resultado al cargar el programa.

8 ¿Como Grabar el PIC?

Al grabar el PIC también se le puede llamar quemar ya que se carga el programa fuente al microcontrolador, este código estará disponible en la memoria flash durante el periodo de vida del MCU. El hardware PICKit3 de Microchip ayudará a esta tarea.

Para lograr grabar el código dentro del circuito integrado primero hay que descargar el software desde la siguiente página.

https://pickit.software.informer.com/Descargar-gratis/

Para instalarlo hay que crear una nueva carpeta en cualquier ruta del computador y dejar el archivo descargado dentro de esa carpeta.

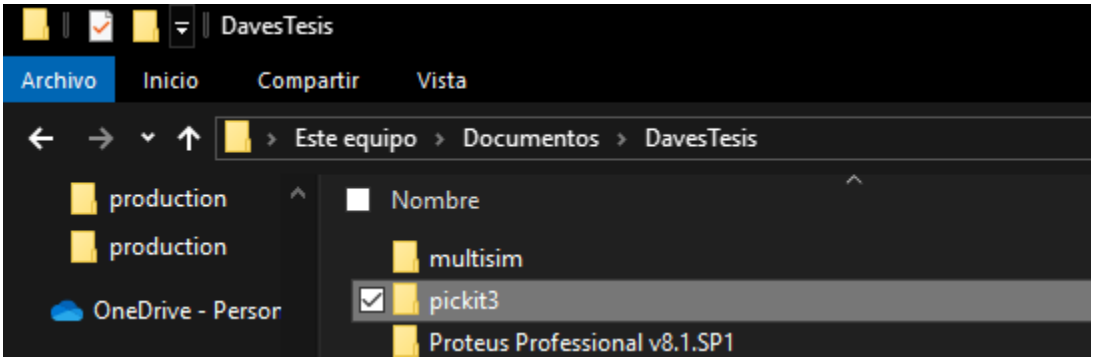


Fig. 8.1 Creación de carpeta pickit3.

Hay que extraer el archivo rar para descomprimir su contenido dentro de esa carpeta.

Fig. 8.2 Archivo comprimido que contiene al programa “pickit3”.

Se descomprimirán los siguientes archivos.

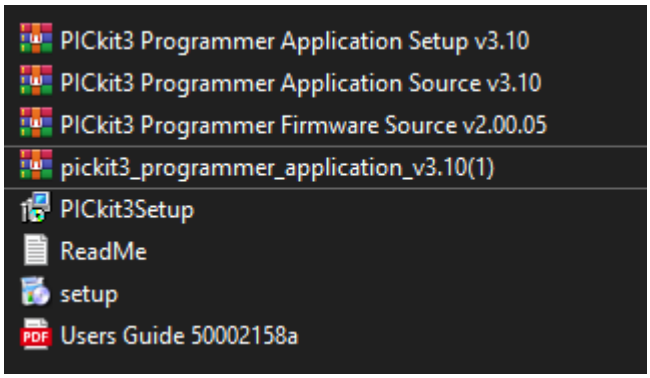


Fig. 8.3 Archivos descomprimidos.

Enseguida ejecutar PICkit3Setup como administrador e instalarlo como cualquier otro programa. Una vez que ya se encuentre instalado en el pc, el PICKit tendrá terminales o pines para alimentación y bus de datos, la configuración que debe llevar el microcontrolador con respecto al hardware es la siguiente.

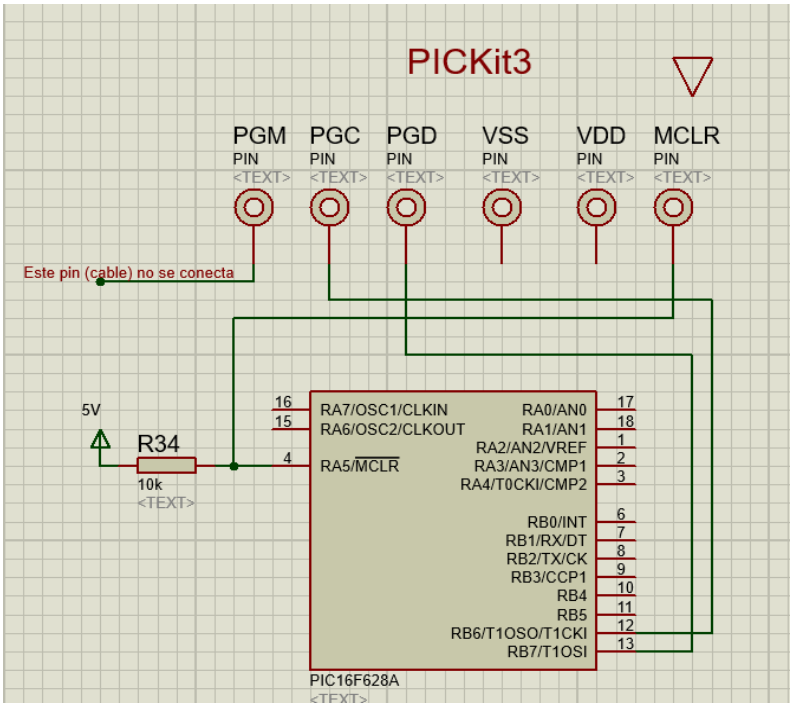


Fig. 8.4 Simulación en proteus, forma de conexión para grabación del PIC16F628.

El Pin VDD(5v) del PICKit se conecta al vdd del microcontrolador y el VSS(GND) se conecta al vss del microcontrolador, en la imagen no se ve porque en PROTEUS no tiene esos pines de alimentación. Cuando se tenga el circuito como en la figura anterior el hardware PICKit3 tiene su cable para la comunicación con la PC mediante puerto USB.

Antes de abrir el software PICKit3 hay que cerrar MPLAB porque MPLAB detecta que se ha instalado el hardware para grabar el PIC. No se grabará desde el entorno de desarrollo en esta ocasión, aunque si se pudiera.

Ahora si se conectará el quemador (hardware PICKit3) y se abrirá el programa de grabado. En primera instancia el programa mostrará un mensaje en color rojo o naranja, esto es porque hay que configurarlo antes.

Dirigirse a la pestaña “tols” enseguida “Download PICKit Operating System”. Se abrirá una ventana como la siguiente.

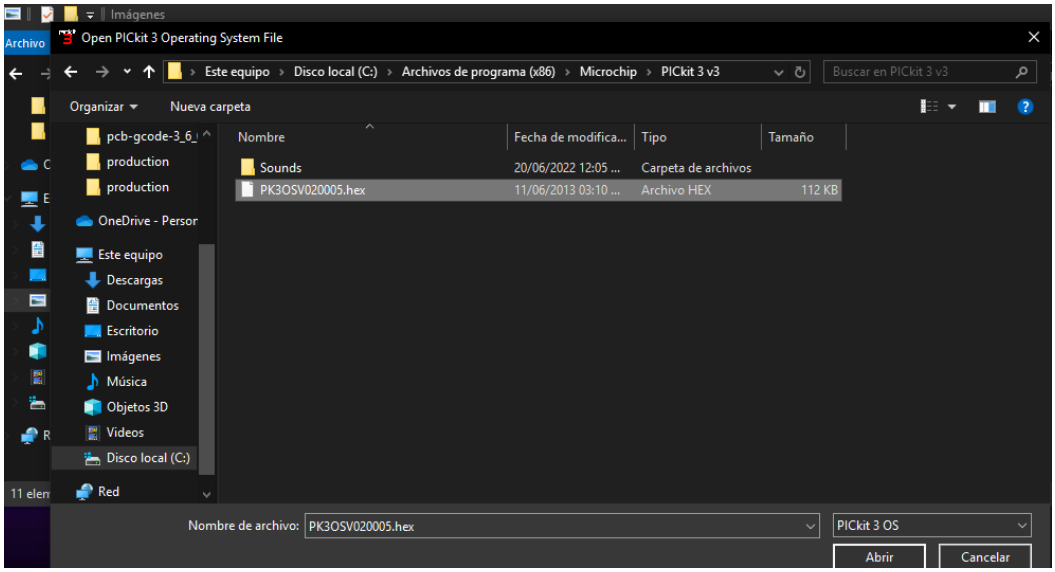


Fig. 8.5 Ventana con archivo de configuración para el hardware PICKit3.

Si no se abre esta ventana hay que ubicarse en la ruta donde se aloja un archivo .hex, como lo muestra la Fig. 8.5, se selecciona y se da clic en abrir. Esta ruta es: *Este equipo>DiscoLocal (C:)>Archivos de programa (x86)>Microchip>PICKit 3 v3*. Después empezará a cargar.

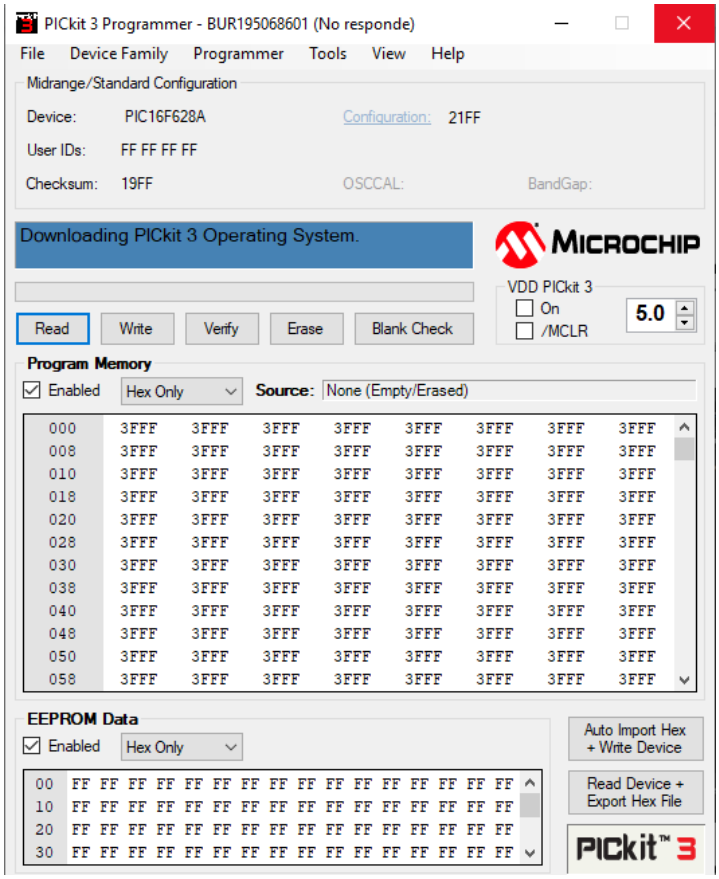


Fig. 8.6 Ventana principal del software PICkit 3.

Cuando termine se debe mostrar como en la Fig. 8.7. Aquí se puede ver como el hardware PICKit3 reconoce el PIC.

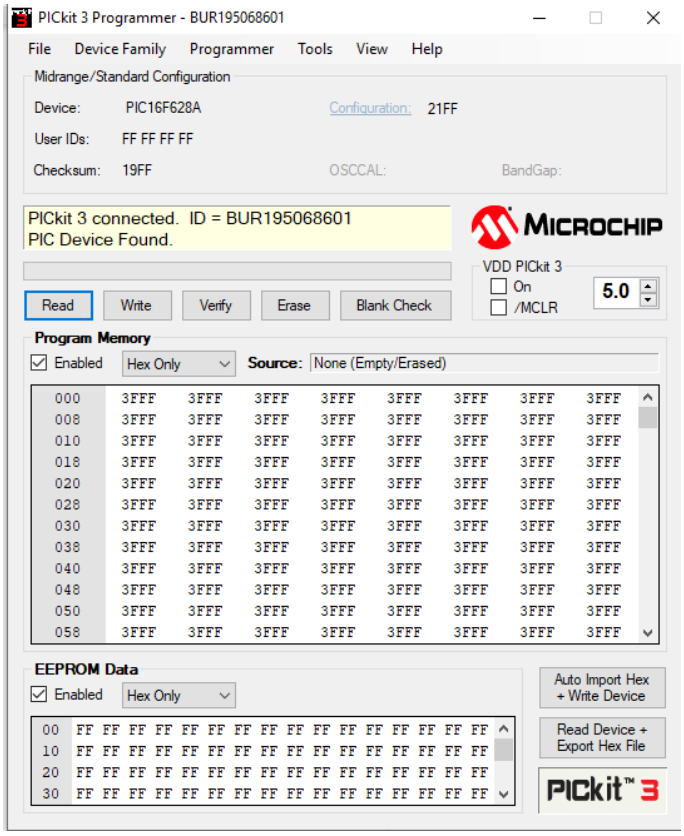


Fig. 8.7 Reconocimiento del PIC16F628.

La activación de la siguiente casilla funciona para que el microcontrolador reconozca la alimentación desde el PICKit3

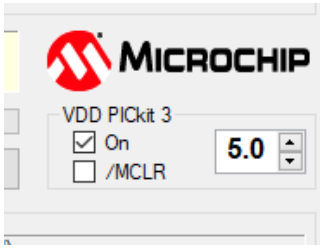


Fig. 8.8 Activación de casilla de alimentación.

Ahora se cargará el archivo prueba (programa de MPLAB), para esto hay que dirigirse a la pestaña “file” enseguida “import hex” del software PICKit3 y ubicar algún programa que se desarrolló en el capítulo 7, por ejemplo, el primero.

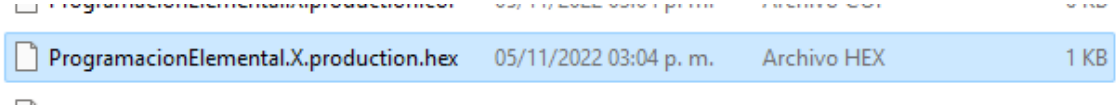


Fig. 8.9 Archivo del programa: Entrada y salida de datos.

Al abrir el archivo el programa mostrará un mensaje de que la importación ha sido satisfactoria.

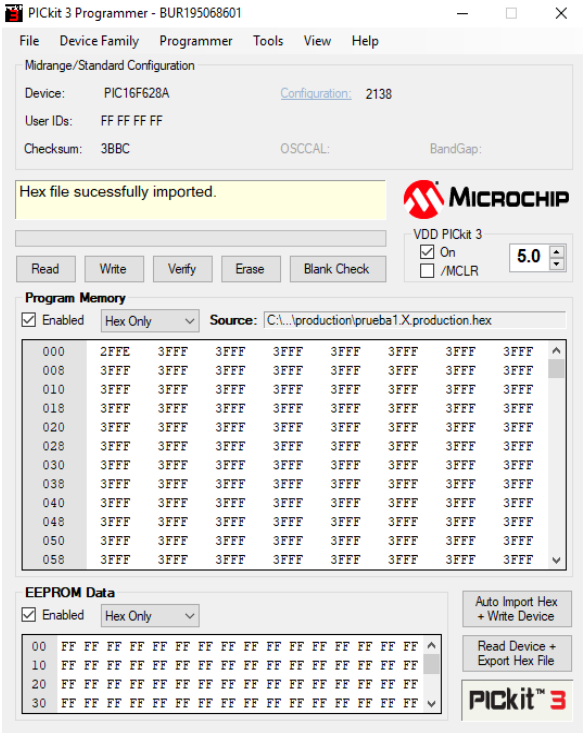


Fig. 8.10 Mensaje “Sucessfully” al importar el programa.

Ahora clic en el botón “Write” para que empiece el grabado del microcontrolador.

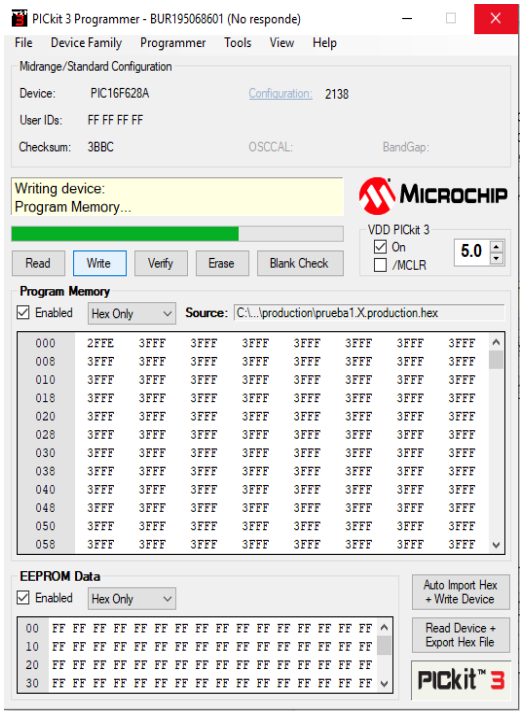


Fig. 8.11 Quemando el PIC16F628.

Aparecerá la ventana donde dice que ya se grabó el PIC.

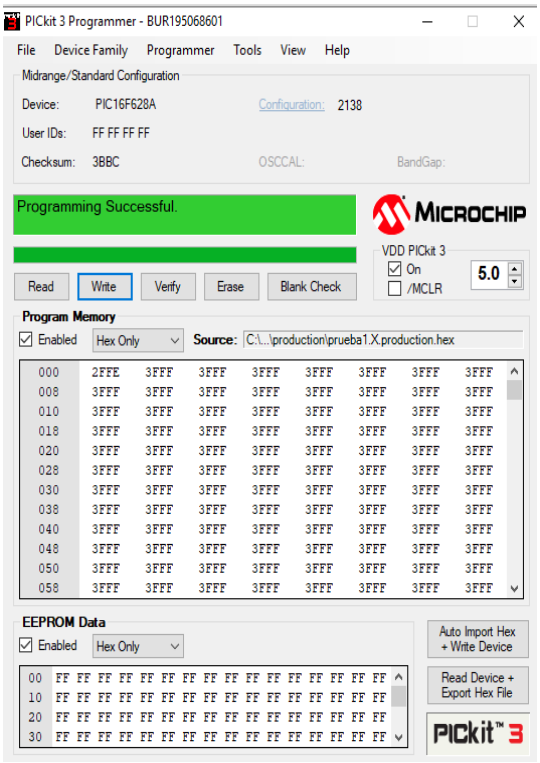


Fig. 8.12 Carga del programa al PIC16F628 completado.

9 Propuesta de tarjeta entrenadora para el PIC16F628

Al probar los programas realizados en este documento, surgen diferentes complicaciones con el armado del circuito físicamente. La protoboard es buena opción, aunque no asegura una conexión cien por ciento entre componentes y la organización no queda del todo, puesto que se necesitan varios componentes, hasta se necesita una segunda protoboard. Las baterías también representan un obstáculo, aunque si eran recargables no habría problema, pero para evitar esos detalles se pensó en una tarjeta PCB.

Se armo el circuito en el programa PROTEUS antes de diseñar la tarjeta en otro software para verificar que las conexiones y voltajes eran adecuados, así mismo se probaban los programas que se vieron en el transcurso del documento sin ningún problema. Se decidió utilizar EAGLE para el diseño PCB de esta tarjeta.

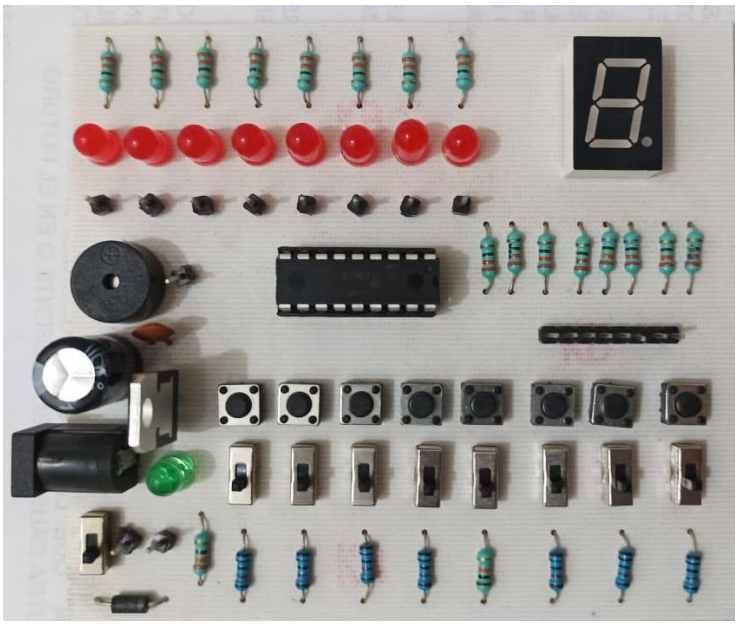


Fig. 9.1 Tarjeta entrenadora para el PIC16F628.

La tarjeta cuenta con 5 etapas.

1. Principal
2. Encendido
3. Control
4. Display y Buzzer
5. Resultado con LED’s

La etapa Principal es donde se encuentra el microcontrolador PIC16F628, puesto que es el cerebro de la tarjeta.

Fig. 9.2 Etapa principal, PIC16F628.

La etapa de encendido es donde se suministra la energía eléctrica a la PCB para que funcione en un rango de 9 a 12 voltios, esto se lo da un cargador o adaptador de corriente que va conectado directamente a la toma de corriente domestica (110 V) una vez conectado el interruptor o dip switch se encarga de energizar el circuito. Si no se cuenta con ese adaptador, la tarjeta cuenta con 2 pines que se encuentran a un costado del dip switch donde se pueden suministrar 5V.

Fig. 9.3 Etapa de encendido.

La etapa de control se compone de 8 push buton y 8 dip switch, por si el programa requiere entrada de datos constantes o solo pulsos, estos a su vez están conectados al puerto A del MCU desde el RA0 al RA7. Hay que recordar que este PIC acepta entradas de tipo pull down, en el capítulo 3 en circuitos periféricos de entrada de datos se explica este tipo de configuración. De derecha a izquierda se leen los bits menos significativos hasta el más significativo.

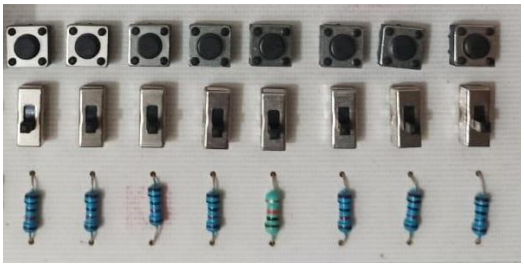


Fig. 9.4 Etapa de control.

El Display de 7 segmentos cuenta con sus pines de alimentación, si se requiere utilizar lo que se tiene que hacer es colocar unos puentes desde la etapa de resultado con LED’s. y lo mismo pasaría con el buzzer, este componente cuenta con su pin a un lado por si se requiere conectarlo de igual manera. De izquierda a derecha en los pines del display se empieza por el LED a, b, c… y así sucesivamente hasta el punto.

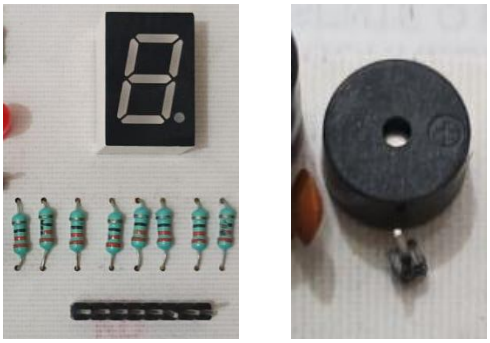


Fig. 9.5 Etapa de display y buzzer.

Los resultados con LED's están conectados directamente al puerto B con unos pines de salida por si se requieren utilizar con otros componentes, como la pantalla LCD. De la misma forma que las entradas, los LED's están organizados de derecha a izquierda como el bit menos significativo al bit más significativo

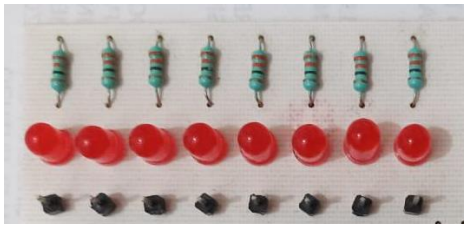


Fig. 9.6 Etapa de resultados con LED's.

A un futuro se podría implementar pines hembra para que la pantalla LCD pueda insertarse y removerse sin ningún problema. De la misma forma si se requiere dar potencia a motores DC o motores a paso se podrían incluir drives de control para estos motores. También que el usuario decida que puerto puede ser de entrada y cuál de salida. Al momento el puerto A solo es entrada y B como salida. Un zócalo ZIF para poder insertar y remover el PIC para cargar el programa fuente quedaría de maravilla. Sin duda una mejora sería poner los pines para que el programador PICkit se coloque sobre la tarjeta sin necesidad de quitar el PIC.

10 Conclusiones

Al desarrollar tecnología con estos microcontroladores, el hombre ha podido llegar a la luna y actualmente sus beneficios para diferentes dispositivos en la vida del ser humano, es de suma importancia pues gracias a los PIC en conjunto con otros componentes electrónicos es más fácil hacer tareas complejas y automatizar algoritmos que permitan ejecutar a la perfección una orden programada.

Es por ello por lo que aprender a desarrollar programas en un lenguaje de bajo nivel ayuda al entendimiento y funcionamiento de los microcontroladores y no solo estos dispositivos si no como también los microprocesadores y demás dispositivos que solo necesiten ceros y unos para funcionar.

Al programar rutinas y subrutinas se conoció las diferencias que hay entre microcontroladores en cuestión de acceso a las partes de memoria ya cada uno tiene diferente capacidad en sus variadas memorias. Es vital recordar la forma en que se debe alojar una variable y el tamaño que debe tener para que no alcance el desbordamiento, así como el tipo de variable a ocupar dentro de los programas. De la misma forma pasaba en los registros, hay banderas o bits en esos bytes que cambian del PIC16f84 al PIC16f628 y la forma de habilitación y des habilitación es donde se debía que prestar atención puesto que un nivel lógico alto podía no hacer funcionar adecuadamente el programa y el circuito.

En las simulaciones se conoce a fondo el movimiento de información entre registros y el valor que tomará en el transcurso de la ejecución del programa gracias a la ejecución en tiempo real y línea por la línea de código en el entorno de desarrollo MPLAB. PROTEUS ayudo mucho a la simulación del circuito puesto que se podía ver si en realidad podría funcionar o no de forma física, también se ajustaban valores de frecuencia en la pantalla LCD para que pudiera estar en sintonía con el PIC.

Se descubrieron archivos dentro de las carpetas de MPLAB, que contienen información de configuración para cada microcontrolador, así como datos precisos de las distribuciones de memoria, esto fue de bastante ayuda puesto que para el desarrollo del programa en la EEPROM la dirección en la que empieza el almacenaje de información era necesario conocerla.

Cuando se diseñó y elaboró la tarjeta entrenadora para el PIC16F68 de forma física se encontró que es de bastante utilidad tener componentes de manera ordenada y con buena conexión entre ellos ya que se visualiza y se interactúa de una manera más confiable con la PCB. Entre otras ventajas, es de gran ayuda tener un circuito como este para probar diferentes ideas de desarrollo.

Sin más preámbulo espero que este documento ayude a los fanáticos de la electrónica o jóvenes estudiantes que se dediquen a esta rama tan fundamental que es la programación de microcontroladores.

11 Bibliografía

[1] Curso práctico sobre microcontroladores CEKIT Tomo1. JUAN ANDRÉS CASTAÑO W., MANUEL FELIPE GONZÁLEZ G., GILBERTO VARGAS C., JUAN DIEGO HERNANDEZ F. Cono Sur. Primera edición. Junio 2002.

[2] Microchip. (enero 2024). MICROCONTROLLERS (MCU). https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors.

[3] Vladimir Alexis. (diciembre 2021). Gama Microcontroladores. https://idoc.pub/documents/gama-microcontroladores-en5k7p5kr5no

[4] M.C. Marco A. Ramírez Barrientos. (20 de abril de 2012). Introducción a la programación de microcontroladores. https://es.slideshare.net/slideshow/sesin-3-introduccion-a-microcontroladores/12616568.

[5] SIISA GLOBAL. (21 de julio de 2021). Microcontroladores. ¿Qué son? y su importancia en la industria. https://www.linkedin.com/pulse/microcontroladores-qu%C3%A9-son-y-su-importancia-en-la-industria-/

[6] Microchip. (2003). PIC16F62X Data sheet.

[7] Fundamentos de sistemas digitales. Thomas L. Floyd. Editorial Pearson. Novena edición. 2006.

[8] Bill Giovino. (24 de julio de 2019). Aproveche el rendimiento y la potencia reducida que un microcontrolador de 16 bits puede ofrecer. https://www.digikey.com.mx/es/articles/take-advantage-of-16-bit-mcu-performance-and-low-power

[9] Ing. Gerardo Macin. (febrero 2014). Características sobre los Microcontroladores de 8 Bits y sus fabricantes. https://eagleheartmc.blogspot.com/p/introduccion-un-microcontrolador.html

[10] Facultad de Ingeniería, UNAM, Departamento de Electrónica. Introducción al microcontrolador MSP430. http://kali.azc.uam.mx/erm/Media/1123021/introduccion_al_msp430.pdf

[11] Urquidez Arcadia Humberto. (2013). Arquitectura de un microcontrolador. https://mio-unidad3.blogspot.com/2015/12/arquitectura-de-un-microcontrolador.html

[12] Electrónica Unicrom. (2024). Niveles Lógicos. https://unicrom.com/niveles-logicos-alto-bajo-0-1-low-high/

[13] Unigal. (2024). Diferencia entre los modos de direccionamiento directo e indirecto. https://unigal.mx/diferencia-entre-los-modos-de-direccionamiento-directo-e-indirecto/

[14] Microcontrolador – PIC16F84 Desarrollo de proyectos. Enrique Palacios, Fernando Remiro, Lucas J. López. Alfaomega. Primera Edición. Agosto 2004.

[15] STMicroelectronics. (2003) datasheet L293D.

[16] Joel Buenrostro. (24 de abril de 2018). Temporizador de encendido (PWRT). https://joelbuenrostroblog.blogspot.com/2018/04/temporizador-de-encendido-pwrt.html

[17] Punto Flotante S.A. Manejo de tablas de datos en microcontroladores con arquitectura Harvard, como el 16F84/F628/F88. https://www.puntoflotante.net/tablas.htm