Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO E IMPLEMENTACIÓN DE PRÁCTICAS DEL LABORATORIO DE MICROCONTROLADORES

Francisco Yovani de León Zamora
Asesorado por el Ing. Enrique Edmundo Ruiz Carballo

Guatemala, febrero de 2015

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE PRÁCTICAS DEL LABORATORIO DE MICROCONTROLADORES

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

FRANCISCO YOVANI DE LEÓN ZAMORA
ASESORADO POR EL ING. ENRIQUE EDMUNDO RUIZ CARBALLO

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, FEBRERO DE 2015

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Narda Lucía Pacay Barrientos
VOCAL V	Br. Walter Rafael Véliz Muñoz
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Julio César Solares Peñate
EXAMINADORA	Inga. Ingrid Salomé Rodríguez de Loukota
EXAMINADOR	Ing. Otto Fernando Andrino González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO E IMPLEMENTACIÓN DE PRÁCTICAS DEL LABORATORIO DE MICROCONTROLADORES

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 3 de mayo de 2013.

Francisco Yovani de León Zamora

Guatemala, 13 de agosto del 2014

Ingeniero
Carlos Eduardo Guzmán Salazar
Coordinador del Área Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala
Presente.

Ingeniero Guzmán:

Por medio de la presente le comunico que he revisado completamente el trabajo de graduación titulado "DISEÑO E IMPLEMENTACIÓN DE PRÁCTICAS DEL LABORATORIO DE MICROCONTROLADORES", desarrollado por el estudiante Francisco Yovani de León Zamora y el mismo cumple con los objetivos propuestos.

Por lo tanto, el autor del mismo y yo como su asesor, nos hacemos responsables del contenido y conclusiones del mismo.

Atentamente.

Enrique Edmundo Ruiz Carballo
Asesor Nombrado
Colegiado 2225

Enrique E Ruiz C
INGENIERO ELECTRICISTA
COL No 2225

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Ref. EIME 41. 2014
Guatemala, 18 de AGOSTO 2014.

Señor Director
Ing. Guillermo Antonio Puente Romero
Escuela de Ingenierìa Mecànica Elèctrica
Facultad de Ingenierìa, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: DISEÑO E IMPLEMENTACIÓN DE PRÁCTICAS DEL LABORATORIO DE MICROCONTROLADORES, del estudiante Francisco Yovani de León Zamora, que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

Ing. Carlos Eduardo Guzmán Salazar
Coordinador Área Electrónica

sro

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 41. 2014.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; FRANCISCO YOVANI DE LEÓN ZAMORA titulado: DISEÑO E IMPLEMENTACIÓN DE PRÁCTICAS DEL LABORATORIO DE MICROCONTROLADORES, procede a la autorización del mismo.

Ing. Guillermo Antonio Puente Romero

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA
DE INGENIERIA
MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 16 DE SEPTIEMBRE 2,014.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 065 .2015

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **DISEÑO E IMPLEMENTACIÓN DE PRÁCTICAS DEL LABORATORIO DE MICROCONTROLADORES,** presentado por el estudiante universitario: **Francisco Yovani de León Zamora,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Murphy Olympo Paiz Recinos
Decano

Guatemala, 17 de febrero de 2015

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

/gdech

ACTO QUE DEDICO A:

Dios	Por ser mi creador y sustentador, quien siempre me ha acompañado en las diferentes etapas de mi vida y me guía cada día hacia la eternidad.
Mis padres	Por que a pesar de las dificultades siempre me dieron el beneficio de la duda.
Mi hermana	De quien siempre he admirado su espíritu de lucha inquebrantable.
Mi hermano	Porque siempre ha estado ahí para ayudarme.

AGRADECIMIENTOS A:

La Universidad de San Carlos de Guatemala	Por ser una institución que me ha dado la oportunidad de continuar con mis estudios superiores, mi alma mater.
Facultad de Ingeniería	Por haber sido mi casa durante estos años de estudio, lugar en el cual he crecido intelectualmente, como también espiritual y socialmente, porque en sus aulas he aprendido no solo sobre una carrera técnica, sino que he conocido personas y situaciones que me servirán para el resto de mi vida.
A mi asesor	El doctor Enrique Ruiz que ha sido una persona que me ha brindado su amistad y consejos muy valiosos durante mi carrera, como para mi vida.
Mis amigos de la Facultad	Aquellas personas que a través de los años en la Facultad he conocido, me han brindado su amistad, motivado en los momentos difíciles, ayudado a superar las diferentes situaciones y sobre todo, porque siempre que los necesité estuvieron ahí, gracias a todos.

Al claustro de catedráticos de la Escuela de Mecánica Eléctrica	Porque han compartido conmigo sus conocimientos sin ningún tipo de reserva. Con quienes siempre mantuve una relación de amistad y cordialidad, gracias a todos.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
TTL	Lógica transistor a transistor
RAM	Memoria de acceso aleatorio
ROM	Memoria de solo lectura
CMOS	Semiconductores complementarios de óxido metálico
ALU	Unidad lógica aritmética
CPU	Unidad central de procesamiento

GLOSARIO

Algoritmo Conjunto ordenado y finito de operaciones que al ejecutarse adecuadamente, permite hallar la solución de un problema.

Binario Elemento capaz de tomar únicamente dos valores.

Bit Acrónimo *Binary digit* (‘dígito binario’), 0 o 1 en lógica digital.

Byte 8 bits.

Circuito Un circuito es una red eléctrica (interconexión de dos o más componentes, tales como resistencias, inductores, condensadores, fuentes, interruptores y semiconductores) que contiene al menos una trayectoria cerrada.

Datasheet Es un documento que resume el funcionamiento y otras características de un componente, con el suficiente detalle para ser utilizado por un ingeniero de diseño y diseñar el componente en un sistema.

Electromagnetismo Es una rama de la física que estudia y unifica los fenómenos eléctricos y magnéticos en una sola teoría, cuyos fundamentos fueron sentados por

	Michael Faraday y formulados por primera vez de modo completo por James Clerk Maxwell.
Lcd	Una pantalla de cristal líquido o LCD (sigla del inglés liquid cristal display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora.
Lenguaje de Programación	Es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.
Microprocesador	Sistema digital, con capacidad para interpretar instrucciones y procesar datos, especificados por un programa.
Software	Conjunto de programas o componentes lógicos que realizan tareas específicas en una computadora.

RESUMEN

El presente trabajo de graduación está constituido en cuatro capítulos, en los cuales se busca trasladar al lector los conceptos básicos de la electrónica y la programación de microcontroladores a través de ejemplos simples y prácticos.

El primer capítulo busca hacer un breve repaso histórico de cómo fue evolucionando el pensamiento del ser humano hasta el presente, en lo concerniente a la electricidad, se hace un enfoque en los teoremas principales de este campo, se resaltan los momentos claves de los microprocesadores, sus características según su modelo, lo básico en electrónica digital y los componentes básicos para poder realizar las prácticas que se plantean en el capítulo IV.

En el segundo capítulo se hace referencia específicamente en los microcontroladores PIC, su funcionamiento interno, como buses, arquitectura, memorias entre otros que son necesarios para su funcionamiento.

En el tercer capítulo se enfoca solamente en la programación, desde como plantear un problema, crear un algoritmo para resolverlo y los pasos utilizados para que el programa sea estructurado y funcional.

En el cuarto y último capítulo se plantean una serie de prácticas sencillas para que el lector tenga una idea de cómo pasar un problema real, plantear el algoritmo, diseñar el diagrama de flujo que lo resuelve y por último plasmar ese

algoritmo en un programa de alto nivel para finalmente poder transferir dicho programa al microcontrolador y ponerle a funcionar en la realidad.

OBJETIVOS

General

Diseñar e implementar prácticas para laboratorio de microcontroladores.

Específicos

1. Introducir al lector en el tema de electricidad, electrónica y procesadores.

2. Orientar sobre los fundamentos de microcontroladores.

3. Orientar sobre fundamentos de programación.

4. Dar ejemplos prácticos sobre programación de microcontroladores.

INTRODUCCIÓN

Desde mediados del siglo pasado se desarrolló en los laboratorios Bell el transistor, esto revolucionó la electrónica y permitió la fabricación de procesadores con lo cual se elaboraron las primeras computadoras, posteriormente se aprovechó este recurso para la automatización de procesos industriales, después con la aparición de la microelectrónica y la creación de procesadores más pequeños, denominados microprocesadores, con mayor capacidad que los procesadores ordinarios, aparecieron los microcontroladores, para trabajos en el área de la automatización teniendo estos la ventaja de incorporar periféricos de salida a los microprocesadores, con lo cual tuvieron gran auge en el campo de la automatización industrial.

Para lograr esto se necesita de herramientas que permitan la comunicación hombre-máquina apareciendo con esto los lenguajes de programación, además es necesario trasladar un problema real a un lenguaje entendible por el microcontrolador, para esto es necesario realizar el paso intermedio de crear el algoritmo que resuelve dicho problema y posteriormente llevar este algoritmo al lenguaje de programación a utilizar.

1. GENERALIDADES SOBRE ELECTRICIDAD, ELECTRÓNICA Y PROCESADORES

Sin duda alguna el avance tecnológico de los últimos cien años contrasta demasiado con el avance de los siglos que le precedieron, para esto basta con observar cómo fueron desarrollándose los pensamientos e ideas que desembocaron con los inventos y descubrimientos de los últimos años, como ejemplo se puede mencionar que para llegar al uso de los números arábigos pasaron varios siglos, entre luchas de territorios, rutas comerciales, ideales políticos y religiosos, que con base en fuerza fueron implementados por los vencedores sobre los vencidos en las distintas guerras, así fue como el sistema de numeración arábigo se hizo universal y no fue solo de las regiones árabes.

Fueron muchos hechos los que llevaron a la humanidad al descubrimiento del fenómeno eléctrico y magnético, aprender a interactuar con ellos y finalmente manipularlos para su beneficio, los grandes pensadores griegos fueron los primeros en notar este fenómeno pero no lograron comprenderlo para poder explotar los beneficios del mismo, ahí es donde más adelante en la línea histórica de la humanidad aparecieron los nombres de Oersted, Coulomb, Maxwell, Faraday, Ampere, Volt, Tesla, Edison, Marconi, quienes haciendo uso de métodos científicos de investigación, ideas y ecuaciones para modelar los campos eléctricos y magnéticos para finalmente desarrollar los campos de la electricidad, electrónica, potencia y diversas ramas que se han desprendido del estudio de los mismos.

De los estudios eléctricos que van desde un circuito serie, circuito paralelo y circuito mixto, además de la generación de la energía eléctrica, el transporte y

distribución de la mismas, así como el debate entre la corriente directa DC y corriente alterna AC que originó la rivalidad entre Edison y Tesla, más adelante en los laboratorios Bell se dio el paso al mundo de la electrónica con la creación del transistor, todo esto desembocó en la automatización de procesos de fabricación, en el cual había necesidad de tomar decisiones con base en situaciones específicas con lo cual se necesitó que alguien o algo tomara esas decisiones.

Ahí es donde el procesador electrónico apareció, para luego ir evolucionando de acuerdo a las necesidades y de acuerdo al avance tecnológico pasó de ser circuitos complicados y que ocupaban un gran espacio, a pequeños encapsulados que pueden ser transportados fácilmente, y de apenas algunos centímetros cuadrados, y al alcance de cualquier persona dado también su costo, siendo este accesible.

1.1. Evolución histórica de la electricidad

El fenómeno eléctrico era conocido ya desde buen tiempo atrás, la carga del ámbar por fricción, el uso del imán en la navegación, son algunos ejemplos de esto, no fue sino hasta el siglo XIX cuando se dieron los primeros pasos en su estudio, Galvani y Volta descubrieron que la electricidad se podía producir por medios químicos, posteriormente Oersted demostró empíricamente que una corriente eléctrica podía desviar la dirección de una brújula, quedando así demostrada que existía una relación entre el campo magnético y el campo eléctrico, lo cual llevó a Ampere a realizar un estudio sobre el tema, para finalmente descubrir que una corriente eléctrica puede generar un campo magnético, además lo relacionó con la regla de Ampere conocida también como regla de la mano derecha.

Faraday descubrió el fenómeno de la inducción electromagnética quien lo indicó en lo que hoy se conoce como la ley de Faraday, que estipula que la magnitud de la tensión inducida es proporcional a la variación del flujo magnético.

Maxwell estudió el fenómeno electromagnético desarrollando la teoría electromagnética en donde sintetiza las anteriores observaciones, leyes y experimentos de electricidad, originalmente eran veinte ecuaciones que después Maxwell redujo a trece. Finalmente en trabajo Gibbs, Hertz y Heaviside redujeron a las cuatro ecuaciones que en la actualidad se conocen como las ecuaciones de Maxwell.

Figura 1. **Ecuaciones de Maxwell**

LEY	FORMA DIFERENCIAL	FORMA INTEGRAL
Gauss	$\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon_0}$	$\oint_S \vec{E} \cdot d\vec{s} = \frac{q}{\epsilon_0}$
Gauss para B	$\vec{\nabla} \cdot \vec{B} = 0$	$\oint_S \vec{B} \cdot d\vec{s} = 0$
Faraday-Henry	$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}$	$\oint_C \vec{E} \cdot d\vec{l} = -\frac{d}{dt}\int_S \vec{B} \cdot d\vec{s}$
Ampere-Maxwell	$\vec{\nabla} \times \vec{B} = \mu_0 \vec{j} + \mu_0 \epsilon_0 \frac{\partial \vec{E}}{\partial t}$	$\oint_C \vec{B} \cdot d\vec{l} = \mu_0 \int_S \vec{j} \cdot \vec{n} \cdot ds + \mu_0 \epsilon_0 \frac{d}{dt}\int_S \vec{E} \cdot \vec{n} \cdot ds$
Fuerza de Lorentz	$\mathbf{F} = q\mathbf{E} + q\mathbf{v} \times \mathbf{B}$	

Fuente: elaboración propia.

Luego Tomás Alva Edison en su fructífera carrera como inventor aprovechó estos conceptos y a través de la experimentación desarrolló entre otros, el generador de corriente continua, se instalaron las primeras hidroeléctricas en los Estados Unidos, también un antiguo ayudante suyo, Nicola Tesla, desarrolló por aparte el generador de corriente alterna, el cual según el era más efectivo que el de corriente continua, generando polémicas alrededor de las dos corrientes eléctricas, que con el paso del tiempo llevó a la consolidación de la corriente alterna sobre la corriente continua en el ámbito de generación, transmisión y distribución de la energía eléctrica, por su contra la corriente continua es la base en la cual se sustenta la electrónica moderna, siendo esta la fuente de potencia con la cual funcionan los dispositivos electrónicos.

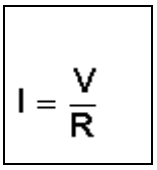
1.2. Teoremas y fundamentos de la electricidad

Después de las aportaciones de grandes pensadores e inventores la electricidad se desarrolló como una ciencia, con teoremas elementales, principios básicos que cualquier persona que esté interesada en su estudio debe conocer, a continuación se enumeran los principales teoremas de la electricidad.

1.2.1. Ley de Ohm

Es la ley básica de la corriente de flujo. Con la Ley de Ohm se calculan corrientes a partir de voltajes y resistencias o viceversa.

Figura 2. **Ley de Ohm**



Fuente: elaboración propia.

La cantidad de corriente que fluye por un circuito formado por resistencias puras es directamente proporcional a la fuerza electromotriz aplicada al circuito, e inversamente proporcional a la resistencia total del circuito.

Esta ley se expresar por la fórmula I = V/R, donde I = intensidad de corriente en amperios; V la fuerza electromotriz en voltios y R la resistencia en ohmios.

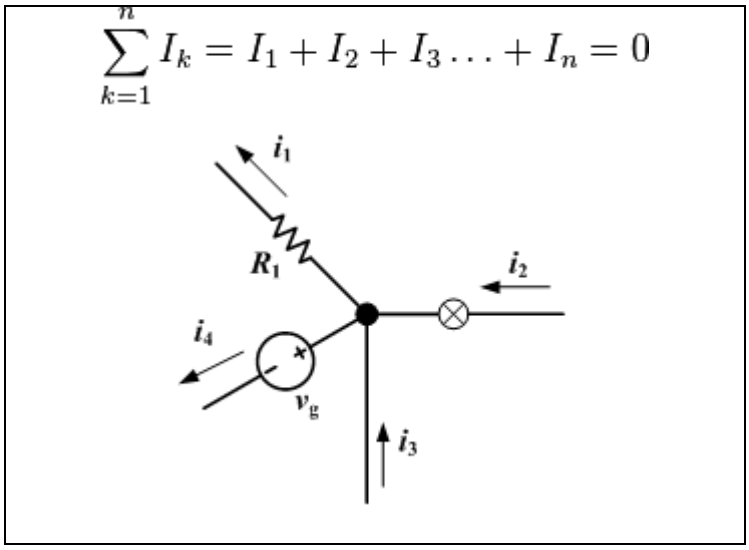
1.2.2. Ley de Kirchhoff

En realidad son dos las leyes de Kirchhoff, una aplica para las corrientes, ley de Kirchhoff de corrientes (LCK), y otra para voltajes, ley de Kirchhoff de voltajes (LVK).

1.2.2.1. Ley de Corrientes de Kirchhoff

Esta ley también es llamada ley de nodos o primera ley de Kirchhoff y es común que se use la sigla LCK para referirse a esta ley. La ley de corrientes de Kirchhoff dice que: en cualquier nodo, la suma de las corrientes que entran en ese nodo es igual a la suma de las corrientes que salen. De forma equivalente, la suma de todas las corrientes que pasan por el nodo es igual a cero.

Figura 3. **Ley de Corriente de Kirchhoff LCK**



Fuente: Wikipedia. *Leyes de Kirchhoff*. http://es.wikipedia.org/wiki/Leyes_de_Kirchhoff. Consulta: 15 de junio de 2014.

1.2.2.2. Ley de Voltajes de Kirchhoff

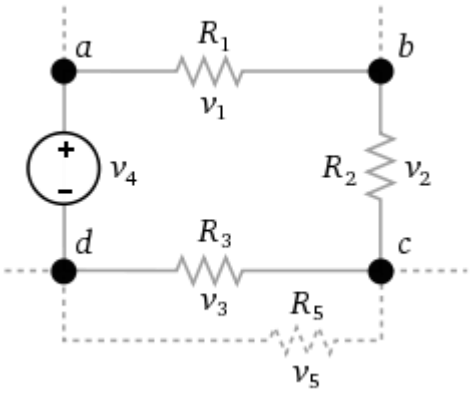
Esta ley es llamada también Segunda ley de Kirchhoff, ley de lazos de Kirchhoff o ley de mallas de Kirchhoff y es común que se use la sigla LVK para referirse a esta ley.

Esta ley dice que: en un lazo cerrado, la suma de todas las caídas de tensión es igual a la tensión total suministrada. De forma equivalente, la suma algebraica de las diferencias de potencial eléctrico en un lazo es igual a cero.

Figura 4. **Ley de Voltajes de Kirchhoff LVK**

$$\sum_{k=1}^{n} V_k = V_1 + V_2 + V_3 \ldots + V_n = 0$$

Continuación de la figura 4.



Fuente: Wikipedia. *Leyes de Kirchhoff*. http://es.wikipedia.org/wiki/Leyes_de_Kirchhoff. Consulta: 15 de junio de 2014.

1.3. Desarrollo de la electrónica

La electrónica es el campo de la física, que se refiere al diseño y aplicación de dispositivos de circuitos electrónicos, cuyo funcionamiento se basa en la conducción y el control del flujo de los electrones u otras partículas cargadas eléctricamente.

La electrónica se puede clasificar en electrónica analógica y electrónica digital. La electrónica analógica es aquella en que se basa en la variación de la magnitud de voltaje o corriente, como ejemplo se puede mencionar la carga y descarga de capacitores e inductores, la electrónica digital es aquella que la base de su funcionamiento es la interacción entre estados de voltajes de manera concreta como por ejemplo tener 5 voltios o 0 voltios que se puede asociar a tener un estado en 1 y otro estado en 0.

La introducción de los tubos de vacío a comienzos del siglo XX propició el rápido crecimiento de la electrónica moderna. Con estos dispositivos se hizo posible la manipulación de señales, algo que no podía realizarse en los antiguos circuitos telegráficos y telefónicos, ni con los primeros transmisores que utilizaban chispas de alta tensión para generar ondas de radio. Por ejemplo, con los tubos de vacío pudieron amplificarse las señales débiles de sonido y radiofrecuencia, y además se pudo lograr superponerse señales de sonido a las ondas de radiofrecuencia.

El desarrollo de una amplia variedad de tubos, diseñados para funciones especializadas, posibilitó el rápido avance de la tecnología de comunicación radial antes de la Segunda Guerra Mundial, y el desarrollo de las primeras computadoras, durante la guerra y poco después de ella.

Hoy día, el transistor, inventado en 1948, ha reemplazado casi completamente al tubo de vacío en la mayoría de sus aplicaciones. Al incorporar un conjunto de materiales semiconductores y contactos eléctricos, el transistor permite las mismas funciones que el tubo de vacío, pero con un costo, peso y potencia más bajos, y una mayor fiabilidad. Los progresos subsiguientes en la tecnología de semiconductores, atribuible en parte a la intensidad de las investigaciones asociadas con la iniciativa de exploración del espacio, llevó al desarrollo, en la década de 1970, del circuito integrado. Estos dispositivos pueden contener centenares de miles de transistores en un pequeño trozo de material, permitiendo la construcción de circuitos electrónicos complejos, como los de los microordenadores o microcomputadoras, equipos de sonido y vídeo, y satélites de comunicaciones.

La electrónica digital ha sido una de las revoluciones tecnológicas más importantes y decisivas de las últimas décadas. Su evolución vertiginosa ha

cambiado el ritmo del tiempo y representa el liderazgo tecnológico de la vida moderna.

Los avances alcanzados en el campo de la electrónica digital han permitido el desarrollo y la fabricación masiva, a bajo costo, de calculadoras, relojes digitales, computadoras personales entre otros.

Esto gracias a la microelectrónica, tecnología que ha permitido fabricar diminutas pastillas de silicio llamadas chips o circuitos integrados, sistemas completos que contienen miles de componente electrónicos.

Conceptualmente, la electrónica digital es, en la teoría y en la práctica, más sencilla que la electrónica análoga. Esto se debe a que los dispositivos digitales trabajan solamente en dos condiciones o estados.

La electrónica digital puede definirse como la parte de la electrónica que estudia circuitos y sistemas digitales, binario y lógicos, a diferencia de la electrónica lineal o análoga que trabajó con señales que pueden adoptar una amplia gama de valores de voltajes, en la electrónica digital solo se tienen dos valores alto o 1 y bajo o 0, que generalmente corresponden a presencia de tensión y ausencia de la misma.

1.3.1. El circuito integrado

Un circuito integrado (CI), también conocido como chip o microchip, es una pastilla pequeña de material semiconductor, de algunos milímetros cuadrados de área, sobre la que se fabrican circuitos electrónicos generalmente mediante fotolitografía y que está protegida dentro de un encapsulado de

plástico o cerámica. El encapsulado posee conductores metálicos apropiados para hacer conexión entre la pastilla y un circuito impreso.

Los avances que hicieron posible el circuito integrado han sido, fundamentalmente, los desarrollos en la fabricación de dispositivos semiconductores a mediados del siglo XX, y los descubrimientos experimentales que mostraron que estos dispositivos podían reemplazar las funciones de las válvulas o tubos de vacío, que se volvieron rápidamente obsoletos al no poder competir con el pequeño tamaño, el consumo de energía moderado, los tiempos de conmutación mínimos, la confiabilidad, la capacidad de producción en masa y la versatilidad de los CI.

Entre los circuitos integrados más complejos y avanzados se encuentran los microprocesadores, que controlan numerosos aparatos, desde teléfonos móviles y hornos de microondas hasta computadoras. Los chips de memorias digitales son otra familia de circuitos integrados, de importancia crucial para la moderna sociedad de la información. Mientras que el costo de diseñar y desarrollar un circuito integrado complejo es bastante alto, cuando se reparte entre millones de unidades de producción, el costo individual de los CI por lo general se reduce al mínimo. La eficiencia de los CI es alta debido a que el pequeño tamaño de los chips permite cortas conexiones que posibilitan la utilización de lógica de bajo consumo (como es el caso de CMOS), y con altas velocidades de conmutación.

Existen al menos tres tipos de circuitos integrados:

- Circuitos monolíticos: están fabricados en un solo monocristal, habitualmente de silicio, pero también existen en germanio, arseniuro de galio, silicio-germanio, entre otros.

- Circuitos híbridos de capa fina: son muy similares a los circuitos monolíticos, pero, además, contienen componentes difíciles de fabricar con tecnología monolítica. Muchos conversores A/D y conversores D/A se fabricaron en tecnología híbrida hasta que los progresos en la tecnología permitieron fabricar resistores precisos.

- Circuitos híbridos de capa gruesa: se apartan bastante de los circuitos monolíticos. De hecho suelen contener circuitos monolíticos sin cápsula, transistores, diodos, entre otros, sobre un sustrato dieléctrico, interconectados con pistas conductoras. Los resistores se depositan por serigrafía y se ajustan haciéndoles cortes con láser. Todo ello se encapsula, en cápsulas plásticas o metálicas, dependiendo de la disipación de energía calórica requerida. En muchos casos, la cápsula no está "moldeada", sino que simplemente se cubre el circuito con una resina epoxi para protegerlo. En el mercado se encuentran circuitos híbridos para aplicaciones en módulos de radio frecuencia (RF), fuentes de alimentación, circuitos de encendido para automóvil, entre otros.

La mayoría de los circuitos integrados digitales vienen en presentación tipo DIP (*Dual In-line Package*) o de doble hilera. El pin Núm. 1 se identifica mediante una ranura o un punto grabado en la parte superior de la cápsula. La enumeración de los pines se realiza en sentido contrario al de las agujas del reloj.

Las configuraciones más comunes de los CI digitales tipo DIP son las de 8, 14, 16, 24, 40 y 64 pines. Estas dos últimas contienen generalmente microprocesadores y otras funciones digitales relativamente complejas.

La cápsula trae impresa la información respecto al fabricante, la referencia del dispositivo y la fecha de fabricación. Cada fabricante de circuitos integrados (National, Texas, Fairchild, Motorola, entre otros) se identifica mediante un logotipo distintivo.

El código de la fecha informa cuando fue manufacturado el chip. Las dos primeras cifras indican el año y las dos últimas se refieren al mes o semana de fabricación. Por ejemplo 8307 significa la séptima semana de 1983.

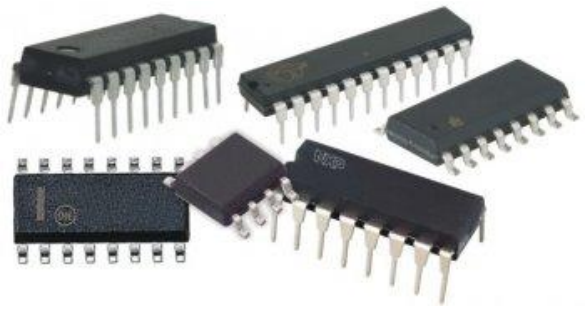
En la presentación tipo DIP, los pines de acceso están espaciados entre sí 2,5 mm. Para efectos de montaje experimental los CI pueden insertarse en un protoboard o tablero sin soldaduras.

Para montajes definitivos en circuito impreso pueden estar soldados directamente al cobre o montados sobre una base o *socket*. La utilización de bases simplifica la instalación durante el ensamble y el remplazo en caso de daño.

Además del tipo DIP, existen otras presentaciones comunes de los circuitos integrados digitales como la cápsula metálica (T0-5), la plana y la tipo *chip carrier.*

Los circuitos integrados digitales se pueden clasificar en dos grandes grupos de acuerdo al tipo de transistores utilizados para implementar sus funciones internas de conmutación: bipolares y MOS.

Figura 5. **Tipos de CI**



Fuente: Burgos, Vicente. *La historia de los circuitos integrados*. http://www.mundodigital.net/la-historia-de-los-circuitos-integrados/. Consulta: 16 de junio de 2014.

Los circuitos integrados bipolares se fabrican con transistores bipolares tipo NPN y PNP, y los de tipo MOS utilizan MOSFET (transistores de efecto de campo de compuerta aislada.

Dentro de cada categoría, los fabricantes han desarrollado una amplia variedad de familias lógicas de circuitos integrados tanto MOS como bipolares.

Una familia lógica es un grupo de chips o módulos funcionales, fabricados de acuerdo a la misma tecnología y eléctricamente compatibles, es decir se pueden interconectar directamente entre sí para configurar cualquier tipo de sistema digital.

Algunas veces es posible interconectar circuitos de dos familias diferentes adaptando los niveles de voltaje entre ellos mediante interface apropiadas.

Las familias bipolares más conocidas son la RTL (lógica de resistor a transistor), la DTL (lógica de diodo a transistor), la TTL (lógica de transistor a transistor), la ECL (lógica de emisor acoplado) y la I^2L (lógica de inyección integrada).

Las familias MOS más conocidas como CMOS (lógica de transistores MOSFET complementarios), la PMOS (lógica de transistores MOSFET canal P) y la NMOS (lógica de transistores MOSFET canal N). Los dispositivos de estas familias se caracterizan por su bajo consumo de potencia y su alta capacidad de integración.

Los circuitos integrados digitales TTL, se caracterizan por su bajo costo, su alta velocidad, su moderada inmunidad al ruido. La serie más popular de esta familia es la 74XX, constituida por los chips cuya referencia comienza por 74 como el 7400, 7404, 74LS04, 74L93, 74ALS1035, entre otros.

Los circuitos integrados CMOS se caracterizan, entre otras cosas, por su amplio rango de voltajes de operación, su bajo consumo de corrientes y alta inmunidad al ruido. Una de las series más populares de esta familia es la 40XXB, constituida por los chips cuya referencia comienza por 40 o 45 y termina en B como 4017B, 40163B, entre otros.

Las características más importantes de un circuito integrado digital son su velocidad, consumo de potencia, inmunidad al ruido y confiabilidad. A continuación se definen estos términos, desde un punto de vista general.

La velocidad mide la rapidez de respuesta de las salidas de un circuito digital a cualquier cambio en sus entradas. La velocidad es una consideración importante en el diseño de sistemas, que deben realizar cálculos numéricos o en circuitos que trabajan con señales de alta frecuencia.

El consumo de potencia mide la cantidad de corriente o potencia que consume un circuito digital en operación.

El consumo de potencia es una consideración importante en el diseño de sistemas operados por baterías.

La inmunidad al ruido mide la sensibilidad de un circuito digital al ruido electromagnético ambiental. La inmunidad al ruido es una consideración importante en el diseño de sistemas que deben trabajar en ambientes ruidosos como automóviles, máquinas, circuitos de control industrial, entre otros.

La confiabilidad mide el período útil de servicio de un circuito digital, es decir, cuánto tiempo se espera que trabaje sin fallar.

1.3.1.1. Familia lógica TTL

La familia lógica TTL es quizás la más antigua y común de todas las familias lógicas de circuitos integrados digitales. La mayor parte de los chips SSI y MSI se fabrican utilizando la tecnología TTL. Los circuitos integrados TTL implementan su lógica interna, exclusivamente, a base de transistores NPN y PNP, diodos y resistencias.

La primera serie de dispositivos digitales TTL fue lanzada por la Texas Instruments en 1964. Los chips TTL se usan en toda clase de aplicaciones digitales, desde el más sencillo computador personal hasta el más sofisticado robot industrial. Los circuitos TTL son rápidos, versátiles y muy económicos.

La familia TTL está disponible en dos versiones la serie 54 y la serie 74. La primera se destina a aplicaciones militares y la segunda a aplicaciones industriales y de propósito general. Los dispositivos de la serie 54 tienen rangos de operación de temperatura y voltaje más flexible (desde -55 hasta 125 grados Celsius contra 0 a 70 de la serie 74).

La familia TTL o bipolar se divide en las siguientes categorías o subfamilias básicas:

- TTL estándar
- TTL Schonttky (S)
- TTL de baja potencia (L)
- TTL Schonttky de baja potencia (LS)
- TTL de alta velocidad (H)
- TTL Schonttky avanzada (AS)
- TTL Schonttky de baja potencia avanzada (ALS)

La más utilizada es la TTL estándar que comprende principalmente los dispositivos que se designan como 74XX. Existen una gran cantidad de funciones lógicas que se realizan con esta tecnología. Entre las principales se tienen compuertas, decodificadores, contadores, *flip-flops*, sumadores, multiplexores entre otros.

Las características más notables de los circuitos integrados de la familia TTL estándar son, a grandes rasgos:

- Alta velocidad de operación: pueden trabajar generalmente a frecuencias de 18 a 20 MHZ y en algunos casos hasta 80 MHZ. La velocidad de operación se expresa generalmente en términos del tiempo o retardo de propagación del chip. El tiempo o retardo de propagación de un circuito digital es el tiempo que toma un cambio lógico en la entrada, en propagarse a través del dispositivo y producir un cambio lógico en la salida. Los tiempos de propagación en TTL son típicamente del orden de 2 a 30 nanosegundos por compuerta.

- Alta disipación de potencia: es una desventaja asociada con la alta velocidad de operación. En general, cuanto más rápido sea un circuito, más potencia consume y viceversa. La mayoría de los circuitos TTL disipan, típicamente, de 1 a 25 milivatios por compuerta.

- Tensión de alimentación nominal de +5V. Los circuitos TTL, en general pueden operar con tensiones de CC entre 4,75 y 5,25 V pero el valor nominal de la tensión de trabajo es +5 V.

- Niveles de voltaje de 0 a 0,8 V para el estado bajo y 2,4 a 5,0 V para el estado alto. En general, los circuitos TTL interpretan cualquier voltaje entre 0 y 0,8 V como un cero (0) lógico o bajo, y cualquier voltaje entre 2,4 y 5 V como uno (1) lógico o alto.

El máximo voltaje positivo que puede aplicarse a una entrada TTL es de +5,5 V y el máximo negativo es -0,5 V. Al excederse estos parámetros, los dispositivos TTL generalmente se destruyen.

La familia TTL utiliza dos parámetros para determinar cuántos dispositivos TTL se pueden conectar entre sí. Estos parámetros se denomina abanico de entrada (*fan in*) y abanico de salida (*fan out*).

El *fan-in* mide el efecto de carga que presenta una entrada a una salida. Cada entrada de un circuito TTL estándar se comporta como una fuente de corriente capaz de suministrar 1,8 mA a este valor de corriente se le asigna un *fan-in* de 1.

El *fan-out* mide la capacidad de una salida de manejar una o más entradas. Cada salida de un circuito TTL estándar se comporta como un

disipador de corriente capaz de aceptar hasta 18 mA, es decir de manejar hasta 10 entradas TTL estándares. Por lo tanto, el *fan-out* de una salida TTL estándar es 10.

Existen dispositivos TTL especiales llamados *buffers* (separadores) y *drivers* (manejadores) que tienen *fan-outs* de 30, 50 e incluso 100. Se utilizan en aplicaciones donde una determinada línea de salida debe manejar al mismo tiempo un gran número de líneas de entrada.

1.3.1.2. Familia CMOS

La familia lógica CMOS es, junto con la TTL, una de las familias lógicas más populares. Utiliza transistores MOSFET complementarios (canal N y canal P) como elementos básicos de conmutación.

CMOS es una abreviación de *Complementary Metal Oxide Semiconductors* (semiconductores complementarios de óxido metálico). Los circuitos integrados digitales fabricados mediante tecnología CMOS, se pueden agrupar en las siguientes categorías o subfamilias básicas:

- CMOS estándar
- CMOS de alta velocidad (HC)
- CMOS compatible con TTL (HCT)
- CMOS equivalente a TTL (C)

La familia CMOS estándar comprende principalmente los dispositivos que se designan como 40XX y 45XX, existen dos series generales de dispositivos CMOS designadas “A” y “B”. Los dispositivos de la serie “A” se designan con el

sufijo A o simplemente no lo traen por ejemplo 4011=4011A. Todos los dispositivos de la serie “B” llevan el sufijo B por ejemplo 4029B.

La principal diferencia entre los dispositivos de las series A y B está en que los CMOS “B” contienen una circuitería interna de protección, que reduce el riesgo de daño del dispositivo por el fenómeno de descarga electrostática.

De otro lado, los dispositivos CMOS “B” tienen frecuencia de operación más altas, tiempos de propagación más cortos y mayor capacidad de salida (*fan-out*) que los dispositivos serie “A”. Las características más sobresalientes de la familia CMOS estándares 40 y 45 son, a grandes rasgos, las siguientes:

- Baja disipación de potencia: es la ventaja más sobresaliente. En estado de reposo, una compuerta CMOS típica consume alrededor de 10 nanovatios. Este bajo consumo de potencia simplifica el diseño y el costo de la fuente de alimentación. Por esta razón, los circuitos integrados CMOS se utilizan extensamente en equipos operados por pilas o baterías.

- Buena velocidad de operación: los circuitos integrados CMOS son típicamente más lentos que los TTL pero suficientemente rápidos para la mayoría de las aplicaciones. Pueden operar a frecuencias hasta de 1 MHz y tienen tiempos de propagación del orden de 10 a 50 nanosegundos por compuerta.

- Amplios márgenes de tensión de alimentación: los dispositivos de la serie 40XXA puede operar con tensiones entre +3 y +15 voltios, y los de la serie 40XXB con tensiones entre +3 y +18 voltios. La tensión de alimentación se designa como VDD. Cuando se emplean circuitos TTL y

CMOS en el mismo sistema, se utiliza generalmente VDD de +5 voltios. Cuando hay circuitos TTL y CMOS trabajando a tensiones diferentes deben hacerse compatibles los niveles lógicos de ambas familias mediante circuitos apropiados de interface.

- Niveles de voltaje de 0 a 0,3 VDD, para el estado bajo y de 0,7 VDD a VDD para el estado alto: por ejemplo, si se utiliza una tensión de alimentación VDD de 10V, los dispositivos CMOS interpretarán un voltaje entre 0 y 3 voltios como un estado lógico bajo o 0, y un voltaje entre 7 y 10 voltios como un estado lógico alto o 1.

- Alta inmunidad al ruido: los circuitos CMOS son esencialmente inmunes al ruido electromagnético externo generado por aparatos eléctricos, líneas de transmisión, descargas atmosféricas. Esta característica los hace excelentes en aplicaciones industriales y automotrices, donde son comunes los altos niveles de ruido.

1.3.2. Resistencias

Se le denomina resistencia eléctrica a la igualdad de oposición que tienen los electrones al desplazarse a través de un conductor. La unidad de resistencia en el Sistema Internacional es el ohmio, que se representa con la letra griega omega (Ω), en honor al físico alemán George Ohm, quien descubrió el principio que ahora lleva su nombre. La resistencia está dada por la siguiente fórmula:

Figura 6. **Estructura física de la resistencia eléctrica**

$$R = \rho \frac{\ell}{S}$$

Fuente: elaboración propia.

En donde ρ es el coeficiente de proporcionalidad o la resistividad del material.

La resistencia de un material depende directamente de dicho coeficiente, además es directamente proporcional a su longitud (aumenta conforme es mayor su longitud), y es inversamente proporcional a su sección transversal (disminuye conforme aumenta su grosor o sección transversal).

Descubierta por Georg Ohm en 1827, la resistencia eléctrica tiene un parecido conceptual a la fricción en la física mecánica. La unidad de la resistencia en el Sistema Internacional de Unidades es el ohmio (Ω). Para su medición, en la práctica existen diversos métodos, entre los que se encuentra el uso de un ohmímetro. Además, su cantidad recíproca es la conductancia, medida en Siemens.

Además, de acuerdo con la ley de Ohm la resistencia de un material puede definirse como la razón entre la diferencia de potencial eléctrico, y la corriente en que atraviesa dicha resistencia, así:

Figura 7. **Estructura de la resistencia en un circuito eléctrico**

$$R = \frac{V}{I}$$

Fuente: elaboración propia.

Donde R es la resistencia en ohmios, V es la diferencia de potencial en voltios e I es la intensidad de corriente en amperios.

También puede decirse que "la intensidad de la corriente que pasa por un conductor es directamente proporcional a la longitud e inversamente proporcional a su resistencia".

Según sea la magnitud de esta medida, los materiales se pueden clasificar en conductores, aislantes y semiconductor. Existen además ciertos materiales en los que, en determinadas condiciones de temperatura, aparece un fenómeno denominado superconductividad, en el que el valor de la resistencia es prácticamente nulo.

1.3.2.1. Resistencia *pull-up* y *pull-down*

En la electrónica digital los valores de tensiones utilizados, va desde los 0 V hasta poco más de 15 V. En las placas de los microcontroladores la tensión está regulada a 5 V.

Los pines que están asociados a los microcontroladores se pueden encontrar en dos estados, en estado de entrada, a la espera de recibir tensión por parte del componente que se una a ese pin o en estado de salida, en cuyo caso suministra tensión, por ejemplo, a un led que se asocie a ese pin.

Al estar hablando de voltajes bajos, puede darse el caso de que se tenga el pin de la placa como entrada, esperando que entre señal, por ejemplo del potenciómetro que se ha conectado a ese pin, y se diera el caso que por culpa del “ruido” ocasionado por alguna interferencia, aumentara, de repente, la tensión en el pin. Esa tensión pasaría por el pin al microcontrolador y este la procesaría como si fuese del potenciómetro. Para evitar estos casos, se conecta una resistencia entre la entrada del pin y la masa (0 V), para eliminar

cualquier fluctuación de tensión que engañe al micro, a esta forma de colocar la resistencia se le llama *pull-down*.

En el caso contrario, si lo que se quiere es mantener el pin a 1 (HIGH), es decir con tensión, se colocaría la resistencia entre la fuente de tensión (5 V) y el pin para mantener el voltaje constante, y que no haya caída de tensión a esta forma de colocar la resistencia se le conoce con *pull-up*.

1.3.2.2. Potenciómetros

Un potenciómetro es un resistor cuyo valor de resistencia es variable. De esta manera, indirectamente, se puede controlar la intensidad de corriente que fluye por un circuito si se conecta en paralelo, o la diferencia de potencial al conectarlo en serie.

Normalmente, los potenciómetros se utilizan en circuitos de poca corriente. Para circuitos de corrientes mayores, se utilizan los reóstatos, que pueden disipar más potencia.

Figura 8. **Potenciómetro comercial**

Fuente: Wikipedia. *Potenciómetros*. http://es.wikipedia.org/wiki/Potenciometros. Consulta: 17 de junio de 2014.

1.3.3. Transistores

Es el transistor el elemento que revolucionó la electrónica, permitió el paso de la electrónica analógica a la electrónica digital y posteriormente la microelectrónica.

1.3.3.1. Transistor de unión bipolar

El transistor de unión bipolar, o BJT por sus siglas en inglés, se fabrica básicamente sobre un monocristal de germanio, silicio o arseniuro de galio, que tienen cualidades de semiconductores, estado intermedio entre conductores como los metales y los aislantes como el diamante. Sobre el sustrato de cristal, se contaminan en forma muy controlada tres zonas, dos de las cuales son del mismo tipo, NPN o PNP, quedando formadas dos uniones NP.

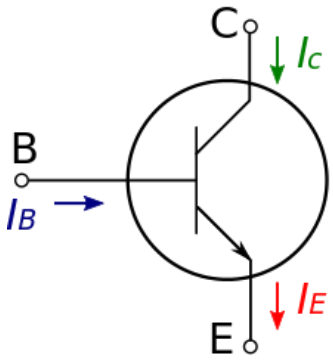
La zona N con elementos donantes de electrones (cargas negativas) y la zona P de aceptadores o «huecos» (cargas positivas). Normalmente se utilizan como elementos aceptadores P al indio (In), aluminio (Al) o galio (Ga) y donantes N al arsénico (As) o fósforo (P).

La configuración de uniones PN, dan como resultado transistores PNP o NPN, donde la letra intermedia siempre corresponde a la característica de la base, y las otras dos al emisor y al colector que, si bien son del mismo tipo y de signo contrario a la base, tienen diferente contaminación entre ellas (por lo general, el emisor está mucho más contaminado que el colector).

El mecanismo que representa el comportamiento semiconductor dependerá de dichas contaminaciones, de la geometría asociada y del tipo de

tecnología de contaminación (difusión gaseosa, epitaxial, entre otros) y del comportamiento cuántico de la unión.

Figura 9. **Diagrama del transistor bipolar**



Fuente: Wikipedia. *Transistores*. http://es.wikipedia.org/wiki/Transistores. Consulta: 17 de junio de 2014.

1.3.3.2. Transistor de efecto de campo

El transistor de efecto de campo de unión (JFET), fue el primer transistor de efecto de campo en la práctica. Lo forma una barra de material semiconductor de silicio de tipo N o P. En los terminales de la barra se establece un contacto óhmico, se tiene así un transistor de efecto de campo tipo N de la forma más básica. Si se difunden dos regiones P en una barra de material N y se conectan externamente entre sí, se producirá una puerta. A uno de estos contactos se llamará surtidor y al otro drenador.

Aplicando tensión positiva entre el drenador y el surtidor y conectando la puerta al surtidor, se establecerá una corriente, a la que se llamará corriente de drenador con polarización cero. Con un potencial negativo de puerta al que se llamará tensión de estrangulamiento, cesa la conducción en el canal.

El transistor de efecto de campo, o FET por sus siglas en inglés, que controla la corriente en función de una tensión; tienen alta impedancia de entrada. Transistor de efecto de campo de unión, JFET, construido mediante una unión PN.

Transistor de efecto de campo de compuerta aislada, IGFET, en el que la compuerta se aísla del canal mediante un dieléctrico.

Transistor de efecto de campo MOS, MOSFET, donde MOS significa Metal-Óxido-Semiconductor, en este caso la compuerta es metálica y está separada del canal semiconductor por una capa de óxido.

1.4. Historia de los microprocesadores

Desde un punto de vista funcional, un microprocesador es un circuito integrado, que incorpora en su interior una unidad central de proceso (CPU) y todo un conjunto de elementos lógicos, que permiten enlazar otros dispositivos como memoria y puertos de entrada o de salida (I/O), formando un sistema completo para cumplir con una aplicación específica dentro del mundo real.

Se dice que los babilónicos utilizaron el ábaco para realizar cuentas, este podría ser considerado la primer forma de un procesador, años después Blas Pascal diseñó su propia calculadora mecánica, para que finalmente en la década de los 40 se elaboraran grandes procesadores con base en tubos de vacíos, este desarrollo fue impulsado debido a las dos guerras mundiales, ya que la humanidad se dio cuenta que la guerra no solo se ganan en los campos de batalla, sino que también en los escritorios con tácticas e intentando adelantarse a los movimientos del enemigo, de ahí que surgen el interés en campos como la codificación de información.

Después de la guerra y con el invento del transistor los procesadores fueron tomando dimensiones pequeñas hasta que finalmente Intel desarrolló el primer microprocesador, este surgió de la evolución de distintas tecnologías predecesoras, básicamente de la computación y de la tecnología de semiconductores. El 15 de noviembre de 1971 Intel lanza su primer microprocesador: el Intel 4004. El Intel 4004 (i4004), un CPU de 4 bits, fue el primer microprocesador en un solo chip, así como el primero disponible comercialmente. Con el Intel 4004 se conseguía situar en placas de 0,25 centímetros cuadrados un circuito integrado que contenía 2 300 transistores, este direccionaba 4 096 localidades de 4 bits, suficiente para aquella época.

Intel considerando el éxito comercial del 4004 sacó al mercado el 1 de abril de 1972, Intel anunció una versión mejorada de su procesador anterior. Era el 8008, y su principal ventaja frente a otros modelos, fue poder acceder a más memoria y procesar 8 bits. La velocidad de su reloj alcanzaba los 740 KHz.

Fue el primer microprocesador de 8 bits, implantado con tecnología PMOS, contaba con 48 instrucciones, podía ejecutar 300 000 operaciones por segundo y direccionaba 16 Kbytes de memoria.

En abril de 1974 Intel lanzó el 8080 con una velocidad de reloj que alcanzaba los 2 MHz. Al año siguiente, aparece en el mercado el primer ordenador personal, de nombre Altaír, basado en la microarquitectura del Intel 8080. El procesador de este computador suponía multiplicar por 10 el rendimiento del anterior, gracias a sus 2 MHz de velocidad.

Este microprocesador también direccionaba 8 bits, tenía 78 instrucciones, su velocidad de operación era 10 veces mayor que la del 8008 y podía direccionar hasta 64 Kbytes de memoria.

En esta época Zilog Inc. construyó el microprocesador Z-80, el cual incorporaba un conjunto de instrucciones más extenso que el 8080, aunque era compatible con este último, este microprocesador ha sido uno de los más utilizados en el campo de control.

También Motorola, desarrolló el 6800, un microprocesador de 8 bits con un diseño completamente distinto pero iguales características al 8080. En junio de 1978 y 1979 hacen su aparición los microprocesadores 8086 y 8088 que pasaron a formar el IBM PC, equipo que salió al mercado en 1981.

Los i8086 e i8088 se basaron en el diseño del Intel 8080 y el Intel 8085, y de hecho son compatibles a nivel de ensamblador con el i8080. Ambos tienen cuatro registros generales de 16 bits, que también pueden ser accedidos como ocho registros de 8 bits, con cuatro registros.

El 16 de octubre de 1985 Intel lanza el i80386, con arquitectura de x86. Fue empleado como la unidad central de proceso de muchos computadores personales desde mediados de los años 80 hasta principios de los 90. También conocido como 386, con una velocidad de reloj entre 16 y 40 MHz. Este producto se destacó principalmente por ser un microprocesador con arquitectura de 32 bits.

Después del procesador de 16 bits, aparecieron los de 32 bits que estuvieron en auge en la primer década del siglo XXI, en la actualidad las computadoras utilizan procesadores de 64 bits, teniendo estos mayor capacidad de procesamiento, velocidad de reloj y con memorias de disco duro del orden de los Terabits.

2. FUNDAMENTOS DE MICROCONTROLADORES

2.1. Microprocesadores y microcontroladores

Puede ser que existan lectores que piensen que al hablar de un microprocesador y un microcontrolador se esté refiriendo a lo mismo, en realidad dado que son muy parecidos, a continuación se explica sus principales diferencias.

El microprocesador es un dispositivo electrónico que hace la función de cerebro dentro de un sistema, es el encargado de ejecutar instrucciones programadas como sumar restar y accesos a memorias, para que el microprocesador pueda realizar una acción real, es necesario conectarlo a otros componentes.

El microcontrolador en cambio fue concedido como un dispositivo programable que puede ejecutar un sinnúmero de tareas y procesos, no necesita otros componentes para su utilización, pues ya tiene integrado los periféricos o componentes necesarios para poder ser aplicado en procesos reales. Algunas diferencias que se pueden mencionar son:

- La CPU del microcontrolador es más simple y sus instrucciones están orientadas, principalmente, a la operación de cada una de las líneas de entrada y salida.
- La memoria RAM (Datos), que ofrece los microcontroladores, es de baja capacidad, la razón es que las aplicaciones de control e instrumentación comunes no necesitan almacenar grandes cantidades de información

temporal, en los microprocesadores pueden acceder a través de los buses a grandes bancos de memoria RAM externa de acuerdo a las necesidades del sistema.

- En los microcontroladores la memoria ROM (Programa) es limitada.
- Con los microcontroladores no es necesario diseñar complejos circuitos decodificadores porque el mapa de memoria y de puestos I/O está incluido, internamente.
- La mayoría de microcontroladores no tienen accesibles los buses de direcciones, de datos y de control del CPU.
- La velocidad de operación de los microcontroladores es más lenta que la que se puede lograr con microprocesadores.

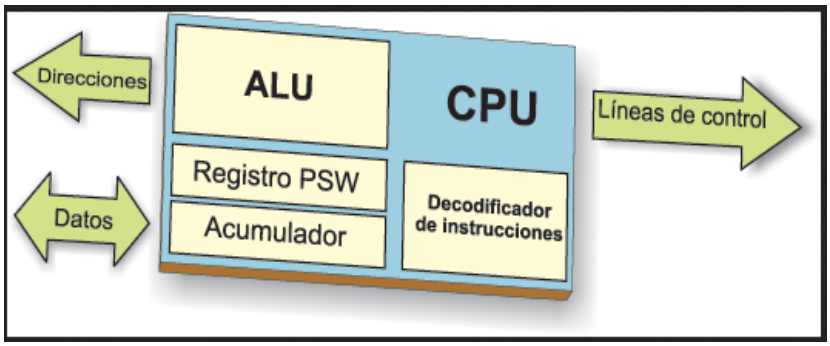
2.2. Características de los microcontroladores

Para empezar el desglose sobre los microcontroladores se hará hablando de la CPU, esta es la “Unidad Central de Procesamiento” en inglés *Central Processor Unit* (CPU), siendo la encargada de controlar todos los procesos dentro del microcontrolador las partes principales son:

- Decodificador de instrucciones: es la parte que descodifica las instrucciones del programa y acciona otros circuitos basándose en esto.
- Unidad lógica aritmética (*Arithmetical Logical Unit* (ALU)): realiza todas las operaciones matemáticas y lógicas sobre datos. El “conjunto de instrucciones” que es diferente para cada familia de microcontrolador expresa las capacidades de este circuito.
- Acumulador o registro de trabajo: es un registro SFR estrechamente relacionado con el funcionamiento de la ALU. Es un tipo de escritorio de trabajo utilizado para almacenar todos los datos, sobre los que se debe realizar alguna operación (sumar, mover). También almacena los

resultados preparados para el procesamiento futuro. Uno de los registros SFR, denominado Registro *Status* (PSW), está estrechamente relacionado con el acumulador. Muestra el "estado" de un número almacenado en el acumulador (el número es mayor o menor que cero entre otros.) en cualquier instante dado. El acumulador es denominado registro de trabajo (*working register*), o sea, registro W o solamente W.

Figura 10. **ALU**



Fuente: MikroElectronika. *Libro de programación en mikrobasic*. http://www.mikroe.com/chapters/view/84/libro-de-la-programacion-de-los-microcontroladores-pic-en-basic-capitulo-1-mundo-de-los-microcontroladores/. Consulta: 18 de junio de 2014.

2.3. Memoria

En los microcontroladores la memoria de instrucciones y datos está integrada en el propio circuito integrado. Una parte debe ser no volátil, tipo ROM, y se destina a contener el programa de instrucciones que gobierna la aplicación. Otra parte de memoria será tipo RAM, volátil, y se destina a guardar las variables y los datos.

Hay varios tipos de memoria dentro del microcontrolador:

2.3.1. Memoria ROM (*Read Only Memory*) - memoria de solo lectura

La memoria ROM se utiliza para guardar permanentemente el programa que se está ejecutando. El tamaño de programa que se puede escribir depende del tamaño de esta memoria. Los microcontroladores actuales normalmente utilizan el direccionamiento de 16 bits, que significa que son capaces de direccionar hasta 64 Kb de memoria, o sea 65 535 localidades. Hay varios tipos de memoria ROM.

2.3.2. ROM de máscara (enmascarada) –MROM

La ROM enmascarada es un tipo de ROM cuyo contenido es programado por el fabricante. El término “de máscara” viene del proceso de fabricación, donde las partes del chip se plasman en las máscaras utilizadas durante el proceso de fotolitografía. En caso de fabricación de grandes series, el precio es muy bajo.

2.3.3. OTP ROM (*One Time Programmable* ROM) - ROM programable una sola vez

La memoria programable una sola vez permite descargar un programa en el chip, pero como dice su nombre, una sola vez. Si se detecta un error después de descargarlo, lo único que se puede hacer es descargar el programa correcto en otro chip.

2.3.4. Memoria Flash

Este tipo de memoria se inventó en los años 80 en los laboratorios de la

compañía INTEL, como forma desarrollada de la memoria UVEPROM. Ya que es posible escribir y borrar el contenido de esta memoria prácticamente un número ilimitado de veces, los microcontroladores con memoria Flash son perfectos para estudiar, experimentar y para la fabricación en pequeña escala. Por la gran popularidad de esta memoria, la mayoría de los microcontroladores se fabrican con tecnología flash hoy en día.

2.3.5. MEMORIA RAM (*Random Access Memory*) - memoria de acceso aleatorio

Al apagar la fuente de alimentación, se pierde el contenido de la memoria RAM. Se utiliza para almacenar temporalmente los datos y los resultados inmediatos creados y utilizados durante el funcionamiento del microcontrolador. Por ejemplo, si el programa ejecuta la adición (de cualquier cosa) es necesario tener un registro que representa lo que se llama “suma” en vida cotidiana. Con tal propósito, uno de los registros de la RAM es denominado “suma” y se utiliza para almacenar los resultados de la adición.

2.3.6. Memoria EEPROM (*Electrically Erasable Programmable* ROM) - ROM programable y borrable eléctricamente

El contenido de la EEPROM se puede cambiar durante el funcionamiento (similar a la RAM), pero se queda permanentemente guardado después de la pérdida de la fuente de alimentación (similar a la ROM). Por lo tanto, la EEPROM se utiliza con frecuencia para almacenar los valores creados durante el funcionamiento, que tienen que estar permanentemente guardados. Por ejemplo, si se ha diseñado una llave electrónica o una alarma, sería estupendo permitir al usuario crear e introducir una contraseña por su cuenta. Por supuesto, la nueva contraseña tiene que estar guardada al apagar la fuente de

alimentación. En tal caso una solución perfecta es el microcontrolador con una EEPROM embebida.

2.4. Bus

El bus está formado por 8, 16 o más cables. Hay dos tipos de buses: el bus de direcciones y el bus de datos. El bus de direcciones consiste en tantas líneas como sean necesarias para direccionar la memoria. Se utiliza para transmitir la dirección de la CPU a la memoria. El bus de datos es tan ancho como los datos, en este caso es de 8 bits o cables de ancho. Se utiliza para conectar todos los circuitos dentro del microcontrolador.

2.4.1. El bus de direcciones

Contiene la información digital que envía el microprocesador a la memoria y demás elementos direccionables del sistema para seleccionar una posición de memoria, una unidad de entrada/salida o un registro en particular. El número de líneas disponible en el bus de direcciones (n) determina el tamaño máximo de memoria que puede ser acomodado en el sistema (2^n).

2.4.2. El bus de datos

Lleva datos e instrucciones hacia y desde el microprocesador. Las instrucciones proceden siempre de la memoria mientras que los datos que procesa u obtiene el programa de instrucciones, puede provenir de o ir hacia la memoria o los módulos de entrada/salida.

Generalmente el número de líneas de entrada es igual al número de líneas de salida, este número define la longitud de la palabra de datos del

microprocesador. Son comunes longitudes de palabra de 4, 8, 16, 32, 64 bits, en microcontroladores los más comunes son de 8 y 16 bits.

2.5. Arquitectura interna

Los microprocesadores utilizan dos formas de intercambiar datos entre la CPU y la memoria, estas son: la arquitectura Von Neumann y la arquitectura Harvard.

2.5.1. Arquitectura Von Neumann

- Los microcontroladores que utilizan la arquitectura Von Neumann disponen de un solo bloque de memoria y de un bus de datos de 8 bits. Como todos los datos se intercambian por medio de estas 8 líneas, este bus está sobrecargado, y la comunicación por sí misma es muy lenta e ineficaz. La CPU puede leer una instrucción o leer/escribir datos de/en la memoria. Los dos procesos no pueden ocurrir a la vez puesto que las instrucciones y los datos utilizan el mismo bus. Por ejemplo, si alguna línea de programa dice que el registro de la memoria RAM llamado "SUM" debe ser aumentado por uno (instrucción: incf SUMA), el microcontrolador hará lo siguiente:

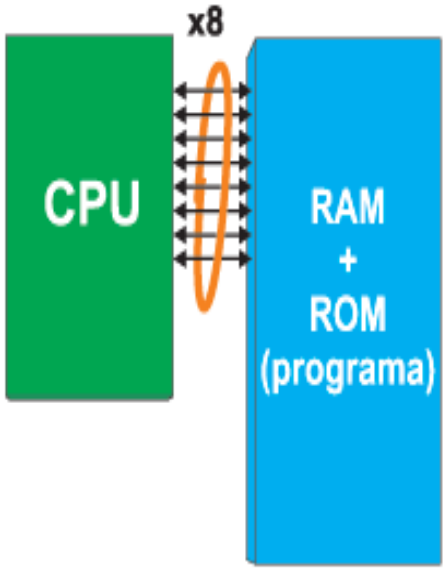
- Leer la parte de la instrucción de programa que especifica QUÉ es lo que debe realizar (en este caso es la instrucción para incrementar "incf").

- Seguir leyendo la misma instrucción que especifica sobre CUÁL dato lo debe realizar (en este caso es el contenido del registro "SUMA").

- Después de haber sido incrementado, el contenido de este registro se debe escribir en el registro del que fue leído (dirección del registro “SUMA”).

El mismo bus de datos se utiliza para todas estas operaciones intermedias al intercambiar los datos entre la CPU y la memoria.

Figura 11. **Arquitectura de Von Neumann**



Fuente: MikroElectronika. *Libro de programación en mikrobasic*. http://www.mikroe.com/chapters/view/84/libro-de-la-programacion-de-los-microcontroladores-pic-en-basic-capitulo-1-mundo-de-los-microcontroladores/. Consulta: 18 de junio de 2014.

2.5.2. Arquitectura Harvard

Los microcontroladores que utilizan esta arquitectura disponen de dos buses de datos diferentes. Uno es de 8 bits de ancho y conecta la CPU con la memoria RAM. El otro consiste en varias líneas (12, 14 o 16) y conecta a la CPU y la memoria ROM. Por consiguiente, la CPU puede leer las instrucciones

y realizar el acceso a la memoria de datos a la vez. Puesto que todos los registros de la memoria RAM son de 8 bits de ancho, todos los datos dentro del microcontrolador que se intercambian son de la misma anchura.

Durante el proceso de la escritura de programa, solo se manejan los datos de 8 bits. En otras palabras, todo lo que usted podrá cambiar en el programa y a lo que podrá afectar será de 8 bits de ancho. Todos los programas escritos para estos microcontroladores serán almacenados en la memoria ROM interna del microcontrolador después de haber sido compilados a código máquina. No obstante, estas localidades de memoria ROM no tienen 8, sino 12, 14 o 16 bits. 4, 6 o 8 bits adicionales representan una instrucción que especifica a la CPU qué hacer con los datos de 8 bits.

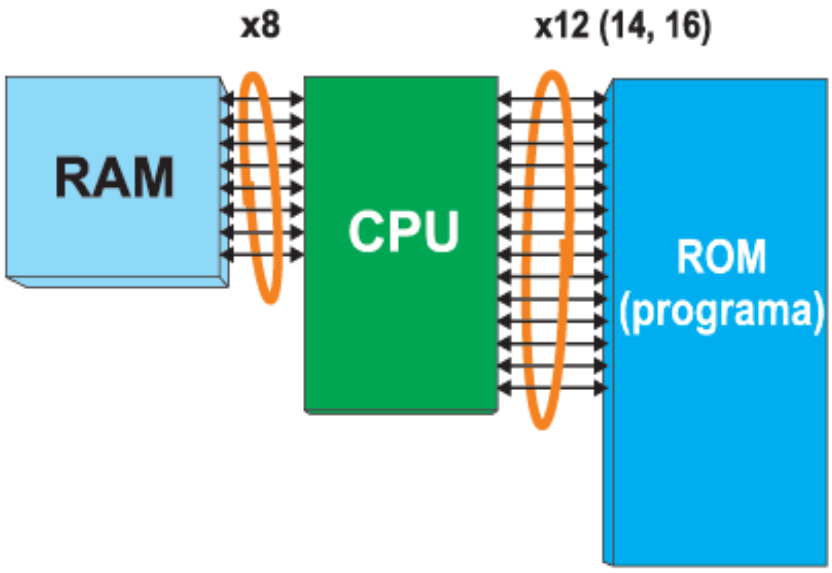
Las ventajas de esta arquitectura son las siguientes:

- Todos los datos en el programa son de un byte (8 bits) de ancho. Como un bus de datos utilizado para lectura de programa tiene unas líneas más (12, 14 o 16), tanto la instrucción como el dato se pueden leer simultáneamente al utilizar estos bits adicionales. Por eso, todas las instrucciones se ejecutan en un ciclo salvo las instrucciones de salto que son de dos ciclos.

- El hecho de que un programa (la ROM) y los datos temporales (la RAM) estén separados, permite a la CPU poder ejecutar dos instrucciones simultáneamente. Dicho de manera sencilla, mientras que se realiza la lectura o escritura de la RAM (que marca el fin de una instrucción), la siguiente instrucción se lee por medio de otro bus.

- En los microcontroladores que utilizan la arquitectura de Von Neumann, nunca se sabe cuánta memoria ocupará algún programa. Generalmente, la mayoría de las instrucciones de programa ocupan dos localidades de memoria (una contiene información sobre QUÉ se debe realizar, mientras que la otra contiene información sobre CUÁL dato se debe realizar). Sin embargo, esto no es una fórmula rígida, sino el caso más frecuente. En los microcontroladores que utilizan una arquitectura Harvard, el bus de la palabra de programa es más ancho que un byte, lo que permite que cada palabra de programa esté compuesto por una instrucción y un dato. En otras palabras, una localidad de memoria - una instrucción de programa.

Figura 12. **Arquitectura Harvard**



Fuente: MikroElectronika. *Libro de programación en mikrobasic*. http://www.mikroe.com/chapters/view/84/libro-de-la-programacion-de-los-microcontroladores-pic-en-basic-capitulo-1-mundo-de-los-microcontroladores/. Consulta: 18 de junio de 2014.

2.6. Juegos de instrucciones

Como ya se dijo un microcontrolador contiene un microprocesador, este a su vez necesita de entender las instrucciones que recibe, para esto cada microprocesador es diseñado para un juego de instrucciones de acuerdo al fabricante, existen tres tipos de juegos de instrucciones CISC, RISC, SISC.

2.6.1. CISC

Complex Instruction Set Computer, Computador con Conjunto de Instrucciones Complejas, es un modelo de arquitectura de computadores. Los microprocesadores CISC tienen un conjunto de instrucciones que se caracteriza por ser muy amplio, y permitir operaciones complejas entre operandos situados en la memoria o en los registros internos, en contraposición a la arquitectura RISC.

Este tipo de arquitectura dificulta el paralelismo entre instrucciones, por lo que, en la actualidad, la mayoría de los sistemas CISC de alto rendimiento implementan un sistema que convierte dichas instrucciones complejas en varias instrucciones simples del tipo RISC, llamadas generalmente microinstrucciones.

Los CISC pertenecen a la primera corriente de construcción de procesadores, antes del desarrollo de los RISC. Ejemplos de ellos son: Motorola 68000, *Zilog*Z80 y toda la familia Intelx86, AMDx86-64 usada en la mayoría de las computadoras personales actuales.

2.6.2. RISC

Reduced Instruction Set Computer, en español Computador con Conjunto de Instrucciones Reducidas, es un tipo de diseño de CPU generalmente utilizado en microprocesadores o microcontroladores con las siguientes características fundamentales:

- Instrucciones de tamaño fijo y presentado en un reducido número de formatos.
- Solo las instrucciones de carga y almacenamiento acceden a la memoria de datos.

Además estos procesadores suelen disponer de muchos registros de propósito general.

El objetivo de diseñar máquinas con esta arquitectura es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria. Las máquinas RISC protagonizan la tendencia actual de construcción de microprocesadores. PowerPC, DEC Alpha, MIPS, ARM, SPARC son ejemplos de algunos de ellos.

RISC es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeñas y simples que toman menor tiempo para ejecutarse. El tipo de procesador más comúnmente utilizado en equipos de escritorio, el x86, está basado en CISC en lugar de RISC, aunque las versiones más nuevas traducen instrucciones basadas en CISC x86 a instrucciones más simples basadas en RISC para uso interno antes de su ejecución.

La idea fue inspirada por el hecho de que muchas de las características que eran incluidas en los diseños tradicionales de CPU, para aumentar la velocidad, estaban siendo ignoradas por los programas que eran ejecutados en ellas. Además, la velocidad del procesador en relación con la memoria de la computadora que accedía era cada vez más alta. Esto conllevó la aparición de numerosas técnicas para reducir el procesamiento dentro del CPU, así como de reducir el número total de accesos a memoria.

2.6.3. SISC

SISC (*Simple Instruction Set Computing*) es un tipo de arquitectura de microprocesadores orientada al procesamiento de tareas en paralelo. Esto se implementa mediante el uso de la tecnología VLSI, que permite a múltiples dispositivos de bajo costo que se utilicen conjuntamente, para resolver un problema particular dividido en partes disjuntas. La arquitectura RISC es un subconjunto del SISC, centrada en la velocidad de procesamiento debido a un conjunto de instrucciones reducido.

Los microprocesadores SISC (o RISC) nunca han logrado amenazar el amplio dominio de los procesadores CISC en los ordenadores personales, debido a su popularidad y al aumento constante en la capacidad de procesamiento de los mismos. Por lo tanto, el uso de RISC y SISC sigue limitado a necesidades muy específicas de procesamiento, como en los procesadores DSP.

2.7. El *datasheet*

Un *datasheet* es un documento que resume el funcionamiento y otras características de un componente (por ejemplo, un componente electrónico) o

subsistema (por ejemplo, una fuente de alimentación) con el suficiente detalle para ser utilizado por un ingeniero de diseño y diseñar el componente en un sistema.

Comienza típicamente con una página introductoria que describe el resto del documento, seguido por los listados de componentes específicos, con la información adicional sobre la conectividad de los dispositivos. En caso de que haya código fuente relevante a incluir se une cerca del extremo del documento o se separa generalmente en otro archivo, por lo general un *datasheet* contiene la siguiente información:

- Datos del fabricante.
- Número y denominación.
- Lista de formatos con imágenes y códigos.
- Propiedades.
- Breve descripción funcional.
- Esquema de conexiones. Habitualmente es un anexo con indicaciones detalladas.
- Tensión de alimentación, consumo.
- Condiciones de operación recomendadas.
- Tabla de especificaciones, tanto en corriente continua como alterna.
- Esquemas.
- Medidas.
- Circuito de prueba.
- Información sobre normas de seguridad y uso.

Tabla I. **Características del pic 16f887**

MICROCHIP

PIC16F882/883/884/886/887

28/40/44-Pin Flash-Based, 8-Bit CMOS Microcontrollers with nanoWatt Technology

High-Performance RISC CPU:

- Only 35 instructions to learn:
  - All single-cycle instructions except branches
- Operating speed:
  - DC – 20 MHz oscillator/clock input
  - DC – 200 ns instruction cycle
- Interrupt capability
- 8-level deep hardware stack
- Direct, Indirect and Relative Addressing modes

Special Microcontroller Features:

- Precision Internal Oscillator:
  - Factory calibrated to ±1%
  - Software selectable frequency range of 8 MHz to 31 kHz
  - Software tunable
  - Two-Speed Start-up mode
  - Crystal fail detect for critical applications
  - Clock mode switching during operation for power savings
- Power-Saving Sleep mode
- Wide operating voltage range (2.0V-5.5V)
- Industrial and Extended Temperature range
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Brown-out Reset (BOR) with software control option
- Enhanced low-current Watchdog Timer (WDT) with on-chip oscillator (software selectable nominal 268 seconds with full prescaler) with software enable
- Multiplexed Master Clear with pull-up/input pin
- Programmable code protection
- High Endurance Flash/EEPROM cell:
  - 100,000 write Flash endurance
  - 1,000,000 write EEPROM endurance
  - Flash/Data EEPROM retention: > 40 years
- Program memory Read/Write during run time
- In-Circuit Debugger (on board)

Low-Power Features:

- Standby Current:
  - 50 nA @ 2.0V, typical
- Operating Current:
  - 11 µA @ 32 kHz, 2.0V, typical
  - 220 µA @ 4 MHz, 2.0V, typical
- Watchdog Timer Current:
  - 1 µA @ 2.0V, typical

Peripheral Features:

- 24/35 I/O pins with individual direction control:
  - High current source/sink for direct LED drive
  - Interrupt-on-Change pin
  - Individually programmable weak pull-ups
  - Ultra Low-Power Wake-up (ULPWU)
- Analog Comparator module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (CV_{REF}) module (% of V_{DD})
  - Fixed voltage reference (0.6V)
  - Comparator inputs and outputs externally accessible
  - SR Latch mode
  - External Timer1 Gate (count enable)
- A/D Converter:
  - 10-bit resolution and 11/14 channels
- Timer0: 8-bit timer/counter with 8-bit programmable prescaler
- Enhanced Timer1:
  - 16-bit timer/counter with prescaler
  - External Gate input mode
  - Dedicated low-power 32 kHz oscillator
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Enhanced Capture, Compare, PWM+ module:
  - 16-bit Capture, max. resolution 12.5 ns
  - Compare, max. resolution 200 ns
  - 10-bit PWM with 1, 2 or 4 output channels, programmable "dead time", max. frequency 20 kHz
  - PWM output steering control
- Capture, Compare, PWM module:
  - 16-bit Capture, max. resolution 12.5 ns
  - 16-bit Compare, max. resolution 200 ns
  - 10-bit PWM, max. frequency 20 kHz
- Enhanced USART module:
  - Supports RS-485, RS-232, and LIN 2.0
  - Auto-Baud Detect
  - Auto-Wake-Up on Start bit
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI (all 4 modes) and I^2C™ Master and Slave Modes with I^2C address mask

Fuente: Datasheet pic 16f887. Consulta: 20 de junio de 2014.

2.8. Laboratorio de microcontroladores

Para que se logre llevar una enseñanza adecuada es necesario contar con al menos ciertas características que ayuden a cumplir este objetivo, por esto a continuación se mencionan los requisitos mínimos con los que debe contar un laboratorio enfocado en la enseñanza de la programación de microcontroladores.

2.8.1. El inmueble

Se debe contar con un lugar que tenga el espacio necesario, según la cantidad de estudiantes por horario, este deberá estar bien iluminado y contar con buena ventilación, deberá contar con pizarrones, marcadores, cañoneras, una instalación eléctrica que esté en capacidad de proporcionar la alimentación necesaria para computadoras como equipos y circuitos de prueba. Además de esto deberá mantenerse limpio, evitando siempre que los componentes electrónicos entren en contacto con humedad y suciedad.

2.8.2. Fuente de alimentación

Puesto que en la electrónica se utilizan diferentes niveles de voltajes según la necesidad del diseñador, como la tecnología de fabricación de los componentes, es necesario que en el laboratorio de microcontroladores tenga a su disposición fuentes de alimentación de voltaje DC de valores constantes, como fuentes de voltajes DC variables, para utilizar los microcontroladores cabe mencionar que deberán ser alimentados con una fuente regulada del voltaje de operación, esto para evitar dañar los microcontroladores ante fluctuaciones de voltaje en la red eléctrica.

2.8.3. Generadores de señal

Ya que en la práctica, la utilización de microcontroladores implica manipulación de datos y comunicaciones, es necesario que un laboratorio de microcontroladores cuente con equipos generadores de señales con voltaje máximo variables como también en frecuencia.

2.8.4. Osciloscopio

Elosciloscopio es un instrumento de visualización electrónico para la representación gráfica de señales eléctricas que pueden variar en el tiempo.

En electrónica en general, si no se cuenta con un osciloscopio para analizar señales, prácticamente se está trabajando al azar, es por esto que es inevitable contar con el en un laboratorio de microcontroladores.

2.8.5. El *Protoboard*

Para realizar pruebas de manera temporal y de manera fácil, rápida y segura es necesario tener galletas o tarjetas de *protoboard* para colocar los componentes electrónicos y verificar su correcto funcionamiento, además servirá para localizar y corregir fallos dentro del diseño.

2.8.6. Tarjetas de prueba

Con respecto a las tarjetas de prueba, estas pueden ser tarjetas diseñadas por los estudiantes según las prácticas a realizar, o se pueden comprar las tarjetas entrenadoras que venden los diferentes fabricantes, como

ejemplo se mencionará la entrenadora EasyPic V7 utilizada para efectuar las pruebas de los programas realizados en el presente trabajo.

2.8.7. Analizador de espectros

Otra herramienta muy útil en un laboratorio de electrónica, como en la enseñanza de programación de microcontroladores, es un analizador de espectro. Este equipo de medición electrónica que permite visualizar en una pantalla, las componentes espectrales en un espectro de frecuencias de las señales presentes en la entrada, pudiendo ser esta cualquier tipo de ondas eléctricas, acústicas u ópticas.

3. FUNDAMENTOS DE PROGRAMACIÓN

3.1. Introducción a la programación

Sin duda alguna nada puede subsistir por sí solo, pues a pesar del gran avance tecnológico de la electrónica y particularmente de los procesadores que llevaron al mundo a la modernización, no sería posible si no existiera algo que le indicara que hacer o cómo reaccionar ante situaciones específicas, es aquí donde surge la necesidad de programar, es decir indicarle al microcontrolador que pines son de entrada, cuales son las salidas, que hacer si tal pin está en alto o en bajo, es por esto que en el presente capítulo se dará los principios para desarrollar programas en el caso más general, ya más adelante se estará enfocando exclusivamente a la programación de microcontroladores.

Un programa es un conjunto de instrucciones que controla o dirige a un microcontrolador; mas formalmente un programa es un conjunto de instrucciones internas utilizadas para ejecutarse en una computadora y que produzca un resultado concreto, otro término utilizado para un programa es software. El proceso de escribir un programa o software se llama programación y el conjunto de instrucciones utilizadas para realizar el programa se conoce como lenguaje de programación y por ende la persona que realiza programas es conocido como programador.

Los lenguajes de programación sirven para escribir programas que permitan la comunicación usuario/máquina. Unos programas especiales llamados traductores (compiladores o interpretes) convierten las instrucciones

escritas en lenguajes de programación en instrucciones escritas en lenguajes de máquina (0 y 1, bits) que esta pueda entender.

Los programas de utilidad facilitan el uso de la computadora, los programas que realizan tareas concretas, nominas, análisis estadísticos, entre otros, se denominan programas de aplicación.

Los lenguajes de los humanos y los lenguajes de la máquina son muy diferentes, ya que las características y posibilidades de las personas y de las máquinas son muy diferentes. Los lenguajes de computadoras permiten a las personas escribir en un lenguaje que sea más apropiado a las características humanas y se puedan traducir al lenguaje máquina de diferentes tipos de máquinas.

Los principales tipos de lenguajes utilizados en la actualidad son tres:

- Lenguajes máquinas
- Lenguaje de bajo nivel (ensamblador)
- Lenguajes de alto nivel

3.1.1. Lenguaje máquina

Los leguajes máquina son aquellos que están escritos en lenguajes directamente inteligibles por la máquina, ya que sus instrucciones son cadenas binarias que especifican una operación, y las posiciones de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina. El código máquina es el conocido código binario.

Las instrucciones en lenguaje máquina dependen del hardware en este caso del microcontrolador y del fabricante. En este caso no se va a programar en dicho lenguaje aunque es necesario que se tenga conocimiento de el.

3.1.2. Lenguaje de bajo nivel

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes de máquina, pero al igual de ellos depende de microcontrolador y del fabricante. El lenguaje de bajo nivel por excelencia es el ensamblador. Las instrucciones en lenguaje ensamblador son instrucciones conocidas como nemotécnicos. Por ejemplo, nemotécnicos típicos de operaciones aritméticas son: en inglés, ADD, SUB, DIV, entre otros, en español se tendrían, SUM, RES, DIV, entre otros.

3.1.3. Lenguajes de alto nivel

Los lenguajes de alto nivel son los más utilizados por los programadores y también es en el cual se va a trabajar. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes de máquina y ensambladores. Otra razón es que un programa escrito en lenguaje de alto nivel es independiente de la máquina en este caso del microcontrolador y del fabricante; esto es, las instrucciones del programa de la computadora no dependen del diseño del hardware o de una computadora en particular.

En consecuencia, los programas escritos en lenguaje de alto nivel son portables o transportables, lo que significa la posibilidad de poder ser ejecutado con poca o ninguna modificación en diferentes tipos de computadoras; al contrario que los lenguajes de máquina y de bajo nivel, que solo se pueden

ejecutar en un determinado tipo de computadora. Los lenguajes de alto nivel presentan las siguientes ventajas:

El tiempo de formación de los programadores es relativamente corto comparado con otros lenguajes. La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos.

- Las modificaciones y puestas a punto de los programas son más fáciles
- Reducción del coste de los programas
- Transportabilidad

Algunas desventajas de los lenguajes de alto nivel son:

- Incremento del tiempo de puesta a punto, al necesitarse diferentes traducciones del programa fuente para conseguir el programa definitivo.
- No se aprovechan los recursos internos de la máquina, que se explotan mucho mejor en lenguajes de máquina o ensambladores.
- Aumento de la ocupación de memoria.
- El tiempo de ejecución de los programas es mucho mayor.
- Al igual que sucede con los lenguajes ensambladores, los programas fuente tienen que ser traducidos por los programas traductores, llamados en este caso compiladores e intérpretes.

Entre lenguajes de alto nivel se pueden mencionar: java, c, c++, c#, basic. Los traductores de lenguaje son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina. Los traductores se dividen en: intérpretes y compiladores.

El intérprete es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta. Los compiladores, a diferencia de los intérpretes, transforman el programa a un programa equivalente en un código objeto (fase de compilación), y en un segundo paso generan los resultados a partir de los datos de entrada (fase de ejecución).

3.2. Conceptos importantes en la programación

Hasta este punto se ha considerado a grandes rasgos el avance de la electrónica, también se han considerado algunos principios de programación, pero ahora se pregunta por qué han surgido estos campos, pues resulta que lo primordial en este desarrollo es en encontrar o darle solución a problemas ya sea de la vida cotidiana, oficina, industria, entre otros.

Entonces resulta que el objetivo tanto de la programación como de la electrónica, en este caso específico de los microcontroladores, es tener un problema que resolver o dicho de otra manera, cubrir una necesidad del usuario valiéndose de herramientas de programación como de la electrónica.

La resolución de un problema en computación se hace escribiendo un problema, que exige al menos los siguientes pasos:

- Definición o análisis del problema
- Diseño de un algoritmo
- Transformación del algoritmo en un programa
- Ejecución y validación del programa

El proceso de resolución de un problema con una computadora conduce a la escritura de un programa y así ejecución en la misma. Aunque el proceso de

diseñar programas es un proceso creativo, se puede considerar una serie de fases o pasos comunes, que generalmente deben seguir todos los programadores.

Las fases de resolución de un problema son:

- Análisis del problema
- Diseño del algoritmo
- Codificación
- Compilación y ejecución
- Verificación
- Depuración
- Mantenimiento
- Documentación

Constituyen el ciclo de vida del programa y las fases o etapas usuales son:

- Análisis: el problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por la persona que encarga el programa.

- Diseño: una vez analizado el problema, se diseña una solución que conducirá a un algoritmo que resuelva el problema.

- Codificación (implementación): la solución se escribe en la sintaxis del lenguaje de alto nivel y se obtiene un programa.

- Compilación, ejecución y verificación: el programa se ejecuta, se comprueba rigurosamente y se eliminan todos los errores (denominados *bugs* en inglés) que puedan aparecer.

- Depuración y mantenimiento: el programa se actualiza y modifica cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.

- Documentación: escritura de las diferentes fases del ciclo de vida del programa, esencialmente el análisis, diseño y codificación, unidos a manuales de usuario y de referencia, así como normas de mantenimiento.

Las dos primeras fases conducen a un diseño detallado escrito en forma de algoritmo. Durante la tercera etapa se implementa el algoritmo en un código escrito en un lenguaje de programación, reflejando las ideas desarrolladas en las fases de análisis y diseño.

La fase de compilación y ejecución traduce y ejecuta el programa. En las fases de verificación y depuración el programador busca errores de las etapas anteriores y los elimina. Comprobará que mientras más tiempo se gaste en la fase de análisis y diseño menos se gastará en la depuración del programa, por último, se debe realizar la documentación del programa.

Se ha venido mencionando la palabra algoritmo, así que se va a considerar su concepto y significado. La palabra algoritmo se deriva de la traducción al latín de la palabra *Al-Juarismi,* nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y

ecuaciones en el siglo IX. Un algoritmo es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

- Preciso (indicar el orden de realización en cada paso)
- Definido (si se sigue dos veces, obtiene el mismo resultado cada vez)
- Finito (tiene fin; un número determinado de pasos)

Un algoritmo debe producir un resultado en un tiempo finito. Los métodos que utilizan algoritmos se denominan métodos algorítmicos, en oposición a los métodos que implican algún juicio o interpretación que se denomina métodos heurísticos. Los métodos algorítmicos se pueden implementar en computadoras, sin embargo, los procesos heurísticos no han sido convertidos fácilmente en las computadoras.

3.3. Análisis del problema

La primera fase de la resolución de un problema con computadora es el análisis del problema. Esta fase requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado solución deseada.

Para poder definir bien un problema es conveniente responder a las siguientes preguntas:

- ¿Qué entradas se requieren? (tipo y cantidad)
- ¿Cuál es la salida deseada? (tipo y cantidad)
- ¿Qué método produce la salida deseada?

3.4. Diseño del algoritmo

En la etapa de análisis del proceso de programación se determina que hace el programa, en la etapa de diseño se determina como hace el programa la tarea solicitada. Los métodos más eficaces para el proceso de diseño se basan en el conocido por divide y vencerás. Es decir, la resolución de un problema complejo se realiza dividiendo el problema en subproblemas y a continuación dividir estos subproblemas en otros de nivel más bajo, hasta que pueda ser implementada una solución en la computadora. Este método se conoce técnicamente como diseño descendente (*top-down*) o modular. El proceso de romper el problema en cada etapa y expresar cada paso en forma más detallada se denomina refinamiento sucesivo.

Cada subprograma es resuelto mediante un módulo que tiene un solo punto de entrada y un solo punto de salida. Cualquier programa bien diseñado consta de un programa principal que llama a subprogramas que a su vez puede llamar otros subprogramas. Los programas estructurados de esta forma se dice que tienen un diseño modular, y el método de romper el programa en módulos más pequeños se llama programación modular.

El proceso que convierte los resultados del análisis del problema en un diseño modular con refinamientos sucesivos, que permitan una posterior traducción a un lenguaje, se denominan diseño del algoritmo.

El diseño del algoritmo es independiente del lenguaje de programación en el que se vaya a codificar posteriormente.

3.4.1. Conceptos y características de algoritmos

Los pasos para la resolución de un problema son:

- Diseño del algoritmo, que describe la secuencia ordenada de pasos –sin ambigüedades- que conducen a la solución de un problema dado –análisis del problema y desarrollo del algoritmo-.
- Expresar el algoritmo como un programa en un lenguaje de programación adecuado –fase de codificación-.
- Ejecución y validación del programa por la computadora.

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa.

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta.

El diseño de la mayoría de algoritmos requiere creatividad y conocimientos profundos de la técnica de la programación. En esencia la solución de un problema se puede expresar mediante un algoritmo.

3.4.2. Características de un algoritmo

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.

- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.
- La definición de un algoritmo debe describir tres partes: entrada, proceso y salida.

Las ventajas más importantes del diseño descendente son:

- El problema se comprende más fácilmente al dividirse en partes más simples denominadas módulos.
- Las modificaciones en los módulos son más fáciles.
- La comprobación del problema se puede verificar fácilmente.

3.5. Instrucciones y tipos de instrucciones

El proceso de diseño del algoritmo o posteriormente de codificación del programa, consiste en definir las acciones o instrucciones que resolverán el problema.

Las acciones o instrucciones se deben escribir y posteriormente almacenar en memoria en el mismo orden en que han de ejecutarse, es decir, en secuencia.

Un programa puede ser lineal o no lineal. Un programa es lineal si las instrucciones se ejecutan secuencialmente, sin bifurcaciones, decisión ni comparaciones. Un programa es no lineal cuando se interrumpe la secuencia mediante instrucciones de bifurcación.

3.5.1. Tipos de instrucciones

Las instrucciones disponibles en un lenguaje de programación dependen del tipo de lenguaje. Por ello, en este apartado se estudián las instrucciones básicas que se pueden implementar de modo general en un algoritmo y que esencialmente soportan todos los lenguajes. La clasificación más usual es la siguiente:

- Instrucciones de inicio/fin.
- Instrucciones de asignación: son aquellas que se utilizan para asignar un valor a una variable. Por ejemplo: RB0 0.
- Instrucciones de lectura: estas instrucciones leen datos de un dispositivo de entrada. Por ejemplo: leer ADC0.
- Instrucciones de escritura: estas instrucciones escriben en un dispositivo de salida. Por ejemplo: escribir A;B;C.
- Instrucciones de bifurcación: el desarrollo lineal de un programa se interrumpe cuando se ejecuta una bifurcación. Las bifurcaciones pueden ser, según el punto del programa a donde se bifurca, hacia adelante o hacia atrás.

Las bifurcaciones en el flujo de un programa se realizaran de modo condicional, en función del resultado de la evaluación de la condición.

Bifurcación incondicional: la bifurcación se realiza siempre que el flujo del programa pase por las instrucciones, sin necesidad del cumplimiento de ninguna condición.

Bifurcación condicional: la bifurcación depende del cumplimiento de una determinada condición. Si se cumple la condición, el flujo sigue ejecutando la acción, si no se cumple, se ejecuta la acción alterna.

3.6. Herramientas de programación

Las dos herramientas más utilizadas comúnmente para diseñar algoritmos son: diagramas de flujo y pseudocódigos.

3.6.1. Diagramas de flujo

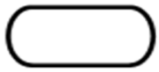
Un diagrama de flujo (*flowchart*) es una representación gráfica de un algoritmo. Los símbolos utilizados han sido normalizados por el Instituto Norteamericano de Normalización (ANSI).

A continuación se detallan algunos de los símbolos más utilizados en los diagramas de flujos.

3.6.1.1. Inicio o fin

- Dentro del símbolo se escribe la palabra inicio o la palabra fin, segun sea el caso.
- Sirve para marcar claramente el inicio y el fin del algoritmo.
- Todo diagrama de flujo debe de tener un inicio y un fin.

Figura 13. **Inicio o fin**



Fuente: elaboración propia.

3.6.1.2. Entrada y salida

- Dentro de este símbolo se escribe el verbo. Leer o el verbo Escribir según sea entrada o salida.
- En seguida del verbo se escriben una lista de variables separadas por comas.

Figura 14. **Entrada y salida**

Fuente: elaboración propia.

3.6.1.3. Proceso

Símbolo de proceso e indica la asignación de un valor en la memoria y/o la ejecución de una operación aritmética.

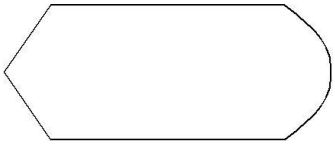
Figura 15. **Proceso**

Fuente: elaboración propia.

3.6.1.4. Salida en pantalla

Indica la salida en pantalla.

Figura 16. **Salida en pantalla**

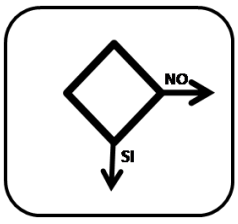


Fuente: elaboración propia.

3.6.1.5. Decisión

- Dentro del símbolo se escribe una expresión de comparación (llámese expresión lógica) cuyo resultado puede ser solamente verdadero (Sí) o falso (No).
- Las flechas indican la dirección del flujo del algoritmo.
- La decisión tiene una sola entrada y dos salidas.
- Si la expresión lógica es verdadera la salida es por el lado de Sí.
- Cuando la expresión lógica es falsa la salida es por el lado de No.

Figura 17. **Decisión**

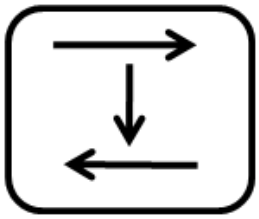


Fuente: elaboración propia.

3.6.1.6. Dirección de flujo

- Las flechas solo pueden ser verticales y horizontales.
- Indican la dirección del flujo, es decir, la secuencia de realización de las instrucciones.
- Todas las líneas deben estar conectadas a un símbolo.

Figura 18. **Flechas que indican la dirección del flujo**



Fuente: elaboración propia.

3.6.2. Pseudocódigo

El pseudocódigo es una herramienta de programación en la que las instrucciones se escriben en palabras similares en inglés o español, que facilitan tanto la escritura como la lectura de programas. En esencia, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Como ejemplo se ha realizado el pseudocódigo de la solución del problema de la suma de dos números:

Inicio
Número1←0

Número2←0
Leer (número1, número2)
Suma=número1+número2
Escribir (suma)
Fin

3.7. Codificación de un programa

Codificación es la escritura en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas precedentes. Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizado para su implementación, el código puede ser escrito con igual facilidad en un lenguaje o en otro.

Para realizar la conversión del algoritmo en programa se deben sustituir las palabras reservadas, y las operaciones/instrucciones indicadas en lenguaje natural, expresarlas en lenguaje de programación correspondiente.

3.8. Compilación y ejecución de un programa

Una vez que el algoritmo se ha convertido en un programa fuente, es preciso introducirlo en memoria mediante el teclado y almacenarlo posteriormente en un disco. Esta operación se realiza con un programa editor, posteriormente el programa fuente se convierte en un archivo de programa que se guarda en disco.

El programa fuente debe ser traducido a lenguaje máquina, este proceso se realiza con el compilador y el sistema operativo que se encarga prácticamente de la compilación.

Si tras la compilación se presentan errores (errores de compilación) en el programa fuente, es preciso volver a editar el programa, corregir los errores y compilar de nuevo.

3.9. Verificación y depuración de un programa

La verificación de un programa es el proceso de ejecución del programa con una amplia variedad de datos de entrada, llamados datos de *test* o prueba, que determinarán si el programa tiene errores (*bugs*). Para realizar la verificación se debe desarrollar una amplia gama de datos de *test*.

La depuración es el proceso de encontrar los errores del programa y corregir o eliminar dichos errores. Cuando se ejecuta un programa se pueden producir tres tipos de errores:

- Errores de compilación: se produce normalmente por un uso incorrecto de las reglas del lenguaje de programación y suelen ser errores de sintaxis. Si existe un error de sintaxis, la computadora no puede comprender la instrucción, no se obtendrá el programa objeto y el compilador imprimirá una lista de todos los errores encontrados durante la compilación.

- Errores de ejecución: estos errores se producen por instrucciones que la computadora puede comprender pero no ejecutar. Ejemplos típicos son: división por cero, en estos casos se detiene la ejecución del programa y se imprime un mensaje de error.

- Errores lógicos: se producen en la lógica del programa y la fuente del error suele ser el diseño del algoritmo. Estos errores son los más difíciles

de detectar, ya que el programa puede funcionar y no producir errores de compilación ni de ejecución, y solo puede advertir el error por la obtención de resultados incorrectos. En este caso se debe volver a la fase de diseño del algoritmo, modificar el algoritmo, cambiar el programa fuente y compilar y ejecutar una vez más.

3.10. Documentación y mantenimiento

La documentación de un problema consta de las descripciones de los pasos a dar en el proceso de resolución de un problema. La importancia de la documentación debe ser destacada por su decisiva influencia en el producto final. Programas pobremente documentados son difíciles de leer, más difíciles de depurar y casi imposibles de mantener y modificar.

La documentación de un programa puede ser interna y externa. La documentación interna es la contenida en líneas de comentarios que acompañan a las líneas de programación. La documentación externa incluye análisis, diagrama de flujo y/o pseudocódigos, manuales de usuario con instrucciones para ejecutar el programa y para interpretar los resultados, que deberán acompañar al programa ya finalizado como apoyo al usuario, para que se facilite su uso y como soporte ante cualquier problema que se le presente, además deberá ayudar para solicitar alguna modificación en el programa.

La documentación es vital cuando se desea corregir posibles errores futuros o bien cambiar el programa. Tales cambios se denominan mantenimiento del programa. Después de cada cambio la documentación debe ser actualizada para facilitar cambios posteriores. Es práctica frecuente numerar las sucesivas versiones de los programas 1.0, 1.1, 2.0, 2.1, entre otros (si los

cambios introducidos son importantes se varía el primer dígito [1.0, 2.0], en caso de pequeños cambios solo se varía el segundo dígito [2.0, 2.1]).

3.11. Datos y tipos de datos

En general una computadora lo que hace es manipular datos, estos datos, pueden ser las cifras de una encuesta, las estadísticas de algún evento, entre otros. En si un dato es la expresión general que describe los objetos con los cuales opera una computadora. La mayoría de las computadoras pueden trabajar con varios tipos de datos. Los algoritmos y los programas correspondientes operan sobre estos tipos de datos.

Los distintos tipos de datos se representan en diferentes formas en la computadora. A nivel de máquina, un dato es un conjunto o secuencia de bits. Los lenguajes de alto nivel permiten basarse en abstracciones e ignorar los detalles de la representación interna. Aparece el concepto de tipo de datos, así como su representación. Los tipos de datos simpes son los siguientes:

- Numéricos (*integer*, real)
- Lógicos (*boolean*)
- Caracter (*char*, *string*)

Existen algunos lenguajes de programación que admiten otros tipos de datos, como por ejemplo los complejos

3.11.1. Datos numéricos

El tipo numérico es el conjunto de los valores numéricos. Estos pueden representarse en dos formas distintas:

- Tipo numérico entero (*integer*)
- Tipo numérico real (real)

3.11.1.1. Enteros

El tipo: el tipo entero es un subconjunto finito de los números enteros. Los enteros son números completos, no tienen componentes fraccionarios o decimales y pueden ser negativos o positivos, ejemplo: 5, -5, 10, -10, 1 500, -1 500

3.11.1.2. Reales

El tipo real consiste en un subconjunto de los números reales. Los números reales siempre tienen un decimal y pueden ser positivos o negativos. Un número real consta de un entero y una parte decimal. Ejemplo: 0,01, -0,01, 10,5, -10,5.

3.11.2. Datos lógicos (booleanos)

También denominado booleano es aquel dato que solo puede tomar uno de dos valores. Este tipo de datos se utiliza para representar las alternativas (si/no) a determinadas condiciones, los valores que toma son los siguientes:

- Cierto o verdadero (*true*)
- Falso (*false*)

3.11.3. Datos tipo carácter y tipo cadena

El tipo caracter es el conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato tipo caracter contiene un solo caracter. Los caracteres que reconocen las diferentes computadoras no son estándar: sin embargo la mayoría reconoce los siguientes caracteres alfabéticos y numéricos:

- Caracteres alfabéticos
- Caracteres numéricos
- Caracteres especiales

El tipo cadena (*string*) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el tipo de lenguaje de programación. La longitud de una cadena de caracteres es el número de ellos comprendidos entre los separadores o limitadores:

"Hola mundo"

3.12. Constantes y variables

Los programas tienen ciertos valores que no deben cambiar durante la ejecución del programa, tales valores se llaman constantes. De igual forma, existen otros valores que cambian durante la ejecución del programa; a estos se les llama variables.

Una constante es una partida de datos que permanecen sin cambios durante todo el desarrollo del algoritmo o durante la ejecución del programa.

La variable es una partida de datos, cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa.

3.13. Programación modular

La programación modular es uno de los métodos de diseño más flexibles y potentes para mejorar la productividad de un programa. En programación modular el programa se divide en módulos, partes independientes, cada una de las cuales ejecuta una única actividad o tarea y se codifican independientemente de otros módulos. Cada uno de estos módulos se analizan, codifican y ponen a punto por separado.

Cada programa contiene un módulo denominado programa principal, que controla todo lo que sucede; se transfiere el control a submódulos, de modo que ellos puedan ejecutar sus funciones, sin embargo, cada submódulo devuelve el control al módulo principal cuando haya completado su tarea.

Dado que los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en diferentes partes del mismo programa. Esto reducirá el tiempo del diseño del algoritmo y posterior codificación del programa. Además, un módulo se puede modificar radicalmente sin afectar otros módulos, incluso sin alterar su función principal.

La descomposición de un programa en módulos independientes más simples se conoce también como el método de divide y vencerás. Se diseña cada módulo con independencia de los demás, y siguiendo un método ascendente o descendente se llegará hasta la descomposición final del problema en módulos en forma jerárquica.

3.14. Programación estructurada

Los términos programación modular, programación descendente y programación estructurada, se introdujeron en la segunda mitad de la década de los setenta y a menudo sus términos se utilizan como sinónimos, aunque no significan lo mismo. La programación modular y descendente ya se han examinado anteriormente. La programación estructurada significa escribir un programa de acuerdo a las siguientes reglas:

- El programa tiene un diseño modular.
- Los módulos son diseñados de modo descendente.
- Cada módulo se codifica utilizando las tres estructuras de control básicas: secuencia, selección y repetición.

El termino programación estructurada se refiere a un conjunto de técnicas que han sido evolucionando desde los primeros trabajos de Edgar Dijkstra. Estas técnicas aumentan considerablemente la productividad del programa reduciendo en elevado grado el tiempo requerido para escribir, verificar, depurar y mantener los programas.

La programación estructurada utiliza un número limitado de estructuras de control que minimizan la complejidad de los programas y por consiguiente, reducen los errores, hace los programas más fáciles de escribir, verificar, leer y mantener. Los programas deben estar dotados de una estructura e incorpora el siguiente conjunto de técnicas:

- Recursos abstractos
- Diseño descendente *top-down*
- Estructuras básicas

3.14.1. Recursos abstractos

La programación estructurada se auxilia de los recursos abstractos, en lugar de los recursos concretos de que dispone un determinado lenguaje de programación.

Descomponer un programa en términos de recursos abstractos consiste en descomponer una determinada acción compleja, en términos de un número de acciones más simples capaces de ejecutarlas o que constituyan instrucciones de computadoras disponibles.

3.14.2. Diseño descendente

El diseño descendente es el proceso mediante el cual un problema se descompone en una serie de niveles o pasos sucesivos de refinamiento. La metodología descendente consiste en efectuar una relación entre las sucesivas etapas de estructuración, de modo que se relacionan unas con otras mediante entradas y salidas de información. Es decir, se descompone el problema en etapas o estructuras jerárquicas, de forma que se puede considerar cada estructura desde dos puntos de vista: qué hace y cómo lo hace.

3.15. Estructuras de control

Las estructuras de control de un lenguaje de programación son métodos de especificar el orden en que las instrucciones de un algoritmo se ejecutarán. El orden de ejecución de las sentencias o instrucciones determina el flujo de control. Estas estructuras de control, son, por consiguiente, fundamentales en los lenguajes de programación y en los diseños de algoritmos, especialmente los pseudocódigos. Las tres estructuras de control básico son:

- Secuencia
- Selección
- Repetición

3.15.1. Estructuras secuencial

Una estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el final del proceso. La estructura secuencial tiene una entrada y una salida.

3.15.2. Estructuras selectivas

La especificación formal de algoritmos tiene realmente utilidad, cuando el algoritmo requiere una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existen un número de posibles alternativas resultantes de la evaluación de una determinada condición. Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también estructuras de decisión o alternativas.

En las estructuras selectivas se evalúa una condición y en función del resultado de la misma se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabras en pseudocódigo (*if, then, else* con su correspondiente en español si, entonces, si no), no con una configuración geométrica en forma de rombo o bien con un triángulo en el interior de una caja rectangular. Las estructuras selectivas o alternativas pueden ser:

- Simples

- Dobles
- Múltiples

3.15.2.1. Alternativa simple (si-entonces/ *if-then*)

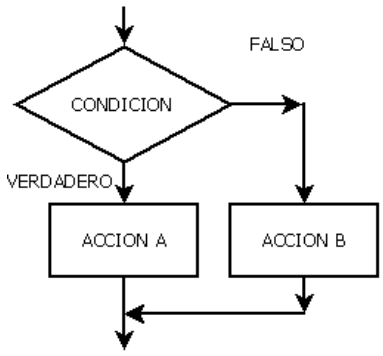
La estructura alternativa simple si-entonces ejecuta una determinada acción cuando se cumple una determinada condición. La selección si-entones evalúa la condición y:

- Si la condición es verdadera, entonces ejecuta la acción indicada
- Si la condición es falsa, entonces no hace nada

3.15.2.2. Alternativa doble (si-entonces-si_no/ *if-then-else*)

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición.

Figura 19. **Diagrama de flujo IF_THEN_ELSE**



Fuente: elaboración propia.

- Si la condición es verdadera se ejecuta la acción indicada
- Si la condición es falsa se ejecuta una acción determinada

3.15.2.3. Alternativa múltiple (según_sea, caso_de/ *case*)

Con frecuencia es necesario que existan más de dos elecciones posibles. Este problema, como se verá más adelante, se podría resolver por estructuras alternativas simples o dobles, anidadas o en cascada; sin embargo, este método si el número de alternativas es grande puede plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad.

La estructura de decisión múltiple evaluará una expresión que podrá tomar n valores distintos, según que elija uno de estos valores en la condición, se realizará una de las n acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los n posibles.

3.15.2.4. Estructuras de decisión anidadas

Las estructuras de selección si-entonces y si-entonces-si_no implican la selección de una de dos alternativas. Es posible también utilizar la instrucción si para diseñar estructuras de selección que contengan más de dos alternativas. Por ejemplo, una estructura si-entonces puede contener otra estructura si-entonces, y así sucesivamente cualquier número de veces; a su vez, dentro de cada estructura puede existir diferentes acciones.

3.15.2.5. Sentencia ir-a (*go to*)

El flujo de control de un algoritmo es siempre secuencial, excepto cuando

las estructuras de control estudiadas anteriormente realizan transferencias de control no secuenciales. La programación estructurada permite realizar programas fáciles y legibles utilizando las tres estructuras ya conocidas: secuenciales, selectivas y repetitivas. Sin embargo, en ocasiones es necesario realizar bifurcaciones incondicionales; para ello se recurre a la instrucción ir-a (*go to*), esta instrucción siempre ha causado polémica entre los programadores ya que algunos consideran que si un programa está bien desarrollando no habría necesidad de utilizar esta sentencia, por lo cual consideran que es una sentencia que no debería existir.

3.15.3. Estructuras repetitivas

Las computadoras están especialmente diseñadas para todas aquellas aplicaciones en las cuales una operación o conjunto de ellas, deben repetirse muchas veces. Un tipo muy importante de estructura es el algoritmo necesario para repetir una o varias acciones un número determinado de veces. Un programa que lee una lista de números puede repetir la misma secuencia de mensajes al usuario e instrucciones de lectura hasta que todos los números de un fichero se lean.

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces, se denominan bucles, y se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones.

3.15.3.1. Estructura mientras (*while*)

La estructura repetitiva mientras (*while*) es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición. Cuando se ejecuta la instrucción mientras, la primera cosa que sucede es que se evalúa la

condición, si se evalúa falsa, no se toma ninguna acción y el programa prosigue en la siguiente instrucción del bucle, si la expresión es verdadera, entonces se ejecuta el cuerpo del bucle, después de lo cual se evalúa de nuevo la expresión booleana. Este proceso se repite una y otra vez mientras la expresión booleana sea verdadera.

3.15.3.2. Estructura hacer-mientras (*do while*)

El bucle mientras al igual que el bucle desde, evalúa la expresión al comienzo del bucle de repetición; siempre que se utilizan para crear bucle *pre-test*. Los bucles *pre-test* se denominan también bucles controlados por la entrada. En ocasiones se necesita que el conjunto de sentencias que componen el cuerpo del bucle, se ejecuten al menos una vez, sea cual sea el valor de la expresión o condición de evaluación. Estos bucles se denominan bucles *post-test* o bucles controlados por la salida.

Este es el caso del bucle hacer-mientras (*do-while*), es análogo al bucle mientras y el cuerpo del bucle se ejecuta una y otra vez mientras la condición es verdadera. Existe, sin embargo, una gran diferencia y es que el cuerpo del bucle está encerrado entre las palabras reservadas, hacer y mientras; de modo que las sentencias de dicho cuerpo se ejecutan, al menos una vez, antes de que se evalúe la expresión booleana. En otras palabras, el cuerpo del bucle siempre se ejecuta, al menos una vez, incluso aunque la expresión booleana sea falsa.

3.15.3.3. Estructura repetir (*repeat*)

Existen situaciones en las que se desea que un bucle se ejecute al menos una vez antes de comprobar la condición de repetición. En la estructura mientras, si el valor de la expresión booleana es inicialmente falso, el cuerpo del

bucle no se ejecutará; por ello, se necesitan otros tipos de estructuras repetitivas.

La estructura repetir (*repeat*) se ejecuta hasta que se cumpla una condición determinada que se comprueba al final del bucle.

El bucle repetir-hasta_que se repite mientras el valor de la expresión booleana de la condición sea falsa, justo la opuesta de la sentencia mientras.

3.15.3.4. Estructura desde/para (*for*)

En ocasiones se conoce de antemano el número de veces que se desean ejecutar las acciones de un bucle. En estos casos, en el que el número de iteraciones es fijo, se debe usar la estructura desde o para (*for*). La estructura desde, ejecuta las acciones del cuerpo del bucle un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle.

3.15.3.5. Sentencia interrumpir (*break*)

En ocasiones, los programadores desean terminar un bucle en un lugar determinado del cuerpo del bucle, en vez de esperar que el bucle termine de modo natural por su entrada o por su salida. Un método de conseguir esta acción es mediante la sentencia interrumpir (*break*) que se suele utilizar en la sentencia según_sea (*switch*).

La sentencia interrumpir se puede utilizar para terminar una sentencia de iteración y cuando se ejecuta, produce que el flujo de control salte fuera a la siguiente sentencia inmediatamente a continuación de la sentencia de iteración.

La sentencia interrumpir se puede colocar en el interior del cuerpo del bucle para implementar este efecto.

3.15.3.6. Sentencia continuar (*continue*)

La sentencia continuar (*continue*) hace que el flujo de ejecución salte el resto de un cuerpo del bucle para continuar con el siguiente bucle o iteración. Esta característica suele ser útil en algunas circunstancias.

4. PROPUESTA DE PRÁCTICAS SOBRE PROGRAMACIÓN DE MICROCONTROLADORES

Después de haber observado principios básicos sobre programación en general, el tema principal es la programación de microcontroladores, para esto es necesario que el lector tenga conocimientos sobre los temas de electrónica que se desarrollarán, puesto que aquí se darán ejemplos de aplicaciones en temas de electrónica, se dará por sentado que ya se saben, por lo tanto las explicaciones serán las más básicas, teniendo mayor relevancia lo relacionado con la aplicación del microcontrolador en dichos temas, por ejemplo el lector deberá tener conocimientos sobre algebra booleana, audio, adc, protocolos de comunicación, entre otros.

4.1. Lenguajes para programación de microcontroladores

Como se ha considerado en el capítulo anterior, existen lenguajes de programación de alto nivel, lenguaje ensamblador y lenguaje máquina, en este trabajo se tratarán únicamente lenguajes de alto nivel, dejando al lector la libertad de investigar sobre los otros dos tipos de lenguajes si es de su interés.

En la actualidad existen varios software que se utilizan para realizar programas para microcontroladores, entre estos se mencionarán a MPLAB, Mikroelektronika, Pic Simulator IDE. MPLAB está enfocado en la programación en C, Mikroelektronika tiene diferentes productos por ejemplo Mikropascal que se enfoca en la programación en pascal, Mikrobasic en la programación en basic, ademas Pic Simulator IDE programación en Basic.

Aquí se tratarán sobre Mikrobasic, por ser uno de los utilizados en el medio. Se verán ejemplos dando una breve explicación sobre la electrónica del mismo, el algoritmo que se utilizará para resolver el problema, aplicación de lo visto en el capítulo anterior sobre programación estructurada, el diagrama de flujo correspondiente y finalmente la solución del problema utilizando los dos lenguajes de programación mencionados, resaltando la sintaxis de cada uno, a manera que el lector pueda tener la idea clara sobre la diferencia de algoritmo y lenguaje de programación utilizado para realizar la solución real del problema planteado.

4.2. Ejemplos de programación de microcontroladores

A continuación se presentan tres programas sencillos, su resolución en diagrama de flujo y posterior traslado al lenguaje de programación de mikrobasic.

4.2.1. Utilización de LCD

Práctica sobre la utilización de la LCD: en esta práctica se utilizará la LCD, que es muy importante, ya que proporciona una comunicación visible con el entorno del microcontrolador y el usuario, la práctica está enfocada en la utilización de la sentencia *if_then, if_then_else, for, while, if* anidados, se verá la forma de declarar variables, desplegar mensajes en la pantalla LCD.

En si el programa inicia, dando un mensaje de bienvenida en la lcd, este mensaje está contenido en la subrutina iniciar_lcd, lo cual también permite utilizar de manera fácil el lema de divide y vencerás, aunque en este caso no es trascendental, se utilizará a manera de ilustrar su utilización.

La lcd es una pantalla de cristal líquido que sirve de comunicación entre un usuario y el microcontrolador, por lo general las más utilizada en el medio es la de 2x16 con iluminación led incluida, esto quiere decir que tiene dos líneas de programación, y que cada línea tiene 16 espacios o casillas en la cual por lo general, se puede escribir letras y números, aunque también se puede generar símbolos según el usuario lo crea necesario.

La lcd de 2x16, cuenta con 16 pines, como se ve en la tabla II, los pines vienen configurados de la siguiente manera:

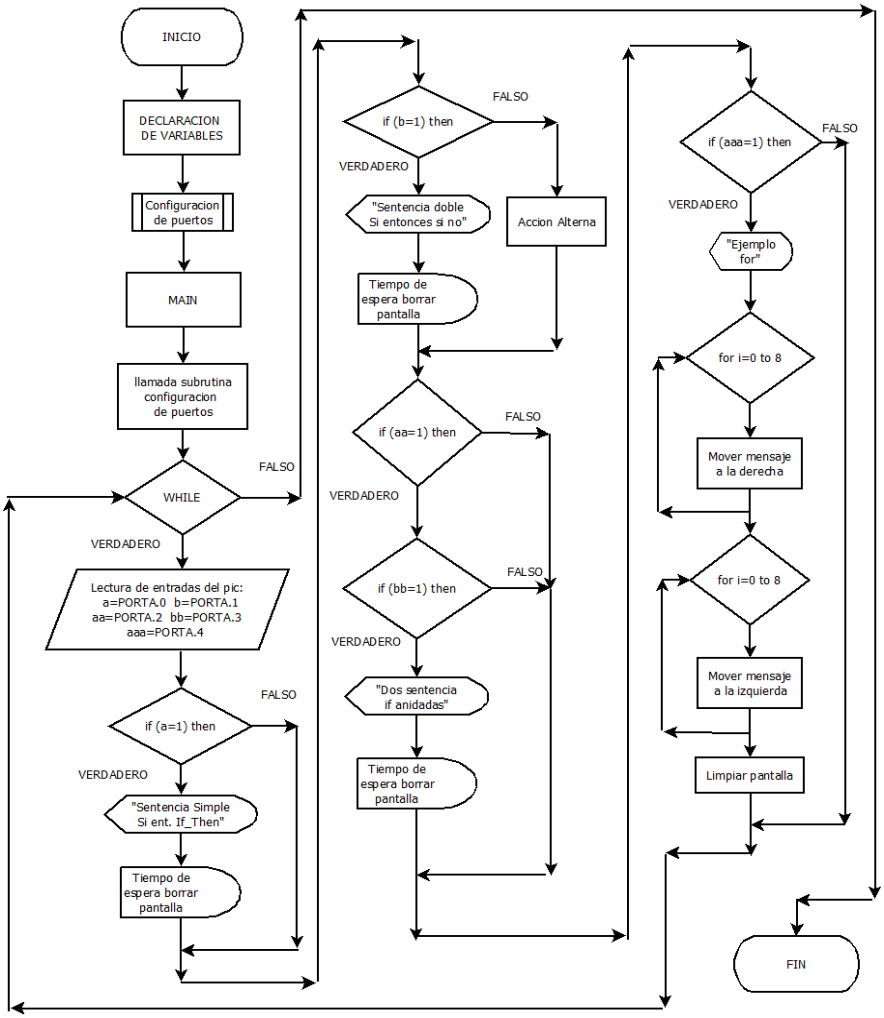
Tabla II. **lcd de 2x16**

Núm. de pin	Símbolo	Conexión externa	Función
1	Vss	Fuente de alimentación	Polaridad – de la fuente
2	Vdd	Fuente de alimentación	Polaridad + de la fuente
3	Vo	Potenciómetro	Segun el valor de tensión, asi sera el contraste de la lcd
4	RS	A pin del PIC	Selecciona el registro de la lcd, 0 instrucciones y 1 de datos
5	R/W	A pin del PIC	Selecciona si se escribe o lee lo que contiene la lcd
6	E	A pin del PIC	Habilita la lcd para escribir o leer
7-10	DB0-DB3	A pines del PIC	Linea de bits datos bajos, generalmene se trabaja solo 4 bits por lo cual estos pines se colocan en 0 lógico
11-14	DB4-DB7	A pines del PIC	Linea de bits de datos alta, estos generalmente van a los pines del pic para transferir información.
15	LED+	Alimentación + del led	Alimentación de led positivo
16	LED-	Alimentación – del led	Alimentación de led negativo

Fuente: elaboración propia.

Los pines de la lcd van conectados según la tabla anterior con la salvedad que los pines de datos, RS, R/W, E van conectados a los pines del pic que sean asignados mediante la programación, por lo general se utilizan transmisión de datos de 4 bits con lo cual se utiliza el puerto B de los pines RB0 ha RB5, en el programa que se ve enseguida RB0-RB3 van con los pines DB4-DB7 en orden, el E (*enable*) va al pin RB5, RS a RB4, como solo se va a escribir en la lcd y no se va a leer nada de ella R/W se llevará a un estado 0 lógico.

Figura 20. **Diagrama de flujo del programa**



Fuente: elaboración propia

```
program lcd

' SECCIÓN DE DECLARACIÓN DE VARIABLES

' CONFIGURACIÓN DE PINES PARA UTILIZACIÓN DE LCD

dimLCD_RS as sbit at RB4_bit
LCD_EN as sbit at RB5_bit
LCD_D4 as sbit at RB0_bit
LCD_D5 as sbit at RB1_bit
LCD_D6 as sbit at RB2_bit
LCD_D7 as sbit at RB3_bit

LCD_RS_Direction as sbit at TRISB4_bit
LCD_EN_Direction as sbit at TRISB5_bit
LCD_D4_Direction as sbit at TRISB0_bit
LCD_D5_Direction as sbit at TRISB1_bit
LCD_D6_Direction as sbit at TRISB2_bit
LCD_D7_Direction as sbit at TRISB3_bit

' DECLARACIÓN DE VARIABLES A UTILIZAR EN EL PROGRAMA

dimtexto1 as char[16]
dimtexto2 as char[16]
dim a, b, aa, bb, aaa as bit
dimi, bbb as byte
dimtiempo as word

'SUB-RUTINA
```

```
subprocedureIniciar_lcd()                ' Función para iniciar lcd
Lcd_Init()                               ' Inicialización de Lcd
Lcd_Cmd(_LCD_CLEAR)                      ' Limpiar la pantalla
Lcd_Cmd(_LCD_CURSOR_OFF)                 ' apaga el cursor
Lcd_Out(1,1,"Bienvenido a lcd")          ' Mensaje en línea 1
Lcd_Out(2,1,"utilizacion de")            ' Mensaje en línea 2
Delay_ms(2000)
Lcd_Cmd(_LCD_CLEAR)                      ' Limpiar la pantalla
Lcd_Out(1,1,"if_then_else for")          ' Mensaje en línea 1
Lcd_Out(2,1,"while e if_then.")          ' Mensaje en línea 2
Delay_ms(2000)
Lcd_Cmd(_LCD_CLEAR)                      ' Limpiar la pantalla

Lcd_Out(1,1,"configuracion de")          ' Mensaje en línea 1
Lcd_Out(2,1,"puerto,variables")          ' Mensaje en línea 2
Delay_ms(2000)
Lcd_Cmd(_LCD_CLEAR)                      ' Limpiar la pantalla

end sub

                'MAIN INDICA EL PROGRAMA PRINCIPAL
        'DESDE MAIN SE LLAMAN A LAS SUBRUTINAS
                'AQUI ES DONDE SE DESARROLLA EL
                'ALGORITMO PLANTEADO

main:

                '  CONFIGURACIÓN DE PUERTOS DEL PIC
```

```
TRISA  = 0xFF' PORTA es puerto de entrada 0xFF=255
TRISB  = 0                ' PORTB es puerto de salida
TRISC  = 0                ' PORTC es puerto de salida
TRISD  = 0                ' PORTD es puerto de salida
TRISE  = 0                ' PORTE es puerto de salida

ANSEL  = 0               ' Configura los pines del puerto A como digitales I/O
ANSELH = 0

  PORTA = 0
PORTB = 0                  ' coloca un 0 lógico en PORTB
PORTC = 0                  ' coloca un 0 lógico en PORTB
PORTD = 0                  ' coloca un 0 lógico en PORTB
  PORTE = 0                ' coloca un 0 lógico en PORTB
tiempo=400
Iniciar_lcd()

while (true)
  a=PORTA.0  'asignando a la variable 'a' el valor del pin RA0
  b=PORTA.1  'asignando a la variable 'b' el valor del pin RA1
aa=PORTA.2  'asignando a la variable 'aa' el valor del pin RA2
bb=PORTA.3  'asignando a la variable 'bb' el valor del pin RA3
aaa=PORTA.4          'asignando a la variable 'aaa' el valor del pin RA4

Lcd_Out(1,1,"¡¡¡HOLA MUNDO!!!")          ' Mensaje en línea 1
Lcd_Out(2,1,"...BIENVENIDOS..")          ' Mensaje en línea 2

if (a=1) then
Lcd_Cmd(_LCD_CLEAR)                              ' Limpiar la pantalla
```

```
Lcd_Out(1,1,"Sentencia simple")                ' Escribir texto en línea 1
Lcd_Out(2,1,"Si_Ent. If_Then")                 ' Escribir texto en línea 2
Vdelay_ms(tiempo)
Lcd_Cmd(_LCD_CLEAR)                            ' Limpiar la pantalla
end if

if (b=1) then
Lcd_Cmd(_LCD_CLEAR)                            ' Limpiar la pantalla
Lcd_Out(1,1,"Sentencia doble.")                ' Escribir texto en línea 1
Lcd_Out(2,1,"Si_Entonces_Sino")                ' Escribir texto en línea 2
Vdelay_ms(tiempo)
Lcd_Cmd(_LCD_CLEAR)                             ' Limpiar la pantalla
else
        'no se realiza ninguna acción
end if

if (aa=1) then
if (bb=1) then
Lcd_Cmd(_LCD_CLEAR)                            ' Limpiar la pantalla
Lcd_Out(1,1,"Senten. anidada")                 ' Escribir texto en línea 1
Lcd_Out(2,1,"dosif anidados.")                 ' Escribir texto en línea 2
Vdelay_ms(tiempo)
Lcd_Cmd(_LCD_CLEAR)                            ' Limpiar la pantalla
end if
end if

if (aaa=1) then
Lcd_Cmd(_LCD_CLEAR)                                ' Limpiar la pantalla
Lcd_Out(1,1,"ejemplo")                         ' Escribir texto en línea 1
```

```
Lcd_Out(2,1,"  for")                          ' Escribir texto en línea 2
Vdelay_ms(tiempo)

for i=0 to 8                                  ' Mover texto 9 espacios a la derecha
Lcd_Cmd(_LCD_SHIFT_RIGHT)
Vdelay_ms(tiempo)
bbb=i
nexti

ifbbb=8 then
fori=0 to 8                                   ' Mover texto 9 espacios a la izquierda
Lcd_Cmd(_LCD_SHIFT_LEFT)
Vdelay_ms(tiempo)
bbb=bbb-1
nexti
end if

Lcd_Cmd(_LCD_CLEAR)                 ' Limpiar la pantalla

endif

wend
end.
```

4.2.2. Programa utilizando lcd, adc, pwm para controlar un motor dc de baja potencia

A continuación se presenta un programa que regula la velocidad de un motor DC, de baja potencia utilizando para esto las librerías de mikrobasic, de pwm, lcd y adc.

La lcd como fue visto en el ejemplo anterior, sirve como medio de comunicación visual entre el entorno del pic y el usuario en este caso, se verá en la lcd el porcentaje de trabajo de la modulación pwm, que equivale al porcentaje de trabajo del motor, la cual se asociará a la velocidad de giro del motor, también es importante explicar que debido a que el motor necesita vencer la inercia del estado de reposo y la carga que pueda tener, este porcentaje no indica directamente la velocidad del motor, ya que algunos necesitan un porcentaje para empezar a girar, como por ejemplo un 30 % de trabajo para empezar a girar, eso si el 100 % de trabajo obtendremos el 100 % de la velocidad del motor.

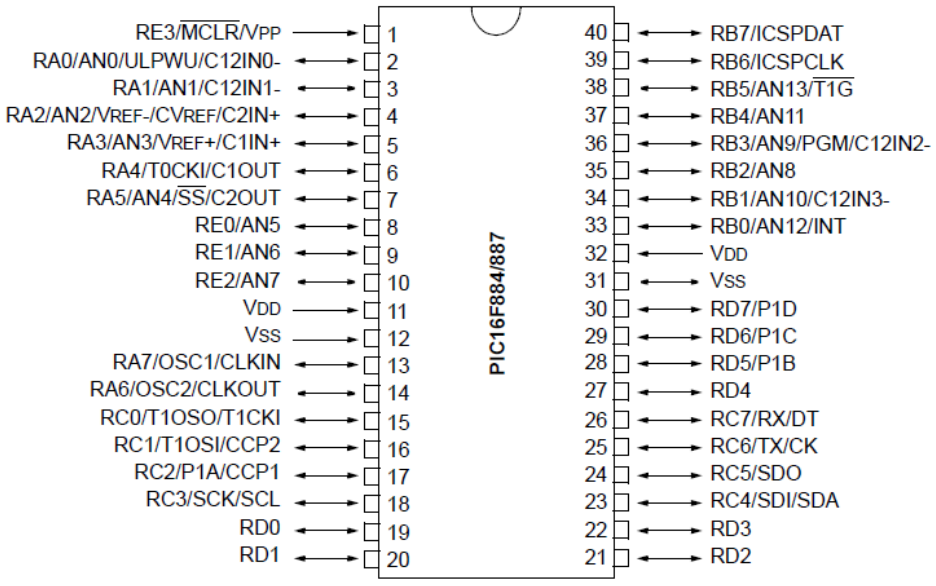
El adc, es un librería de mikrobasic que permite obtener un valor análogo en un pin de entrada y a partir de este obtener una representación binaria del valor, por ejemplo el pic 16f887 incluye 16 pines o canales de entrada analógica de 10 bits, lo cual se interpreta que según el valor de lectura del adc este será representado por 10 bits, siendo el valor mínimo de lectura 0 voltios en la entrada equivalente a 0 y el valor máximo de lectura, por ejemplo 5 voltios igual a 1023 al ser este la cantidad máxima que se puede expresar con 10 bits.

Cabe mencionar que el pic 16f887 tiene dos pines de entrada que puede ser configurados para obtener un valor de referencia mínima y máxima, para

los valores de lectura de los canales analógicos, por ejemplo se menciona que el valor de lectura del adc puede ser de 0 voltios a 5 voltios, esto puede ser modificado por ejemplo de -2,5 voltios a 2,5 voltios, al configurar los pines nombrados como VREF- y VREF+, que están en el pin 4 y 5 del pic.

En este ejemplo se utilizarán los valores de 0 a 5 voltios, 0 a 1 023, este valor se divide entre 4 para obtener una escala de 0 a 255 que equivale a tener 8 bits como representación, esto se hace porque el pwm es una aplicación que funciona a 8 bits, y la variable que lo controla en el programa se obtiene de la lectura del adc de 10 bits se estariá desbordando 4 veces el pwm, luego se divide dentro de 10 para poder obtener la escala de 0 a 100 que se representa en la lcd, como porcentaje, también cabe mencionar que se utiliza la función *WordtoStr*(temp_res,porcentaje) para convertir la variable temp_res de Word a porcentaje en *string* esto para poder visualizarlo en la lcd.

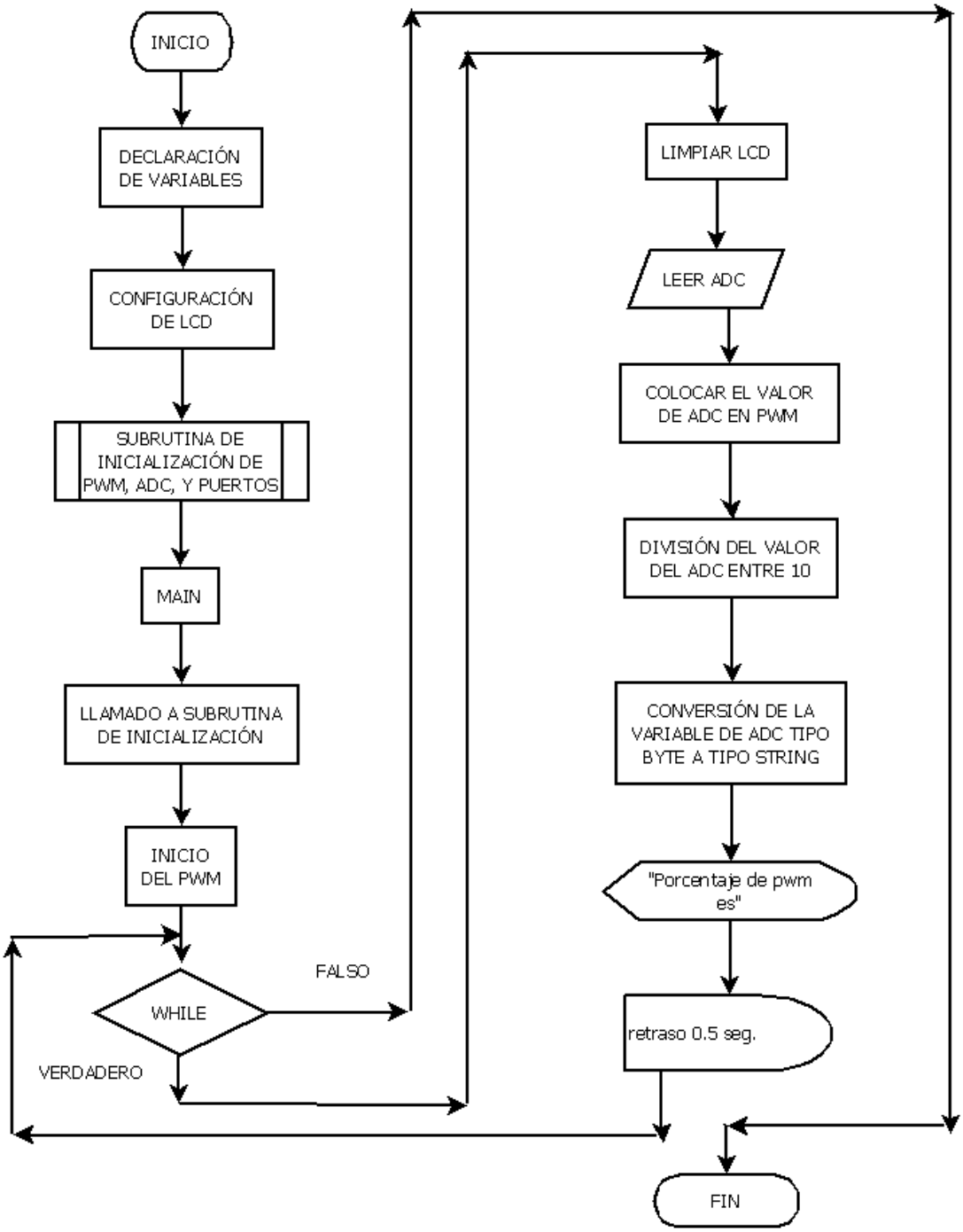
Figura 21. **Identificación de pines del pic 16f887**



.

Fuente: datasheet de pic 16f887.

Figura 22. **Diagrama de flujo ADC y PWM**



Fuente: elaboración propia.

```
programMyProject

dimtemp_res as word

' Conexiones de la LCD
    DimLCD_RS as sbit at RB4_bit
    LCD_EN as sbit at RB5_bit
    LCD_D4 as sbit at RB0_bit
    LCD_D5 as sbit at RB1_bit
    LCD_D6 as sbit at RB2_bit
    LCD_D7 as sbit at RB3_bit

    LCD_RS_Direction as sbit at TRISB4_bit
    LCD_EN_Direction as sbit at TRISB5_bit
    LCD_D4_Direction as sbit at TRISB0_bit
    LCD_D5_Direction as sbit at TRISB1_bit
    LCD_D6_Direction as sbit at TRISB2_bit
    LCD_D7_Direction as sbit at TRISB3_bit

    ' Declaracion de variables a utilizar en el programa

    Dimtexto1 as char[16]
    texto2 as char[16]
    texto3 as char[16]
    texto4 as char[16]
    porcentaje as char[16]

    'sub rutina que inicializa la lcd, pwm y los puertos del pic
```

```
subprocedureInitMain()                    ' Inicio de la sub rutina InitMain
ANSEL  = 0                                ' Configure AN pins as digital I/O
ANSELH = 0

TRISA = 255          ' configura los pines del puerto A como entradas
TRISB = 0            ' configura los pines del puerto B como salidas
TRISC = 0            ' configura los pines del puerto C como salidas
PORTB = 0            ' coloca un 0 lógico en los pines del puerto B
PORTC = 0            ' coloca un 0 lógico en los pines del puerto C

texto1 = "UtilizaciónADC "
texto2 = " para controlar”
texto3 = "motor con PWM y"
texto4 = " porcentaje LCD"

PWM2_Init(5000)     ' InicializacionPWM2 con frecuencia de 5KHz

Lcd_Init()                                ' Inicializa la LCD
Lcd_Cmd(_LCD_CLEAR)                       ' Limpia la pantalla LCD
Lcd_Cmd(_LCD_CURSOR_OFF)'Apaga el cursor, para encender OFF=ON
Lcd_Out(1,1,texto1)          ' Escribe en la primer línea de la LCD
Lcd_Out(2,1,texto2)          ' Escribe en la segunda línea de la LCD
Delay_ms(2000)       'Tiempo en que muestra el mensaje 2 segundos
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
Lcd_Out(1,5,texto3)           ' Escribe mensaje en primer línea de la LCD
Lcd_Out(2,5,texto4)      ' Escribe mensaje en la segunda línea de la LCD
Delay_ms(2000)       ' Tiempo en que muestra el mensaje 2 segundos
end sub              ' finalización de la sub rutina InitMain
```

```
main:
InitMain()
PWM2_Start()                    ' Enciende el PWM2
while(TRUE)                     ' while que siempre es verdadero para tener
                                ‘un ciclo infinito
Lcd_Cmd(_LCD_CLEAR)             ' Limpia la pantalla de la LCD
temp_res = ADC_Read(2)          ' Guarda en la variable temp_res lectura de
                                ' ADC 2
PWM2_Set_Duty(temp_res/4)       ' coloca el valor temp_res como valor para
                                ' el tiempo util del pwm, note que está divido
                                ' por 4 para tener una lectura de 8 bits y no de
                                ' 10 como originalmente lee el ADC 2

temp_res=temp_res/10            'ADC 2 de 10 bits, se tendra en decimales
                                ‘1024 esto se dividirá entre 10 para tener una
                                ‘escala de 0-100
WordtoStr(temp_res,porcentaje)      'se convierte la variable temp_res de
                                    ‘word a string que guardara el resultado
                                    en ‘la variable porcentaje que es un
                                    string
Lcd_Out(1,1,"Velocidad motor")      ' Se escribirá en la línea 1 de la LCD
Lcd_Out(2,1,"es:"+porcentaje+"%")   'se colocara el porcentaje de tiempo de
                                    ‘trabajo del pwm en la línea 2 de la LCD
Delay_ms(500)                       ' tiempo de visualización en pantalla
Wend                                ' se cierra la sentencia wend para volver
                                    ‘a iniciar el ciclo
end.                                ‘fin del programa principal (Main)
```

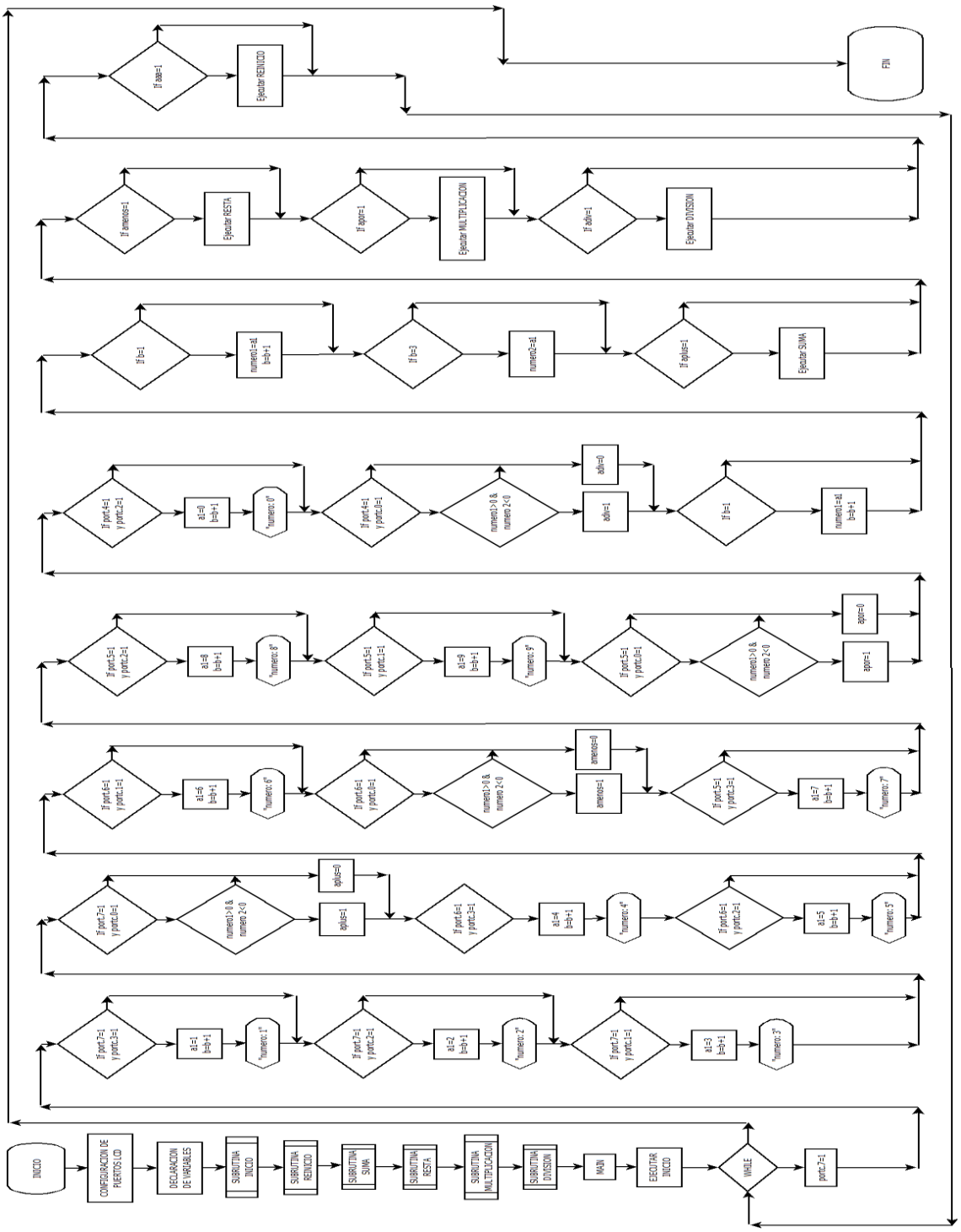
4.2.3. Práctica sobre la realización de una calculadora básica

La presente práctica tiene como finalidad crear con un microcontrolador (pic) una calculadora básica, pero a través de esto mostrar la forma de utilizar las subrutinas, las funciones *if_then, if_then_else, if* anidados, la lcd, el teclado matricial, el cual es de gran utilidad ya que conectar un botón a cada pin del pic, daría el problema de ocupar demasiados pines, como ejemplo se utiliza una matriz de 4x4 y se obtiene 16 botones, si por ejemplo se utiliza una matriz de 6x6 se obtendrían 36 botones con solo 12 pines del pic, lo cual ahorra 24 pines que bien se pueden utilizar para otro tipo de acciones, además se configura el puerto C del pic con 4 pines de salida y cuatro de entrada.

El teclado matricial es una arreglo de interruptores que están conectados de manera que cada botón está conectado a dos líneas del pic, una que da una señal de salida y la otra da una señal de entrada al pic, como se realiza un ciclo o recorrido a través de las líneas de salida del pic se obtendrá que en determinado momento un botón estará teniendo un 1 lógico de entrada y al pulsar dicho botón se enviará un 1 lógico a la entrada del pic, con esto se obtendrá que para cada botón existe una sola condición en la cual una línea de salía del pic esté en 1 lógico, y la otra línea de entrada a través del botón estén también en 1 lógico, con esto se asigna al botón en mención un valor, ya sea numérico o alfabético, en programación tipo *char*, en este caso se le asignará un valor numérico y las funciones de suma, resta, multiplicación, división y un *reset* de la calculadora.

Por otra parte se han creado las subrutinas o subprocedimientos de inicio, suma, resta, multiplicación, división, reinicio para facilitar la programación, también se utiliza un comando que es muy útil en cualquier entorno de programación como lo es el de convertir una variable de un tipo a

Figura 23. **Diagrama de flujo de la calculadora**



Fuente: elaboración propia

otro, por ejemplo en el programa, ha pasado la variable resultado de tipo byte a resultado2 de tipo *char*, esto para que facilite la visualización en la pantalla lcd además de remarcar su utilización.

programteclado2

```
' Conexiones de la LCD
dimLCD_RS as sbit at RB4_bit
LCD_EN as sbit at RB5_bit
LCD_D4 as sbit at RB0_bit
LCD_D5 as sbit at RB1_bit
LCD_D6 as sbit at RB2_bit
LCD_D7 as sbit at RB3_bit
LCD_RS_Direction as sbit at TRISB4_bit
LCD_EN_Direction as sbit at TRISB5_bit
LCD_D4_Direction as sbit at TRISB0_bit
LCD_D5_Direction as sbit at TRISB1_bit
LCD_D6_Direction as sbit at TRISB2_bit
LCD_D7_Direction as sbit at TRISB3_bit
dim b, bb,a1,adiv,aplus,amenos,apor,aa,aaa, resultado, numero1,numero2 as byte
dim resultado2 as char[16]
dimresultado3 as char[4]
dimnumero1char as char[16]
dimnumero2char as char[16]

subprocedure Inicio()          ' Inicio de la sub rutina InitMain
ANSEL  = 0                     ' Configure AN pins as digital I/O
ANSELH = 0
  TRISA = 255                  ' configura los pines del puerto A como entradas
```

```
TRISB = 0                    ' configura los pines del puerto B como salidas
TRISC = %00001111            ' configura los pines altos del puerto C como
salidas
‘y  los pines bajos del puerto C como entradas
PORTB = 0                    ' coloca un 0 lógico en los pines del puerto B
PORTC = 0
a1=0
aplus=0
amenos=0
apor=0
adiv=0
resultado=0
  numero1=0
  numero2=0
  b=0
bb=0

Lcd_Init()                   ' Inicializa la LCD
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
Lcd_Cmd(_LCD_CURSOR_OFF)     ' Apaga el cursor, para encender
OFF=ON
Lcd_Out(1,1,"Realizacion de")    ' Escribe en la primer línea de la LCD
Lcd_Out(2,1,"una calculadora")   ' Escribe en la segunda línea de la LCD
Delay_ms(2000)               ' Tiempo en que muestra el mensaje 2 segundos
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
Lcd_Out(1,2,"usando teclado")    ' Escribe mensaje en primer línea de la
LCD
Lcd_Out(2,4,"matricial")     ' Escribe mensaje en la segunda línea de la LCD
Delay_ms(2000)               ' Tiempo en que muestra el mensaje 2 segundos
```

```
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
end sub                      ' finalización de la sub rutina InitMain

subprocedure reinicio()
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
a1=0
aplus=0
amenos=0
apor=0
adiv=0
resultado=0
  numero1=0
  numero2=0
  b=0
bb=0
end sub

sub procedure suma()
resultado = numero1+numero2
WordtoStr(resultado,resultado2)
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
Lcd_Out(1,1,"resultado suma")            ' Escribe mensaje en primer línea de la
LCD
Lcd_Out(2,1,"="+resultado2)          ' Escribe mensaje en la segunda línea de la
LCD
Delay_ms(2000)               ' Tiempo en que muestra el mensaje 2 segundos
reinicio()
end sub
```

```
subprocedure resta()
resultado = numero1-numero2
WordtoStr(resultado,resultado2)
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
Lcd_Out(1,1,"resultado resta")          ' Escribe mensaje en primer línea de la
LCD
Lcd_Out(2,1,"="+resultado2)          ' Escribe mensaje en la segunda línea de la
LCD
Delay_ms(2000)          ' Tiempo en que muestra el mensaje 2 segundos
reinicio()
end sub

subproceduremultiplicacion()
resultado = numero1*numero2
WordtoStr(resultado,resultado2)
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
Lcd_Out(1,1," multiplicacion")          ' Escribe mensaje en primer línea de la
LCD
Lcd_Out(2,1,"="+resultado2)          ' Escribe mensaje en la segunda línea de la
LCD
Delay_ms(2000)          ' Tiempo en que muestra el mensaje 2 segundos
reinicio()
end sub

sub procedure division()
WordtoStr(numero1,resultado2)
WordtoStr(numero2,resultado3)
Lcd_Cmd(_LCD_CLEAR)          ' Limpia la pantalla LCD
Lcd_Out(1,1," division")          ' Escribe mensaje en primer línea de la LCD
```

```
Lcd_Out(2,1,"="+resultado2+"     /"+resultado3)              ' Escribe mensaje en la
segunda línea de la LCD
Delay_ms(2000)                ' Tiempo en que muestra el mensaje 2 segundos
reinicio()
end sub

main:
Inicio()
while(true)
PORTC.7=1
ifportc.7=1 and (portc.3 = 1) Then
Delay_ms(20)
a1=1
          b=b+1
Lcd_Out(1,1,"numero: 1")
end if
ifportc.7=1 and (portc.2=1) Then
Delay_ms(20)
a1=2
          b=b+1
Lcd_Out(1,1,"numero: 2")
end if
ifportc.7=1 and (portc.1=1) Then
Delay_ms(20)
a1=3
          b=b+1
Lcd_Out(1,1,"numero: 3")
end if
ifportc.7=1 and (portc.0=1) Then
```

```
Delay_ms(20)
if numero1>0 and (numero2>0)  then
aplus=1
end if
else
aplus=0
end if
Delay_ms(20)
portc.7=0

PORTC.6=1
ifportc.6=1 and (portc.3=1) Then
Delay_ms(20)
a1=4
        b=b+1
Lcd_Out(1,1,"numero: 4")
end if
ifportc.6=1 and (portc.2=1) Then
Delay_ms(20)
a1=5
        b=b+1
Lcd_Out(1,1,"numero: 5")
end if
ifportc.6=1 and (portc.1=1) Then
Delay_ms(20)
a1=6
        b=b+1
Lcd_Out(1,1,"numero: 6")
end if
```

```
ifportc.6=1 and (portc.0=1) Then
Delay_ms(20)
if numero1>0 and (numero2>0)  then
amenos=1
end if
else
amenos=0
end if
Delay_ms(20)
portc.6=0

PORTC.5=1
ifportc.5=1 and (portc.3=1) Then
Delay_ms(20)
a1=7
        b=b+1
Lcd_Out(1,1,"numero: 7")
end if
ifportc.5=1 and (portc.2=1) Then
Delay_ms(20)
a1=8
        b=b+1
Lcd_Out(1,1,"numero: 8")
end if
ifportc.5=1 and (portc.1=1) Then
Delay_ms(20)
a1=9
        b=b+1
Lcd_Out(1,1,"numero: 9")
```

```
end if
ifportc.5=1 and (portc.0=1) Then
Delay_ms(20)
if numero1>0 and (numero2>0)  then
apor=1
end if
else
apor=0
end if
Delay_ms(20)
portc.5=0

PORTC.4=1
ifportc.4=1 and (portc.3=1) Then
Delay_ms(20)
aa=1
else
aa=0
end if
ifportc.4=1 and (portc.2=1) Then
Delay_ms(20)
a1=0
          b=b+1
Lcd_Out(1,1,"numero: 0")
end if
ifportc.4=1 and (portc.1=1) Then
Delay_ms(20)
aaa=1
else
```

```
aaa=0
end if
ifportc.4=1 and (portc.0=1) Then
Delay_ms(20)
if numero1>0 and (numero2>0)  then
adiv=1
end if
else
adiv=0
end if
Delay_ms(20)
portc.4=0
if b=1 then
        numero1=a1
        b=b+1
end if
if b=3 then
        numero2=a1
end if
ifaplus=1 then
suma()
end if
ifamenos=1 then
resta()
end if
ifapor=1 then
multiplicacion()
end if
ifadiv=1 then
```

```
division()
end if
ifaaa=1 then
reinicio()
end if

wend
end.
```

CONCLUSIONES

1. Es necesario que las personas interesadas en programar microcontroladores tenga conocimientos básicos sobre electricidad y electrónica, para aprovechar de manera óptima los recursos y ventajas que presentan los microcontroladores.

2. Las personas que deseen realizar tareas con microntroladores, deben conocer las características y recursos del microcontrolador a utilizar para obtener el mayor provecho posible del mismo.

3. Además deben ser capaces de resolver problemas a través de algoritmos para después pasarlo a un lenguaje de alto nivel, ya que estos son de gran ayuda al programador para interactuar con el microcontrolador, puesto que el lenguaje de máquina y *assembler* son más complejos, lo cual da mayor versatilidad para programar, pero genera demasiadas complicaciones para quienes no están familiarizadas con este tipo de programación.

4. El microcontrolador pic 16f887 contiene características básicas para iniciarse en la programación de microcontroladores, desde módulos analógicos, puertos digitales, pwm, uart, entre otros, lo cual es idóneo para que una persona se inicie en el campo de la programación y diseño de circuitos utilizando microcontroladores.

RECOMENDACIONES

1. Considerando la evolución vertiginosa que sufre la electrónica, y específicamente en lo concerniente a los microcontroladores, las personas interesadas en realizar proyectos con estos, deben de tomar en cuenta la resolución de problemas a través de algoritmos y diagramas de flujo, por lo cual es importante que estén en la capacidad de plantear y resolver los problemas por esta vía y convenientemente trasladarlo al lenguaje de programación adecuado para finalmente llevar esta solución al microcontrolador y de esta forma dar por solucionado el problema.

2. Es importante que las personas que necesiten crear soluciones a partir de microcontroladores, deben verificar las características y recursos de los mismos, esto para poder aprovechar de manera óptima al microcontrolador y evitar desperdiciar recursos del mismo.

3. Tomar en cuenta que no solamente existen los microcontroladores pic, además están los embebidos, como los arduinos y los DSP que tienen recursos específicos para determinados problemas, por ejemplo los DSP están diseñados para trabajar con señales de audio, esto debido a la velocidad de operación a la que puede trabajar.

BIBLIOGRAFÍA

1. BREY, Barry B. *Los microprocesadores INTEL, arquitectura, programación e interfaces*. 5a ed. México: Prentice Hall, 1994. 966 p.

2. JOYANES AGUILAR, Luis. *Fundamentos de programación, algoritmos, estructuras de datos y objetos*. 3a ed. España: McGraw-Hill, 1996. 714 p.

3. MikroElectronika. *Libro de programación en MikroBasic*. [en linea] <http://www.mikroe.com/chapters/view/84/libro-de-la-programacion-de-los-microcontroladores-pic-en-basic-capitulo-1-mundo-de-los-microcontroladores/> [Consulta: 15 de junio de 2014].

4. Wikipedia. *Circuito integrado*. [en línea] < http://es.wikipedia.org/wiki/ Circuito_ integrado> [Consulta: 7 de febrero de 2014].

5. _____. *Datasheet.* [en línea] <http://es.wikipedia.org/wiki/ Datasheet> [Consulta: 20 de enero de 2014].

6. _____. *Resistencia eléctrica*. [en línea] <http://es.wikipedia.org/ wiki/Resistencia_electrica> [Consulta: 15 de mayo de 2014].