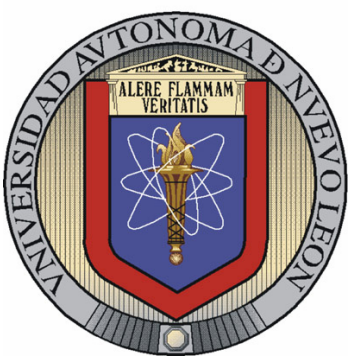
UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISIÓN DE ESTUDIOS DE POST – GRADO



PROGRAMACION DEL MICROCONTROLADOR 68HC12B32

POR

HÉCTOR GILBERTO BARRÓN GONZÁLEZ

TESIS

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERIA ELECTRICA CON ESPECIALIDAD EN ELECTRONICA

SAN NICOLAS DE LOS GARZA, CIUDAD UNIVERSITARIA
18 DE OCTUBRE DEL 2005

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISIÓN DE ESTUDIOS DE POST – GRADO

PROGRAMACIÓN DEL MICROCONTROLADOR 68HC12B32

POR

HÉCTOR GILBERTO BARRÓN GONZÁLEZ

TESIS

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERIA ELECTRICA CON ESPECIALIDAD EN ELECTRONICA

SAN NICOLAS DE LOS GARZA, NUEVO LEON
18 DE OCTUBRE DEL 2005

A mis padres

Agradecimientos

Mi mas sincero y especial agradecimiento a mi asesor MC. José Angel Castillo Castro por haber aportado generosamente sus conocimientos y experiencia en la elaboración de esta tesis, a mis revisores MC. Cesar Augusto Leal Chapa y MC. José Manuel Rocha Núñez por todo el apoyo brindado.

A la Ing. Karla Verenizze Porras Lucio por su apoyo inquebrantable.

PROLOGO

Hoy en día la automatización es un factor primordial en el desarrollo de una empresa, y conocer las herramientas que pueden ser utilizadas al automatizar un proceso es muy importante ya que de esto dependen los costos y la efectividad que se pueda obtener. Es por ello que el desarrollo de los microcontroladores ha venido aumentando considerablemente durante los últimos años, y hoy en día se pueden ver utilizados dentro de una amplia gama de aplicaciones en diferentes campos, por lo que conocer su operación se ha vuelto muy importante ya que son una herramienta económica y flexible que cada día se vuelve más robusta y sencilla de utilizar.

Los microcontroladores pueden ser utilizados en procesos industriales, en la automatización de un dispositivo, en el control de un mecanismo, en sistemas de adquisición de datos y en una gran diversidad de aplicaciones como permitan los recursos con los que cuenta un microcontrolador.

Esta tesis va dirigida a todas las personas que quieran adentrarse en el manejo y operación de un microcontrolador. Se brindan los conocimientos necesarios para programar y utilizar los diferentes recursos con los que cuenta el microcontrolador MC68HC12B32 de Motorola, y se tratan diferentes aplicaciones orientadas a ampliar un panorama que permita ver las distintas aplicaciones que pueden ser desarrolladas con este microcontrolador como herramienta.

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica
División de Estudios de Posgrado

Los miembros del Comité de Tesis, recomendamos que la Tesis **"PROGRAMACION DEL MICROCONTROLADOR 68HC12B32"**, realizada por el alumno Ing. Héctor Gilberto Barrón González con número de matricula 1206655 sea aceptada para su defensa como opción al grado de Maestro en Ciencias de la Ingeniería Eléctrica con Especialidad en Electrónica.

El Comité de Tesis

_____
M.C. JOSE ANGEL CASTILLO CASTRO
ASESOR

_____
M.C. CESAR A. LEAL CHAPA
REVISOR

_____
M.C. JOSE MANUEL ROCHA NUÑEZ
REVISOR

Vo.Bo.

_____
DR. GUADALUPE ALAN CASTILLO RODRIGUEZ
DIVISION DE ESTUDIOS DE POSGRADO

Ciudad Universitaria, a 18 de octubre del 2005

INDICE

SINTESIS

Esta tesis es una guía práctica para conocer el modo de operación y la programación de los periféricos más importantes del microcontrolador MC68HC12B32.

El capitulo 2 menciona la arquitectura del microcontrolador y las diferentes configuraciones básicas de operación. El capítulo 3 abarca la programación del microcontrolador utilizando el programa MiniIde, incluye el desarrollo necesario para la elaboración de un programa, la descripción de este software y también la descripción de las principales instrucciones, interrupciones, modos de operación y tipos de direccionamiento soportados por el microcontrolador.

El capitulo 4 trata la configuración básica de cada periférico del microcontrolador. También se incluye el capitulo 5 el cual trata de interfaces para utilizar con el microcontrolador con la finalidad de poder interactuar con el mundo real.

En el capitulo 6 podemos encontrar una serie de practicas diseñadas para cubrir los principales aspectos de los capítulos 3, 4 y 5.

Con la finalidad de mostrar algunos temas donde se explican algunas aplicaciones que pueden ser utilizadas dentro de la industria, se introdujo el capitulo 7, el cual contiene el diseño de un proyecto multitareas que comprende temas usados dentro de instrumentación y control.

Finalmente en el capitulo 8 se incluyen las conclusiones y recomendaciones.

CAPITULO 1

INTRODUCCIÓN

Objetivo

Se proporcionara a los usuarios las herramientas necesarias acerca del Microcontrolador 68HC12. Herramientas que permitan elaborar aplicaciones más complejas orientadas hacia aplicaciones de control e instrumentación, por lo que se pretende sea útil para quien decida desarrollar aplicaciones con este microcontrolador.

Hipótesis

Si se le da seguimiento al material comprendido en esta tesis la cual abarca desde la configuración básica necesaria para el funcionamiento del microcontrolador, hasta el uso de los diferentes periféricos del mismo, se facilitara el aprendizaje del uso del microcontrolador MC68HC12B32 con lo que será más rápida y sencilla la elaboración de aplicaciones para control e instrumentación.

Justificación

En la actualidad, el desarrollo de nuevas tecnologías que permitan la optimización de procesos industriales es primordial, ya que día con día la competencia entre las empresas demanda costos de producción menores y una máxima calidad en el producto final. Esto solo puede lograrse con la aplicación de tecnologías que abaraten los costos de producción sin disminuir la calidad. Dentro de la automatización de procesos industriales existen aplicaciones que pueden ser automatizadas sin la necesidad de utilizar equipos mas sofisticados y sobre todo con un precio muy alto como son los PLC's (Controladores Lógicos Programables), que no obstante a su gran funcionalidad, su implementación solo se justifica dentro de procesos muy complejos donde se requiere siempre un chequeo de programación centralizada en cada una de las partes del proceso, entre otros.

Para cubrir todos esos espacios donde la implementación de un controlador lógico programable resulta incosteable, y para muchas otras aplicaciones de bajo costo que a la vez pueden requerir de un dispositivo de control preciso y eficaz capaz de efectuar algoritmos de control, como también pueden ser procesos secuenciales sencillos, se han desarrollado los microcontroladores. Los cuales ofrecen grandes ventajas, en términos de versatilidad, consumo de potencia, velocidad de ejecución, tamaño y facilidad de uso, además de que existe una gama muy amplia de microcontroladores para cada necesidad.

Lo que los hace los mejores candidatos para una diversidad muy grande de aplicaciones tanto en procesos pequeños como en procesos industriales complejos.

Es por eso que surge la necesidad para todas las personas involucradas en el área de control y diseño automático, de tener los conocimientos que les permitan desarrollar aplicaciones con microcontroladores, razón por la cual se llego a la idea de desarrollar esta tesis cuya pretensión es la de dotar con las herramientas necesarias tanto teóricas como practicas para el microcontrolador 68HC12B32.

Limites de estudio

La elaboración de esta tesis tuvo varias limitantes, entre las cuales destaca la falta de documentación en nuestro idioma tanto para aplicaciones prácticas, como incluso para describir el funcionamiento de algunos de los periféricos de este microcontrolador, lo cual comúnmente sucede al introducir un dispositivo nuevo.

Es por esto la importancia de la elaboración de este documento, con el que se pretende cubrir mediante la elaboración de aplicaciones prácticas utilizadas en el área de control e instrumentación, el funcionamiento de las partes más importantes del microcontrolador.

Metodología

Para lograr una mejor comprensión y manejo del microcontrolador se han propuesto la siguiente metodología, que comienza con la descripción general del hardware de nuestro microcontrolador y consecutivamente va cubriendo los aspectos primordiales necesarios para la implementación de aplicaciones con el microcontrolador.

1.- Descripción General del Hardware del Microcontrolador.
2.- Programación del Microcontrolador 68HC12.
2.- Modos de operación y configuración de cada uno de los recursos internos del microcontrolador.
3.- Aplicaciones especificas para los diferentes periféricos del microcontrolador.
4.- Interfases para el manejo y protección de los periféricos del microcontrolador.
5.- Aplicaciones del microcontrolador para sistemas de control.
6.- Aplicaciones del microcontrolador dentro de la instrumentación.

CAPITULO 2

INTRODUCCIÓN AL MICROCONTROLADOR MC68HC12B32 DE MOTOROLA

2.1.-Desarrollo del Microcontrolador

En los últimos años la computación ha revolucionado hasta producir computadoras más veloces y de gran capacidad cada vez en tamaños más pequeños. Esta revolución ha ocurrido como resultado del desarrollo de tecnologías Large Scale Integration (LSI) e Very Large-Scale integration (VSLI). Con las cuales es posible integrar decenas de miles de transistores en un solo circuito integrado.

Esto a hecho posible la fabricación del corazón de una microcomputadora llamado "Procesador" dentro de un circuito integrado. Este C.I. con la ayuda de más C.I. auxiliares llamados periféricos (memoria, dispositivos de entrada/salida, timmers etc.), constituyen una microcomputadora.

Estas nuevas tecnologías también han hecho posible la integración de una microcomputadora para aplicaciones de control en un solo C.I. llamado Microcontrolador **(MCU).**

El **Microcontrolador** contiene internamente una unidad central de procesamiento (CPU), y una serie de recursos internos (memoria interna, registros y subsistemas de entradas y salidas (E/S) partes que a la vez están conectadas en el interior del microcontrolador por una serie de buses internos.

2.1.1.- Arquitectura de un Microcontrolador

Un microcontrolador consiste de cuatro partes básicas: unidad central de procesamiento, memoria interna, registros y dispositivos de Entradas y Salidas, las cuales detallamos a continuación.

Unidad Central de Procesamiento (CPU)

El CPU se divide en tres secciones principales:

- Unidad aritmética lógica
- Registros y Acumuladores
- Unidad de control y secuencia.

Unidad Aritmética Lógica

La ALU tiene la capacidad de efectuar operaciones aritméticas y lógicas con operandos; estas operaciones incluyen la suma, resta, multiplicación, división, and, or entre otras. Los operandos se colocan temporalmente en los registros acumuladores y el resultado de una operación regresa de nuevo a los acumuladores.

Registros y Acumuladores

Los registros y acumuladores son localidades de memoria de alta velocidad de transferencia donde se encuentran los datos con los que opera la ALU. Estos intercambian información a través de uno o más buses internos.

Los registros suelen dividirse en dos: registros de propósito general y registros de propósito especial.

1. Registros de Propósito General

Un registro de propósito general puede ser utilizado como registro de datos para operaciones lógicas y aritméticas, o también como un acumulador. Un acumulador es un registro que mantiene el resultado de una operación realizada por la ALU.

Estos registros también pueden ser utilizados como registros de direccionamiento que apunten a una localidad de memoria.

2. Registros de Propósito Especial

Todos los microcontroladores incluyen estos registros (PC, SP etc.), se dedican a funciones específicas. Algunos de estos registros son:

Program Counter (PC). Al principio de la ejecución de un programa, el PC es cargado con la dirección de inicio del programa. Después de eso, este siempre apuntara a la dirección de la próxima instrucción a ejecutar.

Stack Pointer (SP). Este es inicializado por el usuario. La pila va desde direcciones altas hacia direcciones bajas, por lo que al escribir un dato en la pila, el SP se decrementa en 1 o 2 bytes dependiendo del tamaño del dato almacenado temporalmente metido en la pila. Al leer un dato de la pila, SP se incrementa.

Status Register. Este consiste de una bandera de bits y bits de control. Las banderas se configuran automáticamente durante una operación aritmética o lógica. Algunas de estas banderas son N, Z, V, C.

Los bits de control son configurados por el programa para habilitar algunos modos de operación del CPU.

Unidad de Control

La unidad de control cumple con la función de decodificar las instrucciones, enviar señales de reloj a los diferentes componentes del CPU para que avancen e n su secuencia individual pero que en conjunto ejecuten la instrucción

La unidad de control también genera dos grupos de señales:

1. Señal de control interna para activación de la ALU.
2. Señal de control externa que concierne a la memoria y a dispositivos E/S. Esta señal es enviada también, para la activación de transferencia de datos o como respuesta a una interrupción.

Unidad de Memoria

La unidad de memoria contiene el programa a ser ejecutado, los datos que son operados por el programa y también puede contener algunos operandos variables utilizados por el programa principal. La operación de la unidad de memoria es controlada por el CPU.

Cuando el CPU envía un dato a la memoria, esto es llamado operación de escritura, y cuando el CPU recibe un dato desde la memoria, a esto se le llama operación de lectura.

La memoria se divide en diferentes tipos: memoria RAM, ROM, FLASH y EEPROM

Dispositivos de Entrada /Salida

Los dispositivos de entradas y salidas permiten al MCU intercambiar información con el mundo exterior.

Cada puerto de E/S tiene una línea (usualmente 8 líneas) para transferir información entre dispositivos externos y los puertos. Estas líneas pueden ser programadas mediante sus diferentes registros para ser utilizadas como entradas, como salidas o ambas, aunque existen líneas que solo pueden funcionar de una sola manera debido a que comparten el puerto con otros dispositivos internos del microcontrolador.

2.2.-Descripción General del Microcontrolador

El microcontrolador 68HC12B32 es un microcontrolador de 16 bits, con frecuencia de 8 MHZ, su programación es muy sencilla y cuenta con la mayoría de los recursos utilizados por las diferentes marcas de microcontroladores (ver Tabla 2.2.1.- Características de la serie 68HC12B).

Los recursos más importantes del microcontrolador se detallan a continuación:

- CPU de 16 bits
  Compatible con el juego de instrucciones del 68HC11
  Unidad aritmética lógica de 20 bits
- Memoria
  32 Kbytes de memoria FLASH
  1 Kbyte de memoria RAM
  768 bytes de EEPROM
- Modulo de timer estándar de 8 canales (TIM)
- 8 Puertos de 8 bits
- Convertidor analógico a digital de 8 canales y 10 bits (ATD)
- Interfaz de comunicación serial asíncrona (SCI)
- Interfaz de comunicación serial sincrona (SPI)
- Modulador de ancho de pulso (PWM)
  8 bit con 4 canales y 16 bits con 2 canales
- Controlador de Área de Red (CAN)

Tabla 2.2.1.- Características de la serie 68HC12B.

Features	MC68HC912B32	MC68HC12BE32	MC68HC912BC32	MC68HC12BC32
CPU12	X	X	X	X
Multiplexed bus	X	X	X	X
32-Kbyte FLASH electrically erasable, programmable read-only memory (EEPROM)	X		X	
32-Kbyte read-only memory (ROM)		X		X
768-byte EEPROM	X	X	X	X
1-Kbyte random-access memory (RAM)	X	X	X	X
Analog-to-digital (A/D) converter	X	X	X	X
Standard timer module (TIM)	X		X	X
Enhanced capture timer (ECT)		X		
Pulse-width modulator (PWM)	X	X	X	X
Asynchronous serial communications interface (SCII)	X	X	X	X
Synchronous serial peripheral interface (SPI)	X	X	X	X
J1850 byte data link communication (BDLC)	X	X		
Controller area network module (CAN)			X	X
Computer operating properly (COP) watchdog timer	X	X	X	X
Slow mode clock divider	X	X	X	X
80-pin quad flat pack (QFP)	X	X	X	X
Single-wire background debug mode (BDM)	X	X	X	X

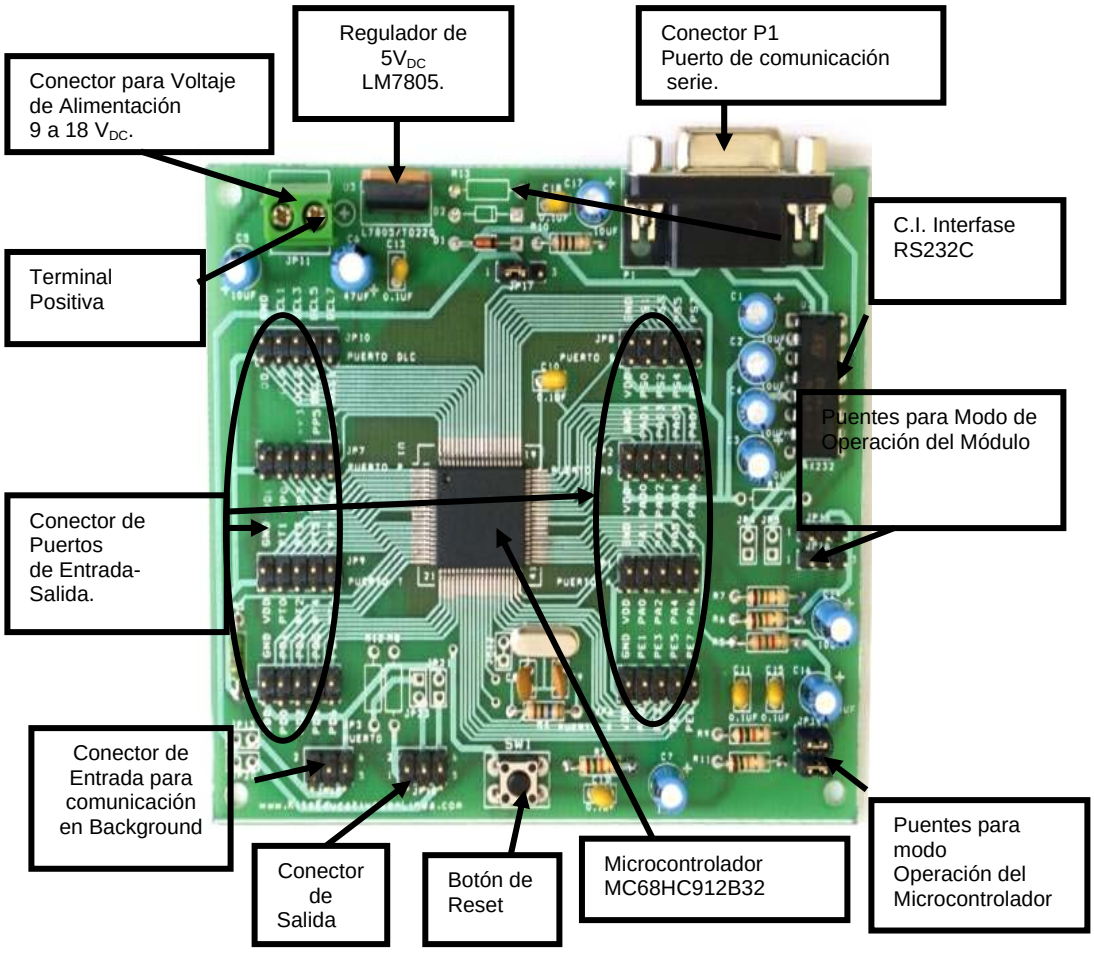
2.3.-Tarjeta Entrenadora

2.3.1.- Descripción

El Módulo de Evaluación MEVHC912B32, consiste una tarjeta de Electrónica, basado en un Microcontrolador de Motorola MC68HC912B32 (Ver figura 1), además de una interfase para comunicación serie en Standard RS232C.

La tarjeta cuenta también, con regulador de 5V (LM7805), 8 conectores tipo poste para acceso a los puertos de entada-salida y una serie de puentes (jumpers) para establecer los diferentes modos de operación del Módulo.

Figura 2.3.1.- Módulo de Evaluación MEV912B32.



2.3.2.- Modos de Operación

El Módulo de Evaluación MEV912B32 puede funcionar en tres diferentes modos de operación, configurables con los puentes JP14 y JP16.

Es importante aclarar que estos son los modos de operación del Modulo, para esto, el Microcontrolador, debe estar operando en modo *Single Chip (Ver sección 5 en documento M68HC12B)* por lo que es importante que los puentes JP19 y JP22 se encuentren siempre cerrados.

La siguiente tabla muestra el estado de JP14 y JP16 para cada uno de los modos

JP16	JP14	MODO	Comentario
Cerrado	Cerrado	EVM	Opera el Sistema Monitor DBUG12
Cerrado	Abierto	EEPROM	Ejecuta programa de usuario que inicia en la Dirección $0D00
Abierto	Cerrado	POD	Opera como controlador de otra Tarjeta que contenga un microcontrolador HC12

Tabla 2.3.2.1. Configuración de puentes JP14 y JP16 para cada Modo de Operación.

2.3.3.- Modo EVB, Tarjeta de Evaluación.

En este modo el Microcontrolador ejecuta el Programa Monitor DBUG12 que reside en la Memoria Interna tipo Flash EEprom.

Este sistema permite al usuario, visualizar y modificar el contenido de la memoria, introducir y ejecutar programas de aplicación, todo esto a través de una computadora personal, ejecutando la aplicación de HYPERTERMINAL.

Para información mas detallada acerca del Sistema Monitor Debug12, consulte el documento DB12RG.

2.3.4.- Modo EEPROM - Ejecución de aplicaciones del usuario en la EEPROM interna.

Cuando el Módulo opera en este Modo, el Microcontrolador ejecuta el programa de usuario previamente cargado en el EEPROM interno, permite que el Microcontrolador ejecute la aplicación sin depender de los comandos del DBUG12. Es decir, se establece al Microcontrolador como un Instrumento Autónomo.
Es requisito que el programa inicie en la dirección $0D00.

2.3.5- Modo POD.

En este modo el Módulo, tiene la capacidad de Manipular la operación de una segunda tarjeta que contenga un Microcontrolador de la familia 912.

CAPITULO 3

PROGRAMACION DEL MICROCONTROLADOR MC68HC12B32

3.1.- Modos de Funcionamiento del Microcontrolador

El microcontrolador 68HC12B32 tiene 8 principales formas de operación. Cada modo de operación tiene un mapa de memoria y configuración de bus externa diferente.

El estado de los pines BKGD, MODB, y MODA durante el reset, determinan el modo de operación posterior al reset. A continuación mostramos la tabla 3.1.1 donde se muestran los diferentes tipos reoperación del microcontrolador así como las diferencias que existen de un modo de operación a otro en algunos de sus buses.

Tabla 3.1.1.- Modos de Operación del Microcontrolador.

BKGD	MODB	MODA	Mode	Port A	Port B
0	0	0	Special single chip	General-purpose I/O	General-purpose I/O
0	0	1	Special expanded narrow	ADDR[15:8] DATA[7:0]	ADDR[7:0]
0	1	0	Special peripheral	ADDR DATA	ADDR DATA
0	1	1	Special expanded wide	ADDR DATA	ADDR DATA
1	0	0	Normal single chip	General-purpose I/O	General-purpose I/O
1	0	1	Normal expanded narrow	ADDR[15:8] DATA[7:0]	ADDR[7:0]
1	1	0	Reserved (forced to peripheral)	—	—
1	1	1	Normal expanded wide	ADDR DATA	ADDR DATA

3.2.-Modos de Direccionamiento

Los modos de direccionamiento son las distintas formas que el CPU tiene para acceder a datos en memoria. Este microcontrolador tiene 12 diferentes tipos de direccionamiento, los más utilizados se muestran en la tabla 3.1.2.

Tabla 3.2.1- Modos de direccionamiento del 68HC12B32

Direccionamiento	Formato	Abreviación	Descripción
Inherente	INS (No necesita de operandos)	INH	Los operandos están dentro del registro del CPU
Inmediato	INS #opr8 INS #opr16	INM	El operando es incluido en la instrucción. 8 o 16 bits
Directo	INS opr8	DIR	Operando de 8 bits dentro del rango $0000-$00FF
Extendido	INS opr16	EXT	El operando es una dirección de 16 bits
Relativo	INS rel8 INS rel16	REL	Se aplica un desplazamiento de del Contador del Programa de +- 128 o 65535 direcciones.
5 bits de Offset Direccionamiento Indexado	INST *oprx5*,*xysp*	IDX	Desplazamiento de 5 bits con signo hacia X, Y, SP O PC.
9 bits de Offset Direccionamiento Indexado	INST *oprx9*,*xysp*	IDX	Desplazamiento de 9 bits con signo hacia X, Y, SP O PC.
16 bits de Offset Direccionamiento Indexado	INST *oprx16*,*xysp*	IDX	Desplazamiento de 16 bits con signo hacia X, Y, SP O PC.
Direccionamiento Indexado Auto Pre/Post Decremento e Incremento	INST *oprx3*,*–xys* INST *oprx3*,*+xys* INST *oprx3*,*xys–* INST *oprx3*,*xys+*	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8 Auto pre-increment x, y, or sp by 1 ~ 8 Auto post-decrement x, y, or sp by 1 ~ 8 Auto post-increment x, y, or sp by 1 ~ 8

3.2.1.-Direccionamiento Inherente

Las instrucciones que usan este modo de direccionamiento no contienen operandos o los operandos utilizados están dentro de los registros internos del CPU.
Ejemplo

NOP; Esta instrucción no contiene operandos

INX; El operando esta dentro de un registro del CPU

3.2.2.- Direccionamiento Inmediato

El operando en este modo de direccionamiento se encuentra dentro de la instrucción, por lo que no es necesario acceder a memoria, el operando (dato) puede ser de 1 o 2 bytes. El símbolo # se utiliza para indicar este modo de direccionamiento. El formato utilizado para este direccionamiento es:

INS #OPR8 ó **INS #OPR16**

Ejemplo: **LDAA #$FF**; Esta instrucción carga el valor hexadecimal FF dentro del acumulador A. El valor $FF se encuentra a continuación de la instrucción, el símbolo $ se utiliza para indicar la base en hexadecimal de un número.

3.2.3.- Direccionamiento Directo

Este modo de direccionamiento actúa con direcciones comprendidas entre $00-$FF. Los primeros 256 bytes de memoria. Este modo solo necesita un byte para especificar la dirección del dato. El formato de este modo de direccionamiento es:

INS OPR8

Ejemplo: LDAA $50; Esta instrucción carga el contenido de la dirección $0050 en el acumulador A.

Esto es debido a que se toma el contenido del byte bajo de la localidad de memoria comprendida entre $0000-$FFFF. Se asume que la mitad del byte alto es 0.

Durante la ejecución de la instrucción el CPU combina el valor $55 de la instrucción con el valor asumido $00 para formar la dirección $0055, la cual se utilizara para acceder al dato que será cargado al acumulador A.

Ejemplo: LDX $40; En este caso el contenido del acumulador x es de 16 bits y mediante un solo byte de dirección podemos acceder a 2 Bytes de localidad de memoria. Suponiendo que el registro x tiene almacenado el valor 5578, al enviar el valor a la dirección 40, quedaran almacenados el 55 (parte alta del 16 bits) dentro de la dirección $0040 y a continuación el CPU almacenara el valor 78 (parte baja 16 bits) dentro de la dirección siguiente es decir $0041.De igual forma utilizando los registros X y Y podemos acceder a localidades de memoria indicando siempre la primera localidad que contenga el primer valor a cargar dentro de estos dos registros y el CPU podrá acceder de forma automática a la siguiente localidad de memoria para obtener el byte bajo del registro.

3.2.4.-Direccionamiento Extendido

En este modo de direccionamiento el dato se encuentra dentro de los 16 bits de la localidad de memoria especificada. Este modo de direccionamiento puede ser utilizado para acceder a una localidad de memoria dentro de los 64 Kbytes del mapa de memoria por lo que la dirección ocupa 2 bytes.

El formato de este modo de direccionamiento es:

INS opr16

Ejemplo: LDAA #$80; En esta instrucción mediante modo inmediato colocamos el numero 80h dentro del acumulador A.

STAA $FF55; Mediante direccionamiento extendido se coloca el contenido del acumulador A (80h) dentro de la localidad de memoria $FF55

3.2.5.-Direccionamiento Relativo

Este modo de direccionamiento solo se utiliza con las instrucciones de Salto (BRANCH). Estas indican al CPU que realice un salto de tantos bytes hacia adelante o hacia atrás. Se dice que el desplazamiento tiene signo y es de 1 byte es decir solo puede hacerse una bifurcación 128 bytes hacia atrás o 127 hacia adelante:
Ejemplo:

```
Ciclo   ....            ; Dirección 0
        ....
        BNE   Ciclo; Dirección 128
```

El salto a efectuar lo calcula automáticamente el ensamblador. Solo es necesario saber que los instrucciones Branch solo pueden hacerse a posiciones de memoria que estén 128 bytes por debajo y 127 bytes por arriba, por lo que si este limite es sobrepasado el ensamblador generara un mensaje de error.
Ejemplo:

```
Ciclo   Inst1           ; Dirección 0
        .....
        .....
        BNE   Ciclo   ; Dirección 150
```

Este programa genera un mensaje de error debido a que el salto que la bifurcación realiza es de más de 128 bytes hacia arriba. Para realizar bifurcaciones a cualquier localidad de memoria se realizan las instrucciones JUMP y JSR que ocupan 3 bytes.

3.2.6.- 5 bits de Offset Direccionamiento Indexado

Este modo de direccionamiento indexado utiliza 5 bits de desplazamiento con signo incluidos en la instrucción. Este pequeño desplazamiento o corrimiento se agrega a la dirección que contienen los registros índices (X, Y, SP o PC) para formar la dirección final que será afectada por la instrucción. Este nos da un rango de un valor desde -16 hasta 15 para el valor que contiene el registro índice.

Ejemplo:

```
LDAA    0,X
STAB    -8,Y
```

Para este ejemplo asumimos que X tiene la dirección $1000 y Y el la dirección de $2000 antes de la ejecución de la instrucción. El valor de 5 bits de desplazamiento no cambia el valor del registro índice

En el primer ejemplo el valor que será cargado se encuentra en la dirección $1000, y el segundo el acumulador B se colocara en la dirección $1FF8 (2000-8).

3.2.7.- 9 bits de Offset Direccionamiento Indexado

Este modo de direccionamiento indexado utiliza 9 bits de desplazamiento con signo incluidos en la instrucción. Este pequeño desplazamiento o corrimiento se agrega a la dirección que contienen los registros índices (X, Y, SP o PC) y nos da un rango que va desde-255 hasta 255 para encontrar la dirección final que contiene el valor que va a ser afectado por la instrucción.

EJEMPLO

```
LDAA $FF,X
LDAB -20, Y
```

Para este ejemplo asumimos que el registro índice X contiene la dirección $1000 y el registro índice Y contiene la dirección $2000 antes de ejecutar la instrucción.

El primer valor que será cargado en el acumulador A será el que contenga la dirección $10FF y la segunda instrucción cargara en el acumulador B el contenido de la dirección $1FEC.

La principal diferencia de este modo de direccionamiento es que el valor del desplazamiento cubre el mismo rango para valores positivos, tanto como para valores negativos.

3.2.8.- 16 bits de Offset Direccionamiento Indexado

El modo de direccionamiento indexado contiene un dato de 16 bits agregado en la instrucción, con los que se encontrara una dirección en que contendrá el valor de la dirección final que contiene el valor a ser cargado en el registro índice. En este modo de direccionamiento se utilizan los corchetes para distinguirlo de otros modos de direccionamiento.

Ejemplo:

```
LDAA [10, X]
```

Para este ejemplo vamos a suponer que el registro índice X contiene la dirección de $1000 y que la dirección 2000 esta a colocada en las direcciones $100A y $100B. De esta forma al agregar 10 a el registro X se forma la dirección $100A que contiene la dirección final $2000 en la que se encuentra el valor que será cargado al Acumulador A.

3.2.9.- Direccionamiento Indexado Auto Pre/Post Decremento e Incremento

Este modo de direccionamiento provee de cuatro formas básicas para cambiar la dirección que contiene el registro índice. El registro índice puede decrementarse o incrementarse antes o después de que el direccionamiento toma efecto. La dirección a acceder puede contenerse en los registros X, Y o SP.

Las versiones Pre-decremento y Pre-incremento modifican el valor del registro índice antes de acceder a la localidad de memoria afectada por la instrucción, después que la instrucción se ejecuta se retiene este cambio en el registro índice. Los direccionamiento con Post-incremento o post-decremento utilizan la dirección del registro índice para acceder a la localidad de memoria y después incrementan el contenido del registro índice. El CPU permite un decremento o incremento para un valor con un rango de -8 hasta -1 ó desde 1 hasta 8.

Ejemplo

```
STAA 1,–SP
STX  2,–SP
LDX  2,SP+
LDAA 1,SP+
```

En el ejemplo STAA 1,–SP, el puntero de pila es pre-decrementado con uno y después es colocado el contenido del acumulador A dentro del puntero de pila (Stack pointer). De forma similar en el ejemplo LDX 2, SP+, primero se carga x en el puntero de pila y después se incrementa el SP con 2.

3.3.-Instrucciones del Microcontrolador

A continuación se presentan en las siguientes tablas las instrucciones soportadas por el microcontrolador, en las tablas se detalla el código nemonico de la instrucción, su función y por ultimo su operación.

Para conocer más a detalle el funcionamiento de cada instrucción diríjase el apéndice B de esta tesis.

Las instrucciones de carga copian el contenido de una memoria dentro de un acumulador o registro, el contenido de la memoria no se ve afectado por esta operación. Estas instrucciones afectan directamente el registro de código de condición por lo que no se necesita de otras instrucciones para conocer el estado de estos bits.

Instrucciones de Carga		
Nemonico	**Función**	**Operación**
LDAA	Carga en A	(M) $\Rightarrow$ A
LDAB	Carga en B	(M) $\Rightarrow$ B
LDD	Carga en D	(M:M+1) $\Rightarrow$ (A:B)
LDS	Carga en S	(M:M+1) $\Rightarrow$ SP_H:SP_L
LDX	Carga registro índice en X	(M:M+1) $\Rightarrow$ X_H:X_L
LDY	Carga registro índice en Y	(M:M+1) $\Rightarrow$ Y_H:Y_L
LEAS	Carga la dirección en SP	Dirección efectiva $\Rightarrow$ SP
LEAX	Carga la dirección en X	Dirección efectiva $\Rightarrow$ X
LEAY	Carga la dirección en Y	Dirección efectiva $\Rightarrow$ Y

Las instrucciones de almacenamiento copian el contenido de un registro del cpu dentro de una memoria, el contenido de los registros o acumuladores no se ve afectado por esta operación.

Al utilizar este tipo de instrucciones se actualiza el estado de los bits N y Z del código de condición por lo que no se necesita de otras instrucciones para conocer el estado de estos bits.

Instrucciones de Almacenamiento		
Nemonico	**Función**	**Operación**
STAA	Guarda en A	(A) $\Rightarrow$ M
STAB	Guarda en B	(B) $\Rightarrow$ M

STD	Guarda en D	(A) $\Rightarrow$M, (B) $\Rightarrow$M+1
STS	Guarda en SP	**(**SP_H:SP_L**)** $\Rightarrow$ M:M+1
STX	Guarda en X	**(**X_H:X_L**)** $\Rightarrow$ M:M+1
STY	Guarda en Y	**(**Y_H:Y_L**)** $\Rightarrow$ M:M+1

Las instrucciones de transferencia copian el contenido de un registro o acumulador dentro de otro registro o acumulador. El contenido del la fuente no cambia por la operación.

Instrucciones de Transferencia		
Nemonico	**Función**	**Operación**
TAB	Transfiere A hacia B	(A) $\Rightarrow$B
TAP	Transfiere A hacia CCR	(A) $\Rightarrow$CCR
TBA	Transfiere B hacia A	(B) $\Rightarrow$A
TFR	Transfiere Registro a Registro	(A, B, CCR, D, X, Y OR SP) $\Rightarrow$()(A, B, CCR, D, X, Y OR SP)
TPA	Transfiere CCR hacia A	(CCR) $\Rightarrow$A
TSX	Transfiere SP hacia X	(SP) $\Rightarrow$X
TSY	Transfiere SP hacia Y	(SP) $\Rightarrow$Y
TXS	Transfiere X hacia SP	(X) $\Rightarrow$SP
TYS	Transfiere Y hacia SP	(Y) $\Rightarrow$SP

Las instrucciones de intercambio, intercambian el contenido de un par de registros.

Instrucciones de Intercambio		
Nemonico	**Función**	**Operación**
EXG	Intercambia Registro a Registro	(A, B, CCR, D, X, Y OR SP) $\Rightarrow$(A, B, CCR, D, X, Y OR SP)
XGDX	Intercambia D con X	(D) $\Rightarrow$(X)
XGDY	Intercambia D con Y	(D) $\Rightarrow$(Y)

Esta instrucción se usa en casos especiales, como cuando se va a transferir números 2dos complemento con signo extendido.

Instrucción de Extensión de Signo		
Nemonico	**Función**	**Operación**
SEX	Extensión de signo a operando de 8 bits	Signo extendido (A, B o CCR) $\Rightarrow$ (D, X, Y OR SP)

Las instrucciones Move, mueven o copian bytes de datos o palabras desde una fuente a un destino en memoria. Existen 6 combinaciones de direccionamiento (IMM a EXT, IMM a IDX, EXT a EXT, EXT a IDX, IDX a EXT, IDX a IDX) para especificar la fuente o la dirección de destino.

Instrucciones de Movimiento		
Nemonico	**Función**	**Operación**
MOVB	Mueve un Byte (8 bits)	$(M_1) \Rightarrow M_2$
MOVW	Mueve una Palabra (16 bits)	$(M: M+M_1) \Rightarrow M: M+ 1_2$

Las instrucciones de Suma de 8 bits con signo y sin signo se pueden llevar a cabo entre registros o entre registros y acumuladores. Las instrucciones con el bit de acarreo del CCR nos facilitan realizar operaciones con mayor precisión.

Las instrucciones de Resta de 8 y 16 bits se desarrollan entre registros o entre registros y memoria. Las instrucciones con el bit de acarreo del CCR nos facilitan realizar operaciones con mayor precisión.

Instrucciones de Suma		
Nemonico	**Función**	**Operación**
ABA	Suma B a A	$(A)+(B) \Rightarrow A$
ABX	Suma B a X	$(B)+(X) \Rightarrow X$
ABY	Suma B a Y	$(B)+(Y) \Rightarrow Y$
ADCA	Suma con acarreo a A	$(A)+(M)+C \Rightarrow A$
ADCB	Suma con acarreo a B	$(B)+(M)+C \Rightarrow B$
ADDA	Suma sin acarreo a A	$(A)+(M) \Rightarrow A$
ADDB	Suma sin acarreo a A	$(B)+(M) \Rightarrow B$
ADDD	Suma a D	$(A:B)+(M:M+1) \Rightarrow A:B$
Instrucciones de Resta		
SBA	Substrae B de A	$(A)-(B) \Rightarrow A$
SBCA	Substrae con préstamo a A	$(A)-(M)-(C) \Rightarrow A$
SBCB	Substrae con préstamo a B	$(A)-(M)-C \Rightarrow B$
SUBA	Substrae memoria a A	$(A)-(M) \Rightarrow A$
SUBB	Substrae memoria a B	$(A)-(M) \Rightarrow B$
SUBD	Substrae memoria a D	$(D)+(M:M+1) \Rightarrow D$

Las instrucciones de decremento e incremento estad optimizadas para operaciones de sumas y restas de 8 y 16 bits. Se utilizan generalmente para implementar contadores.

Instrucciones de Decremento		
Nemonico	**Función**	**Operación**
DEC	Decrementa una Memoria	(M)-$01 ⇒ M
DECA	Decrementa A	(A)-$01 ⇒ A
DECB	Decrementa B	(B)-$01 ⇒ B
DES	Decrementa SP	(SP)-$0001 ⇒ SP
DEX	Decrementa X	(X)-$0001 ⇒ X
DEY	Decrementa Y	(Y)-$0001 ⇒ Y
Instrucciones de Incremento		
INC	Incrementan una Memoria	(M)+$01 ⇒ M
INCA	Incrementa A	(A)+$01 ⇒ A
INCB	Incrementa B	(B)+$01 ⇒ B
INS	Incrementa SP	(SP)+$0001 ⇒ SP
INX	Incrementa X	(X)+$0001 ⇒ X
INY	Incrementa Y	(Y)+$0001 ⇒ Y

Las instrucciones de prueba y comparación desarrollan una substracción entre un par de registros o entre un registro y memoria, el resultado no se almacena pero si altérale código de condición se activa por esta operación.

Estas instrucciones son utilizadas para establecer condiciones para instrucciones de saltos tipo Branco.

Instrucciones de Comparación		
Nemonico	**Función**	**Operación**
CBA	Compara A con B	(A)-(B)
CMPA	Compara A con una memoria	(A)-(M)
CMPB	Compara B con una memoria	(B)-(M)
CPD	Compara D con memoria de 16 bits	(A:B)--(M:M+1)
CPS	Compara SP con memoria de 16 bits	(SP)--(M:M+1)
CPX	Compara X con memoria de 16 bits	(X)--(M:M+1)
CPY	Compara Y con memoria de 16 bits	(Y)--(M:M+1)
Instrucciones de Prueba		
TST	Prueba si M=0 o M<0	(M)-$00
TSTA	Prueba si A=0 o M<0	(A)-$00
TSTB	Prueba si B=0 o M<0	(B)-$00

Este tipo de instrucciones desarrollan operaciones lógicas entre un acumulador o CCR con un valor de memoria.

Instrucciones de Lógica Boleana		
Nemonico	**Función**	**Operación**
ANDA	AND A con memoria	(A)*(M) $\Rightarrow$ A
ANDB	AND B con memoria	(B)*(M) $\Rightarrow$ B
ANDCC	AND CCR con memoria (limpia CCR)	(CCR)*(M) $\Rightarrow$ CCR
EORA	OR Exclusivo de A con memoria	(A)*(M) $\Rightarrow$ A
EORB	OR Exclusivo de B con memoria	(B)*(M) $\Rightarrow$ B
ORAA	OR de A con memoria	(A)+(M) $\Rightarrow$ A
ORAB	OR de B con memoria	(B)+(M) $\Rightarrow$ A
ORCC	OR CCR con memoria (activa bits CCR)	(CCR)+(M) $\Rightarrow$ CCR

Cada una de estas instrucciones desarrolla una función binaria específica dentro de un valor de un acumulador o una memoria. Las operaciones de Borrado o limpieza colocan un 0 como valor, las operaciones de complemento remplazan el valor con uno complemento, y las operaciones de negación remplazan el valor con dos complemento.

Instrucciones de Negación, Complemento y Borrado de Bits		
Nemonico	**Función**	**Operación**
CLC	Borra bit C en CCR	0 $\Rightarrow$ C
CLI	Borra bit I en CCR	0 $\Rightarrow$ 1
CLR	Borra memoria	\$00 $\Rightarrow$ M
CLRA	Borra A	\$00 $\Rightarrow$ A
CLRB	Borra B	\$00 $\Rightarrow$ B
CLV	Borra bit V en CCR	0 $\Rightarrow$ V
COM	Uno complemento a memoria	\$FF-(M) $\Rightarrow$ M $\oplus$ (M) $\Rightarrow$ M
COMA	Uno complemento a A	\$FF-(A) $\Rightarrow$ A $\oplus$ (A) $\Rightarrow$ A
COMB	Uno complemento a B	\$FF-(B) $\Rightarrow$ B $\oplus$ (B) $\Rightarrow$ B
NEG	Dos complemento a una memoria	\$00-(M) $\Rightarrow$ M $\oplus$ (M)+1 $\Rightarrow$ M
NEGA	Dos complemento a A	\$00-(A) $\Rightarrow$ A $\oplus$ (A)+1 $\Rightarrow$ A
NEGB	Dos complemento a B	\$00-(B) $\Rightarrow$ B $\oplus$ (B)+1 $\Rightarrow$ B

Las instrucciones de multiplicación son de 8 y 16 bits con signo, si se multiplican operandos de 8 bits se obtiene un producto de 16 bits, y son operandos de 16 bits se obtiene un producto de 32 bits.

Las instrucciones de división entera y fraccionaria tienen un dividendo, divisor, cociente y residuo de 16 bits. Las instrucciones de división extendida usan un dividendo de 32 bits y un divisor de 16 bits para producir un cociente y un residuo de 16 bits.

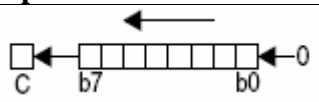
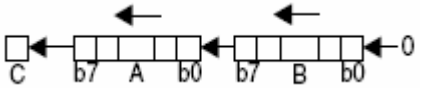
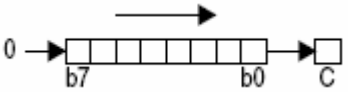
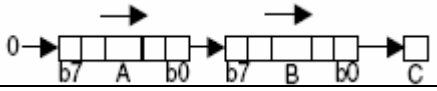
Instrucciones de Multiplicación		
Nemonico	**Función**	**Operación**
EMUL	Multiplica 16X16 sin signo	(D)X (Y) ⇒ Y: D
EMULS	Multiplica 16X16 con signo	(D)X (Y) ⇒ Y: D
MUL	Multiplica 8X8 sin signo	(A)X(B) ⇒ A:B
Instrucciones de División		
EDIV	Divide 32 entre 16 sin signo	(Y: D) / (X) ⇒ Y, Residuo ⇒ D
EDIVS	Divide 32 entre 16 con signo	(Y: D) / (X) ⇒ Y, Residuo ⇒ D
FDIV	Divide 16 entre 16 con fracción	(D) / (X) ⇒ X, Residuo ⇒ D
IDIV	División entera de 16 entre 16 sin signo	(D) / (X) ⇒ X, Residuo ⇒ D
IDIVS	División entera de 16 entre 16 con signo	(D) / (X) ⇒ X, Residuo ⇒ D

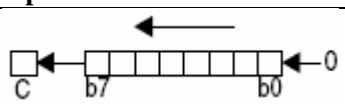
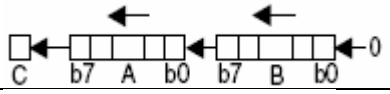
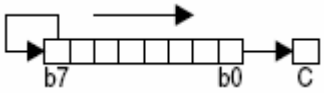
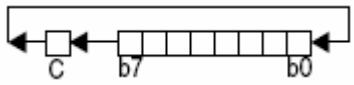
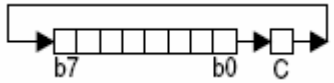
Las operaciones de prueba y manipulación de bits usan un valor enmascarado para probar o cambiar el valor de un bit individual dentro de un acumulador o una memoria.

Instrucciones como BITA y BITB proveen los medios convenientes para pruebas de bits sin alterar los operandos.

Instrucciones de Prueba y Manipulación de Bits		
Nemonico	**Función**	**Operación**
BCLR	Borra los bits en una memoria	(M)*(mm) ⇒ M
BITA	Prueba los bits de A	(A)*(M)
BITB	Prueba los bits de B	(B)*M
BSET	Activa los bits en memoria	(M)+(mm) ⇒ M

Existen instrucciones para rotación y desplazamientos para todos los acumuladores y para bytes de memoria.

Instrucciones de Desplazamiento		
Nemonico	**Función**	**Operación**
LSL	Desplaza a la izquierda a M	
LSLA	Desplaza a la izquierda a A	
LSLB	Desplaza a la izquierda a B	
LSLD	Desplaza a la izquierda a D	
LSR	Desplaza a la derecha a M	
LSRA	Desplaza a la derecha a A	
LSRB	Desplaza a la derecha a B	
LSRD	Desplaza a la derecha a D	

Instrucciones de Desplazamiento Aritmético		
Nemonico	**Función**	**Operación**
ASL	Desplazamiento aritmético a la izquierda de M	
ASLA	Desplazamiento aritmético a la izquierda de A	
ASLB	Desplazamiento aritmético a la izquierda de B	
ASLD	Desplazamiento aritmético a la izquierda de D	
ASR	Desplazamiento aritmético a la derecha de M	
ASRA	Desplazamiento aritmético a la derecha de A	
ASRB	Desplazamiento aritmético a la derecha de B	
Instrucciones de Rotación		
ROL	Rotación a la izquierda de M con acarreo	
ROLA	Rotación a la izquierda de A con acarreo	
ROLB	Rotación a la izquierda de b con acarreo	
ROR	Rotación a la derecha de M con acarreo	
RORA	Rotación a la derecha de A con acarreo	
RORB	Rotación a la derecha de B con acarreo	

Las instrucciones de salto corto operan cuando una condición esperada es encontrada y agregan un desplazamiento de 8 bits al contador de programa, por lo que el programa se continuara ejecutando a partir de la nueva dirección.

El rango numérico del desplazamiento corto es de $80 (-128) hasta $7F (127) para direccional la próxima localidad de memoria después de obtener el valor del desplazamiento.

Instrucciones de Salto Incondicional		
Nemonico	**Función**	**Operación**
BRA	Salta Siempre	1 =1
BRN	Nunca Salta	1 =0
Saltos Simples		
BCC	Salta si acarreo es igual a 0	C =0
BCS	Salta si acarreo igual a 1	C =1
BEQ	Salta si es igual	Z =1
BMI	Salta si es menor	N =1
BNE	Salta si no es igual	Z =0
BPL	Salta si es positivo	N =0
BVC	Salta si no existe desbordamiento	V =0
BVS	Salta si existe un desborde	V =1

Saltos Sin Signo			
Nemonico	**Función**	**Relación**	**Operación**
BHI	Salta si es mayor	R > M	C +Z =0
BHS	Salta si es mayor o igual	R ≥ M	C =0
BLO	Salta si es menor	R< M	C =1
BLS	Salta si es menor o igual	R ≤M	C +Z =1
Saltos con Signo			
BGE	Salta si mayor que o igual	R ≥ M	N ⊕ V =0
BGT	Salta si mayor que	R> M	Z + (N ⊕ V) =0
BLE	Salta si menor que o igual	R ≤ M	Z +(N ⊕ V) =1
BLT	Salta si menor que	R < M	N ⊕ V =1

Las instrucciones de salto largo operan cuando una condición esperada es encontrada y agregan un desplazamiento de 16 bits al contador de programa, por lo que el programa se continuara ejecutando a partir de la nueva dirección.

El rango numérico del desplazamiento corto es de $8000 (-32,768) hasta $7FFF (32,768) para direccional la próxima localidad de memoria después de obtener el valor del desplazamiento.

Instrucciones de Saltos Largos Incondicionales		
Nemonico	Función	Operación
LBRA	Salto largo incondicional	1 =1
LBRN	Salto largo nunca	1 =0
Saltos Largos Simples		
LBCC	Salto largo si acarreo igual a 0	C =0
LBCS	Salto largo si acarreo igual a 1	C =1
LBEQ	Salto largo si igual	Z =1
LBMI	Salto largo si menor	N = 1
LBNE	Salto largo si no es igual	Z =0
LBPL	Salto largo si es positivo	N =0
LBVC	Salto largo si no existio desbordamiento	V =0
LBVS	Salto largo si existe desbordamiento	V =1
Saltos Largos Sin Signo		
LBHI	Salto largo si mayor	C +Z = 0
LBHS	Salto largo si mayor o igual	C =0
LBLO	Salto largo si menor	Z =1
LBLS	Salto largo si menor o igual	C +Z = 1

Saltos Largos con Signo		
LBGE	Salto largo si mayor que o igual	$N \oplus V = 0$
LBGT	Salto largo si mayor que	$Z + (N \oplus V) = 0$
LBLE	Salto largo si menor que o igual	$Z + (N \oplus V) = 1$
LBLT	Salto largo si menor que	$N \oplus V = 1$

Los saltos de condición de bits son realizados cuando un bit en dentro de un byte de memoria se encuentra en un estado especifico. Se utiliza un operando enmascarado para probar esa localidad especifica. Si todos los bits de esa localidad corresponde a cada uno de los bits de la mascara que estén activos (BRSET) o borrados (BRCLR), el salto es realizado.

El rango numérico del desplazamiento corto es de $80 (-128) hasta $7F (127) para direccionar la próxima localidad de memoria después de obtener el valor del desplazamiento.

Instrucciones de Salto con Condición de Bits		
Nemonico	**Función**	**Operación**
BRCLR	Salta si los bits seleccionados son iguales a 0	$(M(mm) = 0$
BRSET	Salta si los bits seleccionados son iguales a 1	$(\overline{M})*(mm) = 0$

Las instrucciones de ciclo primitivo fueron pensadas para realizar el conteo de los saltos. Estas instrucciones prueban si el valor de un contador dentro de un registro o acumulador es 0 o diferente de 0 dependiendo de la condición de salto.

El rango numérico para el desplazamiento de 9 bits esta dentro de $100 (256) a $0FF (255) para direccionar la próxima localidad de memoria después de obtener el valor del desplazamiento.

Instrucciones de Ciclos Primitivos		
Nemonico	**Función**	**Operación**
DBEQ	Decrementa contador si salto = 0 (contador = A, B, D, X, Y, ó SP)	(contador) – 1 $\Rightarrow$ contador Sí (contador) = 0, salta; de lo contrario continua la siguiente instrucción.
DBNE	Decrementa un contador si salto ≠0 (contador = A, B, D, X, Y, ó SP)	(contador) – 1 $\Rightarrow$ contador Sí (contador) no = 0, salta; de lo contrario continua la siguiente instrucción.
IBEQ	Incrementa un contador si salto = 0 (contador = A, B, D, X, Y, ó SP)	(contador) + 1 $\Rightarrow$ contador Sí (contador) = 0, salta; de lo contrario continua la siguiente instrucción.

IBNE	Incrementa un contador si salto ≠0 (contador = A, B, D, X, Y, ó SP)	(contador) + 1 ⇒ contador Sí (contador) no = 0, salta; de lo contrario continua la siguiente instrucción.
TBEQ	Prueba un contador y brinca si = 0 (contador = A, B, D, X, Y, ó SP)	Sí (contador) = 0, salta; de lo contrario continua la siguiente instrucción.
TBNE	Prueba el contador y salta si ≠0 (contador = A, B, D, X, Y, ó SP)	Sí (contador) no = 0, salta; de lo contrario continua la siguiente instrucción.

Las instrucciones JUMP (salto) causan un cambio inmediato en la ejecución del programa. Estas instrucciones cargan al contador del programa con una dirección comprendida dentro de los 64 Kbytes de memoria del mapa.

Las instrucciones para manejo de subrutina optimizan el proceso de transferir el control de un segmento de código que desarrolla una tarea particular que al ser finalizada debe de incluir una instrucción de retorno de subrutina como es (RTS) que regresa el contador del programa a la siguiente línea del programa donde se estableció el salto (JSR, BSR, CALL).

Instrucciones de Subrutina y Salto (Jump)		
Nemonico	**Función**	**Operación**
BSR	Salto a subrutina	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ Dirección de subrutina $\Rightarrow PC$
CALL	Llamada a subrutina en memoria expandida	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP$ $(PPAGE) \Rightarrow M_{(SP)}$ Page $\Rightarrow$ PPAGE Dirección de subrutina $\Rightarrow PC$
JMP	Salto	Dirección $\Rightarrow PC$
JSR	Salto a subrutina	$SP - 2 \Rightarrow SP$ $RTNH : RTNL \Rightarrow M_{(SP)} : M_{(SP+1)}$ Dirección de subrutina $\Rightarrow PC$
RTC	Regreso de llamada	$M(SP) \Rightarrow$ PPAGE $SP + 1 \Rightarrow SP$ $M_{(SP)} : M_{(SP+1)} \Rightarrow PC_H : PC_L$ $SP + 2 \Rightarrow SP$
RTS	Regreso desde subrutina	$M_{(SP)} : M_{(SP+1)} \Rightarrow PC_H : PC_L$ $SP + 2 \Rightarrow SP$

Las instrucciones de retorno de interrupción RTI se utilizan para finalizar la rutina de servicio a una interrupción, estas instrucciones restauran el valor de CCR, B, A, X, Y, y regresa la dirección almacenada en la pila para continuar con el programa normal en la ultima dirección, antes de que se ejecutara la interrupción.

Instrucciones de Interrupción		
Nemonico	**Función**	**Operación**
RTI	Regreso de interrupción	$(M_{(SP)}) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow B : A; (SP) + \$0002 \Rightarrow SP\ (M_{(SP)}: M_{(SP+1)}) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow SP\ (M_{(SP)}: M_{((SP+1))} \Rightarrow PC_H : PC_L; (SP) + \$0002 \Rightarrow SP\ (M_{(SP)}) : M_{(SP+1))} \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow SP$
SWI	Interrupción por software	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)}: M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)}: M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : XL \Rightarrow M_{(SP)}: M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)}: M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$
TRAP	Unimplemented opcode interrupt	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)}: M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : YL \Rightarrow M_{(SP)}: M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)}: M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)}: M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$

Las instrucciones de código de condición son instrucciones especiales de matemáticas y transferencia que permiten cambiar el estado del código de condición.

Instrucciones de Código de Condición		
Nemonico	**Función**	**Operación**
ANDCC	AND CCR con memoria	$(CCR) \bullet (M) \Rightarrow CCR$
CLC	Borra bit C	$0 \Rightarrow C$
CLI	Borra bit I	$0 \Rightarrow I$
CLV	Borra V bit	$0 \Rightarrow V$
ORCC	OR CCR con memoria	$(CCR) + (M) \Rightarrow CCR$
PSHC	Coloca CCR dentro de la Pila	$(SP) - 1 \Rightarrow SP; (CCR) \Rightarrow M(SP)$
PULC	Carga CCR con la Pila	$(M(SP)) \Rightarrow CCR; (SP) + 1 \Rightarrow SP$
SEC	Activa bit C	$1 \Rightarrow C$
SEI	Activa bit I	$1 \Rightarrow I$
SEV	Activa bit V	$1 \Rightarrow V$
TAP	Transfiere A hacia CCR	$(A) \Rightarrow CCR$
TPA	Transfiere CCR hacia A	$(CCR) \Rightarrow A$

Las instrucciones de paro y espera ponen en estado de espera al procesador. La instrucción STOP coloca dentro de la pila el contenido de los registros y acumuladores, también detienen todos los relojes del sistema.

La instrucción WAI coloca dentro de la pila el contenido de los registros y acumuladores, posteriormente espera la solicitud de un servicio de atención a una interrupción, mas sin embargo los relojes continúan activos.

Instrucciones de Paro y Espera		
Nemonico	**Función**	**Operación**
STOP	Alto	$SP - 2 \Rightarrow SP$; $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP$; $CCR \Rightarrow M_{(SP)}$ Para el Reloj del CPU
WAI	Espera una interrupción	$SP - 2 \Rightarrow SP$; $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP$; $CCR \Rightarrow M_{(SP)}$

3.4.- Interrupciones

Las interrupciones son señales externas o internas de microcontrolador que indican al CPU la detención de ejecución del programa principal para atender a una rutina de servicio de interrupción.
Después de que se ejecuta la rutina de servicio de la interrupción el programa continúa con la ejecución del programa normal que estaba ejecutando antes de la interrupción.

Las direcciones de la rutina de servicio de interrupción se encuentran en una tabla de vectores de interrupción, localizada en la parte alta del mapa de memoria (Ver Tabla 3.4.4).

3.4.1-Tipo de interrupciones

Existen 3 tipos de interrupciones, las interrupciones no enmascarables, las no enmascarables y las interrupciones por software.

Interrupciones Enmascarables: Son interrupciones que pueden ser enmascaradas. Para permitirlas solamente es necesario poner en 0 el bit I del CCR y para inhibirlas se coloca un uno lógico en el mismo bit, lo que se logra mediante las instrucciones CLI pone en 0 el bit I y la instrucción SEI que pone en uno lógico el bit I del CCR.

Interrupciones No Enmascarables: Son interrupciones que no se pueden inhibir como la instrucción (señal en la terminal de reset) Reset y la instrucción (señal en la terminal de t) XIRQ.

Interrupciones por Software: Se producen cada que el Programa de Usuario lo indica. Este tipo de interrupción se produce con la instrucción SWI y es una interrupción no enmascarable.

3.4.2-Prioridad de Interrupciones

Las interrupciones tienen diferente prioridad, hay algunas que tienen una prioridad superior a las demás. Esto se debe a un caso donde se producen 2 interrupciones a la vez, donde se ejecuta la que tenga mayor prioridad en primer termino y después la segunda.

Las interrupciones no enmascarables (RESET, XIRQ) tienen mayor prioridad. Las demás interrupciones se determinan por el Hardware, aunque esta prioridad puede cambiarse escribiendo ciertos valores en algunos registros como el HPBRIO.

3.4.3-Proceso de Interrupción

Si al ejecutar programa ocurre una interrupción, si las interrupciones están permitidas (Bits X e I del CCR) se almacenan todos los registros en la pila, se inhiben las interrupciones y se obtiene de la tabla de vectores de interrupción la dirección a la que tiene que bifurcar el CPU.

Las interrupciones se inhiben para que no se produzcan interrupciones mientras se esta atendiendo a una rutina de servicio de interrupción, es decir, que no se produzca un anidamiento de interrupciones

Aunque el programador puede habilitar las interrupciones no es recomendable hacerlo ya que produciríamos un anidamiento de interrupciones. Si se está ejecutando la rutina de servicio de una interrupción enmascarable y se produce una interrupción no enmascarable, en este caso la interrupción no enmascarable tiene una prioridad máxima por lo que el CPU procede a atenderla dejando a la anterior en espera (colgada).

Cuando termina de atender la rutina de servicio de la no enmascarable vuelve con la enmascarable. Todas las rutinas de servicio de interrupción deben acabar con la instrucción RTI que indica al CPU que la interrupción ha sido atendida y que se puede retornar al programa anterior: La CPU toma de la pila todos los registros que había guardado al ejecutarse la interrupción y continúa ejecutando el programa que había sido interrumpido.

En el microcontrolador existen muchos periféricos integrados que pueden producir interrupciones en cualquier momento. Para generar una interrupción activan un bit de un determinado registro del periférico. Este bit indica a la CPU que el periférico en cuestión ha solicitado interrupción. Cuando el CPU pasa a ejecutar la rutina de servicio del periférico, la primera acción a tomar es desactivar ese bit. Si no se hace, al regresar de la interrupción con RTI, el bit seguirá activo y el CPU lo interpretará como una nueva interrupción, con lo que se vuelve a ejecutar la rutina de servicio y permanecería el CPU en un bucle infinito.

3.4.4-Vectores de interrupción

Dirección del Vector	Fuente de Interrupción	Mascara del CCR	Habilitación Local		Valor del HPBRIO
			Registro	Bits(s)	
$FFFE, $FFFF	Reset	None	None	None	—
$FFFC, $FFFD	COP clock monitor fail reset	None	COPCTL	CME, FCME	—
$FFFA, $FFFB	COP failure reset	None	None	COP rate	—
$FFF8, $FFF9	Unimplemented instruction trap	None	None	None	—
$FFF6, $FFF7	SWI	None	None	None	—
$FFF4, $FFF5	XIRQ	X bit	None	None	—
$FFF2, $FFF3	IRQ	I bit	INTCR	IRQEN	$F2
$FFF0, $FFF1	Real-time interrupt	I bit	RTICTL	RTIE	$F0
$FFEE, $FFEF	Timer channel 0	I bit	TMSK1	C0I	$EE
$FFEC, $FFED	Timer channel 1	I bit	TMSK1	C1I	$EC
$FFEA, $FFEB	Timer channel 2	I bit	TMSK1	C2I	$EA
$FFE8, $FFE9	Timer channel 3	I bit	TMSK1	C3I	$E8
$FFE6, $FFE7	Timer channel 4	I bit	TMSK1	C4I	$E6
$FFE4, $FFE5	Timer channel 5	I bit	TMSK1	C5I	$E4
$FFE2, $FFE3	Timer channel 6	I bit	TMSK1	C6I	$E2
$FFE0, $FFE1	Timer channel 7	I bit	TMSK1	C7I	$E0
$FFDE, $FFDF	Timer overflow	I bit	TMSK2	TOI	$DE
$FFDC, $FFDD	Pulse accumulator overflow	I bit	PACTL	PAOVI	$DC
$FFDA, $FFDB	Pulse accumulator input edge	I bit	PACTL	PAI	$DA
$FFD8, $FFD9	SPI serial transfer complete	I bit	SP0CR1	SPIE	$D8
$FFD6, $FFD7	SCI 0	I bit	SP0CR1	TIE, TCIE, RIE, ILIE	$D6
$FFD4, $FFD5	Reserved	I bit	—	—	$D4
$FFD2, $FFD3	ATD	I bit	ATDCTL2	ASCIE	$D2
$FFD0, $FFD1	BDLC	I bit	BCR1	IE	$D0
$FF80–$FFC1	Reserved (not implemented)	I bit	—	—	$80–$C0
$FFC2–$FFC9	Reserved (implemented)	I bit	—	—	$C2–$C8
$FFCA, $FFCB	Pulse accumulator B overflow	I bit	PBCTL	PBOVI	$CA
$FFCC, $FFCD	Modulus down counter underflow	I bit	MCCTL	MCZI	$CC
$FFCE, $FFCF	Reserved (implemented)	I bit	—	—	$CE

Tabla 3.4.4.1- Vectores de Interrupción

3.5.- Tipos de Lenguajes de Programación.

Existen diferentes tipos de lenguaje de programación, orientadas a diferentes tipos de aplicaciones, pero para la programación de los microcontroladores los más utilizados son los lenguajes de bajo, medio y alto nivel.

Entre los lenguajes de bajo nivel se encuentra el Lenguaje maquina y el lenguaje ensamblador, y el lenguaje Basic que también forman parte de los lenguajes de nivel medio. Y entre los lenguajes de Alto Nivel se encuentra el lenguaje C que aparte de tener la propiedad de funcionar como un intérprete entre el ordenador y el hombre también puede actuar a nivel binario es decir como un lenguaje de bajo nivel.

3.5.1.- Lenguaje Maquina

El lenguaje maquina es un lenguaje de 0s y 1s es decir, que solo puede ser comprendido por las computadoras. Sus instrucciones no son nada más que cadenas de números equivalentes a 0s y 1s, estas especifican las instrucciones a ejecutar, registros del procesador, localidades de memoria implicadas etc.

Cada tipo de ordenadores tiene sus diferentes versiones de lenguaje maquina, es decir, las instrucciones son especificas de cada fabricante, por ejemplo la instrucción de suma es diferente en el lenguaje maquina de motorola y en el de microchip.

La dificultad que tenia este tipo de programación para los programadores los llevo ha buscar lenguajes diferentes para la programación de los ordenadores, obteniendo así el lenguaje ensamblador entre otros.

3.5.2.-Lenguaje Ensamblador

El lenguaje ensamblador es considerado un lenguaje de nivel medio, donde no se programa con 0s y 1s como en el lenguaje maquina si no a través de instrucciones conocidas como mnemónicos.

Se inventó para facilitar la tarea de los primeros programadores que hasta ese momento tenían que escribir los programas directamente en código binario como programa.

El código simbólico puede parecer de difícil acceso, pero es más fácil de recordar e interpretar que el binario o el hexadecimal. Este código simbólico no puede ser ejecutado directamente por un ordenador, por lo que es preciso traducirlo previamente.

Pero la traducción es un proceso mecánico y repetitivo, que se presta a su realización por un programa de ordenador Los programas que traducen código simbólico al lenguaje de máquina se llaman ensambladores ("assembler", en inglés), porque son capaces de ensamblar el programa traducido a partir de procedimientos o subrutinas.

Mientras que una computadora reconoce la instrucción de máquina :

10110000 01100001

Para los programadores es mucho más fácil reconocer dicha instrucción empleando lenguaje ensamblador:

```
mov  al, 0x61
```

Que significa mover el valor hexadecimal 61 (97 decimal) al registro 'al'.

En este lenguaje de mnemónicos el programador debe conocer la arquitectura interna del ordenador para el que se programa tanto como su juego de instrucciones.

Hoy en día el lenguaje ensamblador es el lenguaje más utilizada para aplicaciones basadas en microcontroladores y microprocesadores.

3.5.3.-Lenguaje de Alto Nivel C

El lenguaje de alto nivel no usa cadenas de números o mnemónicos como instrucciones sino palabras ordinarias. El programador tampoco tiene conocimiento sobre la arquitectura interna del computador que se quiere programar ni conoce su juego de instrucciones. Es por eso los lenguajes de alto nivel necesitan un compilador que entienda el código escrito en alto nivel y lo traslade a bajo nivel para que pueda ser comprendido por el ordenador a programar.

Uno de los lenguajes de alto nivel mas utilizados es el Lenguaje C, en el que destaca una de sus principales ventajas en contra de los ensambladores que consiste en que a diferencia del lenguaje ensamblador el lenguaje C mantiene un estándar, es decir, el programador puede programar aplicaciones para diferentes tipos de ordenadores con el mismo código. También existen diferentes compiladores con características propias de tal forma que el programador pueda utilizar el de mayor conveniencia.

Ejemplo de Código C

```
#include <stdio.h>

main()
{
   printf("¡Hola, Mundo!\n");
}
```

3.6.- Software de Desarrollo MiniIDE Versión 1.17

MiniIDE consiste en un ambiente de desarrollo integrado junto con un cross-assembler (ensamblador). El ensamblador es un macro ensamblador para las familias de los microcontroladores 68HC11 y 68HC12. Esta herramienta fue diseñada con el propósito de ser utilizada principalmente con la tarjeta de evaluación del microcontrolador 68HC12B32 (M68EVB912B32EVB), sin embargo puede ser utilizada para la elaboración de proyectos con los microcontroladores HC11 y HC12.

Este programa incorpora un editor, una ventana de terminal y un cross ensamblador integrado. Los archivos fuentes pueden ser fácilmente editados, ensamblador y descargados dentro de los microcontroladores, lo que hace a este programa una herramienta ideal para pequeños proyectos con tarjetas de evaluación o tarjetas entrenadoras del microcontrolador 68HC12.

En la figura 3.6.1.1 podemos observar al MiniIDE con sus principales herramientas.

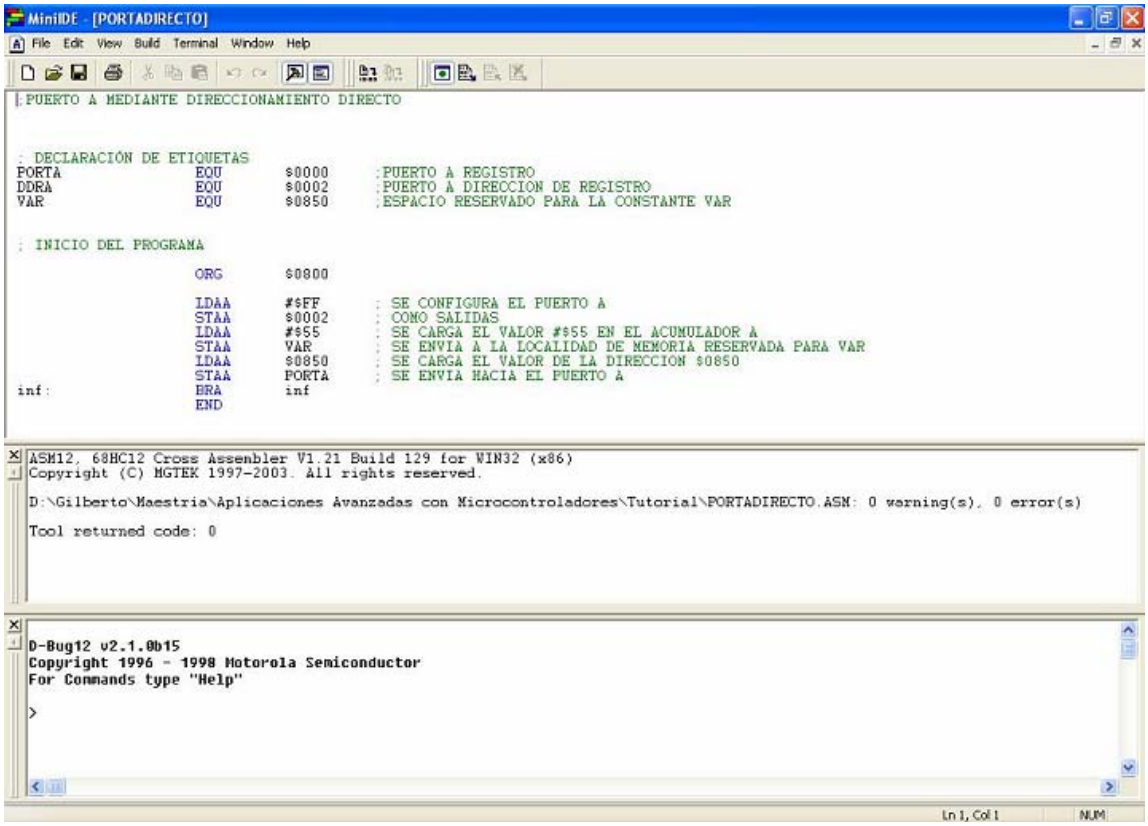


Figura 3.6.1.1- MiniIde

A continuación haremos una descripción de las herramientas básicas de MiniIDE.

Editor: Una de las principales ventajas de este programa es la capacidad de distinguir entre las instrucciones, operandos y etiquetas ya que cada uno de estos elementos se muestra por el programa con fuentes de diferente color. En la Figura 3.6.1 se muestra el Editor.

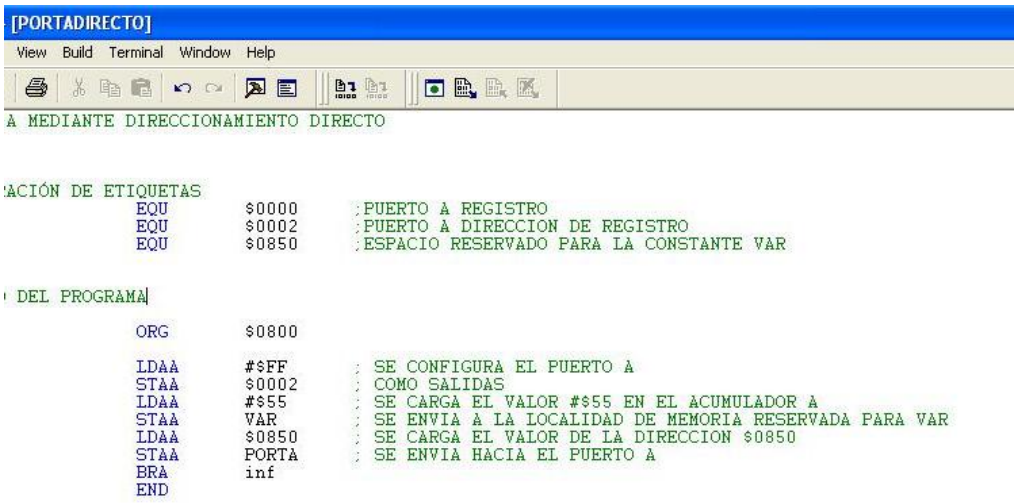


Figura 3.6.1.2.- Editor del MiniIDE

Ventana del Ensamblador. En esta ventana nosotros podemos observar el resultado de el ensamblado de nuestro programa, ENCASO de existir algún error el ensamblador nos arroja un mensaje de error con un numero especifico para el error, y en donde se encuentra. Lo que nos puede servir consultar la ayuda y buscar la causa de nuestro error.

En caso de no existir errores de ensamblaje el ensamblador nos arroja un mensaje de ensamblado exitoso. En la figura 3.6.2 podemos ver la ventana del ensamblador.

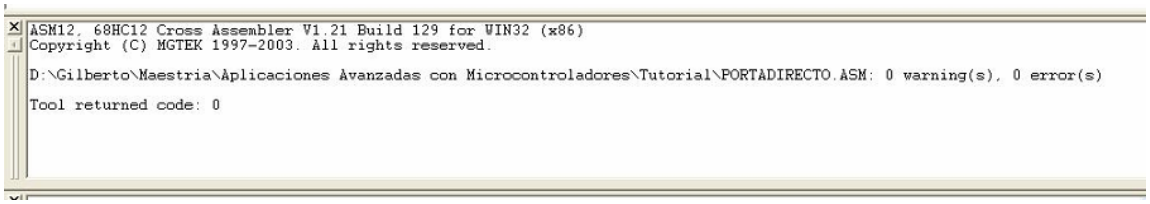


Figura 3.6.1.3.- Ventana del Ensamblador

Terminal: Esta ventana es una conexión en modo Hyper Terminal con el Dbug12 cargado en la memoria flash del microcontrolador. Esta conexión también puede hacerse manualmente con la Hyper Terminal de Windows pero al estar integrada en este ambiente de desarrollo, resulta mas practica aun, esta ventana se muestra en la figura 3.6.1.4.

Figura 3.6.1.4.- Ventana Hyper Terminal

Esta conexión hay que configurarla manualmente en el menú terminal-options con los siguientes valores: Port: COM1, Baud Rate: 9600, Data Bit: 8, Stop Bit: 1, Parity: None como se muestra en la figura 3.6.3.2. Al configurar la conexión es necesario aplicar un reset al microcontrolador para establecer la comunicación

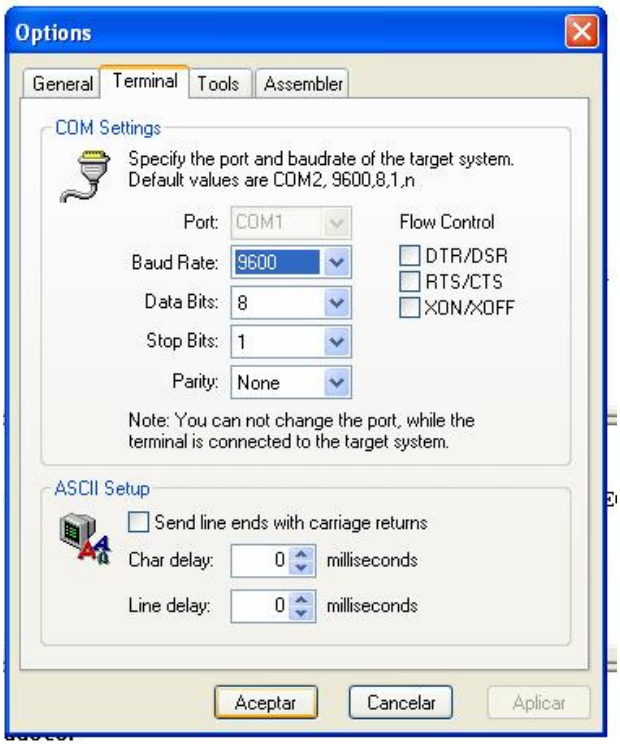


Figura 3.6.1.4.1.- Configuración de Terminal

3.6.1.- Elaboración de un Programa.

Un programa esta formado básicamente de dos partes un encabezado en el que incluimos una declaración de constantes e incluso librerías y un cuerpo de programa al cual lo constituyen principalmente etiquetas, instrucciones, operandos y comentarios.

El cuerpo del programa puede incluir subrutinas dentro de el o separadas, mismas que también están formadas por los mismos elementos que el cuerpo del programa.

A continuación describiremos las partes principales del siguiente ejemplo:

```
; PUERTO A MEDIANTE DIRECCIONAMIENTO DIRECTO

PORTA     EQU   $0000          ; PUERTO A REGISTRO
DDRA      EQU   $0002          ; PUERTO A DIRECCION DE REGISTRO
VAR       EQU   $0850          ; ESPACIO RESERVADO PARA LA CONSTANTE VAR

; INICIO DEL PROGRAMA

          ORG       $0800

; CUERPO DE EL PROGRAMA

          LDAA      #$FF      ; SE CONFIGURA EL PUERTO A
          STAA      $0002     ; COMO SALIDAS
          LDAA      #$55      ; SE CARGA EL VALOR #$55 EN EL ACUMULADOR A
          STAA       VAR      ; SE ENVIA A LA LOCALIDAD DE MEMORIA
                               RESERVADA PARA VAR
          LDAA      $0850     ; SE CARGA EL VALOR DE LA DIRECCION $0850
          STAA      PORTA     ; SE ENVIA HACIA EL PUERTO A
inf:      BRA       inf
          END
```

En la parte de encabezado están declarados valores mediante directivas del lenguaje ensamblador, las palabras PORTA, DDRA, VAR son etiquetas que van a tener un valor constante. Después hay una parte intermedia en la cual se declara con la instrucción ORG la dirección de inicio de nuestro programa, el valor $0800 es un operando que especifica el valor de la dirección de inicio donde esta situada la memoria RAM en nuestro microcontrolador.

Posteriormente podemos observar en el cuerpo de nuestro programa mas instrucciones y operandos que nos sirven para inicializar recursos del microcontrolador como en el caso de las dos primeras líneas LDAA #$FF, STAA $0002. Después podemos observar que existe un comentario separado por un **;** de las instrucciones de nuestro programa. Durante todo el desarrollo podemos hacer comentarios comenzándolos siempre por un símbolo de ; que sirve para que el ensamblador reconozca las palabras posteriores a este símbolo como un comentario.

Para probar el ejemplo anterior hay que escribirlo en el editor del programa MiniIDE y guardarlo con extensión .ASM de lo contrario al momento de ensamblarlo el ensamblador no encontrara el archivo con esta extensión y nos arrojara un mensaje de error.

3.6.2.- Ensamblando el programa.

A programa escrito en lenguaje ensamblador se le llama archivo fuente, este tiene extensión .ASM. Y al ser ensamblado se crean dos archivos mas uno con extensión .LST y otro con la extensión .S19 los cuales describimos a continuación:

El archivo de listado (.LST): es un archivo que se genera de dos partes principales, en la parte izquierda se observan 3 columnas, una donde se encuentran las direcciones en las que es colocado el programa y otras 2 que corresponden al valor en hexadecimal a la instrucción ejecutada y al operando de la instrucción.

Este archivo de listado también contiene los mnemónicos de es programa así como las etiquetas que fueron declaradas por el programador. A continuación mostramos un pequeño ejemplo de un archivo listado.

Ejemplo de Archivo de Listado .LST

```
0800   86  FF     LDAA   #$FF        ; SE CONFIGURA EL PUERTO A
0802   5A  02     STAA   $0002       ; COMO SALIDAS
0804   86  01     LDAA   #$01        ; SE CARGA EL VALOR #$01 EN EL ACUMULADOR A
0806   5A  00     STAA   $0000       ; SE ENVIA HACIA EL PUERTO A
```

Archivo .S19: Este archivo contiene al programa en lenguaje maquina con números en formato hexadecimal, que es resultado del ensamblado de nuestro programa escrito en mnemónicos. La extensión .S19 es la extensión valida de lenguaje maquina para la familia de productos de motorola, en algunos otros microcontroladores como los microchip este archivo tiene una extensión .HEX.

A continuación escribimos un segmento de código en formato .S19 el cual no es nada más que una ristra de números que son interpretadas por el microcontrolador.

Ejemplo de código en formato .S19.

```
S0030000FC
S10D080086FF5A0286555A0020FEB6
S9030000FC
```

3.6.3.- Programación en Modo Autónomo

Cuando se requiera desarrollar una aplicación cuyas características requieran que el microcontrolador sea utilizado en modo autónomo, puede utilizarse la memoria EEPROM de nuestro microcontrolador para alojar el programa a ser ejecutado. De esta forma una vez que se ha programado el microcontrolador, este continuara programado después de un reset o un apagado de la fuente de alimentación.

Los pasos principales para programar de forma autónoma a un microcontrolador son los siguientes:

1.- El primer paso consiste en adecuar el programa fuente inicializando algunas opciones para que pueda ser ejecutado, estas opciones son:

a) Cambiar el origen a la EEPROM
   ```
   ORG    $0D00
   ```
b) Deshabilitar el Watchdog
   ```
   CLR    COPCTL
   ```
c) Inicializar el Stack Pointer
   ```
   LDS    #$0C00
   ```

Estas tres líneas de código deben estar siempre al inicio del programa para asegurarnos con ello de obtener un funcionamiento correcto.

2.- El segundo paso consiste simplemente en ensamblar nuestro programa para obtener el archivo de salida con formato .s19.

3.- El tercer pasó y el de mayor importancia consiste en configurar los puentes (jumpers) JP14 y JP16 de forma que al continuar después de un reset, el microcontrolador arranque en modo de programación boot loader.

Normalmente al estar ejecutándose el debug 12 estos puentes están colocados como se muestra en la figura 3.6.3.1, por lo que es necesario cambiarlos a la posición que se

muestra en la figura 3.6.3.2 y posteriormente aplicar un reset con lo que podremos cargar nuestro programa dentro de la memoria EEPROM.

Figura 3.6.3.1.-Modo Dbug12

Figura 3.6.3.2.- Modo de Programación

4.- El cuarto paso consiste en preparar al microcontrolador para la carga del programa. Una vez seguidos los pasos anteriores, en nuestra ventana de Terminal podremos observar un mensaje como el que se muestra en la figura 3.6.3.3. Este mensaje indica que esta operando el modo de programación boot loader. Por lo que bastara con escribir la letra **L + ENTER** para que el microcontrolador este listo para recibir el código del programa a ser enviado (ver figura 3.6.3.4).

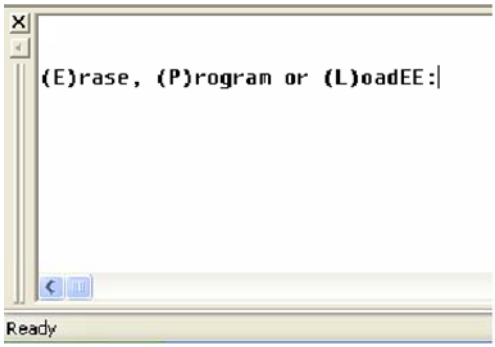


Figura 3.6.3.3.- Ventana Terminal

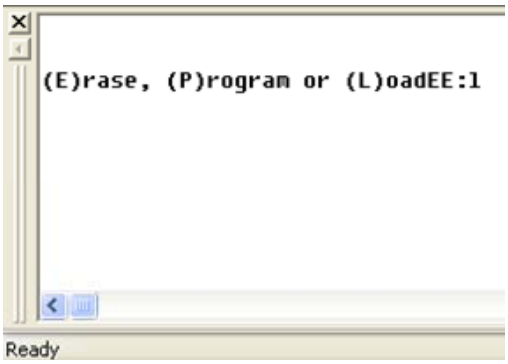


Figura 3.6.3.4.- Opción L

5.- El quinto paso consiste en transferir el fichero de salida .s19 a nuestro microcontrolador por lo que es necesario dirigirse a la opción Terminal> del menú principal y después a la opción download file to EEPROM> como se muestra en la figura 3.6.3.4. Posteriormente seleccionaremos el archivo que va a ser cargado dentro del microcontrolador (ver figura 3.6.3.5).

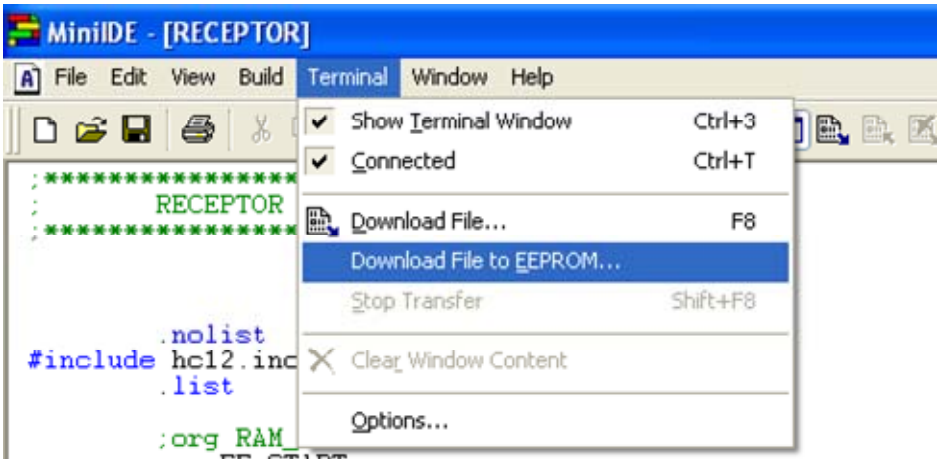


Figura 3.6.3.5.- Cargar en Eeprom

Una vez que se ha transferido el archivo hacia el microcontrolador observaremos el mensaje “Programmed” que aparecerá en la ventana de la Terminal, semejante al que podemos ver en la figura 3.6.3.6. Este mensaje nos indica que el microcontrolador ha sido programado satisfactoriamente.

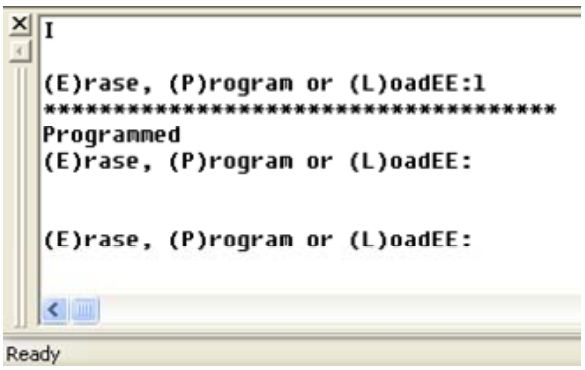


Figura 3.6.3.6.- Programación.

6.- Finalmente el sexto y último paso consiste simplemente en colocar los puentes JP14 y JP15 como se muestra en la figura 3.6.3.7 y después aplicar un reset.

En este punto después de ejecutar el reset podremos observar el funcionamiento de nuestro programa, el cual será satisfactorio si se han seguido de forma correcta los pasos anteriores y el programa actúa de la forma prevista.

Figura 3.6.3.7.- Programa en EEPROM

CAPITULO 4

RECURSOS INTERNOS DEL MICROCONTROLADOR

4.1.- EEPROM

La memoria EEPROM (por sus siglas en ingles **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory) es un tipo especial de memoria PROM que puede ser programada por una carga eléctrica. La característica principal de esta memoria es que es capaz de retener los datos inclusive después de un corto de energía, por lo que es utilizada grabar programas que puedan ejecutarse de forma autónoma en cualquier dispositivo que la incluya.

El microcontrolador 68HC12B32 cuenta con una memoria EEPROM de 768 Bytes situados desde la dirección $0D00 hasta $0FFF, que aunque es limitada puede servir para muchas aplicaciones. El acceso para la lectura de la memoria EEPROM es habilitado o deshabilitado mediante el bit de control EEON situado en el registro de inicialización de la EEPROM INITEE. También puede habilitarse la escritura o el borrado mediante el registro de control de la EEPROM.

4.1.1.- Registros de la EEPROM

Registro de configuración del modulo EEPROM (EEMCR)

Dirección $00F0

Bit 7	6	5	4	3	2	1	Bit 0
1	1	1	1	1	EESWAI	PROTLCK	EERC

Figura 4.1.1.- Registro de configuración del modulo EEPROM (EEMCR)

EESWAI: Bit de Stop en Modo de Espera

0= El modulo no es afectado durante el modo de espera.
1= El modulo detiene el cronometraje durante el modo de espera.

PROTLCK: Bit de Protección de Block contra Escritura
0 = El bits de protección de block y el bit de borrado de bloque pueden ser escritos.
1 = Los bits de protección de bloque están bloqueados
Lectura en cualquier momento. Escritura en modo de operación normal (SMODN=1;).
Activación y borrado puede realizarse en cualquier momento en modo especial (SMODN=0).

EERC — Bit de Reloj para la Carga de capacitores de la EEPROM.
0 = El reloj del sistema es usado como una fuente interna para la carga de los capacitores. El reloj oscilador RC se detiene.
1 = el reloj oscilador interno RC maneja la carga de los capacitores.
El oscilador RC se requiere cuando la frecuencia del sistema de reloj es mas bajo que fPROG.
Lectura y escritura en cualquier momento.

Registro de Protección de Block EEPROT

Dirección $00F1

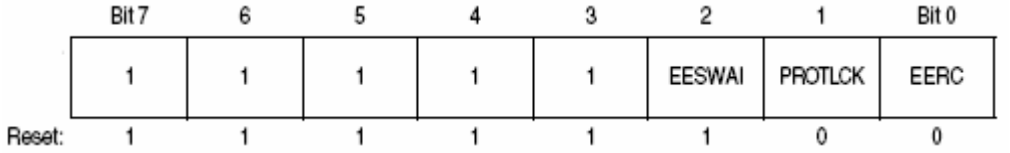


Figura 4.1.2.-Registro de Protección de Block EEPROT

El registro EEPROT previene la escritura accidental en la memoria EEPROM.

La lectura de este registro puede ser durante cualquier momento, pero la escritura permanecerá deshabilitada durante la programación de la memoria, EEPGM = 0. Solo podemos escribir en esta memoria cuando EEPGM = 0 y PROTLCK = 0.

BPROT4 - BPROT0 : EEPROM bits de protección del block.

0 = El block de memoria EEPROM asociado con este bit puede ser programado y borrado.

1 = El block de memoria EEPROM asociado con este bit esta protegido desde el inicio de un proceso de programación o borrado.

Tabla 4.1.- Block de protección de los 768 bytes de EEPROM

Nombre del Bit	Block Protegido	Tamaño del Block
BPROT4	$0D00 to $0DFF	256 bytes
BPROT3	$0E00 to $0EFF	256 bytes
BPROT2	$0F00 to $0F7F	128 bytes
BPROT1	$0F80 to $0FBF	64 bytes
BPROT0	$0FC0 to $0FFF	64 bytes

Registro de Control de la EEPROM

Dirección: $00F3

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	BULKP	0	0	BYTE	ROW	ERASE	EELAT	EEPGM
Reset:	1	0	0	0	0	0	0	0

Figura 4.1.3.-Registro de Control de la EEPROM

BULKP: Bit de Protección contra Borrado de Bloque.

0 = El bloque de la EEPROM puede ser borrado.

1 = La EEPROM esta protegida contra borrado de bloque o hilera.

Lectura en cualquier momento. Escritura en cualquier momento si EEPGM = 0 y PROTLCK = 0.

BYTE: Bit de Borrado de Byte o Palabra

0 = Borrado de bloque o hilera habilitado.

1 = Solamente se borra un byte o una palabra.

Lectura en cualquier momento. Escritura en cualquier momento si EEPGM = 0.

ROW — Bit de Borrado de Hilera o Bloque (cuando BYTE = 0).

0 = Borrado completo de un arreglo de EEPROM.

1 = Borrado solamente de una hilera de 32 bytes.

Lectura en cualquier momento. Escritura en cualquier momento si EEPGM = 0.

ERASE: Bit de Control de Borrado

0 = Configura la EEPROM para programación y lectura.

1 = Configura la EEPROM para borrado seguro.

EELAT: Bit de Control de Candado (Latch) EEPROM.

0 = La EEPROM se configura como lectura normal.

1 = El bus de datos y direccionamiento se bloquea para programación y borrado de la EEPROM.

EEPGM: Bit de Habilitación para Programación y Borrado.
0 = Deshabilita el voltaje de programación y borrado de la EEPROM.
1 = aplica el voltaje de programación y borrado para la EEPROM.

Los bits BULKP, BYTE, ROW, ERASE, y EELAT no pueden cambiar de estado cuando el EEPGM esta activado. Al completarse un ciclo de programación, borrado o escritura el bit EEPGM se borra y el bit EELAT se checa antes de leer o programar un dato. Una escritura en la EEPROM no tiene efecto si el EEPGM esta activo.
Una operación de programación o borrado debe seguir la siguiente secuencia:
1.- Para escribir un Byte, Hilera o Borrar un valor, EELAT = 1.
2.- Escribir un Byte o palabra en una dirección de la EEPROM.
3.- Escribir EEPGM = 1.
4.-Para programar o borrar esperar un retraso de tiempo.
5.- Escribir EEPGM = 0 y EELAT = 0.

Para programar o borrar mas bytes o palabras sin leer la EEPROM solo se escribe EEPGM = 0 en el paso 5, dejando EELAT =1, y brincar al paso 2.

4.2.- Puertos Paralelos de Entrada y Salida

El Microcontrolador cuenta con 8 puertos paralelos de 8 bits cada uno, 7 puertos digitales bidireccionales en su mayoría de terminales y 1 puerto análogo - digital unidireccional de entrada.

Cada uno de los puertos comparte sus terminales con otras aplicaciones específicas del microcontrolador, por lo que para poder utilizarlos normalmente deben de estar deshabilitadas las otras funciones del puerto, que por default lo están.

Todos los puertos cuentan con un registro de control y la dirección de registro de datos del puerto. En el registro de control del puerto, se elije el funcionamiento del puerto, es decir si va a funcionar como entrada o salida (discretas), si colocamos ceros (#$00) en el registro de control, el puerto se habilitara como un puerto de entradas discretas, y si colocamos unos en el registro del puerto (#$FF), el puerto será habilitado como salidas discretas.

En la dirección de el registro de datos del puerto, si un determinado puerto a sido habilitado como puerto de salida, colocamos el valor que deseamos tener en las terminales del puerto, es decir, si colocamos el valor #$FF en la dirección del puerto, tendremos un nivel alto (5 Volts) en los 8 bits o terminales del puerto y si colocamos el valor #$00, tendremos un nivel bajo (0 Volts aprox.) en los 8 bits del puerto.

A continuación se describen las direcciones el registro de datos y la dirección de los registros de dirección de los puertos del microcontrolador.

PUERTO A: Bidireccional
PORTA: $0000
DDRA: $0002

PUERTO B: Bidireccional
PORTB: $0001
DDRB: $0003

PUERTO E: PE0-PE1 de Entrada, PE2-PE7 Bidireccionales

PORTE: $0008

DDRE: $0009

PUERTO DLC: Bidireccional

PORTDLC $00FE

DDRDLC $00FF

PUERTO AD: Unidireccional de Entrada

PORTAD: $006F

PUERTO S: Bidireccional

PORTS: $00D6

DDRS: $00D7

PUERTO P: Bidireccional

PORTP: $0056

DDRP: $0057

PUERTO T: Bidireccional

PORTT: $00AE

DDRT: $00AF

4.3.- Modulador de Ancho de Pulso (PWM).

El modulo PWM, puede generar cuatro formas de onda independientes de 8 bits, o 2 formas de onda de 16 bits o finalmente una combinación de una forma de onda de 16 bits y otra de 8 bits. Ya que consta de 4 canales para realizar estas operaciones.

Cada canal tiene un periodo y una duración de ciclo programable así como un contador para cada canal. La flexibilidad de el reloj de este modulo (ver figura 11-3 del documento M68HC12B) permite seleccionar 4 fuentes independientes para cada contador, con lo que pueden generarse formas de onda independientes.

Otra ventaja de este modulo es que la duración del ciclo puede ir desde el 0% del valor del periodo hasta el 100% de este, y la salida del modulo PWM puede programarse para alinearse a la izquierda (ver figura 4.3.1) o al centro del periodo (Ver figura 4.3.2).

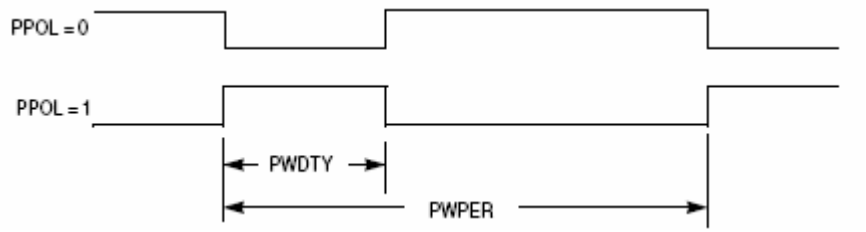


Figura 4.3.1.- Salida del canal PWM alineada a la izquierda.

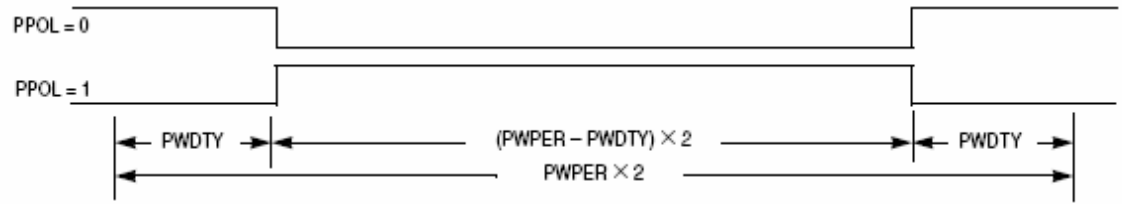


Figura 4.3.2.- Salida del canal PWM alineada al centro.

Una de las características principales de este modulo es que el valor de el periodo y duración de ciclo están almacenados en un buffer doble, es decir, el valor de la salida de los canales no cambia mientras se esta ejecutando un ciclo con valores previamente determinados, con lo que se asegura que el valor en ejecución por el canal no se vera

modificado si no hasta que este ciclo concluya. Lo que concluye en que cada que sea escrito un nuevo valor en los registros de periodo y duración, este valor será ejecutado hasta que termine de ejecutarse el ciclo con los valores anteriores.

El escribir un valor en los registros de periodo y duración de ciclo, causa a la vez que la salida de cada canal cambie y se adapte a los nuevos valores.

Para poder utilizar este modulo es necesario habilitarlo mediante su registro de control, de no hacerlo, los pines que comparten este puerto con el modulo funcionaran como puerto bidireccional discreto.

En el subtema siguiente describimos los registros principales que conciernen al modulo PWM.

Registro de Concatenamiento y de Reloj del PWM

Dirección: $0040

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	CON23	CON01	PCKA2	PCKA1	PCKA0	PCKB2	PCKB1	PCKB0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.3.- Registro de Concatenamiento y de Reloj del PWM (PWCLK).

CON23: Concatena los canales 2 y 3 del PWM

Cuando están concatenados el canal 2 tiene el byte de mayor peso y el canal 3 el de menor peso. El pin 2 del canal de salida es usado como salida de 16 bits del PWM (bit 2 de puerto P), y los bits de control del canal 3 seleccionan la fuente de reloj.

0 = Canal 2 y 3 separados, de 8 bits cada uno.

1 = Canal 2 y 3 se concatenan para formar un solo canal de 16 bits del PWM.

CON01: Concatena los canales 0 y 1 del PWM

Cuando están concatenados el canal 0 tiene el byte de mayor peso y el canal 1 el de menor peso. El pin 0 del canal de salida es usado como salida de 16 bits del PWM (bit 0 de puerto P), y los bits de control del canal 1 seleccionan la fuente de reloj.

0 = Canal 0 y 1 separados, de 8 bits cada uno.

1 = Canal 0 y 1 se concatenan para formar un solo canal de 16 bits del PWM.

PCKA2 – PCKA0: Bits escalares para el reloj A

El reloj A es una de dos fuentes de reloj que pueden ser seleccionadas para el canal 0. Estos 3 bits determinan la velocidad del reloj A (Ver tabla 4.3.1).

PCKB2-PCKB0: Bits escalares para el reloj B.

El reloj B es una de dos fuentes de reloj que pueden ser seleccionadas para el canal 2. Estos 3 bits determinan la velocidad del reloj B (Ver tabla 4.3.1).

Tabla 4.3.1.- Escalares para el reloj A y B.

PCKA2 (PCKB2)	PCKA1 (PCKB1)	PCKA0 (PCKB0)	Vaor del Clock A (B)
0	0	0	E
0	0	1	E ÷ 2
0	1	0	E ÷ 4
0	1	1	E ÷ 8
1	0	0	E ÷ 16
1	0	1	E ÷ 32
1	1	0	E ÷ 64
1	1	1	E ÷ 128

Registro de Selección de Reloj y Polaridad del PWM

Dirección: $0041

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	PCLK3	PCLK2	PCLK1	PCLK0	PPOL3	PPOL2	PPOL1	PPOL0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.4.- Registro de Selección de Reloj y Polaridad del PWM (PWPOL).

PCLK3: Bit de selección de reloj del canal 3 del PWM.

0 = Reloj B es la fuente de reloj para el canal 3.

1 = Reloj S1 es la fuente de reloj para el canal 3.

PCLK2: Bit de selección de reloj para el canal 2 del PWM.
0 = Reloj B es la fuente de reloj para el canal 2.
1 = Reloj S1 es la fuente de reloj para el canal 2.

PCLK1: Bit de selección de reloj para el canal 1 del PWM.
0 = Reloj A es la fuente de reloj para el canal 1.
1 = Reloj S0 es la fuente de reloj para el canal 1.

PCLK0: Bit de selección de reloj para el canal 0 del PWM.
0 = Reloj A es la fuente de reloj para el canal 0.
1 = Reloj S0 es la fuente de reloj para el canal 0.

Si el reloj seleccionado se cambia cuando la señal PWM se esta generando, esta se truncaría o podría ocurrir un pequeño pulso durante la transición.

PPOL3: Bit de polaridad del canal 3 PWM.
0 = La salida del canal 3 tendría un nivel bajo durante el principio del periodo y un nivel alto al alcanzar la duración del ciclo.
1 = La salida del canal 3 tendría un nivel alto durante el principio del periodo y un nivel bajo al alcanzar la duración del ciclo.

PPOL2: Bit de polaridad del canal 2 PWM.
0 = La salida del canal 2 tendría un nivel bajo durante el principio del periodo y un nivel alto al alcanzar la duración del ciclo.
1 = La salida del canal 2 tendría un nivel alto durante el principio del periodo y un nivel bajo al alcanzar la duración del ciclo.

PPOL1: Bit de polaridad del canal 1 PWM.
0 = La salida del canal 1 tendría un nivel bajo durante el principio del periodo y un nivel alto al alcanzar la duración del ciclo.

1 = La salida del canal 1 tendría un nivel alto durante el principio del periodo y un nivel bajo al alcanzar la duración del ciclo.

PPOL0: Bit de polaridad del canal 0 PWM.
0 = La salida del canal 0 tendría un nivel bajo durante el principio del periodo y un nivel alto al alcanzar la duración del ciclo.
1 = La salida del canal 0 tendría un nivel alto durante el principio del periodo y un nivel bajo al alcanzar la duración del ciclo.

Dependiendo del bit de polaridad, el registro de duración también puede contener un conteo de tiempo alto o un conteo de tiempo bajo. Si el bit de polaridad es igual a 0 y se selecciona el alineamiento a la izquierda, el registro de duración contiene un conteo de tiempo bajo.

Si el bit de polaridad es 1, el registro de duración contiene un conteo de tiempo alto.

Registro de Habilitación del PWM

Dirección: $0042

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	0	0	0	0	PWEN3	PWEN2	PWEN1	PWEN0
Escritura								
Reset	0	0	0	0	0	0	0	0

Figura 4.3.5.- Registro de Habilitación del PWM (PWEN).

Este registro es utilizado para habilitar nuestros canales del modulo PWM. Si cualquier bit de este registro es activado (escribiendo un 1 lógico), el correspondiente bit del puerto P es configurado como salida independientemente del valor que pueda contener el registro DDRP sin cambiar el valor de este. Pero al deshabilitar este mismo bit (escribiendo un 0 lógico), el registro DDRP vuelve a tener el control de la dirección de este bit.

Cuando algún bit de este registro es activado, se activa el divisor de reloj, si los 4 canales del PWM están deshabilitados el contador divisor se apaga para almacenar

energía. También existe un detector de flanco para garantizar que la sincronización del reloj se habilite o deshabilite solo cuando el flanco cambie.

PWEN3: Bit de habilitación del canal 3 del PWM.
0 = Canal 3 deshabilitado.
1 = Canal 3 habilitado.
PWEN2: Bit de habilitación del canal 2 del PWM.
0 = Canal 2 deshabilitado.
1 = Canal 2 habilitado.
PWEN1: Bit de habilitación del canal 1 del PWM.
0 = Canal 1 deshabilitado.
1 = Canal 1 habilitado
PWEN0: Bit de habilitación del canal 0 del PWM.
0 = Canal 0 deshabilitado.
1 = Canal 0 habilitado.

Contadores de los Canales 0-3 del PWM.

Cada contador puede leerse en cualquier momento sin afectar la cuenta o operación del correspondiente canal PWM. Escribir en cualquier contador causa que este sea reiniciado a $00 y forza una carga nueva de los valores del registro de duración y periodo con los nuevos valores.

Para evitar truncar el periodo del PWM, hay que escribir en el contador mientras este se encuentre deshabilitado.

A continuación se muestran las figuras 4.3.6-4.3.9 que detallan la composición y dirección de los contadores.

Dirección: $0048

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.6.- Contador 0 del canal PWM (PWCNT0).

Dirección: $0049

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.7.- Contador 1 del canal PWM (PWCNT1).

Dirección: $004A

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.8.- Contador 2 del canal PWM (PWCNT2).

Dirección: $004B

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.9.- Contador 3 del canal PWM (PWCNT3).

Registro de Periodo de los Canales PWM.

El valor del registro de periodo determina el periodo del canal PWM. Si se escribe en el registro mientras el canal es habilitado, el nuevo valor toma efecto cuando el periodo existente termina, forzando el contador a reiniciarse. El nuevo periodo es enlazado hasta que un nuevo periodo sea escrito. Si leemos este registro obtenemos el valor más reciente escrito.

A continuación se muestran las figuras 4.4.10-4.4.13 que detallan la composición y dirección de los registros de periodo.

Dirección: $004C

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.10.- Registro de periodo 0 del canal PWM (PWPER0).

Dirección: $004D

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.11.- Registro de periodo 0 del canal PWM (PWPER1).

Dirección: $004E

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.12.- Registro de periodo 0 del canal PWM (PWPER2).

Dirección: $004F

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figura 4.3.13.- Registro de periodo 0 del canal PWM (PWPER3).

Registros de duración de los canales 0-3 del PWM.

El valor de registro de determinación nos sirve para determinar la duración del canal asociado a este. Cuando el valor de duración es igual al valor del contador, nuestra salida cambia de estado. Si se escribe el registro mientras el canal esta habilitado, el nuevo valor permanecerá en el búfer hasta que el contador reinicie la cuenta o este canal sea deshabilitado.

La lectura de este registro nos entrega el valor mas escrito mas reciente. Si el valor del registro de duración es igual al valor del registro de periodo, no se registrara un cambio de estado.

Si el registro de duración contiene un valor de #FF, la salida estará siempre en el estado opuesto al valor del registro PPOLx.

A continuación se muestran las figuras 4.4.10 - 4.4.14 que detallan la composición y dirección de los registros de duración.

Dirección: $0050

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	1	1	1	1	1	1	1	1

Figura 4.3.14.- Registro de duración del canal 0 del PWM (PWDTY0).

Dirección: $0051

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	1	1	1	1	1	1	1	1

Figura 4.3.15.- Registro de duración del canal 1 del PWM (PWDTY1).

Dirección: $0052

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	1	1	1	1	1	1	1	1

Figura 4.3.16.- Registro de duración del canal 2 del PWM (PWDTY2).

Dirección: $0053

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	1	1	1	1	1	1	1	1

Figura 4.3.17.- Registro de duración del canal 2 del PWM (PWDTY3).

Registro de control del modulo PWM (PWCTL).

Dirección: $0054

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	0	0	0	PSWAI	CENTR	RDPP	PUPP	PSBCK
Reset	0	0	0	0	0	0	0	0

= SIN IMPLEMENTAR

Figura 4.3.18.- Registro de control del modulo PWM (PWCTL).

Puede ser leído y escrito en cualquier momento.

PSWAI: Bit de Interrupción en Modo de Espera del PWM.

0 = El principal generador continua mientras este en modo de espera.

1 = Detiene el reloj principal mientras esta en modo de espera.

CENTR: Bit de Modo de Salida Alineado o al Centro del PWM.

Para evitar irregularidades al activar este modo de salida, hay que escribir este bit solo cuando los canales estén deshabilitados.

0 = Los canales del PWM operan en el modo de salida alineado a la izquierda.

1 = Los canales del PWM operan en el modo de salida alineado al centro.

RDPP: Bit de Reducción de Corriente del Puerto P.

0 = Máxima corriente de salida para todos los pines del puerto P.

1 = Reduce la corriente de salida para todos los pines del puerto P.

PUPP: Bit de Habilitación de Resistencias de PULLUP para el Puerto P.

0 = Deshabilita las resistencias de Pullup para el puerto P.

1 =Habilita las resistencias para todos los pines de entrada del puerto P.

PSBCK: Bit de Detención en Modo Background.

0 = Permite continuar al PWM mientras esta en modo Background.

1 = Deshabilita la entrada de reloj mientras esta en modo Background.

Todos los registros descritos anteriormente son los mas utilizados en operaciones sencillas que involucran este modulo, pero existen mas registros como el Registro escalar (PWSCALx), Registro contador escalar (PWSCNTx), cuya función es de generar las fuentes de reloj S1 y S2 a partir dividir los relojes A y B por un valor almacenado en estos registros. También existen otros registros como el Registro de Modo Especial (PWTST).

Si se desea obtener más información acerca de estos registros consulte el documento MC68HC12B Family Technical Data.

4.4.- Modulo de Contador “Timer” (TIM).

El modulo timmer estándar consiste en un divisor de 16 bits programado por software. Esta formado por 8 canales de entrada de captura o salida de comparación, ambos son de 16 bits y cuenta también con un acumulador de pulsos de 16 bits.

El modulo timmer generalmente es utilizado en aplicaciones en tiempo real, es decir, para medir formas de onda en los canales de entrada, para generar formas de onda de salida en base a medición de tiempo, o sencillamente hacer las dos cosas simultáneamente. Las frecuencias de tiempo que es capaz de medir van desde los microsegundos hasta varios segundos. Este modulo en versiones anteriores de el microcontrolador que no contaban con el modulo PWM, era muy utilizado para generar señales de este tipo.

A continuación describiremos los registros mas utilizados en el empleo del modulo timmer.

Registro de Selección de las Entradas de Captura o Salidas de Comparación del Timmer TIOS.

Dirección: $0080

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
Reset	0	0	0	0	0	0	0	0

Figura 4.4.1.- Registro de Selección de la Entrada de Captura o Salida de Comparación del Timer.

Puede ser leído y escrito en cualquier momento.

IOS7 – IOS0: Canales de entrada o Comparación.
0 = El canal correspondiente actúa como entrada de captura
1 = El canal correspondiente actúa como salida de comparación.

Registros de Control del Timer.

Registro de Control del Timmer TCTL3

Dirección: $008A

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	EDG3B	EDG0A	EDG3B	EDG0A	EDG3B	EDG0A	EDG3B	EDG0A
Reset	0	0	0	0	0	0	0	0

Figura 4.4.2.- Registro de Control del Timer TCTL3

Registro de Control del Timmer TCTL4

Dirección: $008B

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
Reset	0	0	0	0	0	0	0	0

Figura 4.4.3.- Registro de Control del Timer TCTL3

Puede ser leído y escrito en cualquier momento.

EDGnB Y EDGnA: Bits de Control de Captura de Flanco.

Estos 8 pares de bits de control configuran el circuito detector de entrada de captura de flanco como se muestra en la tabla 4.5.1.

Tabla 4.4.1.- Configuración del circuito detector de flanco

EDGnB	EDGnA	Configuración
0	0	Captura deshabilitada
0	1	Captura solamente en flanco de subida
1	0	Captura solamente en flanco de bajada
1	1	Captura en cualquiera de los flancos

Registros de Selección de Mascara de interrupción del Timer

Dirección: $008C

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
Reset	0	0	0	0	0	0	0	0

Figura 4.4.4.- Registro de Mascara de Interrupción del Timer (TMSK1)

Estos Bits corresponden exactamente a los bits del registro de estado TFLG1, Si están en 0, la bandera correspondiente de interrupción esta deshabilitada, si están en 1 la bandera correspondiente se habilita a causa de una interrupción.

Dirección: $008D

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	TOI	0	PUPT	RDPT	TCRE	PR2	PR1	PR0
Reset	0	0	0	0	0	0	0	0

Figura 4.4.5.- Registro de Mascara de Interrupción del Timer (TMSK2)

TOI: Bit de Habilitación de Interrupción por Desbordamiento del Timer

0= Interrupción Inhibida.

1= Solicitud de interrupción por hardware cuando la bandera TOF se active.

PUP: Bits de habilitación de las Resistencias de Pull-Up del Timer.

Estos bits habilitan las resistencias de Pull-up cuando algun pin del timer es configurado como entrada.

0= Deshabilita las resistencias Pull-Up

1= Habilita las resistencias Pull-Up

RDPT: Bits de Reducción de Corriente del Timer.

Estos bits reducen la corriente efectiva que maneja el timer en una salida, reduciendo la corriente entregada por la fuente de alimentación

0= Capacidad normal de corriente de salida

1= Reduce la capacidad de corriente de salida.

TCRE: Bits de habilitación de Reset del Contador del Timer.

Este bit le permite aplicar un reset al contador del timer después de ocurrir después de ocurrir 7 eventos del Output Compare.

0= El Reset del contador permanece inhibido, y este corre libremente

1= Reset del contador a partir de un 7 evento del Output Compare

PR2, PR1, PR0: Bits de Selección del Divisor del Timer.

Estos 3 bits especifican el numero de divisiones entre 2 para el modulo de reloj y el contador.

PR2	PR1	PR0	Factor Divisor
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	Reservado
1	1	1	Reservado

Tabla 4.4.2.- Selección del Divisor

Registros de Bandera de Interrupción del Timer

Dirección: $008E

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura / **Escritura**	**C7F**	**C6F**	**C5F**	**C4F**	**C3F**	**C2F**	**C1F**	**C0F**
Reset	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Figura 4.4.6.- Registro de Banderas de Interrupción del Timer (TFLG1).

TFLG1: Indica cuando ha ocurrido una interrupción. Estos bits se borran escribiendo un 1 sobre el bit que quiera ser borrado.

4.5.- Interfaz Serial

La interfaz serial esta formada por dos subsistemas interfaces seriales independientes de entrada/salida:

- Interfaz de Comunicación Serial (SCI)
- Interfaz Periférica Serial (SPI)

Cada pin de este puerto comparte funciones de propósito general con el puerto S. Debido a su diseño esta interfaz serial es compatible con los sistemas RS232 estándar, así como también el SPI es compatible con el sistema similar del M68HC11.

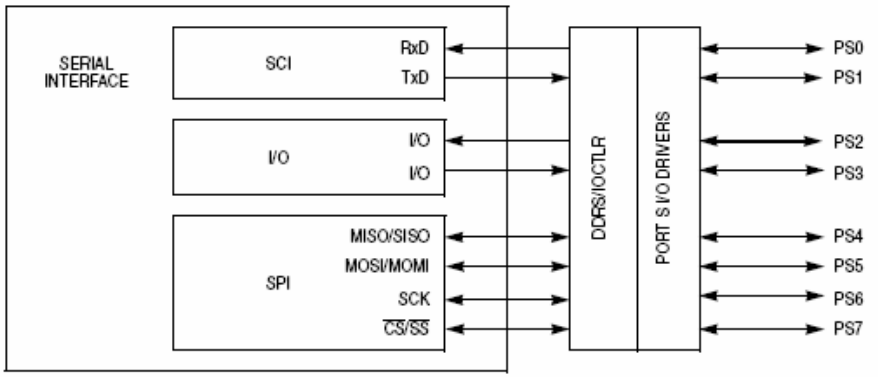


Figura 4.5.1.- Diagrama de Bloques de la Interfaz Serial

4.5.1.- Interfaz de Comunicación Serial (SCI).

Esta interfaz permite realizar comunicaciones asíncronas de 8 o 9 bits con un rango independiente de velocidades. Debido a que la los subsistemas de transmisión y recepción son independientes entre si, se puede realizar una comunicación bidireccional a la vez (Full Duplex). El formato del paquete de datos debe contener un bit de inicio, 8 o 9 bits de datos y un bit de stop.

Este modulo cuenta con un bit de paridad que permite detectar errores mediante la activación de banderas. La velocidad de operación opera basada en un modulo contador muy flexible que permite configurar diferentes velocidades.

Este modulo también cuenta con diferentes características para la recepción de errores las cuales se detallan en los registro de estado.

Transmisión del SCI

Al escribir el programa del usuario en el buffer del registro de datos, el dato es enviado a un registro de desplazamiento, en el cual se desplaza hacia la derecha a través del pin TxD. Los bits de inicio y paro (Stara y stop) son agregados al dato en este registro siempre que este programado el bit M del registro SC0CR1.

Receptor del SCI

El programa del usuario habilita un registro de desplazamiento que recibe los bits en serie a través del pin RxD. Cuando se detecta el bit de stop, se transfiere el dato al registro de datos (SCODRH-L) en forma paralela.

Generación del Rango de Velocidad en Baudios.

Este modulo básicamente esta formado por un contador, este contador da al generador la flexibilidad necesaria para lograr un nivel razonable de independencia de las frecuencias a las que opera el CPU lo que produce velocidades estándar con un mínimo de error. La fuente del generador.

Rango de Baudios Deseado	BR Divisor for P = 4.0 MHz	BR Divisor for P = 8.0 MHz
110	2273	4545
300	833	1667
600	417	833
1200	208	417
2400	104	208
4800	52	104
9600	26	52
14,400	17	35
19,200	13	26
38,400	—	13

Tabla 4.5.1.- Generación de Rango de Velocidad en Baudios.

Registros de Control de Rango de Baudios.

Dirección: $00C0

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	BTST	BSPL	BRDL	SBR12	SBR11	SBR10	SBR9	SBR8
Reset	0	0	0	0	0	0	0	0

Figura 4.5.2.- Registro de control de Rango de Baudios (SC0BDH)

Dirección: $00C1

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
Reset	0	0	0	0	0	0	0	0

Figura 4.5.3.- Registro de control de Rango de Baudios (SC0BDL)

SC0BDH – SCOBDL son considerados juntos como un registro de control de velocidad de transferencia (En baudios) de 16 bits. El Valor de SBR12-SBR0 determina el rango de baudios del SCI.

Este rango puede ser determinado también por la siguiente formula:

$$\text{SCI rango de baudios} = \frac{MCLK}{16XBR}$$

Lo que es equivalente a:

$$\text{BR} = \frac{MCLK}{16XSCIrangodebaudios}$$

BR es la velocidad escrita en los bits SBR12-SBR0 para establecer la velocidad de transferencia en Baudios.

Registro de Control 1 del SCI

Dirección: $00C2

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	LOOPS	WOMS	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

Figura 4.5.4.- Registro de Control 1 del SCI (SC0CR1).

LOOPS: Modo Loop del SCI / Bit de Habilitación de Pin para Propósito General

0 = La sección de transmisión y recepción del SCI opera normalmente.

1 = La sección de recepción del SCI es desconectada del pin RxD, por lo que este queda habilitado para ser usado como pin de propósito general.

La entrada del receptor se determina mediante el bit RSRC. La salida del transmisor es controlada por el estado del bit DDRS asociado. Al inicio el transmisor y el receptor deben habilitarse para usarse en modalidad de Loop o como pines de propósito general Para mas detalles acerca del Funcionamiento en Modo Loop diríjase a la sección 14.2.3 del documento M68HC12B Family Technical Data.pdf.

WOMS: Modo Wired-OR para Pins Seriales.

Este pin controla los dos pines (TXD y RXD) asociados con la sección SCI.

0 = Estos pines operan normalmente con un inicio en alto y bajo manejo de corriente. Para afectad el pin RxD este debe estar configurado como salida como en el caso de operar como pin de propósito general. Es decir operan como salidas CMOS.

1 = Cada pin opera como colector abierto si es declarado como salida.

RSRC: Bit Fuente de Receptor.

Cuando LOOPS=1, el bit RSRC determina el curso de retroalimentación para el receptor mas no al pin TxD.

0 = La entrada receptora esta conectada internamente con el transmisor.

1 = La entrada del receptor esta conectada al pin TXD.

M: Bit de modalidad (Selecciona el formato del carácter).
0 = Un bit de inicio, 8 de datos, y un bit de stop.
1 = Un bit de inicio, 8 u 9 bits de datos, y un bit de stop.

M — Mode Bit (select character format)
0 = One start, eight data, one stop bit
1 = One start, eight data, ninth data, one stop bit

WAKE: Bit de Activación por Línea Libre/Marca.
0 = Activación por línea libre.
1 = Activación por marca.

ILT: Bit Tipo Línea Libre.

Este bit determina cual de los 2 tipos de detección de línea libre son usados por el receptor SCI.
0 = Modo de habilitación de detección de línea libre corto.
1 = Modo de habilitación de detección de línea libre largo.

En la modalidad corto, el circuito SCI empieza a contar un segundo en búsqueda de línea libre inmediatamente después del bit de inicio. Lo que quiere decir que el bit de stop y cualquier bit que fue antes del bit de stop podían ser contados en una cadena de 1segundo, resultando con esto en un mayor reconocimiento de línea libre.

En la modalidad largo, el circuito SCI no empieza a contar 1 segundo en búsqueda de la línea libre, si no hasta que el bit de stop es recibido.

PE: Bit de Habilitación de Paridad
0 = Paridad deshabilitada.
1 = Paridad Habilitada.

PT: Bit de Tipo de Paridad.

Si la paridad es habilitada, este bit determina paridad par o impar para el inicio del receptor y transmisor. Un número par de 1 segundo en la cadena de caracteres causa que el bit de paridad sea 0 y un número impar de 1 segundo causa que el bit de paridad sea 1.

0 = Selecciona paridad par.

1 = Selecciona paridad impar.

Registro de Control 2 del SCI

Dirección: $00C3

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

Figura 4.5.5.- Registro de Control 2 del SCI (SC0CR2).

TIE: Bit de Habilitación de Interrupción de Transmisión.

0 = Interrupción TDRE deshabilitada.

1 = Solicitud de interrupción del SCI cuando la bandera TDRE este activa.

TCIE: Bit de Habilitación de Interrupción de Transmisión Completa.

0 = Interrupción TC deshabilitada.

1 = Solicitud de interrupción del SCI cuando la bandera TC este activa.

RIE: Bit de Habilitación de Interrupción del Receptor.

0 = Interrupción de RDRF y OR deshabilitadas.

1 = Solicitud de interrupción del SCI cuando las bandera RDRF u OR estén activas.

ILIE: Bit de Habilitación de Interrupción de Línea Libre.

0 = Interrupción IDLE deshabilitada.

1 = Solicitud de interrupción del SCI cuando las bandera IDLE este activa.

TE: Bit de Habilitación de Transmisión.

0 = Transmisor Deshabilitado.

1 = El bit de transmisión del SCI es habilitado u el pin TXD (Port S bit 1) es dedicado al transmisor.

RE: bit de Habilitación del Receptor.

0 = Receptor deshabilitado.

1 = Habilita el circuito de recepción del SCI.

RWU: Bit de Control de Activación del Receptor.

0 = Receptor Normal.

1 = Habilita la función de despertar e inhibe las futuras interrupciones. Normalmente el hardware despierta estas interrupciones automáticamente borrando este bit.

SBK: Bit de Envió de Señal BREAK.

0 = Generador de señal Break apagado.

1 = Genera un código break, de al menos 10 u 11 continuos 0s.

4.5.2.- Interfaz SPI

La interfaz periférica serial (SPI) es un sistema de comunicación serie sincrono de alta velocidad, generalmente es utilizado para comunicar el microcontrolador con dispositivos como LCD o DAC's de entrada serial entre otros, o inclusive con otros microcontroladores.

Este dispositivo tiene dos formas principales de operación, que son el modo maestro y el modo esclavo. Estos modos de operación intervienen por ejemplo al realizar una interconexión entre 2 o más dispositivos, en estos casos, solo se permite que un solo dispositivo opere en modo maestro y se permite la operación de varios dispositivos en modo esclavo. También es posible cambiar el rol de cada dispositivo y hacer que cambien su modalidad de operación, es decir de maestro a esclavo y viceversa.

Otra principal característica de esta interfaz serial es la de permitir comunicaciones full duplex (comunicación bidireccional simultanea) entre un dispositivo maestro y otro esclavo, lo que le da una gran flexibilidad al momento de diseñar un sistema de comunicación entre varios dispositivos.

Descripción del Protocolo de Comunicación del SPI

El funcionamiento del protocolo utilizado es muy sencillo, cuando el dispositivo maestro requiere enviar un dato a uno o mas dispositivos, tiene que proceder antes que cualquier cosa a la habilitación del dispositivo de destino (esta operación es similar a la señal chip enable). Después de que se ha activado el dispositivo esclavo, este procede con la recepción del dato, utilizando una señal de reloj conjunta como medio de sincronización.

Cuando se escribe un dato en el registro SP0DR del dispositivo maestro, este es el dato de salida para el esclavo. Y el dato leído del registro SP0DR del dispositivo maestro después de una operación de transferencia es la entrada de datos desde el esclavo

Un bit de control de face de reloj y un bit de control de polaridad de reloj en el registro de control SP0CR1, seleccionan uno de los cuatro posibles formatos de reloj que

pueden ser usados en el SPI. El bit CPOL simplifica la selección de un reloj invertido o no invertido. Para ver mas detalles consulte el documento M68HC12B Family Technical Data.pdf en la sección 14.3.2.

En algunas ocasiones, el dispositivo esclavo después de recibir un dato, puede enviar un dato de respuesta. Para que esto sea posible es necesario que permanezca activa la señal de habilitación procedente del dispositivo maestro.

Existen 4 líneas básicas en la interfaz SPI para establecer los modos de operación, que son las siguientes:

MOSI (Mater Out, Slave In).-

En esta línea se envían los datos que el dispositivo maestro envía a los esclavos Por lo anterior esta señal será la salida de datos para el maestro y la señal de entrada de datos para los esclavos.

MISO (Master In, Slave Out).-

En esta línea se envían los datos del dispositivo esclavo hacia el maestro. Por lo anterior será una señal de entrada para el maestro y una señal de salida para los esclavos.

SCK (Señal de Reloj Serial).-

Esta es la señal de reloj que se utiliza como señal de sincronía. Aunque la velocidad de transferencia puede ser configurada en cada dispositivo, la velocidad de transferencia que prevalecerá será siempre la indicada por la señal del dispositivo maestro.

Esta señal será siempre de salida para los maestros y de entrada para los esclavos.

SS (Slave Select).-

Esta línea representa la señal de habilitación para los dispositivos esclavos. En el caso del dispositivo maestro esta línea es una señal de salida utilizada para otros propósitos.

Descripción de Registros del SPI

Registro de Control 1 del SPI

Dirección: $00D0

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	SPIE	SPE	SWOM	MSTR	CPOL	CPHA	SSOE	LSBF
Reset	0	0	0	0	0	1	0	0

Figura 4.5.6.- Registro de Control 1 (SP0CR1).

SPIE: Bit de Habilitación de Interrupciones del SPI

0 = Las interrupciones del SPI son inhibidas.

1 = Una secuencia de interrupción por hardware es solicitada cada vez que las banderas SPIF o MODF se activen.

SPE: Bit de Habilitación del Sistema SPI.

0 = El hardware interno del SPI es inicializado y es estado de bajo consumo es deshabilitado.

1 = PS4 Y PS7 son dedicados a funciones del SPI

Cuando MODF este activo, SPE siempre regresara 0. SP0CR1 debe ser escrito como parte de una secuencia de recuperación.

SWOM: Bit de Habilitación del Puerto S para Propósito General

Los pines PS4-PS7 pueden ser usados como de pines de propósito general cuando no son utilizados por el SPI.

0 = El SPI y/o el buffer de las salidas PS4-PS7 opera normalmente.

1 = El SPI y/o el buffer de las salidas PS4-PS7 opera como colector abierto.

MSTR: Bit de Selección de Modo Maestro o Esclavo.

0 = Modo Esclavo.

1 = Modo Maestro.

CPOL Y CPHA: Bit de Polaridad y Face del Reloj.

Estos dos bits son utilizados para especificar el formato del reloj del SPI. Cuando el bit de polaridad es borrado y el dato ha empezado a transferirse. El pin SCK del dispositivo maestro esta en nivel Bajo (SCK=0). Cuando CPOL esta activo, el SCK esta en nivel alto. Ver figuras 14-12 y 14-13 del documento M68HC12B Family Technical Data.pdf.

SSOE: Bit de Habilitación de Esclavo
La salida SS es habilitada solo en el modo maestro mediante la activación de los bits SSOE y DDS7.

LSBF: Bit de Habilitación del Primer LSB del SPI.
0 = El primer dato a transferir es el bit mas significativo (MSB).
1 = El segundo dato a transferir es el bit menos significativo (LSB).

LSBF — SPI LSB First Enable Bit
0 = Data is transferred most-significant bit (MSB) first.
1 = Data is transferred least-significant bit (LSB) first.
Normalmente, el dato transferido es el MSB primero, pero esto no afecta la posición del los bits MSB y LSB de el registro de datos ya que la escritura o lectura del registro de datos siempre coloca al bit mas significativo (MSB) en el bit 7.

Registro de Control 2 del SPI

Dirección: $00D1

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	**0**	**0**	**0**	**0**	**PUPS**	**RDS**	**0**	**SPC0**
Escritura								
Reset	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**

= SIN IMPLEMENTAR

Figura 4.5.7.- Registro de Control 1 (SP0CR2).

Lectura y Escritura habilitadas en cualquier momento

PUPS: Bit de Habilitación de Resistencias Internas de Pullup.

0 = Deshabilitación de resistencias de pullup.

1 = Las resistencias de pullup están habilitadas en todas las entradas del puerto. Si un pin es programado como salida, se deshabilitan las resistencias de pullup.

RDS: Bit de Reducción de Corriente para el Puerto P.

0 = Las salidas del puerto P opera normalmente.

1= Todas los pines de salida manejan baja capacidad de corriente y menos ruido.

SPC0 — Bit de Control 0 del Pin Serial
Este bit decide el pin de configuración como bit de control MSTR.

Modo de Pin		**SPC0(1)**	**MSTR**	**MISO(2)**	**MOSI(3)**	**SCK(4)**	**SS(5)**
#1	Normal	0	0	Slave out	Slave in	SCK in	SS in
#2			1	Master in	Master out	SCK out	SS I/O
#3	Bidireccional	1	0	Slave I/O	Propósito general/O /	SCK in	SS in
#4			1	Propósito general/O	Master I/O	SCK out	SS I/O

Tabla 4.5.2.- Pin de Control Serial.

1.- El pin serial control 0 habilita la configuración bidireccional.

2.- El salida del esclavo se habilita si DDS4 = 1, SS = 0, y MSTR = 0. (#1, #3).

3.- La salida del maestro se habilita si DDS5 = 1 y MSTR = 1. (#2, #4).

4.- La salida SCK se habilita si DDS6 = 1 y MSTR = 1. (#2, #4).

5.- La salida SS se habilita si DDS7 = 1, SSOE = 1, y MSTR = 1. (#2, #4).

Registró De Rango de Baudios del SPI.

Dirección: $00D2

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	**0**	**0**	**0**	**0**	**0**	**SPR2**	**SPR1**	**SPR0**
Escritura								
Reset	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

= SIN IMPLEMENTAR

Figura 4.5.8.- Registro de Control 1 (SP0BR).

Lectura y escritura habilitadas en cualquier momento.

Después de un reset, el reloj se divide entre 2.

SPR2-SPR1: Bits de Selección de Rango del Reloj SPI (SCK).

Estos bits son utilizados especificar el rango del reloj SPI. Ver tabla 4.7.3.

Tabla 4.5.3.- Selección de Rango de Reloj del SPI:

SPR2	SPR1	SPR0	E Clock Divisor	Frecuencia con Reloj E = 4 MHz	Frecuencia con E Reloj E = 8 MHz
0	0	0	2	2.0 MHz	4.0 MHz
0	0	1	4	1.0 MHz	2.0 MHz
0	1	0	8	500 kHz	1.0 MHz
0	1	1	16	250 kHz	500 kHz
1	0	0	32	125 kHz	250 kHz
1	0	1	64	62.5 kHz	125 kHz
1	1	0	128	31.3 kHz	62.5 kHz
1	1	1	256	15.6 kHz	31.3 kHz

Registro de Estado del SPI.

Dirección: $00D3

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	**SPIF**	**WCOL**	**0**	**MODF**	**0**	**0**	**0**	**0**
Escritura								
Reset								

= SIN IMPLEMENTAR

Figura 4.5.9.- Registro de Control 1 (SP0SR).

Lectura en cualquier momento, escritura no tiene efecto.

SPIF: Bit de Solicitud de Interrupción.

Este bit se activa después de 8 ciclos del SCK en una transferencia de datos, y se borra mediante la lectura de datos del SP0SR seguido por un acceso al registro de datos del SPI.

WCOL: Bandera de Estado de Colisión por Escritura.

La escritura del MCU se desactiva para evitar la sobre escritura del dato en transferencia.

0 = No hay posibilidad de colisión al escribir.

1 = Indica que hubo una transferencia de datos cuando el MCU trata de escribir un dato en el registro SP0DR. Este bit se borra automáticamente por la lectura del SP0SR (cuando WCOL=1) seguida de un acceso al registro de datos SP0DR.

MODF: Bandera de Interrupción Estado de de Modo no Valido del SPI.

Este bis se activa automáticamente por el hardware API, si el bit de control MSTR esta activo y la entrada de selección de esclavo es 0. Esta condición no esta permitida en modo de operación normal.

En el caso donde DDRS bit 7 este activo, el pin PS7 es duna salida de propósito general o el pin de salida SS es muy pocas veces de entrada para el sistema SPI. En este caso en especial la modalidad de falla es inhibida y MODF permanece desactivado.

Esta bandera se desactiva automáticamente por la lectura del SP0SR (si MODF =1) seguida de una escritura en SP0CR1.

Registro de Datos del SPI.

Dirección: $00D5

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Escritura								
Reset								

Figura 4.5.10.- Registro de Datos d (SP0DR).

Lectura en cualquier momento si la bandera SPIF esta activa. Escritura en cualquier momento.

Este registro de 8 bits es de entrada y salida para el registro de datos. La lectura en este registro esta doblemente almacenada en buffer, pero la escritura causa que el dato se dirija directamente al registro de desplazamiento serial.

En el sistema SPI, los 8 bits del registro de datos en el maestro y en el esclavo se ligan por los pines MOSI y MISO para formar un registro distribuido de 16 bits. Cuando una operación de transferencia de datos es realizada, los 16 bits del registro son desplazados serialmente 8 posiciones por medio del reloj SCK maestro y los datos son intercambiados efectivamente entre el maestro y el esclavo.

4.6.- Convertidor Analógico a Digital (ATD).

El convertidor analógico a digital (ADC por sus siglas en español, y ATD por sus siglas en ingles), consiste de un modulo de 8 canales de 10 bits. Diseñado para trabajar mediante la técnica de aproximaciones sucesivas, cuenta con una preescisión de ±2 bits menos significativos (LSB).

Cuenta principalmente con 32 bytes de memoria mapeados en un block de registro de control, utilizado para control, prueba y configuración

El modo de operación de este modulo es muy sencillo. Una secuencia de conversión sencilla consiste en 4 o 8 conversiones, dependiendo del estado del bit S8CM del registro ATDCTL5. Este modulo también tiene 8 modos básicos de conversión.

En el modo no-escaneo, el SCF bit se activa después de que la secuencia de 4 o 8 conversiones a sido realizada por lo que el modulo ATD se detiene.

En el modo de escaneo, el bit SCF se activa después de que la primera de 4 o 8 conversiones a sido realizada y el modulo ATD continua restableciéndose la secuencia.

En modo de arranque el bit CCF asociado con cada registro se activa cuando se carga en cada registro el resultado de la conversión. La bandera se limpia automáticamente cuando el resultado es leído del registro.

La conversión empieza en el momento que es registro de control es escrito.

Registros de control del modulo ATD (ATDCTLx).

A continuación describimos los registros de control utilizados más frecuentemente, ya que existen otros registros como el ATDCTL0-2 no utilizados tan comúnmente. Si se quiere obtener más información acerca de estos diríjase a la sección 17 del M68HC12B Family Technical Data.

Registro de Control 2 del ATD

Dirección: $0062

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	ADPU	AFFC	AWAI	0	0	0	ASCIE	ASCIF
Reset	0	0	0	0	0	0	0	0

= SIN IMPLEMENTAR

Figura 4.6.1.- Registro de Control 2 del ATD (ATDCTL2).

Lectura y Escritura en cualquier momento, excepto el bit 0 que no puede ser escrito.

ADPU: El programa puede deshabilitar la señal de reloj del ATD, asi como apagar el circuito análogo para reducir el consumo de poder. Si se resetea a 0, el bit ADPU aborta cualquier secuencia de conversión en progreso debido a que se corta la corriente de bias del circuito análogo. Una vez que se activa el bit ADPU, el modulo ATD requiere de un periodo de recuperación para la estabilización del circuito análogo.

0 = Deshabilita el ATD, incluyendo la sección analógica para reducir el consumo de corriente.

1 = Permite al modulo ATD funcionar normalmente.

AFFC: Bit para limpiar la bandera.

0 = El bit de bandera de limpieza opera normalmente es decir, si se lee el registro de estado antes de leer el registro de resultado, causa que se limpie el bit CCF asociado.

1 = Cambia el estado de todas las banderas de conversión completa en una secuencia de limpieza rápida. Cualquier acceso al registro de resultado (ATD0-ATD7) causa que la bandera asociada al bit CCF se borre (limpie).

AWAI: Bit de stop en modo de espera.

0 = El modulo ATD continua activo cuando el MCU esta en modo de espera.

1 = El modulo ATD se detiene durante el modo de espera para reducir el consumo de energía.

ASCIE: Bit de habilitación de interrupción de secuencia completa.

0 = Deshabilita la interrupción.

1 = Habilita la interrupción de secuencia completa.

ASCIF: Bandera de interrupción de secuencia completa.

No puede ser escrito en ningún modo.

0 = No ocurren interrupciones en el ATD.

1 = Secuencia completa del ATD.

Registro de Control 3 del ATD

Dirección: $0063

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	0	0	0	0	0	0	FRZ1	FRZ0
Reset	0	0	0	0	0	0	0	0

= SIN IMPLEMENTAR

Figura 4.6.2.- Registro de Control 3 del ATD (ATDCTL3).

Este registro es utilizado para el encendido del modulo, control de interrupción y control de detención. Escribir en este registro aborta cualquier secuencia de conversión y suspende la operación del modulo al encontrar cualquier punto de ruptura (breakpoint).

FRZ1 – FRZ0: Bit de habilitación en modo de depuración Background.

Cuando se depura una aplicación, resulta muy útil en algunos casos detener el modulo ATD cuando se encuentra un punto de ruptura. Estos dos bits determinan como puede responder el modulo ATD cuando el modo de depuración Background se activa.

Ver la tabla 4.9.1.

Tabla 4.6.1.- Respuesta al Modo Background.

FRZ1	FRZ2	Respuesta del ATD
0	0	Continua la conversión en modo Background
0	1	Reservado
1	0	Si finaliza la conversión en curso, entonces se detiene.
1	1	Se detiene cuando el modo Background esta activo

Registro de Control 4 del ATD.

Dirección: $0064

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	S10BM	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
Reset	0	0	0	0	0	0	0	0

Figura 4.6.3.- Registro de Control 4 del ATD (ATDCTL4).

Este registro activa la fuente de reloj y activa el divisor. Escribir en este registro inicializa una nueva secuencia de conversión.

Si ocurre una operación de escritura en este registro mientras una secuencia de conversión se encuentra activa, la conversión en progreso se interrumpe y el ATD se detiene hasta que ocurra una operación de escritura en ATDCTL5.

S10BM: Bit de control de modo de operación de 10 Bits.

0 = Opera con 8 bits.

1 = Opera con 10 Bits.

SMP1 – SMP0: Bit se selección de tiempo de muestreo.

Estos bits son utilizados para seleccionar uno de cuatro tiempos de muestreo después de que la muestra se almacena en el buffer y la transferencia ha ocurrido. Ver la tabla 4.6.2.

Tabla 4.6.2.- Selección de Tiempo de Muestreo Final.

SMP1	SMP0	Tiempo final de Muestreo	Tiempo de Conversión Total de 8 Bits	Tiempo de Conversión Total de 10 bits
0	0	2 Periodos de reloj del ATD	18 Periodos de reloj del ATD	20 Periodos de reloj del ATD
0	1	4 Periodos de reloj del ATD	20 Periodos de reloj del ATD	22 Periodos de reloj del ATD
1	0	8 Periodos de reloj del ATD	24 Periodos de reloj del ATD	26 Periodos de reloj del ATD
1	1	10 Periodos de reloj del ATD	32 Periodos de reloj del ATD	34 Periodos de reloj del ATD

PRS4 – PRS0: Bits de Selección del Divisor para el P-Clok.

Un valor escrito en estos bits selecciona el divisor para el modulo de conteo basado en un divisor, Los periodos de reloj son divididos por un valor positivo, que alimenta un divisor por dos para generar una frecuencia de reloj para el modulo ATD, lo que asegura la simetría de la señal de reloj.

La tabla 4.6.3 muestra el factor de división y el rango de la frecuencia a la que se trabaja.

Tabla 4.6.3.- Valor del Divisor de Reloj

Valor Divisor	**Total Divisor**	**Max P Clok (1)**	**Min P Clok (2)**
00000	2	4 MHz	1 MHz
00001	4	8 MHz	2 MHz
00010	6	8 MHz	3 MHz
00011	8	8 MHz	4 MHz
00100	10	8 MHz	5 MHz
00101	12	8 MHz	6 MHz
00110	14	8 MHz	7 MHz
00111	16	8 MHz	8 MHz
01xxx	No están en uso		
1xxxx			

Registro de Control 5 del ATD.

Dirección: $0065

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura		S8CM	SCAN	MULT	CD	CC	CB	CA
Reset	0	0	0	0	0	0	0	0

= SIN IMPLEMENTAR

Figura 4.6.4.- Registro de Control 5 del ATD (ATDCTL5).

Este registro es utilizado para seleccionar el modo de conversión, los canales de conversión así como también para inicializar la conversión. Cualquier escritura en este registro ocasiona la inicialización de una nueva secuencia de conversión, por lo que si esta operación de escritura ocurre mientras se realiza una conversión, la secuencia corriente se cancela iniciándose otra nueva secuencia.

SC8M: Selecciona las Secuencias de Conversión

0 = La secuencia de conversión consiste en 4 conversiones.

1 = La secuencia de conversión consiste en 8 conversiones.

SCAN: Bit de Habilitación de Escaneo de Canal Continuo.

Cuando se inicializa una secuencia de conversión el usuario puede seleccionar una secuencia de 4 u 8 conversiones (dependiendo del bit SC8M) o realizar una secuencia continua de 4 u 8 conversiones.

0 = Secuencia de conversión sencilla.

1 = Secuencia de conversión continua (Modalidad de escaneo).

MULT: Bit de Habilitación de Múltiples Canales de Conversión.

0 = El ATD corre una secuencia de 4 u 8 conversiones en un solo canal seleccionado vía los bits CD, CC, CB y CA.

1 = El ATD corre una secuencia de 4 u 8 conversiones en una secuencia de canales especificados (Vea la tabla 4.9.4.- Registros de Asignación de Resultado en la Modalidad Multicanal.).

CD, CC, CB y CA: Bits de selección de Canales de Conversión.

Tabla 4.6.4.- Registros de Asignación de Resultado en la Modalidad Multicanal.

S8CM	CD	CC	CB	CA	Canal	Resultado en ADRx si MULT = 1
0	0	0	0	0	AN0	ADR0
			0	1	AN1	ADR1
			1	0	AN2	ADR2
			1	1	AN3	ADR3
0	0	1	0	0	AN4	ADR0
			0	1	AN5	ADR1
			1	0	AN6	ADR2
			1	1	AN7	ADR3
0	1	0	0	0	Reservado	ADR0
			0	1	Reservado	ADR1
			1	0	Reservado	ADR2
			1	1	Reservado	ADR3
0	1	1	0	0	V_{RH}	ADR0
			0	1	V_{RL}	ADR1
			1	0	$(V_{RH}+ V_{RL})/2$	ADR2
			1	1	Test/ Reservado	ADR3
1	0	0	0	0	AN0	ADR0
			0	1	AN1	ADR1
			1	0	AN2	ADR2
			1	1	AN3	ADR3
		1	0	0	AN4	ADR4
			0	1	AN5	ADR5
			1	0	AN6	ADR6
			1	1	AN7	ADR7
1	1	0	0	0	Reservado	ADR0
			0	1	Reservado	ADR1
			1	0	Reservado	ADR2
			1	1	Reservado	ADR3
		1	0	0	V_{RH}	ADR4
			0	1	V_{RL}	ADR5
			1	0	$(V_{RH}+ V_{RL})/2$	ADR6
			1	1	Test/ Reservado	ADR7

Registros de Estado del ATD.

Dirección: $0066

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	SCF	0	0	0	0	CC2	CC1	CC0
Escritura								
Reset	0	0	0	0	0	0	0	0

= SIN IMPLEMENTAR

Figura 4.6.5.- Registro de Estado del ATD (ATDSTAT).

Dirección: $0067

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
Escritura								
Reset	0	0	0	0	0	0	0	0

= SIN IMPLEMENTAR

Figura 4.6.5.1.- Registro de Estado del ATD (ATDSTAT).

Este registro es de solo lectura, misma que puede realizarse en cualquier momento. Solo puede escribirse SCF y CCF en modo especial.

El registro de estado contiene las banderas que indican que se ha completado una secuencia de conversión.

SCF: Bandera de Secuencia Completa.

Este bit se activa al finalizar una secuencia de conversión, en la modalidad de secuencia de conversión sencilla (SCAN = 0 en ATDCTL5). Y también se activa al finalizar la primera secuencia de conversión en la modalidad de secuencia de conversión continua (SCAN =1 en ATDCTL5).

Si AFFC = 0, SCF se limpia cuando se realiza una escritura en ATDCTL5, inicializando una nueva secuencia de conversión. Si AFFC = 1, SCF se limpia después de que es leído el primer resultado de el registro.

CC2 – CC0: Bits de Contador de Conversión para 4 u 8 Conversiones

Estos 3 bits reflejan el contenido del puntero de conteo de conversión en 4 u 8 secuencias de conteo. Este valor también refleja que registro de resultado será el siguiente en escribirse, lo cual indica que canal se ha empezado a convertir.

CCF7-CCF0: Bandera de Conversión Completa.

Cada CCF bit es asociado con un registro de resultado individual del ATD. Para cada registro, este bit se activa al finalizar la conversión del canal asociado a este, y permanece activo hasta que el registro de resultado es leído.

Este bit puede ser borrado en cualquier momento si el bit AFFC se activa sin tomar en cuenta si se ha realizado una lectura al registro de estado.

Registros de Resultado del ATD.

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	**Bit 15**	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**
Escritura								
Reset								

= SIN IMPLEMENTAR

Figura 4.6.6.- Registro de Resultado High del ATD (ADRxH).

ADRx0H: $0070
ADRx1H: $0072
ADRx2H: $0074
ADRx3H: $0076
ADRx4H: $0078
ADRx5H: $007A
ADRx6H: $007C
ADRx7H: $007E

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Escritura								
Reset								

= SIN IMPLEMENTAR

Figura 4.6.6.1- Registro de Resultado Low del ATD (ADRxL).

ADRx0L: $0071
ADRx1L: $0073
ADRx2L: $0075
ADRx3L: $0077
ADRx4L: $0079
ADRx5L: $007B

ADRx6L: $007D
ADRx7L: $007F

ADRxH [15:8] –ADRxH [7:0]: Bits de Resultado de Conversión del ATD.

Estos bits son de solo lectura, y contienen el resultado de la conversión. El canal del que fue obtenido el resultado depende del modo de conversión seleccionado.

CAPITULO 5

INTERFACES

Las interfaces son sistemas diseñados a partir de circuitos electrónicos con el fin de permitir la interconexión para diferentes propósitos entre dispositivos electrónicos, y es por esto que sirven de gran ayuda para adaptar a un microcontrolador con dispositivos de hardware externos tanto analógicos como digitales.

Los microcontroladores están diseñados para manejar niveles ttl en sus puertos digitales, y solamente de 0 a 5 Volts en sus puertos analógicos, por lo que al no respetar estos estándares podríamos dañar permanentemente la estructura interna del microcontrolador lo que culminaría en la perdida del mismo.

Estas características limitan la funcionalidad del microcontrolador a trabajar dentro del rango de los sistemas digitales, por lo que se requiere de diferentes interfaces que puedan permitir al microcontrolador trabajar con niveles de voltaje diferentes para los que a sido diseñado y contando siempre con un aislamiento adecuado entre el microcontrolador y los diferentes dispositivos externos.

En este capitulo se trataran los circuitos electrónicos básicos necesarios para diseñar interfaces de los siguientes tipos:

- Interfaces de aislamiento
- Interfaces de adquisición de datos
- Interfaces de acondicionamiento de señal

5.1.- Dispositivos de Entrada Digitales.

5.1.1.- Switchs Mecánicos.

Los switches mecánicos mas comúnmente utilizados son los de 1 polo 1 tiro, 1 polos 2 tiros, además de otras combinaciones existentes.

Estos switches o interruptores nos permiten introducir datos de nivel lógico TTL es decir 0 o 1 en los puertos configurados como entradas digitales. La configuración típica consiste en conectar un interruptor a 5 volts y al estar abierto conduzca el voltaje al pin del microcontrolador y al cerrarse lo envié a tierra logrando así los niveles del 0 y el 1 respectivamente. Esta configuración se muestra en la figura 5.1.1.

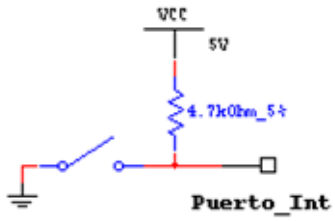


Figura 5.1.1.– Configuración Típica de un switch

El único problema que surge al utilizar estos interruptores, es la generación de pequeñas variaciones de voltaje al momento de activar el interruptor comúnmente llamado efecto “Rebote”, lo que puede derivar en una lectura errónea del nivel lógico de entrada. Por esto existen dos técnicas comúnmente utilizadas para resolver este inconveniente, estas técnicas son:

1. Usar un flip-flop R-S
2. Usar un retardo de tiempo generado por software.

La segunda opción es la más utilizada comúnmente ya que elimina la necesidad de agregar mas elementos a nuestra interfaz generando un retraso no menor a 20 milisegundos a partir de una rutina de software dentro del programa del usuario.

5.1.2.- Teclados Matriciales.

Un teclado matricial esta diseñado a partir de una matriz de interruptores conectados de esta forma para maximizar el uso de un puerto. Esta ventaja nos permite conectar 16 interruptores, 8 en filas y 8 en columnas en un puerto de 8 bits en el caso de ser un arreglo pequeño como se muestra en la figura 5.1.2.

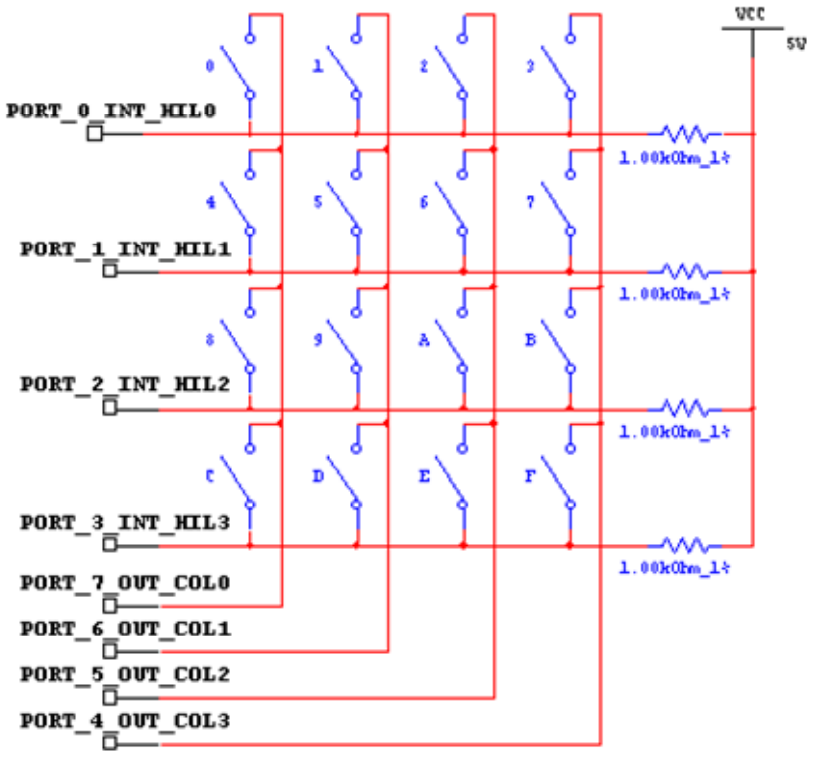


Figura 5.1.2.-Teclado Matricial

Cuando se requiera conectar matrices de mayor tamaño es necesario utilizar decodificadores como el MC14028 para 24 teclas, el MM74C922 para 16 teclas o el MM74C923 para 20 teclas por mencionar algunos, ya que estos nos permiten conectar estos dispositivos al mismo puerto de 8 bits.

El principio de funcionamiento de este tipo de teclado es muy sencillo. Primero se conectan todas las hileras a 5 volts con lo que nos aseguramos de tener un 1 lógico en los pines configurados como entrada. De esta forma si se enviara un 0 lógico por la columna 0 y se detectara este nivel bajo en la hilera 0, esto nos indicaría que la tecla o interruptor oprimido corresponde al de la etiqueta “0”. Si el cero lógico fuera

enviado por la columna 4 y se detectara un nivel bajo en la hilera 0 esto indicaría que la tecla oprimida seria la de la etiqueta “3”.

Como se trata de interruptores, existe el problema llamado “rebote” que es eliminado con un retraso de tiempo por software al detectar un primer cambio de nivel lógico en las hileras, después de que el retraso de tiempo llego a su fin se procede con la lectura de la hilera de la manera ya mencionada.

5.2.- Dispositivos de Salida Digitales.

Son circuitos electrónicos como circuitos integrados, transistores con arreglos especiales u otros dispositivos de potencia que permiten conectar elementos externos que excedan la corriente o voltaje manejada por el microcontrolador.

5.2.1.- Drivers de Corriente y Voltaje.

Existen en una amplia variedad, algunos como los circuitos integrados del tipo CMOS que nos permiten manejar dispositivos externos que no son del tipo TTL, es decir, adaptan diferentes niveles de voltaje de diferentes familias entre si, como es el caso del circuito integrado 74HC4050 que también brinda protección contra descargas electrostáticas.

Este circuito en particular y otros similares son útiles como interfaces de entrada de datos procedentes de circuitos que operan a diferentes niveles lógicos. En la figura 5.2.1 se muestra la distribución de las terminales donde se puede apreciar básicamente una serie de buffer no inversores para realizar la interconexión entre dispositivos.

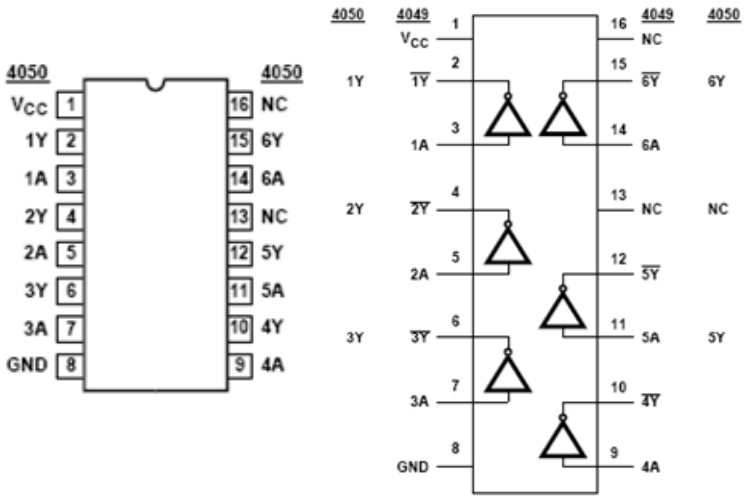


Figura 5.2.1 .- Buffer no inversor 74HC4050

Otro circuito auxiliar para el manejo de cargas con alto consumo de corriente es el ULN2003. Este circuito integrado cuenta con 8 entradas compatibles a nivel TTL, cuenta además con diodos supresores en cada una de las 8 salidas para el manejo de cargas inductivas, y es capaz de entregar hasta 600 mA por salida.

La figura 5.2.2 muestra la configuración de las terminales del circuito.

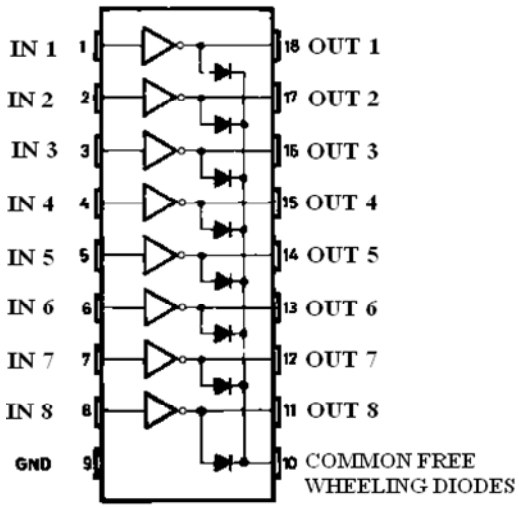


Figura 5.2.2.– ULN2803A

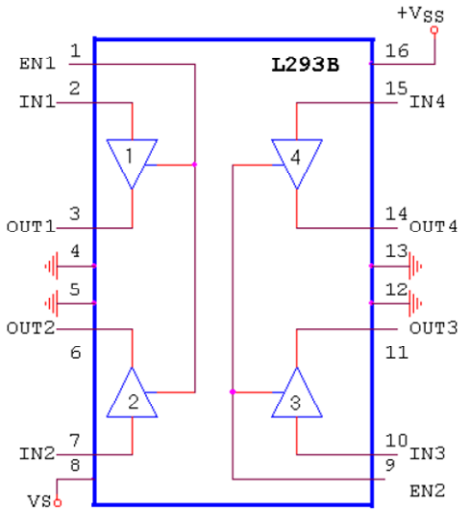


Figura 5.2.3.- Circuito L293B

Existe también el circuito integrado L293B, el cual cuenta con 4 canales de entrada compatibles con nivel TTL, además de contar con una Terminal de alimentación independiente para el manejo de las cargas y una terminal de habilitación para cada par de canales.

Este circuito entrega 1 ampere de corriente nominal y picos de corriente de hasta 2 amperes por canal.

Como podemos observar la distribución de sus terminales es muy practica tanto para este circuito como para el circuito anterior lo que los hace sumamente prácticos para el manejo de muchos dispositivos de salida como los motores a pasos por ejemplo.

La figura 5.2.3 muestra la distribución de las terminales del circuito así como su diseño interno básico.

5.2.2.- Drivers para Displays.

Cuando surge la necesidad de manejar indicadores visuales como los displays de 7 segmentos, es necesario utilizar circuitos auxiliares compatibles con entrada en formato BCD.

La serie de circuitos integrados LS4, LS48, LS49 son los más utilizados para manejo de displays ya que cuentan con numerosas ventajas en las que destacan principalmente las salidas de tipo colector abierto y las resistencias internas de pull-up, con lo que se eliminan las resistencias externas.

Otra de las ventajas del circuito es que una vez alimentado, puede utilizarse sin las demás líneas de control que pose, únicamente necesita las entradas de código a mostrar.

En la figura 5.2.2 se muestra la distribución de las terminales del circuito y la distribución de los segmentos de un display. Esta figura fue tomada directamente de la tabla de datos del fabricante.

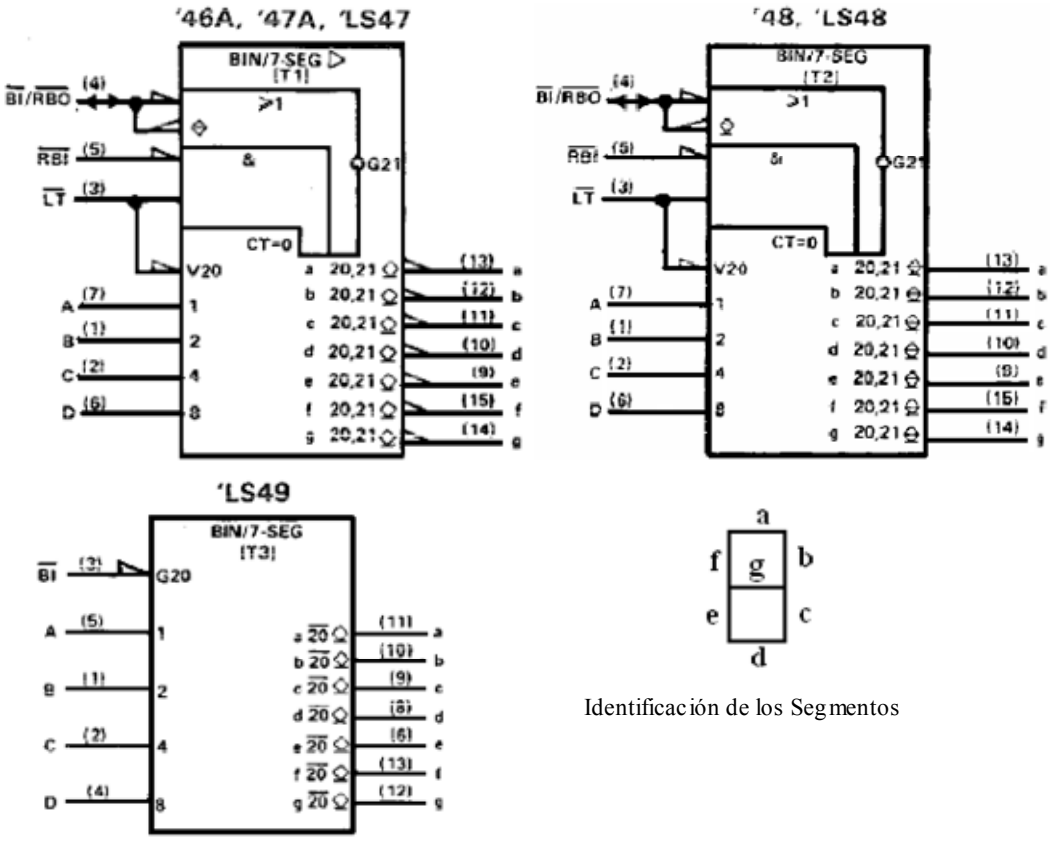


Figura 5.2.4.– Diagrama de Terminales de los circuitos LS46, LS47, LS48, e identificación de los segmentos de un display.

5.2.3.- Opto Acopladores.

Un optoacoplador es un dispositivo semiconductor formado por un fotoemisor, un fotoreceptor y entre ambos un material que sirve de camino para transmitir la luz, como se muestra en la figura 5.2.4. Todos estos elementos se encuentran dentro de un encapsulado que por lo general es del tipo DIP.

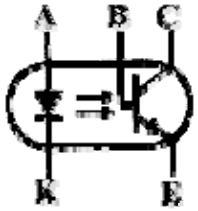


Figura 5.2.5.– Optoacoplador

El funcionamiento de este dispositivo consiste en aplicar la señal de entrada al fotoemisor y la salida es tomada del fotoreceptor. Los optoacopladores son capaces de convertir una señal eléctrica en una señal luminosa modulada y volver a convertirla en una señal eléctrica.

Es por esto que un optoacoplador tiene la gran ventaja de proveer un aislamiento eléctrico que puede establecerse entre los circuitos de entrada y salida brindando una protección muy fuerte contra descargas de voltaje de cualquier tipo.

Existen diferentes tipos de optoacopladores entre los cuales destacan los siguientes:

Fototransistor: se compone de un optoacoplador con una etapa de salida formada por un transistor BJT.
Fototriac: se compone de un optoacoplador con una etapa de salida formada por un triac.
Fototriac de paso por cero: Optoacoplador en cuya etapa de salida se encuentra un triac de cruce por cero. El circuito interno de cruce por cero conmuta al triac sólo en los cruce por cero de la corriente alterna.

Un circuito práctico para el manejo de cargas con voltaje alterno es el MOC3020, el cual cuenta con un triac a la salida y entradas compatibles al nivel TTL, lo que permite trabajar con voltajes alternos directamente a la salida del microcontrolador contando con la protección del aislamiento óptico.

La distribución de las terminales y su diseño interno se muestra en la figura 5.2.5.

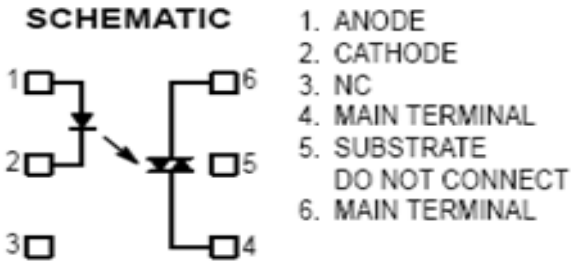


Figura 5.2.6.– MOC3020 para Cargas de Corriente Alterno

Otro circuito igual de práctico es el 4N32, pero a diferencia del anterior este cuenta con un arreglo de transistores darlington a la salida y tiene una mayor sensitividad a entradas de corriente muy bajas. Este optoacoplador resulta muy útil cuando la carga requiere una cantidad de corriente continua mayor a la que puede entregar el microcontrolador.

En la figura 5.2.6 se observa la distribución de las terminales así como su diseño interno.

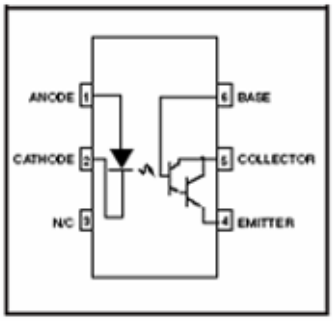


Figura 5.2.7.– Optoacoplador Photodarlington

Es importante mencionar que hay que incluir una resistencia en serie a la entrada del optoacoplador para limitar la corriente que entra y no dañar el fotodiodo.

5.3.- Acondicionamiento de Señales.

Cuando el microcontrolador maneja datos de entrada o salida digitales, estos se encuentran en nivel TTL por lo que no presentan mayor inconveniente, pero cuando es necesario manejar datos provenientes del mundo real a través del convertidor analógico a digital nos encontramos con señales eléctricas de diferente naturaleza que suelen ser incompatibles con los niveles de voltaje a los que opera el convertidor. Es por esto que es necesario adaptar estas señales para que puedan ser procesadas por el convertidor analógico a digital, a esta adecuación de la señal proveniente del sensor para que pueda ser utilizada por el convertidor, se le llama acondicionamiento de señales.

El acondicionamiento de señales, se realiza a partir de componentes de electrónica analógica generalmente, aunque hoy en día existen nuevos dispositivos digitales capaces de colaborar en ciertos puntos del acondicionamiento, pero básicamente los principios básicos del acondicionamiento pueden ser realizados a partir de la electrónica analógica convencional.

Las funciones de los dispositivos electrónicos que realizaran el acondicionamiento de señal pueden dividirse en los siguientes puntos

- Proveer la fuente de alimentación para el sensor.
- Compensar corrimiento (offset) de señal.
- Proveer calibración para el sistema y sensores.
- Proveer amplificación de la señal de entrada para acoplarse a los requerimientos de entrada del convertidor.
- Compensar por diferencias en referencias de potencial cero (común).
- Limitar ancho de banda de la señal.

Para realizar estas funciones, se usan varios dispositivos electrónicos. Es decir, los primeros 3 puntos son provistos por una unidad llamada acondicionador de señal, los 2 siguientes por un amplificador y el ultimo punto por un filtro.

En la práctica, estas unidades pueden presentarse en un solo ensamble o en varias piezas. Pero podríamos dividir funcionalmente el acondicionamiento de señal en tres unidades: Acondicionado, Amplificador y Filtro.

A continuación detallamos cada uno de los puntos que formal el acondicionamiento de la señal.

Proveer la fuente de alimentación para el sensor.-

La mayoría de los sensores carecen de su fuente de alimentación, es por ello que esta etapa consiste en una fuente de alimentación en forma de voltaje o corriente continua regulados con capacidad de medición remota, limitación de corriente para proteger el sensor, buena estabilidad, ruido mínimo y debe ser fácilmente ajustable.

Compensar corrimiento (offset) de señal.-

Cuando los límites del voltaje de operación superior e inferior de nuestro sensor no son iguales a los límites en los que opera el convertidor, es necesario utilizar un circuito electrónico para ajustar el rango a los valores utilizados por el convertidor, lo que repercute en brindar mayor resolución a la señal a convertir.

Calibración.-

Consiste en una etapa que nos permite realizar ajustes finos a las señales provenientes de los sensores para corregir pequeñas variaciones ocasionadas por los sensores.

Amplificación de la Señal.-

Las funciones básicas del amplificador son: escalar las entradas de los convertidores y compensar diferencias en potencial cero (tierra) que pueden existir entre el sensor y el sistema de datos.

Debido a que los convertidores operan con rangos fijos (± 5V, ± 10V), es importante utilizar toda la resolución del convertidor escalando apropiadamente la señal de entrada. Para lograr esto se puede utilizar un amplificador de salida sencilla o diferencial.

Si existe una diferencia en el potencial de cero (tierra) entre el sensor y el sistema de datos, se prefiere un amplificador diferencial, que aísle los puntos de referencia del convertidor y el sensor.

Compensar por diferencias en referencias de potencial cero (común).-

Debido a que los sensores de medición frecuentemente están localizados en forma remota al sistema de adquisición de datos, es muy probable que las señales se encuentren

en una región de potencial cero que difiere del potencial cero del sistema de datos, por lo que es necesario verificar el rechazo de modo común de los amplificadores, balancear las líneas de las señales y aumentar las impedancias de fuga.

Limitar ancho de banda de la señal.-

Esta medida nos es útil para evitar la contaminación por ruido de nuestra señal y se logra aplicando filtros que delimiten el ancho de banda de nuestra señal.

5.4.- Indicadores Visuales

5.4.1.- Diodos Emisores de Luz (LED).

Básicamente un LED es un diodo que emite luz, y es nombrado así por sus siglas en ingles “Light Emitting Diode”. Estos dispositivos consumen alrededor de 4 mA, y operan a diferentes voltajes siendo 5V y 12V los más utilizados comúnmente.

El funcionamiento de un led es muy sencillo y prácticamente el color de estos esta definido por los umbrales de conducción de los materiales es decir si la energía que se necesita es pequeña, se tendrá que dicha energía se emitirá en ondas infrarrojas de relativamente baja frecuencia, si el material necesitara mas energía para que se produzca el paso de la corriente, las ondas que emitirá el diodo tendrían mas energía y se pasaría de emitir luz infrarroja a roja, naranja, amarilla, verde, azul, violeta y ultravioleta, por lo que cabe mencionar que el diodo emitiría luz monocromática en el espectro visible y más allá.

Otro aspecto importante es que a mas alta frecuencia mayor será la caída de tensión por lo que pasaremos de 0.6v de caída para un diodo normal a 1,3 v para un led infrarrojo, 1,8 v para un led rojo, 2,5 v para uno verde, y 4,3v para un led azul y más de 5v para un led ultravioleta.

A continuación la figura 5.4.1 muestra el encapsulado típico T1 $\frac{3}{4}$ de 5mm de diámetro con sus partes principales.

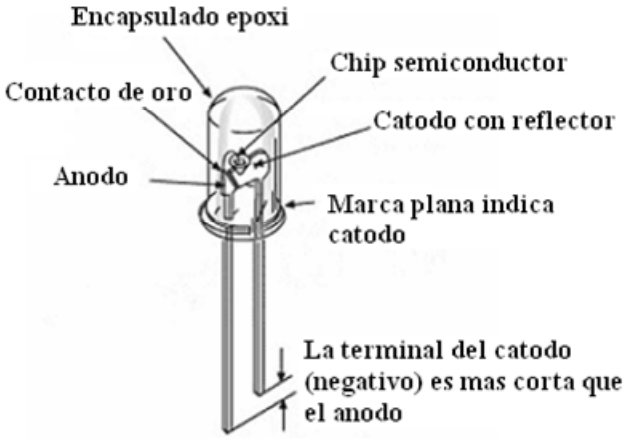


Figura 5.4.1– Partes de un Led.

5.4.2.- Displays de 7 segmentos

Los displays son ampliamente utilizados para visualizar valores o datos obtenidos a través del microcontrolador, los displays mas utilizados por su costo y versatilidad son los displays de 7 segmentos de cátodo o ánodo común, sencillos o dobles, como los que muestra la figura 5.4.2 donde podemos observar la distribución de sus terminales.

Estos displays consumen alrededor de 4 mA por segmento, y para el desarrollo de las prácticas descritas en esta tesis es conveniente utilizar los que operan a 5 volts, aunque también los hay para otros voltajes.

Para poder utilizar correctamente estos dispositivos es necesario que el código a mostrar este en formato BCD, de lo contrario los datos se visualizaran de forma errónea por los displays, para esto puede utilizarse como interfaz los circuitos LS47, LS48 y LS49 mencionados en el subtema 5.2.2.

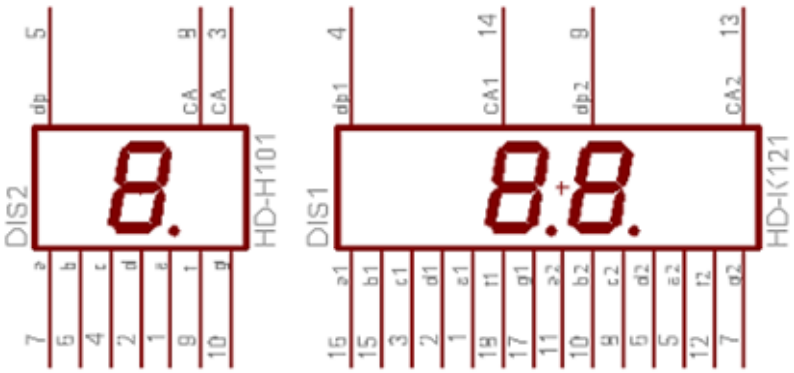


Figura 5.4.2.– Display sencillo y doble de 7 segmentos con punto decimal.

En la figura 5.4.3 se muestran los números que pueden ser mostrados por los displays a través de las diferentes combinaciones de datos de entrada en decimal que se muestran en la tabla 5.4.1.

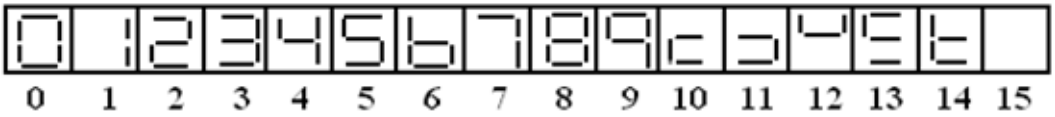


Figura 5.4.3 - Designación numérica con datos de entrada en decimal..

	Entradas en BCD				Salidas						
Decimal	D	C	B	A	a	b	c	d	e	f	g
0	L	L	L	L	ON	ON	ON	ON	ON	ON	OFF
1	L	L	L	H	OFF	ON	ON	OFF	OFF	OFF	OFF
2	L	L	H	L	ON	ON	OFF	ON	ON	OFF	ON
3	L	L	H	H	ON	ON	ON	ON	OFF	OFF	ON
4	L	H	L	L	OFF	ON	ON	OFF	OFF	ON	ON
5	L	H	L	H	ON	OFF	ON	ON	OFF	ON	ON
6	L	H	H	L	OFF	OFF	ON	ON	ON	ON	ON
7	L	H	H	H	ON	ON	ON	OFF	OFF	OFF	OFF
8	H	L	L	L	ON	ON	ON	ON	ON	ON	ON
9	H	L	L	H	ON	ON	ON	OFF	OFF	ON	ON
10	H	L	H	L	OFF	OFF	OFF	ON	ON	OFF	ON
11	H	L	H	H	OFF	OFF	ON	ON	OFF	OFF	ON
12	H	H	L	L	OFF	ON	OFF	OFF	OFF	ON	ON
13	H	H	L	H	ON	OFF	OFF	ON	OF	ON	ON
14	H	H	H	L	OFF	OFF	OFF	ON	ON	ON	ON
15	H	H	H	H	OFF	OFF	OFF	OFF	OFF	OFF	OFF

Tabla 4.1.1.– Combinaciones de entrada y salidas respectivas.

Cabe mencionar que cada puerto de 8 bits puede manejar de forma directa un display doble, pero existen otras formas de multiplexar datos de salida y habilitar displays como la técnica de barrido, en la cual se conectan dos displays dobles a un puerto y a través de otras cuatro salidas digitales se habilita un numero del display a la vez, permitiendo así conectar más de 1 dispositivo por puerto.

5.5.- Convertidor de Digital a Analógico

La función de los convertidores de digital a analógico es la de convertir un valor en código digital en una señal analógica ya sea en corriente o voltaje de magnitud equivalente.

Esto es necesario debido a que la mayoría de los dispositivos controlados por un microcontrolador son dispositivos analógicos que operan con información digital, por lo que es necesario convertir la información que envía el microcontrolador o cualquier sistema digital a un formato analógico.

Un ejemplo de esto seria el control de una válvula proporcional, la cual opera mediante una señal de voltaje analógico, por lo que el microcontrolador necesita convertir el valor digital a su equivalente en magnitud de voltaje analógico para realizar el control de esta.

Debido a esto los convertidores de digital a analógico son ampliamente utilizados como interfaces de señales producidas por computadoras o sistemas digitales hacia dispositivos de control de procesos.

Hoy en día existen diferentes tipos de convertidores de digital a análogo como los convertidores de resistencias ponderadas, o los convertidores en red R2R. Los cuales se detallan a continuación:

Convertidor de Resistencias Ponderadas:

El bit más significativo (de más peso) esta representado por el interruptor S1 y el de menos peso por S4. Así tenemos que se produce un voltaje analógico a la salida que tiene relación con el valor del código digital en la entrada. Dicho de otra forma, las corrientes a través de las resistencias de entrada se suman para determinar la corriente que fluye a través de la resistencia de retroalimentación que define el valor del voltaje de salida y por otra parte, de resistencias ponderadas porque las resistencias guardan en su valor una relación de 2, lo que le da el peso en relación al lugar que ocupa cada bit en el código digital.

Otra forma de realizar un análisis de circuitos con amplificadores operacionales es asumiendo que la entrada inversora está a cero volts, calculando la corriente de entrada

como el voltaje de entrada dividido por la resistencia de entrada (si hay voltaje en varias resistencias de entrada se hace la suma de sus corrientes) y multiplicando esa corriente calculada por la resistencia en retroalimentación, con lo que se determina el voltaje de salida.

Si el voltaje de entrada es el mismo para todas las resistencias, al ir duplicando el valor de las resistencias de entrada se reduce a la mitad la corriente de entrada y en esa misma proporción cambiará el voltaje de salida. La tabla 5.5.1 muestra el voltaje de salida con diferentes combinaciones de entrada.

La figura 5.5.1 muestra el diagrama de un convertidor de resistencias ponderadas de 4 bits.

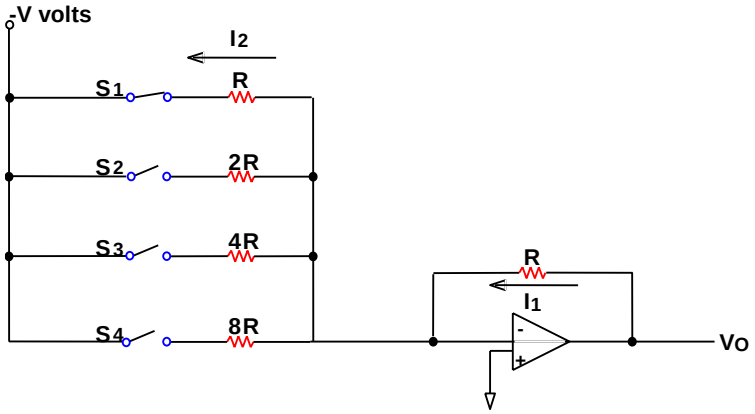


Figura 5.5.1.– Convertidor de Resistencias Ponderadas.

Estados de los interruptores				Voltaje de salida del amplificador
S_1	S_2	S_3	S_4	V_0
Abierto	Abierto	Abierto	Abierto	Cero
Abierto	Abierto	Abierto	Cerrado	1/8 V
Abierto	Abierto	Cerrado	Abierto	1/4 V
Abierto	Abierto	Cerrado	Cerrado	3/8 V
Abierto	Cerrado	Abierto	Abierto	1/2 V
Abierto	Cerrado	Abierto	Cerrado	5/8 V
Abierto	Cerrado	Cerrado	Abierto	3/4 V
Abierto	Cerrado	Cerrado	Cerrado	7/8 V
Cerrado	Abierto	Abierto	Abierto	V
Cerrado	Abierto	Abierto	Cerrado	1 1/8 V
Cerrado	Abierto	Cerrado	Abierto	1 1/4 V
Cerrado	Abierto	Cerrado	Cerrado	1 3/8 V
Cerrado	Cerrado	Abierto	Abierto	1 1/2 V
Cerrado	Cerrado	Abierto	Cerrado	1 5/8 V
Cerrado	Cerrado	Cerrado	Abierto	1 3/4 V
Cerrado	Cerrado	Cerrado	Cerrado	1 7/8 V

Tabla 5.5.1.– Estados de los interruptores.

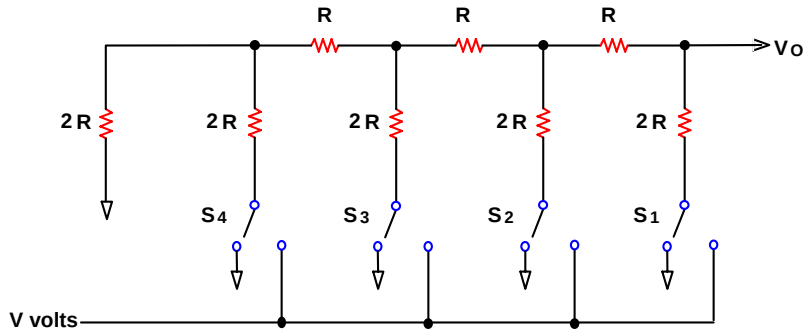
Convertidor mediante red R2R:

En este caso, cada interruptor esta conectado al voltaje de referencia V, o a tierra. Si se aterrizan todos los interruptores, el voltaje de salida es cero debido a que no existe voltaje ni corriente en el circuito.

Al conectarse el voltaje de referencia a cualquiera de las resistencias 2R, la corriente de entrada siempre es la misma, pero al llegar al nodo superior se divide entre 2.

Debido a esto, a medida que la resistencia 2R se encuentra más alejada (a la izquierda) del amplificador, la corriente con que contribuye a la entrada es menor. La división entre 2 realiza la ponderación del peso del bit a su contribución en el voltaje de salida.

La figura 5.5.2 muestra el diagrama para el convertidor mediante red R2R.



Así, en forma semejante al análisis del amplificador sumador, si se realizan combinaciones para aterrizar los interruptores se obtiene una amplia variedad de valores de salida.

La tabla 5.5.2 presenta las diferentes combinaciones que se pueden realizar dependiendo de los interruptores que se encuentren abiertos y cerrados.

Estados de los interruptores				Voltaje de salida del amplificador
S_1	S_2	S_3	S_4	V_O
Tierra	Tierra	Tierra	Tierra	Cero
Tierra	Tierra	Tierra	V	1/16 V
Tierra	Tierra	V	Tierra	1/8 V
Tierra	Tierra	V	V	3/16 V
Tierra	V	Tierra	Tierra	1/4 V
Tierra	V	Tierra	V	5/16 V
Tierra	V	V	Tierra	3/8 V
Tierra	V	V	V	7/16 V
V	Tierra	Tierra	Tierra	1/2 V
V	Tierra	Tierra	V	9/16 V
V	Tierra	V	Tierra	5/8 V
V	Tierra	V	V	11/16 V
V	V	Tierra	Tierra	3/4 V
V	V	Tierra	V	13/16 V
V	V	V	Tierra	7/8 V
V	V	V	V	15/16 V

Tabla 5.5.2.– Estados de los interruptores del DAC R2R.

De nueva cuenta se tienen 16 voltajes de salida posibles con cuatro interruptores, aún cuando los voltajes son diferentes a los de la tabla 2.1.

La mayor ventaja del DAC escalera red R-2R es que la técnica es válida sin importar el número de bits, sólo se requieren dos tipos de resistencias y pueden ser seleccionadas para tener una resistencia moderada y prevenir la carga del voltaje de referencia.

CAPITULO 6

PRACTICAS

A continuación se elaboro una serie de prácticas divididas en 3 subtemas diseñadas para lograr un mejor aprendizaje en el manejo del microcontrolador.

En un principio se trata de utilizar los diferentes dispositivos de interfaz para adaptar nuestro microcontrolador al mundo real. Para esto se elaboro una serie de prácticas muy sencillas que muestran el uso de los diferentes dispositivos electrónicos mas utilizados tanto para interfaz como para acondicionamiento.

Posteriormente siguiendo una secuencia cronológica en el aprendizaje se muestra con otra serie de prácticas el manejo de las instrucciones básicas de diferentes tipos con el fin de familiarizar al lector con el tipo de instrucciones más elementales usadas en el desarrollo de diversos programas.

Y finalmente se trata otra serie de prácticas diseñadas con el fin de mostrar la configuración estándar para los principales periféricos de nuestro microcontrolador.

6.1.- Practicas Para Capitulo 3

Ejemplo 1.- Direccionamiento Inmediato

En este programa se utiliza el direccionamiento inmediato para almacenar #$FF en nuestro acumulador, mismo que posteriormente se envía hacia el registro de dirección del puerto a configurándolo como salidas. Posteriormente mediante direccionamiento extendido se carga el acumulador a con el contenido de la dirección $0850 que contiene el valor #$55 para finalmente enviar este valor hacia el puerto A. Si el valor que deseamos enviar estuviera en una dirección que necesitara solamente un byte para especificar su ubicación, es decir $00FF o menor, pudiéramos utilizar direccionamiento directo, pero como en este ejemplo se necesitan 2 bytes para especificar la dirección ($0850) es necesario utilizar direccionamiento extendido.

En los términos del lenguaje maquina para el 68HC12 la instrucción load (cargar) significa copiar, por lo que la instrucción LDAA $0850 transfiere una copia del contenido de esta localidad dentro del acumulador A.

Con la ayuda de un indicador visual como el circuito 1 del apéndice A, podemos observar el valor enviado hacia el puerto.

```
; DECLARACIÓN DE ETIQUETAS
PORTA  EQU    $0000  ; REGISTRO DE DATOS DEL PUERTO A
DDRA   EQU    $0002  ; REGISTRO DIRECCION DEL PUERTO A
VAR    EQU    $0850  ; DEFINE UNA LOCALIDAD DE MEMORIA PARA
                     ; LA CONSTANTE VAR
; INICIO DEL PROGRAMA

         ORG  $0800

         LDAA   #$FF  ; SE CONFIGURA EL PUERTO A
         STAA   $0002 ; COMO SALIDAS
         LDAA   #$55  ; SE CARGA EL VALOR #$55 EN EL
                      ; ACUMULADOR A
         STAA   VAR   ; SE ENVIA A LA LOCALIDAD DE MEMORIA
                      ; RESERVADA PARA VAR
         LDAA $0850   ; SE CARGA EL VALOR DE LA DIRECCION
                      ; $0850 EN EL ACUMULADOR A
         STAA PORTA   ; SE ENVIA HACIA EL PUERTO A
FIN:     BRA  FIN     ; SE CICLA EL PROCRAMA MEDIANTE UN
                      ; BRINCO INCONDICIONAL
         END          ; FIN DEL PROGRAMA
```

Ejemplo 2.- Direccionamiento Indexado

.

En el siguiente programa se accede ahora al valor enviado por el puerto mediante direccionamiento indexado. Esto se logra colocando la dirección de inicio de la variable VAR en el registro índice X de la forma LDX VAR, posteriormente se carga el acumulador con la dirección de inicio de la variable VAR mas un offset de #$10 mediante la instrucción LDAA $10,X con lo que se obtiene la dirección final que contiene el valor que será enviado por el puerto a.

Puede utilizar el circuito 2 del apéndice A para visualizar el resultado.

```
; PUERTO A MEDIANTE DIRECCIONAMIENTO INDEXADO DE 9 BITS DE
OFFSET

; DECLARACIÓN DE ETIQUETAS
PORTA   EQU     $0000   ; REGISTRO DE DATOS DEL PUERTO A
DDRA    EQU     $0002   ; REGISTRO DIRECCION DEL PUERTO A

; INICIO DEL PROGRAMA
        ORG     $0800

INI     LDAA    #$FF    ; SE CONFIGURA EL PUERTO A
        STAA    $0002   ; COMO SALIDAS
        LDX     #VAR     ; SE INICIALIZA EL REGISTRO X CON LA
                         ; DIRECCION DE VAR
        LDAA    $10,X   ; SE CARGA EN A EL CONTENIDO DE LA
                        ; DIRECCIÓN ALAMCENADA
                         ; EN EL REGISTRO X CON UN OFSET DE #$10
        STAA    PORTA   ; SE ENVIA A EL PUERTO A
        BRA     INI     ; SE CICLA EL PROGRAMA INDEFINIDAMENTE

VAR DC.B    $00; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 00
    DC.B    $01; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 00
    DC.B    $02; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $03; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 55
    DC.B    $04; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $05; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $06; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $07; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $08; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $09; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $10; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $11; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $12; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $13; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $14; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $15; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
    DC.B    $16; DEFINE UNA CONSTANTE DE UN BYTE CON EL VALOR 01
```

Ejemplo 3.- Suma de 2 localidades de Memoria con Direccionamiento Extendido

Caso 1

A continuación se realiza una suma de un byte sin acarreo. Previamente se almacenan los operandos en dos direcciones $0900 y $0902, las cuales serán utilizadas para efectuar la multiplicación, finalmente se envía el resultado de la operación haciendo el ajuste a decimal del mismo hacia el puerto a.

Para poder visualizar el resultado de los 2 casos utilice el circuito 2 en el apéndice A

$04 + 12 = C \rightarrow 12$

Caso 1

```
#INCLUDE hc12.inc     ; ESTA LIBRERIA INCLUYE TODOS LOS
PRINCIPALES
                      ; PERIFERICOS DEL PUERTO

        ORG RAM_START

        LDAA    #$FF    ; CONFIGURAMOS EL PUERTO A
        STAA    DDRA    ; COMO SALIDAS
        LDAA    #$04    ; ALMACENAMOS UN #$04
        STAA    $0900   ; EN LA DIRECCION $0900
        LDAA    #$08    ; ALMACENAMOS UN #$08
        STAA    $0902   ; EN LA DIRECCION $0902
        LDAA    $0900   ; CARGAMOS EL ACUMULADOR A CON
                        ; EL CONTENIDO DE LA DIREECION $0900
        ADDA    $0902   ; AGREGAMOS AL ACUMULADOR A EL CONTENIDO
                        ; DE LA DIRECCION $0902
        DAA             ; AJUSTE A DECIMAL DEL RESULTADO 0C->12
        STAA    PORTA   ; ENVIAMOS EL RESULTADO HACIA EL PUERTO A
FIN     BRA     FIN    ; CICLO INFINITO
```

Caso 2

En este otro caso como el resultado de la suma es un 14 compatible con el formato BCD, no es necesario efectuar ningún ajuste a decimal y se envía el resultado hacia el puerto.

```
#INCLUDE hc12.inc     ; ESTA LIBRERIA INCLUYE TODOS LOS
PRINCIPALES
                      ; PERIFERICOS DEL PUERTO
; INICIO DEL PROGRAMA

        ORG RAM_START


        LDAA    #$04    ; ALMACENAMOS UN #$08
        STAA    $0900   ; EN LA DIRECCION $0900
        LDAA    #$10    ; ALMACENAMOS UN #$12
        STAA    $0902   ; EN LA DIRECCION $0902
        LDAA    $0900   ; CARGAMOS EL ACUMULADOR A CON
                        ; EL CONTENIDO DE LA DIREECION $0900
        ADDA    $0902   ; AGREGAMOS AL ACUMULADOR A EL CONTENIDO
                        ; DE LA DIRECCION $0902
        STAA    PORTA   ; ENVIAMOS EL RESULTADO "14" AL PUERTO A
FIN     BRA     FIN    ; CICLO INFINITO
```

Ejemplo 4.- Suma de Dos localidades de Memoria con Acarreo Mediante Direccionamiento Extendido.

Ahora se realiza una suma de 2 bytes con acarreo, almacenando mediante el doble acumulador D los operandos dentro de dos direcciones, $0900 y $0902. Al utilizar el acumulador D formado por los acumuladores A y B, los bytes menos significativos se guardan en B y los mas significativos se guardan en A, por lo que al almacenar estos

valores en una localidad de memoria se almacena el byte mas significativo en la dirección indicada y el byte menos significativo en la siguiente localidad de memoria.

Por lo anterior al realizar una suma de 2 bytes, se realiza primero la adición de los dos bytes menos significativos y después se efectúa la adición de los dos bits más significativos junto con el acarreo en caso de haberlo.

En este ejemplo no es necesario efectuar el ajuste a decimal, y se utiliza el circuito 2 en el apéndice A para visualizar el resultado.

```
#INCLUDE hc12.inc      ; ESTA LIBRERIA INCLUYE TODOS LOS
PRINCIPALES
                       ; PERIFERICOS DEL PUERTO

        ORG RAM_START


        LDAA    #$FF    ; CONFIGURAMOS LOS PUERTOS:
        STAA    DDRA    ; A COMO SALIDAS
        STAA    DDRB    ; B COMO SALIDAS
        LDD     #$5077  ; ALMACENAMOS UN #$5077
        STD     $0900   ; EN LA DIRECCION $0900
        LDD     #$2589  ; ALMACENAMOS UN #$2589
        STD     $0902   ; EN LA DIRECCION $0902
        LDAA    $0901   ; CARGAMOS EL ACUMULADOR A CON
                        ; EL CONTENIDO DE LA DIREECION $0901
        ADDA    $0903   ; AGREGAMOS AL ACUMULADOR A EL CONTENIDO
                        ; DE LA DIRECCION $0903
        STAA    PORTB   ; ENVIAMOS EL RESULTADO AL PUERTO B
        LDAA    $0900   ; CARGAMOS EL ACUMULADOR B CON
                        ; EL CONTENIDO DE LA DIREECION $0900
        ADCA    $0902   ; AGREGAMOS AL ACUMULADOR A EL CONTENIDO
                        ; DE LA DIRECCION $0902 Y EL ACARREO
        STAA    PORTA   ; ENVIAMOS EL RESULTADO AL PUERTO A

FIN   BRA   FIN         ; CICLO INFINITO
```

Ejemplo 5.- Resta al contenido del Acumulador A

Caso 1

En este programa se realiza una resta de 1 byte en el acumulador A utilizando la instrucción SUBA, posteriormente se envía el resultado al puerto A.

Para ambos casos utilice el circuito 2 del apéndice A para visualizar el resultado.

```
#INCLUDE hc12.inc       ; ESTA LIBRERIA INCLUYE TODOS LOS PRINCIPALES
                        ; PERIFERICOS DEL PUERTO
```

```
        ORG RAM_START

        LDAA    #$FF    ; CONFIGURAMOS EL PUERTO A
        STAA    DDRA    ; COMO SALIDAS

        LDAA    #$09    ; ALMACENAMOS UN #$09
        SUBA    #$05    ; SE LE RESTA UN #$05 AL ACUMULADO0R

        STAA    PORTA   ; ENVIAMOS EL RESULTADO #$04
                        ; HACIA EL PUERTO A
FIN     BRA     FIN     ; CICLO INFINITO
```

Caso 2

Este otro ejemplo realiza la misma resta pero con la instrucción SBA, esta instrucción nos permite realizar la operación sin declarar el valor que se restara al acumulador A, ya que este se encuentra almacenado en los acumuladores A y B. Posteriormente se envía el resultado al puerto A

```
        LDAA    #$FF    ; CONFIGURAMOS EL PUERTO A
        STAA    DDRA    ; COMO SALIDAS

        LDAA    #$09    ; ALMACENAMOS UN #$09
        LDAB    #$05    ; SE LE RESTA UN #$05 AL ACUMULADO0R
        SBA
        STAA    PORTA   ; ENVIAMOS EL RESULTADO #$04
                        ; HACIA EL PUERTO A
FIN     BRA     FIN     ; CICLO INFINITO
```

Ejemplo 6.- Multiplicación de 8 bits.

Las multiplicaciones que puede ejecutar el microcontrolador son de dos tipos, es decir entre factores de 8 bits o de 16 bits, en el primer caso es de esperarse un resultado de 16 bits y para el segundo un número de 32 bits.

A continuación realizaremos una multiplicación de 8 bits, para la cual es necesario almacenar en los acumuladores A y B los factores a multiplicar.

Finalmente la operación de multiplicación la realiza la instrucción MUL que al ser ejecutada coloca la parte baja del resultado en el acumulador B y la parte alta en el acumulador A.

Se recomienda utilizar el circuito 2 del apéndice A para visualizar el resultado.

```
#INCLUDE hc12.inc       ; ESTA LIBRERIA INCLUYE TODOS LOS PRINCIPALES
                        ; PERIFERICOS DEL PUERTO

        ORG RAM_START

        LDAA    #$FF    ; CONFIGURAMOS LOS PUERTOS
        STAA    DDRA    ; A Y B COMO SALIDAS
        STAA    DDRB    ;
        LDAA    #$78    ; ALMACENAMOS UN #$78 EN A
        LDAB    #$34    ; ALMACENAMOS UN #$32 EN B
        MUL             ; SE EJECUTA LA MULTIPLICACIÓN
        STAB    PORTB   ; COMO EL RESULTADO ES DE  16 BITS
                        ; SE ENVIA LA PARTE BAJA "60" AL PUERTO B
        STAA    PORTA   ; Y LA PARTE ALTA "18" HACIA EL PUERTO A
                        ; QUEDANDO DE RESULTADO 1860 EN HEXADECIMAL
                        ; COMO SE MUESTRA EN AMBOS PUERTOS
FIN     BRA     FIN     ; CICLO INFINITO
```

Ejemplo 7.- Multiplicación de 16 bits.

Este ejemplo realiza una multiplicación de un factor de 16 bits por otro de 8 bits. Para que esto sea posible se almacenan los factores en los registros D y Y, ambos de 16 bits. El resultado se almacena en estos mismos registros, en D la parte baja de los 32 bits y en Y la parte alta.

Para este caso en particular el resultado que se obtiene no es mayor de 16 bits por lo que solo enviaremos la parte baja “D” a los puertos A y B para su posterior visualización.

Cabe mencionar que el acumulador D se compone a la vez por los registros acumuladores A como parte alta y B como parte baja.

Para visualizar el resultado de la operación utilice el circuito 2 del apéndice A.

```
#INCLUDE hc12.inc       ; ESTA LIBRERIA INCLUYE TODOS LOS PRINCIPALES
                        ; PERIFERICOS DEL PUERTO

        ORG RAM_START

        LDAA    #$FF    ; CONFIGURAMOS LOS PUERTOS
        STAA    DDRA    ; A Y B COMO SALIDAS
        STAA    DDRB    ;
        LDD     #$2222  ; ALMACENAMOS UN #$2222 EN D
        LDY     #$02    ; ALMACENAMOS UN #$03 EN Y
        EMUL            ; SE EJECUTA LA MULTIPLICACIÓN
        STAB    PORTB   ; COMO EL RESULTADO ES MENOR A 32 BITS
                        ; SE ENVIA SOLAMENTE LA PARTE BAJA D
        STAA    PORTA   ; A LOS PUERTOS A Y B
                        ; QUEDANDO DE RESULTADO 4444 EN HEXADECIMAL
                        ; COMO SE MUESTRA EN AMBOS PUERTOS
FIN     BRA     FIN     ; CICLO INFINITO
```

Ejemplo 8.- División de 16 bits.

El microcontrolador soporta diferentes tipos de divisiones, es decir, entre números enteros o fracciones, con signo o sin signo, así como pueden tener un dividendo de 16 o 32 bits pero todas cuentan con un divisor de 16 bits.

El ejemplo a continuación muestra como realizar una división con un dividendo y divisor ambos de 16 bits, esperando obtener un cociente de tipo entero ya que la instrucción que ejecuta la operación solo soporta este formato de operación aritmética.

Para poder realizar esta operación se coloca el dividendo dentro del doble acumulador D, el divisor dentro del registro índice X, el cociente es colocado en X y finalmente el residuo en el doble acumulador D. La instrucción para ejecutar este tipo de división es IDIV.

Para visualizar el resultado de la operación utilice el circuito 2 del apéndice A.

```
#INCLUDE hc12.inc       ; ESTA LIBRERIA INCLUYE TODOS LOS PRINCIPALES
                        ; PERIFERICOS DEL PUERTO

        ORG RAM_START

        LDAA    #$FF    ; CONFIGURAMOS LOS PUERTOS
        STAA    DDRA    ; A Y B COMO SALIDAS
        STAA    DDRB    ;
        LDD     #$9999  ; ALMACENAMOS UN #$9999 EN D
        LDX     #$1450  ; ALMACENAMOS UN #$1450 EN Y
        IDIV            ; SE EJECUTA LA DIVISIÓN
        XGDX            ; COMO EL RESULTADO SE ENCUENTRA EN X
                        ; LO INTERCAMBIAMOS CON D MEDIANTE
                        ; ESTA INSTRUCCIÓN


        STAB    PORTB   ; COMO EL RESULTADO ES MENOR A 32 BITS
                        ; SE ENVIA SOLAMENTE LA PARTE BAJA D
        STAA    PORTA   ; A LOS PUERTOS A Y B
                        ; QUEDANDO DE RESULTADO #$07 EN HEXADECIMAL
                        ; COMO SE MUESTRA EN AMBOS PUERTOS
FIN     BRA     FIN     ; CICLO INFINITO
```

Ejemplo 9.- División de 32 bits.

El ejemplo a continuación muestra como realizar una división donde existe un dividendo de 32 bits y un divisor de 16 bits.

Para poder realizar esta operación se coloca la parte baja del dividendo dentro del doble acumulador D, la parte alta dentro del doble acumulador Y, y finalmente el divisor se almacena en el doble acumulador X.

Después de ejecutarse la división el divisor es colocado en el doble acumulador D y el cociente en el registro acumulador Y.

La instrucción para ejecutar este tipo de división es EDIV como veremos a continuación.

Para visualizar el resultado de la operación utilice el circuito 2 del apéndice A.

```
#INCLUDE hc12.inc       ; ESTA LIBRERIA INCLUYE TODOS LOS PRINCIPALES
                        ; PERIFERICOS DEL PUERTO
```

```
        ORG RAM_START

        LDAA    #$FF    ; CONFIGURAMOS LOS PUERTOS
        STAA    DDRA    ; A Y B COMO SALIDAS
        STAA    DDRB    ;
        LDD     #$0000  ; ALMACENAMOS UN #$0000 EN D
        LDY     #$1000  ; ALMACENAMOS UN #$1000 EN Y
        LDX     #$FFFF  ; SE ALMACENA EL DIVISOR EN X
        EDIV            ; SE EJECUTA LA DIVISIÓN
        XGDY            ; COMO EL RESULTADO SE ENCUENTRA EN Y
                        ; LO INTERCAMBIAMOS CON D MEDIANTE
                        ; ESTA INSTRUCCIÓN


        STAB    PORTB   ; SE LEE EL ACUMULADOR D
                        ; PARA ENVIAR LA PARTE BAJA AL PUERTO B
        STAA    PORTA   ; Y LA PARTE ALTA AL PUERTO A

FIN     BRA     FIN     ; CICLO INFINITO
```

6.2.- Practicas Para Capitulo 4

Ejemplo 10.- Conversión de Analógico a Digital

El siguiente ejemplo muestra como realizar una conversión a través del convertidor analógico a digital, configurado para trabajar con una resolución de 8 bits y adquiriendo la señal por el puerto 6 (PAD6).

Con la ayuda de un potenciómetro de 5 kohms conectado al puerto de entrada podemos variar el valor de una señal analógica que estará dentro del rango de 0 a 5 Volts. Para poder visualizar el resultado de la conversión se utilizara una serie de leds conectados en el puerto A, como se muestra en el circuito 1 del apéndice A. Los cuales reflejaran el valor en numeración hexadecimal a través de los leds.

```
; DECLARACIÓN DE ETIQUETAS
PORTA:      EQU   $0000 ; PUERTO A REGISTRO
DDRA:       EQU   $0002 ; PUERTO A DIRECCION DE REGISTRO
ATDCTL2:    EQU  $0062  ; ATD REGISTRO DE CONTROL 2
ATDCTL3:    EQU   $0063 ; ATD REGISTRO DE CONTROL 3
```

```
ATDCTL4:    EQU   $0064 ; ATD REGISTRO DE CONTROL 4
ATDCTL5:    EQU   $0065 ; ATD REGISTRO DE CONTROL 5
ATDSTATH:   EQU   $0066 ; ATD REGISTRO DE ESTADO
ADR2H:      EQU   $0074 ; ATD REGISTRO DE RESULTADOS DE CONVERSION 2


;INICIO DEL PROGRAMA

            ORG   $0800

            LDAA    #$FF      ; SE CONFIGURA EL PUERTO A
            STAA    $0002     ; COMO SALIDAS
MAIN:       BSR     INIT      ; SALTO A INIT SUPRUTINA PARA INICIALIZAR
                                EL ADC
            JSR     CONVERT   ; SALTO A CONVERT SUBRUTINA PARA
                                CONVERSION
            LDAA    ADR2H     ; CARGA EL ACUMULADOR A CON EL VALOR DE
                                LA VARIABLE TEMPORAL
            LDX     #02H      ; SE DIVIDE ENTRE 2
            IDIV
            STX     PORTA     ; SE ALMACENA EL RESULTADO EN PUERTO A
            LDAB    PORTA     ; CARGAR PORTB EN ACUMUADOR B
            STAB    PORTA     ; SE ENVIA EL RESULTADO AL PUERTO A
            BRA   MAIN        ;



;_____________________________________________________
;
; SUBRUTINA INIT PARA INICIALIZAR EL CONVERTIDOR
;_____________________________________________________


INIT:
            LDAA  #$80        ; PERMITE AL CONVERTIDOR FUNCIONAR
                                NORMALMENTE
            STAA  ATDCTL2     ; SE DESHABILITAN INTERRUPCIONES Y SE
                                CONFIGURA AL ADC PARA FUNCIONAR
                                NORMALMENTE
            BSR DELAY         ; SALTO A SUBRUTINA DELAY, ORIGINA UN
                                RETRASO DE 100 uS
            LDAA  #$00        ; SELECCIONA CONVERSION CONTINUA EN MODO
                                BGND
            STAA  ATDCTL3     ; IGNORA FREEZE EN ADCTL3
            LDAA  #$01        ; SELECCIONA LA VELOCIDAD DEL ADC A 2 A/D
                                CLOCKS
            STAA  ATDCTL4     ; SELECCIONA EL DIVISOR POR 4
            RTS               ; REGRESO DE SUBRUTINA

;_____________________________________________________________________
; CONFIGURA E INICIA LA CONVERSION Y PONE EL RESULTADO EN UNA LOCALIDAD
; DE MEMORIA
; CONFIGURA E INICIA LA CONVERSION
; SEÑAL ANALOGA DE ENTRADA EN PUERTO AD6
; CONVERT: USA UN SOLO CANAL
; EL RESULTADO PODRA SER ENCONTRADO EN ADR2H
```

```
;_____________________________________________________________________________

CONVERT:
            LDAA  #$06          ; INICIALIZA ATD: SCAN=0,MULT=0,PAD6.
                                  ESCRIBE LIMPIAR BANDERAS
            STAA  ATDCTL5       ; 4 CONVERSIONES EN UNA SECUENCIA DE
                                  CONVERSION SENCILLA
WTCONV:     BRCLR ATDSTATH,#$80,  WTCONV  ; ESPERA A LA BANDERA DE
                                            SECUENCIA COMPLETADA
            LDD   ADR2H         ; CARGA EL RESULTADO DE LA CONVERSION EN
                                  ADR2H, DENTRO DEL ACUMULADOR

            RTS                 ; REGRESO DESDE SUBRUTINA

;_____________________________________________________________
; SUBRUTINA DELAY 100 uS
;RETRASO REQUERIDO PARA QUE EL ATD SE ESTABILICE (100 uS)
;_____________________________________________________________

            LDAA  #$C8          ; CARGA EL ACUMULADOR CON "100 uS,
                                  VALOR DEL RETRASO (DELAY)"
DELAY:      DECA                ; DECREMENTA EL ACUMULADOR
            BNE   DELAY         ; BRANCH SI NO ES IGUAL A ZERO
            RTS                 ; REGRESO DESDE SUBRUTINA

            END                 ; FIN DEL PROGRAMA
                                ; FIN DEL PROGRAMA
```

Ejemplo 11.- Generación de señal PWM.

Este programa genera una señal a partir del modulo PWM. Básicamente el modulo nos permite seleccionar el tiempo del periodo de la señal, la duración de el ciclo y su alineación.

Para este caso se selecciono un tiempo de periodo fijo dividiendo la señal de reloj entre 128, obteniendo un periodo de 62.5 Hz, y se muestreara un valor de 8 bits mediante el canal análogo 7, el cual será colocado como duración de ciclo.

De esta manera, con la ayuda de un potenciómetro podremos variar el valor de la señal adquirida y este se vera reflejado en la duración del periodo. Para visualizar el resultado podremos colocar un simple led en el canal PWM 0 o muestrearlo con un osciloscopio.

Puede adaptarse el circuito 1 del apéndice A para poder visualizar el funcionamiento de la señal.

```
; DECLARACIÓN DE REGISTROS DEL PWM
```

```
PWCLK   EQU     $0040   ;PWM CLOCKS AND CONCATENATE REGISTER
PWPOL   EQU     $0041   ;PWM CLOCKT SELECT AND POLARITY REGISTER
PWEN    EQU     $0042   ; PWM ENABLE REGISTER
PWPER0  EQU     $004C   ; DIRECCION DEL REGISTRO DE PERIODO 0
PWTDY0  EQU     $0050   ; DIRECCION DEL REGISTRO DE DURACION DE ANCHO
                        ; DEL PULSO 0
PWCTL   EQU     $0054   ; DIRECCION DEL REGISTRO DE CONTROL DEL PWM
ATDCTL2 EQU     $0062   ; ATD REGISTRO DE CONTROL 2
ATDCTL3 EQU     $0063   ; ATD REGISTRO DE CONTROL 3
ATDCTL4 EQU     $0064   ; ATD REGISTRO DE CONTROL 4
ATDCTL5 EQU     $0065   ; ATD REGISTRO DE CONTROL 5
ATDSTATH EQU    $0066   ; ATD REGISTRO DE ESTADO
ADR3H   EQU     $0076   ; REGISTRO DE RESULTADO DE CONVERSION

; INICIO DEL PROGRAMA
        ORG $0800

; CONFIGURACION DE REGISTROS DEL PWM
        CLR     CH7
        MOVB    #$1C, PWCLK     ; SELECCIONO LOS DOS RELOJES PARA CANAL
                                ; 0,1,2 Y SE DIVIDE EL RELOJ ENTRE 128 =
                                ; 62.5 HZ
        MOVB    #$FF, PWPOL     ; SELECCIONO POLARIDAD POSITIVA Y FUENTES
                                ; DE RELOJ A,B PARA TODOS LOS CANALES
        MOVB    #$08, PWCTL     ; CONFIGURO EL PWM PARA FUNCIONAR EN EL
                                ; CENTRO DEL PERIODO
        MOVB    #$01, PWEN      ; HABILITO  CANAL 0.

; CICLO DE CONTROL PRINCIPAL
MAIN:   MOVB    #$FF, PWPER0    ; DURACION DEL PERIODO FIJAA 62.5 HZ
        MOVB    CH7, PWTDY0     ; LA DURACIÓN DEL CICLO ESTA DADA POR EL
                                ; VALOR DEL CANAL ANALOGICO
        BSR     INIT            ; SALTO A SUBRUTINA PARA INICIALIZAR EL ADC
        JSR     CONVERT         ; SALTO A CONVERT SUBRUTINA PARA CONVERSION
        BRA     MAIN            ; REGRESO DE SUBRUTINA PRINCIPA
;
; SE INICIALIZA EL CONVERTIDOR

INIT:
        LDAA    #$80    ; PERMITE AL CONVERTIDOR FUNCIONAR NORMALMENTE
        STAA    ATDCTL2 ; SE DESHABILITAN INTERRUPCIONES Y SE CONFIGURA
                        ; AL ADC PARA FUNCIONAR NORMALMENTE
        BSR     DELAY   ; SALTO A SUBRUTINA DELAY, RETRASO DE 100 uS

        LDAA    #$00    ; SELECCIONA CONVERSION CONTINUA EN MODO BGND
        STAA    ATDCTL3 ; IGNORA FREEZE EN ADCTL3

        LDAA    #$01    ; SELECCIONA LA VELOCIDAD DEL ADC A 2 A/D CLOCKS
        STAA    ATDCTL4 ; SELECCIONA EL DIVISOR POR 4
        RTS             ; REGRESO DE SUBRUTINA


; CONFIGURA E INICIA LA CONVERSION Y COLOCA EL RESULTADO EN UNA LOCALIDAD DE
; MEMORIA
; CONFIGURA E INICIA LA CONVERSION
; SEÑAL ANALOGA DE ENTRADA EN PUERTO AD6
; CONVERT: USA UN SOLO CANAL
; EL RESULTADO PODRA SER ENCONTRADO EN ADR2H

CONVERT:
        LDAA    #$17            ; INICIALIZA ATD: SCAN=0, MULT=1,PAD7.
                                ; ESCRIBE LIMPIAR BANDERAS
        STAA    ATDCTL5         ; 4 CONVERSIONES EN UNA SECUENCIA DE
                                ; CONVERSION SENCILLA
WTCONV: BRCLR   ATDSTATH,#$80,  WTCONV  ; ESPERA A LA BANDERA DE SECUENCIA
                                        ; COMPLETADA
```

```
        LDAA    ADR3H           ; SE CARGA EN A EL RESULTADO DE LA
                                ; CONVERSIÓN
        STAA    CH7
        RTS

; SUBRUTINA DELAY 100 uS
; RETRASO REQUERIDO PARA QUE EL ATD SE ESTABILICE (100 uS)

        LDAA    #$C8    ; CARGA EL ACUMULADOR CON "100 uS, VALOR DEL RETRASO
                        ; (DELAY)"
DELAY:  DECA            ; DECREMENTA EL ACUMULADOR
        BNE     DELAY   ; BRANCH SI NO ES IGUAL A ZERO
        RTS
        BRA     MAIN    ; BRINCO A INICIO DEL PROGRAMA (MAIN)

        ORG     $0950   ; DIRECCIÓN DE INICIO DE DECLARCIÓN DE
                        ; VARIABLES

CH7     DS.B    1       ; SE RESERVA UNA LOCALIDAD DE 1 BYTE PARA LA
                        ; VARIABLE CH7
```

Ejemplo 12.- Comunicación por Puerto Serie

La interfaz de comunicación serial es muy sencilla de utilizar, y puede establecerse entre la hiperterminal y el microcontrolador para ejercicios básicos. Pero para establecerla es necesario indicar la velocidad (baudios), y establecer protocolos de comunicación muy sencillos.

Básicamente estos protocolos verifican que la transmisión y recepción se efectué correctamente, lo cual se logra monitoreando los registros que indican si algún dato a llegado, o si se puede transmitir un dato o se esta en proceso de envió de alguno.

El programa a continuación esta elaborado de tal forma que el microcontrolador esta a la espera de recibir un dato de tipo “ASCCI”, el cual es comparado con el carácter A y de ser igual, se procede al envió de una cadena de caracteres. De llegar un carácter no esperado, la comparación falla y se continúa con la rutina de espera del dato deseado.

Este circuito no requiere de ninguna interfaz adicional, basta con observar la ventana de terminal del programa MiniIde.

```
        .nolist
#include hc12.inc
        .list

; INICIO DEL PROGRAMA

        ORG RAM_START

; CONFIGURACIÓN DE REGISTROS DEL SCI
```

```
        LDD     #$0034  ; SE CONFIGURA EL PUERTO SCI
        STD     SC0BDH  ; PARA TRABAJAR A 9600 BAUDIOS
        CLR     SC0CR1  ; SE HABILITA LA TRANSMISIÓN Y RECEPCIÓN
        LDAA    #$0C    ; SE HABILITA EL CIRCUITO DE RECEPCIÓN
        STAA    SC0CR2  ; Y EL CIRCUITO DE TRANSMISIÓN


;     CICLO DE CONTROL PRINCIPAL

INICIO  LDX     #HELLO  ; SE INICIALIZA EL REGISTRO INDICE
                        ; CON LA DIRECCIÓN DE LA CADENA DE CARACTERES

RECEP   LDAB    SC0SR1  ; SE ALBACENA EN B EL RESULTADO DEL REGISTRO
        BITB    #$20    ; PARA PROBAR SI ESTA ACTIVO EL BIT RDRF, SI ES
                        ; ASI SE LEE EL REGISTRO DE DATOS
                        ; SI NO ES ASI SE REALIZA LA COMPARACIÓN
        BEQ     RECEP   ; INDEFINIDAMENTE
        LDAB    SC0DRL  ; SE LEE EL REGISTRO DE DATOS
        CMPB    #$41    ; Y SE LE COMPARA CON EL VALOR ESPERADO
        BNE     RECEP   ; SI ES DIFERENTE SE REGRESA A INICIO
BYTE    LDAA    0, X    ; SE CARGA EN A LA DIRECCIÓN DE LA CADENA DE
                        : CARACTERES
TRANSM  LDAB    SC0SR1  ; SE OBTIENE EL VALOR DE LAS BANDERAS DE ESTADO
        BITB    #$80    ; SE VERIFICA SI SE ACTIVO LA BANDERA DE FIN DE
                        ; TRANSMISIÓN
        BEQ     TRANSM  ; DE NO SER ASI SE CONTINUA CON EL ENVIO
        STAA    SC0DRL  ; SE ALMACENA EN A EL CONTENIDO DEL REGISTRO DE
                        : DATOS
        INX             ; SE INCREMENTA EL REGISTRO INDICE X
        TSTA            ; SE PRUEBA SI EL CONTENIDO DEL REGISTRO DE
                        ; DATOS ES 0
        BNE     BYTE    ; SI NO ES ASI, REGRESA A CARGAR EL SIGUIENTE
                        ; CARACTER
        BRA     INICIO  ; SALTO INCONDICIONAL A INICIO


;       DECLARACIÓN DE VARIABLES

HELLO:      DC.B  "HELLO WORLD",$0D,$0A,$0
```

Ejemplo 13.- Comunicación por Puerto SPI

La interfaz de comunicación serial es utilizada comúnmente para enviar datos a dispositivos externos que trabajen a la misma velocidad que el puerto serie, los cuales también pueden operar como maestros o esclavos.

En este ejemplo se le utiliza para enviar un dato al registro de corrimiento 74HC595, el cual tiene una entrada de reloj de sincronía, entrada serial y salida paralela de 8 bits. El circuito recomendado es el circuito 3 del apéndice A.

```
; DECLARACION DE ETIQUETAS
SP0CR1  EQU     $00D0   ; REGISTRO DE CONTROL 1 DEL SPI
SP0CR2  EQU     $00D1   ; REGISTRO DE CONTROL 2 DEL SPI
SP0BR   EQU     $00D2   ; REGISTRO DE RANGO DE BAUDIOS DEL SPI
SP0SR   EQU     $00D3   ; REGISTRO DE ESTADO DEL SPI
SP0DR   EQU     $00D5   ; REGISTRO DE DATOS DEL SPI
PORTS   EQU     $00D6   ; REGISTRO DE DATOS DEL PUERTO S
DDRS    EQU     $00D7   ; REGISTRO DE DIRECCIÓN DEL PUERTO S

;********************
; INICIO DEL PROGRAMA
;********************

        ORG     $0800   ; ORIGEN DEL VECTOR DE RESSET

; LOOP DE CONTROL

        BSET    PORTS, #$80     ; SE ACTIVA A NIVEL ALTO LA LINEA SS
        MOVB    #$19, SP0CR1    ; CONFIGURA SPI: MSTR=1,CPOL=1,LSBF=1
        CLR     SP0CR2          ; PUPS=0,RDS=0,SPC0=0.
        MOVB    #$02, SP0BR     ; SCK=1, ECLOCK/8
        MOVB    #$E0, DDRS      ; CONFIGURA PUERTO S; MOSI, SCK, SS =
                                ; SALIDAS, MISO=ENTRADA

        LDAA    SP0SR           ; 1ER PASO LIMPIAR LA BANDERA SPIF,
                                ; LEER SP0SR
        LDAA    SP0DR           ; 2DO PASO LIMPIAR LA BANDERA SPIF,
                                ; ACCESAR A SP0DR
        BSET    SP0CR1,#$40     ; PS4-PS7 SE HABILITAN PARA FUNCIONAR
                                ; CON EL  SPI
OTROX   LDAA    #$0F            ;
```

```
        STAA    SP0DR           ; SE ALMACENA EL DATO EN EL REGISTRO DE
                                ; DATOS
NOTX    BRCLR   SP0SR,#$80,NOTX ; SE ESPERA LA ACTIVACIÓN DE LA BANDERA
                                ; SPIF
        BRA     OTROX           ; REGRESA A ENVIAR EL PROXIMO DATO
```

Ejemplo 14.- Calculo de Frecuencia

El modulo del timmer se utiliza en este ejemplo para capturar flancos de subida y medir la frecuencia con que se presentan en un intervalo de tiempo (Hertz). Pero para poder realizar esto el timmer captura y cuenta el número de flancos de subida que ocurren durante un periodo de tiempo especificado, para este caso concreto se miden los flancos en intervalos de tiempo de 250 Khz y pueden detectarse frecuencias comprendidas entre los rangos de 1000 a 10000 Hertz.

El canal de captura utilizado es el canal 0, y para mostrar el resultado del calculo de la frecuencia se utiliza una serie de displays de 7 segmentos para los cual fue necesario utilizar una subrutina que convirtiera el resultado de la frecuencia de hexadecimal a código BCD. El diagrama del circuito utilizado es el circuito 2 del apéndice A.

```
        .nolist
#include hc12.inc
        .list
        org EE_START

;*****************************************
; SUBRUTINA DE INICIALIZACION DE REGISTROS  *
;*****************************************

                CLR     TIOS    ; SE CONFIGURAN TODOS LOS CANALES DEL
                                ; TIMER COMO ENTRADA
                LDAA    #$05    ; SE CONFIGURA EL TIMMER EN EL FLANCO
                STAA    TCTL4   ; DE SUBIDA
                LDAA    #$80    ; SE HABILITA EL TIMMER
                STAA    TSCR    ; PARA FUNCIONAR NORMALMENTE
                CLR     TMSK1   ; SE DESHABILITAN LAS INTERRUPCIONES
                                ; POR HARDWARE
                LDAA    #$05    ; SE SELECCIONA EL DIVISOR A 32
                STAA    TMSK2   ; PARA TRABAJAR A 250 KHZ.

;*************************************
; SUBRUTINA DE CALCULO DE FRECUENCIA   *
;*************************************

FRECUENCIA      BRCLR   TFLG1, $01, FRECUENCIA  ; SE ESPERA LA BANDERA
                                ; DE INTERRUPCION
                LDD     TC0H    ; SE CARGA EN D LOS PULSOS QUE
                                ; CONTIENE EL REGISTRO DE CAPTURA
                STD     N2      ; SE ENVIA EL NUMERO DE PULSOS A N2
```

```
                SUBD    N1      ; SE LE RESTA EL NUMERO DE PULSOS DE LA
                                ; SECUENCIA ANTERIOR
                STD     N       ; SE ENVIA EL RESULTADO A N
                MOVW    N2, N1  ; ENVIA N2 A N1 PARA INICIALIZARLA EN
                                ; LA SIGUIENTE SECUENCIA
                LDY     #$03    ; CARGA UN 250000
                LDD     #$D090  ;100
                LDX     N       ; CARGA N COMO DIVISOR
                EDIV            ; EJECUTA LA DIVISION
                STY     VAR3    ; ENVIA EL RESULTADO HACIA VAR3,A LA
                                ; QUE SE ACCEDE PARA MOSTRAR EN DISPLAY

                LDAA    #$01    ; CARGA UN #01 EN A
                STAA    TFLG1   ; BORRA BANDERA DE CONTEO DE PULSOS
                JSR     DISPLAYFREC     ; BRINCO A SUBRUTINA DE
                                        ; CONVERSION A CODIGO BCD
                BRA     FRECUENCIA      ; SALTO AL INICIO DEL PROGRAMA
                                        ; PRINCIPAL
;*****************************************************

DISPLAYFREC     CLR     PORTA
                CLR     PORTB
                LDD     VAR3    ; CARGA EN A EL VALOR DE VAR3
                STD     DATOHEX ; ENVIA EL VALOR HACIA DATOHEX
                JSR     HEX_BCD ; BRINCA A CONVERTIR A BCD
                LDAA    $0952   ; CARGA EN A EL BIT MENOS SIGNIFICATIVO
                STAA    PORTB   ; Y LO ENVIA AL PUERTO B
                LDAA    $0951   ; CARGA EN A EL BIT MAS SIGNIFICATIVO
                STAA    PORTA   ; Y LO ENVIA AL PUERTO A
                RTS             ; RETORNO DE LA SUBRUTINA

;*********************************
; SUBRUTINA PARA CONVERTIR A BCD *
;*********************************

HEX_BCD:        MOVW    #$0,DATOBCD     ; INICIALIZA RESULTADO BCD
                clr     DATOBCD+2
                ldd     DATOHEX         ; LEE DATO HEXADECIMAL
                cpd     #$09            ; SI ES MENOR QUE 9 NO
                                        ; CONCIERTE
                bhi     CONVIERTE       ; SI ES MAYOR QUE 9 CONVIERTE
                stab    DATOBCD+2       ; ES MENOR QUE NUEVE, COLOCA EL
                                        ; RESULTADO
                bra     FIN_BCD         ; TERMINA

CONVIERTE:      ldx     #0AH            ; PREPARA DIVISION ENTRE BASE 10
                idiv                    ; PREPARA DIVISION
                stab    DATOBCD+2       ; PRIMER RESIDUO, COLOCA EN
                                        ; RESULTADO
                xgdx                    ; CAMBIA COCIENTE A D (A:B)
                cpd     #09H            ; SI ES MAYOR A 9 EFECTUA LA
                                        ; SEGUNDA DIVISIÓN
                bhi     DIV_2
                lslb                    ; COCIENTE ES MENOR QUE 9
                lslb                    ; PREPARA EL COCIENTE
                lslb
                lslb
                addb    DATOBCD+2       ; JUNTA DIGITOS DE PESO Y 1
                stab    DATOBCD+2
                bra     FIN_BCD         ; TERMINA

DIV_2:          ldx     #$0A            ; PREPARA DIVISIÓN ENTRE BASE 10
                idiv                    ; EJECUTA LA DIVISION
                lslb                    ; PREPARA EL SEGUNDO RESIDUO
                lslb
```

```
                lslb
                lslb
                addb    DATOBCD+2       ; JUNTA DIGITOS DE PESO 0 Y 1
                stab    DATOBCD+2
                xgdx                    ; CAMBIA COCIENTE A D
                cpd     #09H            ; SI ES MENOR A 9 EFECTUA LA
                                        ; TERCERA DIVISION
                bhi     DIV_3
                stab    DATOBCD+1       ; COCIENTE ES MENOR A 9,COLOCA
                                        ; DIGITO PESO 2
                bra     FIN_BCD         ; TERMINA

DIV_3:          ldx     #$0A            ; PREPARA TERCERA DIVISION
                idiv                    ; EJECUTA DIVISION
                stab    DATOBCD+1       ; COLOCA DIGITO PESO 2
                xgdx                    ; CAMBIA COCIENTE A D
                cpd     #09H            ; SI ES MAYOR A 9 EFECTUA
                                        ; CUARTA DIVISION
                bhi     DIV_4
                lslb                    ; COCIENTE ES MENOR A 2
                lslb                    ; PREPARA DIGITO PESO 3
                lslb
                lslb
                addb    DATOBCD+1       ; COMPLETA DIGITO DE PESO 2 Y 3
                stab    DATOBCD+1
                bra     FIN_BCD         ; TERMINA

DIV_4:          ldx     #$0a            ; PREPARA CUARTA DIVISION
                idiv                    ; EJECUTA LA DIVISION
                lslb                    ; PREPARA DIGITO DE PESO 3
                lslb
                lslb
                lslb
                addb    DATOBCD+1       ; COMPLETA DIGITO DE PSO 2 Y 3
                stab    DATOBCD+1
                xgdx                    ; CAMBIA ULTIMO COCIENTE A D
                stab    DATOBCD         ; COLOCA DIGITO DE PESO 4
FIN_BCD:        RTS
;*******************************************************************

                ORG     $950    ; DEFINE EL ORIGEN A PARTIR DE LA
                                ; LOCALIDAD $0950

DATOBCD:        DS.B    3       ; DEFINE LOCALIDAD DE MEMORIA DE 3 BYTE
DATOHEX:        DS.B    2       ; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE
VAR3            DS.B    2       ; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE
                                ; PARA LA VARIABLE VAR3
N2              DS.W    1       ; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE
                                ; PARA LA VARIABLE N2
N1              DS.W    1       ; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE
                                ; PARA LA VARIABLE N1
N                DS.W   1       ; DEFINE LOCALIDAD DE MEMORIA DE  ; 1
                                ; BYTE PARA LA VARIABLE N
```

6.3. - Practicas Para Capitulo 5

Ejemplo 15.- Switchs como elemento de Entrada de Datos

El siguiente programa nos es útil para conocer el manejo de switches como elementos de adquisición de datos. Básicamente realiza la función de escanear constantemente el puerto de entrada A para conocer el estado en que se encuentra cada bit del puerto y posteriormente se refleja el valor obtenido en el puerto B previamente declarado como salida.

Para poder realizar esta practica no se necesita mas que un switch conectado al puerto de entrada como el que se muestra en el circuito 4 del apéndice A y cualquier dispositivo que nos permita ver si alguno de los puertos de salida esta encendido como leds ó displays como se muestra en el circuito 1 y 2 del apéndice A. Se debe tener la precaución de conectar a tierra los puertos de entrada que no estén encendidos de lo contrario se correrá el riesgo de obtener una lectura errónea y por consiguiente un valor de salida erróneo.

```
; DECLARACIÓN DE ETIQUETAS
PORTA           EQU     $0000   ; PUERTO A REGISTRO
DDRA            EQU     $0002   ; PUERTO A DIRECCION DE REGISTRO
PORTB           EQU     $0001
DDRB            EQU     $0003

;*****************************
; INICIO DEL PROGRAMA       *
;*****************************
                ORG     $0800   ; VECTOR DE RESET
;_______________________________
;
; CONFIGURACION DE PUERTOS
;_______________________________

                LDAA    #$00    ; SE CONFIGURA EL PUERTO A
                STAA    $0002   ; COMO ENTRADAS
                LDAA    #$FF    ; SE CONFIGURA EL PUERTO B
                STAA    $0003   ; COMO SALIDAS
;_______________________________
;
; LOOP DE CONTROL
;_______________________________

INICIO          LDAA    $0000   ; SE LEE EL PUERTO A
                STAA    $0001   ; SE ENVIA AL PUERTO B
                BRA     INICIO  ; SE CICLA EL PROGRAMA  INDEFINIDAMENTE
```

EJEMPLO 16.- Contador Binario de 8 bits en Puerto B

Este programa varia el estado del Puerto B configurado como salida basándose en un contador ascendente. Es un poco más elaborado que el anterior ya que este utiliza subrutinas de retardo de tiempo para variar el cambio de estado de cada bit del puerto y a

la vez hace uso de una variable auxiliar que almacena la cuenta que se va realizando dentro del Puerto.

Para poder visualizar el estado del contador se recomienda utilizar el circuito 1 del apéndice A. Para poder disminuir la velocidad del contador solamente hay que incrementar el valor del retardo.

```
; DECLARACIÓN DE ETIQUETAS
PORTA   EQU     $0000   ; PUERTO A REGISTRO
DDRA    EQU     $0002   ; PUERTO A DIRECCION DE REGISTRO
PORTB   EQU     $0001
DDRB    EQU     $0003

;*****************************
; INICIO DEL PROGRAMA       *
;*****************************
        ORG     $0800   ; VECTOR DE RESET

        LDAA    #$FF    ; SE CONFIGURA EL PUERTO B
        STAA    $0003   ; COMO SALIDAS

; LOOP DE CONTROL

        CLR     MEM1    ; SE BORRA EL CONTENIDO DE LA MEMORIA
INICIO  LDAA    MEM1    ; SE CARGA EN EL A EL CONTENIDO DE LA MEMORIA
        STAA    $0001   ; SE ENVIA AL PUERTO B
        ADDA    #$01    ; SE LE AGREGA UN #$01 AL ACUMULADOR A
        STAA    MEM1    ; SE ENVIA EL NUEVO VALOR DE A+1 A LA MEMORIA
        JSR     RET1    ; SALTO A SUBRUTINA DE RETARDO
        BRA     INICIO  ; SE CICLA EL PROGRAMA INDEFINIDAMENTE

; SUBRUTINA DE RETARDO

RET1    LDAB    #$02    ; SE ALMACENA UN #$02 EN EL ACUMULADOR B
DECR1   LDY     #$FFFF  ; SE CARGA UN VALOR GRANDE EN EL REGISTRO Y
DECR    DEY             ; SE DECREMENTA EN 1 EL CONTENIDO DE Y
        BNE     DECR    ; SE BRINCA SI NO ES IGUAL A 0
        DECB            ; SE DECREMENTA EL CONTENIDO DE B
        BNE     DECR1   ; BRINCA SI NO A LLEGADO A 0
        RTS     ; RETORNO DE SUBRUTINA

; TABLA DE DECLARACIÓN DE VARIABLES

         ORG    $950    ; ORIGEN DE VARIABLES
MEM1     DS.B   1       ; DEFINE LOCALIDAD DE 1 BYTE
```

EJEMPLO 17.- Manejo del Teclado Matricial

Los teclados matriciales son muy útiles si se quieren ingresar datos al microcontrolador, además de que su diseño nos permite conectar matrices de teclas de hasta 16 dígitos en un solo puerto de 8 bits.

A continuación se elaboro un programa para el funcionamiento del teclado, el cual básicamente se enfoca en desplegar en el puerto A, el valor asignado a la tecla oprimida.

Para poder realizar esta labor, se conecta el teclado a un puerto de 8 bits configurando la parte alta como salida y la parte baja como entrada, para este ejemplo el puerto T. Posteriormente se envía constantemente un 0 lógico a la parte de salida, y la parte de entrada al estar conectada permanentemente a Vcc, detectara un 1 lógico si no a sido oprimida ninguna tecla.

Pero si detecta que existe un cambio de nivel a 0 lógico en alguna terminal de los bits de entrada, se realizara una subrutina de servicio de interrupción la cual ejecutara un desplazamiento a la izquierda donde se intentara detectar la columna donde esta la tecla oprimida ingresando un 0 lógico al bit 7 con lo que sabremos la posición de la tecla al llegar este 0 al acarreo. El siguiente paso consiste en obtener el código de la tecla equivalente al de la tecla presionada a través de una tabla de datos para después finalmente desplegar el valor de la tecla mediante el circuito 2 del Apéndice A.

El circuito del teclado es el circuito 5 del apéndice A.

```
; DECLARACION DE ETIQUETAS
#include hc12.inc
        .list

;*******************
; INICIO DEL PROGRAMA*
;*******************
        org RAM_START

; LOOP DE CONTROL

TECLADO:        LDAA    #$FF    ; CONFIGURA COMO SALIDAS
                STAA    DDRA    ; AL PUERTO A
                CLR     PORTA   ; LIMPIA EL PUERTO A
                LDAA    #$F0    ; INICIALIZA PUERTO T
                STAA    DDRT    ; LA PARTE ALTA COMO SALIDA
                                ; Y LA BAJA COMO ENTRADAS
                CLR     PORTT   ; LIMPIA EL PUERTO T
                cli             ; SE HABILITAN LAS INTERRUPCIONES
CICLO           BRA     CICLO   ; SE CICLA INDEFINIDAMENTE
;***********************************
;PROGRAMA DE ATENCION A INTERRUPCION*
;***********************************

IRQ:            JSR     DELAY25 ; RETARDO PARA ELIMINAR REBOTE
                LDAA    PORTT   ; LEE FILAS
                                ; LEYO CUALQUIERA DE ESTOS
                                  DATOS:07,0D,0B,0E

; SUBRUTINA PARA DETECTAR LA TECLA OPRIMIDA

                CLRB            ; BORRA EL CONTENIDO DEL PUERTO B
LEE_TECLA:      LSRA            ; ENVIA BIT 0 AL BIT C (ACARREO)
                BCC     LEE_COLUMNA     ; SI C=O, YA SE ENCONTRO
COLUMNA
                INCB                    ; SI C=1, INCTREMENTA B
                BRA     LEE_TECLA       ; LEE LA SIGUIENTE
```

```
LEE_COLUMNA:    LDAA    #$F0    ; PREPARA CORRIMIENTO DE UN CERO LOGICO
                                  EN COLUMNAS
                STAA    PORTT           ;
                CLC             ; BORRA ACARREO, EN LA SIGUIENTE
                                  INSTRUCCION
COLUMNA:        ROR     PORTT   ; LO INTRODUCE POR EL PB0, SE
                                  SELECCIONA COLUMNA 0
                LDAA    PORTT   ; LEE FILAS
                ANDA    #0FH    ; SOLO INTERESAN PB0-PB3
                CMPA    #0FH    ; ALGUNA FILA EN CERO?
                BNE     LEE_COD ; SI NO LEYO F0, ESTA ES LA COLUMNA
                ADDB    #04     ; SI NO ES ESTA COLUMNA, PUEDE SER
                                ; CUALQUIERA DE LAS
                                ; CUATRO TECLAS DE LA SIGUIENTE COLUMNA
                SEC             ; ACARREO=1, EL CERO YA ESTA DENTRO DEL
                                  PUERTO B
                BRA     COLUMNA ; VE A SELECCIONAR LA SIGUIENTE COLUMNA


; SUBRUTINA PARA LEER CODIGO DE TECLA OPRIMIDA

LEE_COD:        LDX     #T_TECLADO; PREPARA APUNTADOR DE TABLA DE
                                    CODIGOS
                ABX             ; EL RESULTADO DE LA SUMA APUNTA AL
                                  CODIGO
                LDAB    0,X     ; LEE CODIGO
                STAB    PORTA   ; ESCRIBE B EN PUERTO A

;LAZO PARA ESPERAR A QUE SE SUELTE LA TECLA

TEC_SUELTA:     LDAA    PORTT           ; LEE FILAS
                ANDA    #0FH            ; SOLO INTERESAN PB0-PB3
                CMPA    #0FH            ; SE SOLTO LA TECLA?
                BNE     TEC_SUELTA      ; SI NO LEYO F0, SIGUE TECLA
                                          OPRIMIDA
                JSR     DELAY25         ; RETARDO PARA ELIMINAR REBOTE
                CLR     PORTT           ; LIMPIA AL PUERTO T
                RTI

;RUTINA DE RETARDO DE 25 MS.
;1 CICLO MAQUINA= .125 USEG......8 CICLOS=1 USEG
;25 MSEG= 8X1000x25= 200,000 CICLOS
DELAY25:                LDX     #50000     ;200000 CICLOS/4
TOLP:                   DEX                    ;1
                        BNE     TOLP           ;3
                        RTS


;TABLA CON DATOS DE TECLADO
T_TECLADO:              DC.B    $00     ;
                        DC.B    $01
                        DC.B    $02
                        DC.B    $03
                        DC.B    $04
                        DC.B    $05
                        DC.B    $06
                        DC.B    $07
                        DC.B    $08
                        DC.B    $09
                        DC.B    $10
                        DC.B    $11
                        DC.B    $12
                        DC.B    $13
                        DC.B    $14
```

```
                        DC.B    $15

;TABLA DE VECTORES DE INTERRUPCION

                        ORG     $0B32
                        DC.W    IRQ
```

EJEMPLO 18.- Contador Decimal mediante Displays

Existen diferentes tipos de displays que pueden ser utilizados para desplegar datos ya sea numéricos o alfanuméricos. Uno de los displays más sencillos de utilizar para desplegar valores numéricos son los displays de 7 segmentos, de ánodo común o cátodo común.

Estos dispositivos utilizan el circuito integrado auxiliar 74LS47 o 74LS48 según sea el caso, el cual utiliza un formato de datos de entrada en código BCD, por lo que para poder mostrar algún dato proveniente del microcontrolador es necesario utilizar una programa para convertir de hexadecimal a código BCD, o si es posible utilizar la instrucción DAA la cual realiza un ajuste a decimal después de una operación de suma.

El siguiente programa realiza la función de un contador de 16 bits el cual maneja 2 memorias, una donde se localizan los 8 bits menos significativos MEM1, la cual se envía al puerto B. Y otra llamada MEM2 que contiene los 8 bits mas significativos, la cual se envía al puerto A.

Para desplegar los valores de los puertos se requiere de 2 pares de displays de 7 segmentos una para la parte alta de el contador y el otro para la baja por lo que se recomienda utilizar el circuito 2 del apéndice A.

```
;*******************
;CONTADOR 16 BITS  *
;*******************

; DECLARACIÓN DE ETIQUETAS


PORTA   EQU     $0000; DIRECCION DEL PUERTO A
DDRA    EQU     $0002; DIRECCION DEL REGISTRO DE DIRECCION DEL PUERTO A
PORTB   EQU     $0001; DIRECCION DEL PUERTO B
DDRB    EQU     $0003; DIRECCION DEL REGISTRO DE DIRECCION DEL PUERTO B
PORTT   EQU     $00AE; DIRECCION DEL PUERTO T
DDRT    EQU     $00AF; DIRECCION DEL REGISTRO DE DIRECCION DEL PUERTO T

; INICO DEL PROGRAMA
        ORG     $0800

;LOOP DE CONTROL PRINCIPAL

        LDAA    #$FF    ; CARGA #$FF EN EL ACUMULADOR A
        STAA    DDRA    ; ENVIA EL CONTENIDO DEL ACUMULADOR A HACIA EL
                          REGISTRO PARA CONFIGURAR PUERTO A COMO
                         SALIDAS.
        STAA    DDRB    ; ENVIA EL CONTENIDO DEL ACUMULADOR A HACIA EL
                         REGISTRO PARA CONFIGURAR PUERTO B COMO SALIDA.
        CLR     MEM1    ; BORRA EL CONTENIDO DE MEM1
        CLR     MEM2    ; BORRA EL CONTENIDO DE MEM2
CICLO   JSR     DISPLAY ; SALTO A LA SUBRUTINA DISPLAY
        JSR     RET1    ; SALTO A LA SUBRUTINA RET1
        LDAA    MEM1    ; CARGA EL CONTENIDO DE MEM1 EN ACULULADOR A.
        ADDA    #$01    ; LE SUMA UN 1 AL ACUMULADOR A
        DAA             ; REALIZA EL AJUSE A DECIMAL
        STAA    MEM1    ; ENVIA EL RESULTADO DEL AJUSTE A LA MEMORIA
                         MEM1
        LDAA    MEM2    ; CARGA EL CONTENIDO DE MEM2 EN ACUMULADOR A
        ADCA    #$00    ; SUMA EL ACARREO Y "0" AL ACUMULADOR A.
        DAA             ; AJUSTA A DECIMAL EL CONTENIDO DEL ACUMULADOR
        STAA    MEM2    ; SE ENVIA EL RESULTADO DEL AJUSTE A LA MEMORIA
                         MEM2
        BRA     CICLO   ; BRINCO INCONDICIONAL HACIA CICLO

; SUBRUTINA PARA MANEJO DEL DISPLAY

DISPLAY LDAA    MEM1    ; CARGA EN ACUMULADOR A EL VALOR DE MEM1
        STAA    PORTB   ; ENVIA EL CONTENIDO DEL ACUMULADOR AL PUERTO B
        LDAA    MEM2    ; CARGA EN ACUMULADOR A EL CONTENIDO DE MEM2
        STAA    PORTA   ; ENVIA EL CONTENIDO DEL ACUMULADOR AL PUERTO A
        RTS             ; RETORNO DE SUBRUTINA

; SUBRUTINA DE RETARDO DE TIEMPO

RET1    LDAB    #$1     ; ESTE ES EL MISMO RETARDO CALCULADO EN TODOS
                         LOS PROGRAMAS
DECR1   LDY     #$2211
DECR    DEY
        BNE     DECR
```

```
        DECB
        BNE     DECR1
        RTS

; DECLARACION DE VARIABLES EN MEMORIA

        ORG     $950
MEM1    DS.B    1       ; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE
MEM2    DS.B    1       ; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE
```

EJEMPLO 19.- Convertidor de Digital a Analógico con puerto B usando una Función Rampa

El siguiente programa esta basado en un contador de 8 bits, el cual debido a su funcionamiento ira incrementando el puerto B de salida en una posición hasta reiniciarse. Esto pretende crear una función rampa a través de un convertidor de digital a análogo ya que al incrementar en una localidad el contador, este se convertirá en un salto en el convertidor con un valor de .019 mV.

El programa esta diseñado para utilizar el puerto B como salida del contador, y puede variarse el tiempo de la función rampa incrementando o reduciendo el tiempo del retardo de la subrutina RET1. El convertidor se muestra en el circuito 6 del apéndice A.

```
; DECLARACIÓN DE ETIQUETAS
PORTB   EQU     $0001   ; DIRECCION DEL PUERTO B
DDRB    EQU     $0003   ; REGISTRO DE DIRECCION DEL PUERTO B

; INICO DEL PROGRAMA
        ORG     $0800

; LOOP DE CONTROL PRINCIPAL
        LDAA    #$FF    ; CARGA #$FF EN EL ACUMULADOR A
        STAA    DDRB    ; ENVIA EL CONTENIDO DEL ACUMULADOR A HACIA EL
                          REGISTRO PARA CONFIGURAR PUERTO B COMO
                          SALIDAS.
        CLR     MEM1    ; BORRA EL CONTENIDO DE MEM1
CICLO   JSR     DISPLAY ; SALTO A LA SUBRUTINA DISPLAY
        JSR     RET1    ; SALTO A LA SUBRUTINA RET1
        LDAA    MEM1    ; CARGA EL CONTENIDO DE MEM1 EN ACULULADOR A.
```

```
        ADDA    #$01    ; LE SUMA UN 1 AL ACUMULADOR A
        STAA    MEM1    ; ENVIA EL RESULTADO DEL AJUSTE A LA MEMORIA
                          MEM1
        BRA     CICLO   ; BRINCO INCONDICIONAL HACIA CICLO

; SUBRUTINA PARA MANEJO DEL DISPLAY
DISPLAY LDAA    MEM1    ; CARGA EN ACUMULADOR A EL VALOR DE MEM1
        STAA    PORTB   ; ENVIA EL CONTENIDO DEL ACUMULADOR AL PUERTO B
; SUBRUTINA DE RETARDO DE TIEMPO
RET1    LDAB    #$1     ; ESTE ES EL MISMO RETARDO CALCULADO EN TODOS
                          LOS PROGRAMAS
DECR1   LDY     #$2211
DECR    DEY
        BNE     DECR
        DECB
        BNE     DECR1
        RTS
; DECLARACION DE VARIABLES EN MEMORIA
        ORG     $950
MEM1    DS.B    1       ; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE
```

CAPITULO 7

APLICACIONES PARA CONTROL E INSTRUMENTACIÓN

El microcontrolador por su diseño orientado hacia el ambiente de control es capaz de cubrir cualquier aplicación en estas áreas que este dentro de sus características de operación como serian su velocidad, tipos de cálculos, temperaturas de operación etc.

Al contar con una variedad de periféricos que cubre diferentes funciones, nosotros encontramos que podemos calcular frecuencias mediante el modulo Timmer y a la vez podemos convertirla a voltaje mediante un convertidor tipo DAC por ejemplo. Ó también quizás podríamos detectar velocidad mediante el convertidor ADC y aplicar algún tipo de algoritmo de control como ON- OFF a un motor.

Y así al estudiar los diferentes periféricos con los que cuenta este dispositivo se nos abrirá una amplia gama de aplicaciones que podemos desarrollar dentro de la industria, así como dentro de algunas otras áreas de la ingeniería mismas que contaran con la robustez con la que fue diseñado este dispositivo.

7.1.- Diseño de un Proyecto Multitareas

Esta aplicación esta dividida en 3 partes básicas cada una de las cuales esta cubierta por un microcontrolador las tareas de cada microcontrolador se explican a continuación:

Modulo Receptor

1. Este modulo efectuara un control tipo ON-OFF sobre dos variables analógicas, muestreadas a través de los canales 6 y 7 del convertidor de analógico a digital.
2. Encenderá un alarma visual por cada canal que indicara cuando la variable del canal correspondiente este por debajo o arriba del rango establecido.
3. Obtendrá el rango de operación de las variables analógicas a través del puerto serie y lo utilizara para el control ON-OFF sobre las variables.
4. Muestreara una señal a través del modulo de captura del Timmer y calculara su frecuencia.
5. Desplegara los valores reales de frecuencia y de las variables analógicas controladas a través de un modulo de displays conectados 2 puertos paralelos de salida
6. Obtendrá datos de entrada a través de un puerto paralelo, los cuales indicaran que datos se desplegaran en el display.

Modulo Transmisor

1. Leerá los datos provenientes de un teclado matricial conectado a uno de sus puertos paralelos y los almacenara en varias variables.
2. Identificara los códigos presionados que le indicaran que variable enviar a través del puerto serie.

Modulo Generador de Frecuencia

1. Leerá el valor de 8 bits de un canal analógico a digital y lo utilizara para variar la frecuencia de una señal modulada por ancho de pulso con 8 bits de resolución generada por el modulo PWM a través del canal 0, la cual oscilara entre 1 y 10 Khz.

Desarrollo

A continuación se explica el funcionamiento detallado de cada modulo y la manera en que quedo configurado para cumplir las tareas asignadas.

Modulo Transmisor:

Es un microcontrolador programado en modo autónomo, el cual tiene la tarea de decodificar el teclado para capturar los valores y códigos de los setpoints y posteriormente enviarlos por el puerto serie.

Cuenta con un circuito auxiliar que despliega el valor de las variables oprimidas en el teclado, lo cual sirve para cerciorarnos del valor actual en el teclado, y cuenta con una tecla “F” la cual activa al ser presionada el envió del dato por el puerto serie, que a su vez es mostrado en el byte menos significativo del display.

El teclado esta conectado al puerto T y el display a los puertos A y B, el teclado se enlaza a la vez con el receptor mediante el puerto serie.

A continuación se observa el código de programa empleado para este modulo.

```
;*******************************
; TRANSMISOR MEDIANTE TECLADO   *
;*******************************
        .nolist
#include hc12.inc
        .list
```

```
          org EE_START   ;DIRECCIÓN DE INICIO DE LA EEPROM


         CLR     COPCTL
         LDS     #$0C00
INICIO   LDAA    #$FF    ; CONFIGURA LOS PUERTOS
         STAA    DDRB    ; A Y B COMO SALIDAS
         STAA    DDRA    ;
         CLR     PORTA   ; LIMPIA EL PUERTO A
         CLR     PORTB   ; LIMPIA EL PUERTO B
         LDD     #$0034  ; CONFIGURA EL PUERTO SERIE
         STD     SC0BDH  ; A 9600 BAUDIOS
         CLR     SC0CR1  ; BORRA EL REGISTRO
         LDAA    #$0C    ; HABILITA LA TRANSMICION
         STAA    SC0CR2  ; Y RECEPCION
         LDAA    #$F0    ; CONFIGURA LA PARTE ALTA DEL
                                 PUERTO
         STAA    DDRT    ; COMO SALIDAS Y LA BAJA COMO
                           ENTRADAS
         CLR     PORTT   ; LIMPIA EL PUERTO
         CLR     TECPEN  ; LIMPIA LA VARIABLE

         CLR     CODTEC  ; LIMPIA LA VARIABLE
         CLR     MEM1    ; LIMPIA LA VARIABLE
         CLR     MEM2    ; LIMPIA LA VARIABLE
         CLI             ; HABILITA INTERRUPCIONES

TECLA:   TST     TECPEN  ; PRUEBA SI EXISTE UN VALOR MAYOR A CERO
         BNE     REVISA  ; SI ES DIFERENTE BRINCA A REVISA
         BRA     TECLA   ; BRINCO INCONDICIONAL A TECLA

REVISA:          CLR     TECPEN  ; BORRA LA VARIABLE
                 LDAB    CODTEC  ; CARGA EN B EL CODIGO DE LA
                                           TECLA
                 BPL     NUMERO  ; REVISA SI ES POSITIVO
                 CMPB    #$80    ; COMPARA CONTRA #$80
                 BNE     TECLA   ; SI NO ES IGUAL BRINCA A TECLA
                 JSR     ENVIAR  ; SI ES IGUAL BRINCA A LA
                                           SUBRUTINA ENVIAR
                 BRA     TECLA   ; BRINCO INCONDICIONAL A TECLA


NUMERO           JSR     CORRIMIENTO     ; SALTO A SUBRUTINA
                 JSR     DISPLAY         ; SALTO A SUBRUTINA
                 BRA     TECLA           ; BRINCO INCONDICIONAL A TECLA

CORRIMIENTO      LDAA    #$04
                 LSLB
                 LSLB
                 LSLB
                 LSLB
CORRIMIENTO1     LSLB
                 ROL     MEM1
                 ROL     MEM2
                 DECA
                 BNE     CORRIMIENTO1
                 RTS

DISPLAY          LDAA    MEM1    ; CARGA EN A EL CONTENIDO DE MEM1
                 STAA    PORTB   ; Y DESPUES LO ENVIA AL PUERTO B
                 LDAA    MEM2    ; CARGA EN A EL CONTENIDO DE MEM2
                 STAA    PORTA   ; Y DESPUES LO ENVIA AL PUERTO A
                 RTS             ; RETORNO DESDE SUBRUTINA
ENVIAR           LDAA    MEM1    ; CARGA EN A EL CONTENIDO DE MEM1
NOTX             LDAB    SC0SR1  ; PRUEBA SI SE ACTIVO LA BANDERA
```

```
                BITB    #$80    ; DE TRANSMISION LIBRE
                BEQ     NOTX    ; BRINCA SI NO SE A ACTIVADO LA
                                  BANDERA
                STAA    SC0DRL  ; ENVIA EL CONTENIDO DEL
                                  REGISTRO DE DATOS
                RTS             ; RETORNO DESDE SUBRUTINA


;*************************
; PROGRAMA DE INTERRUPCION*
;*************************

IRQ:            JSR     DELAY25         ; RETARDO PARA ELIMINAR REBOTE
                LDAA    PORTT           ; LEE FILAS
                                        ; LEYO CUALQUIERA DE ESTOS
                                          DATOS:07,0D,0B,0E

                CLRB
LEE_TECLA:      LSRA                    ; ENVIA BIT 0 AL BIT C(ACARREO)
                BCC     LEE_COLUMNA     ; SI C=O,YA SE ENCONTRO COLUMNA
                INCB                    ; SI C=1, INCTREMENTA B
                BRA     LEE_TECLA       ; LEE LA SIGUIENTE

LEE_COLUMNA:    LDAA    #$F0            ; PREPARA CORRIMIENTO DE UN
                                          CERO LOGICO EN COLUMNAS
                STAA    PORTT
                CLC     ; BORRA ACARREO, EN LA SIGUIENTE INSTRUCCION

COLUMNA:        ROR     PORTT   ; LO INTRODUCE POR EL PB0, SE
                                          SELECCIONA COLUMNA 0
                LDAA    PORTT   ; LEE FILAS
                ANDA    #0FH    ; SOLO INTERESAN PB0-PB3
                CMPA    #0FH    ; ALGUNA FILA EN CERO?
                BNE     LEE_COD ; SI NO LEYO F0, ESTA ES LA COLUMNA
                ADDB    #04     ; SI NO ES ESTA COLUMNA, PUEDE SER
                                  CUALQUIERA DE LAS
                                ; CUATRO TECLAS DE LA SIGUIENTE COLUMNA
                SEC             ; ACARREO=1, EL CERO YA ESTA DENTRO DEL
                                  PUERTO B
                BRA     COLUMNA ; VE A SELECCIONAR LA SIGUIENTE
                                  COLUMNA

LEE_COD:        LDX     #T_TECLADO; PREPARA APUNTADOR DE TABLA DE
                                    CODIGOS
                ABX             ; EL RESULTADO DE LA SUMA APUNTA AL
                                  CODIGO
                LDAB    0,X     ; LEE CODIGO
                STAB    CODTEC
                LDAA    #$01
                STAA    TECPEN

                                ;LAZO PARA ESPERAR A QUE SE SUELTE LA TECLA
TEC_SUELTA:     LDAA    PORTT           ; LEE FILAS
                ANDA    #0FH            ; SOLO INTERESAN PB0-PB3
                CMPA    #0FH            ; SE SOLTO LA TECLA?
                BNE     TEC_SUELTA      ; SI NO LEYO F0, SIGUE TECLA
                                          OPRIMIDA
                JSR     DELAY25         ; RETARDO PARA ELIMINAR REBOTE
                clr     portT
                RTI
```

```
;************************************************
; RUTINA DE RETARDO DE 25 MS.
; 1 CICLO MAQUINA= .125 USEG......8 CICLOS=1 USEG
; 25 MSEG= 8X1000x25= 200,000 CICLOS
;************************************************

DELAY25:                LDX      #50000  ; 200000 CICLOS/4
TOLP:                   DEX              ; 1
                        BNE      TOLP    ; 3
                        RTS

; TABLA CON DATOS DE TECLADO
T_TECLADO:              DC.B     $00
                        DC.B     $01
                        DC.B     $02
                        DC.B     $03
                        DC.B     $04
                        DC.B     $05
                        DC.B     $06
                        DC.B     $07
                        DC.B     $08
                        DC.B     $09
                        DC.B     $0A
                        DC.B     $0B
                        DC.B     $0C
                        DC.B     $0D
                        DC.B     $0E
                        DC.B     $80

                        ORG      $0B32   ; TABLA DE VECTORES

                        DC.W     IRQ     ; IRQ


                ORG     $950; DEFINE ORIGEN PARA VARIABLES
TECPEN          DS.B    1   ; RESERVA UN BYTE PARA LA VARIABLE TECPEN
PARO            DS.B    1   ; RESERVA UN BYTE PARA LA
                                VARIABLE PARO
CODTEC          DS.B    1   ; RESERVA UN BYTE PARA LA VARIABLE CODTEC
MEM2            DS.B    1   ; RESERVA UN BYTE PARA LA VARIABLE MEM2
MEM1            DS.B    1   ; RESERVA UN BYTE PARA LA
                                VARIABLE MEM1
```

Modulo Receptor:

Es un segundo microcontrolador programado en modo autónomo, el cual realiza las siguientes tareas mediante su programación en la secuencia indicada:

Primero.- Se ejecuta una subrutina de inicialización de los periféricos del microcontrolador utilizados en el programa como el timmer, puerto serie, puertos a y b etc. Así como las inicializaciones requeridas de algunas de las variables utilizadas.

Segundo.- Al ejecutarse un reset el programa principal ejecuta una rutina en la que monitorea las variables en las que se almacenan los setpoints, a estas variables les es colocado un valor a través de la rutina de servicio de interrupción del puerto serie, ya que al ocurrir una interrupción la subrutina del SCI obtiene el valor que a llegado para evaluarlo mediante una comparación con el código indicado para cada setpoints .Por esto al recibir el primer código, se ordena hacia donde va a dirigirse el segundo dato que ingrese por el puerto serie.

Una vez que la subrutina de interrupción ha evaluado los códigos de los setpoints y ha decidido en donde almacenarlos, hay una subrutina final que se encarga de verificar que los dos setpoints hayan sido asignados. Una vez que a ocurrido esto se ejecuta la continuación del programa principal, de lo contrario, si falta algún setpoint por asignar, la subrutina se cicla para monitorear la asignación de el total de setpoints que son 2.

Cabe mencionar que esta subrutina solo se efectúa después de una interrupción de reset, por lo que para ingresar nuevos valores es necesario aplicar un reset al microcontrolador.

Tercero.- Una vez que se han inicializado los periféricos y obtenido los setpoints, es necesario calcular un margen de banda muerta para cada variable analógica, ya que son necesarios para efectuar la labor de monitoreo de los valores que activaran la alarma visual. La subrutina BANDA es la encargada de realizar esta labor. Esto lo realiza sumando un 10 al setpoint para definir su límite superior y restando un 10 para definir el límite inferior, esto lo realiza con cada uno de los setpoints.

Una vez que ha realizado dicha operación se continúa con la ejecución del programa principal.

Cuarto.- A continuación, el microcontrolador ejecuta la subrutina que va a adquirir el valor actual de las señales analógicas monitoreadas, y una vez que obtenga el valor de

cada canal (canales 6 y 7) lo almacenara en las variables CH7 y CH6 las cuales serán utilizadas en el resto del programa.

Esta subrutina al finalizar el programa es el nuevo punto de partida del programa principal debido a que es necesario mantener siempre el valor mas actual de las variables analógicas, y a partir de esta subrutina se volverá a ejecutar el programa principal el cual al finalizar regresara a esta misma subrutina indefinidamente.

Quinto.- Situándonos dentro de la subrutina LOOP_CONTROL, cuando ya se calcularon los setpoints con sus respectivos limites superior e inferior, y se conoce el valor de cada canal analógico correspondiente a cada setpoint, continua con la ejecución del programa principal esta subrutina la cual se encargara de efectuar la labor de control sobre las variables analógicas monitoreadas. Esto lo realizara comparando el valor de cada canal análogo y comparándolo con sus límites superior e inferior, cuando el valor de un canal este dentro de los limites establecidos el led correspondiente (conectados en PORTDLC6 Y 5) conectado como alarma visual permanecerá apagado, pero, si el valor del canal se encuentra fuera de los limites permitidos su alarma visual se encenderá.

El mismo control se realiza con las dos variables, una vez que ha evaluado las variables y ejecutado el control correspondiente de cada una de ellas se continúa con la ejecución del programa principal.

Sexto.- Después de efectuar el control de las variables analógicas, se ejecuta la subrutina FRECUENCIA, la cual muestreara y calculara la frecuencia registrada en el canal 0 del timmer.

Posteriormente colocara este valor dentro de la variable VAR3 que será utilizada en las subrutinas siguientes, antes de salirse de la subrutina se inicializan las variables y registros requeridos para el siguiente ciclo.

Séptimo.- Finalmente la subrutina final SELECTOR consiste en monitorear el estado de los bits, 0, 1, 2 de entrada, para realizar la prueba de cada bit por separado. Los bits 0 y 1 corresponden a los canales 7 y 6 del convertidor y al ser seleccionados alguno de los dos, se ejecuta una subrutina correspondiente a estos en la cual se muestra el valor actual de cada canal.

El bit 2 sirve para seleccionar la frecuencia, si se activa se ejecuta la subrutina que muestra el valor de la frecuencia en el display.

El valor de los canales 7 y 6 se muestra en formato hexadecimal y el de la frecuencia en formato decimal, se opto por lo anterior para poder observar un formato de valor similar al que es enviado por el teclado y el correspondiente a la lectura de cada canal analógico.

Finalmente después de la primera corrida del programa al regresar de la última subrutina, el programa se vuelve a iniciar a partir de la subrutina convertidor como lo mencionamos anteriormente y así permanece ciclado de forma indefinida. Por lo anterior si queremos actualizar nuestros setpoints debemos hacerlo después de aplicar un reset al microcontrolador.

```
;***************************
;       RECEPTOR            *
;***************************

        .nolist
#include hc12.inc
        .list

        ;org RAM_START
        org EE_START

        CLR     COPCTL
        LDS     #$0C00


        JSR     INICIALIZACION  ; SE INICIALIZAN TODOS LOS RECURSOS
                                  UTILIZADOS DEL MICRO.
        CLI                     ; SE HABILITAN LAS INTERRUPCIONES.
        JSR     ESPERA          ; SE OBTIENEN LOS SETPOINTS DE LAS
                                  VARIABLES.
        JSR     BANDA           ; CALCULA LOS LIMITES PARA LOS
                                  SETPOINTS.
INICIO  JSR     CONVERTIDOR     ; OBTIENE EL VALOR DE LAS VARIABLES
                                  ANALOGICAS.
        JSR     LOOP_CONTROL    ; EJECUTA EL LOOP DE CONTROL DE LAS
                                  VARIABLES ANALOGICAS
        JSR     FRECUENCIA
        JSR     SELECTOR        ; SELECCIONA LA VARIABLE A MOSTRAR EN
                                  EL DISPLAY
        LBRA    INICIO


;*****************************************
; SUBRUTINA DE INICIALIZACION DE VARIABLES  *
;*****************************************

INICIALIZACION
        LDAA    #$FF    ; CARGA #$FF EN EL ACUMULADOR A
        STAA    DDRA    ; ENVIA EL CONTENIDO DEL ACUMULADOR A HACIA EL
                          REGISTRO
                        ; PARA CONFIGURAR PUERTO A COMO SALIDAS.
```

```
        STAA    DDRB    ; ENVIA EL CONTENIDO DEL ACUMULADOR A HACIA EL
                          REGISTRO
                        ; PARA CONFIGURAR PUERTO B COMO SALIDAS.
        LDAA    #$F0    ; SE CONFIGURA LA PARTE ALTA DEL PUERTO DLC
                          COMO SALIDAS
        STAA    DDRDLC  ; Y LA PARTE BAJA COMO ENTRADAS
        LDAA    #$02    ; SE INICIALIZA LA VARIABLE CUENTA CON EL VALOR
        STAA    CUENTA  ; DE #$02

        CLR     CH7     ; SE LIMPIAN LA VARIABLE CH7
        CLR     CH6     ; SE LIMPIA LA VARIABLE CH6
        CLR     V1      ; SE LIMPIA LA VARIABLE V1
        CLR     V2      ; SE LIMPIA LA VARIABLE V2
        CLR     SP1     ; SE LIMPIA LA VARIABLE SP1
        CLR     SP2     ; SE LIMPIA LA VARIABLE SP2


        CLR     TIOS    ; SE CONFIGURAN TODOS LOS CANALES DEL
                          TIMER COMO ENTRADA
        LDAA    #$05    ; SE CONFIGURA EL TIMMER EN EL FLANCO
        STAA    TCTL4   ; DE SUBIDA
        LDAA    #$80    ; SE HABILITA EL TIMMER
        STAA    TSCR    ; PARA FUNCIONAR NORMALMENTE
        CLR     TMSK1   ; SE DESHABILITAN LAS INTERRUPCIONES POR
                          HARDWARE
        LDAA    #$05    ; SE SELECCIONA EL DIVISOR A 32
        STAA    TMSK2   ; PARA TRABAJAR A 250 KHZ.

        LDD     #$0034  ; SE CONFIGURA EL PUERTO SERIE
        STD     SC0BDH  ; PARA TRABAJAR A 9600 BAUDIOS
        CLR     SC0CR1  ; SE CONFIGURA EL PUERTO SERIE PARA FUNCIONAR
                          NORMALMENTE
        LDAA    #$2C    ; SE ACTIVA LA INTERRUPCION DEL RECEPTOR Y SE
                          CONFIGURAN
        STAA    SC0CR2  ; LOS PINES DE RECEPCION Y TRANSMISIÓN
        RTS             ; RETORNO DE LA SUBRUTINA

;*****************************************
;     SUBRUTINA DE ASIGNACION DE SETPOINTS  *
;*****************************************

ESPERA  LDAA    V1        ; ALMACENA EL VALOR DE V1 EN EL ACUMULADOR A
        CMPA    #$01      ; COMPARA V1 CON EL CODIGO DE SETPOINT 1
        BEQ     SETPOINT1; BRINCA SI EL CODIGO Y EL VALOR DE V1 SON
                          IGUALES HACIA
                        ; LA SUBRUTINA INDICADA
                        ; SI SON DIFERENTES SE EJECUTA LA SIGUIENTE
                          INSTRUCCION
        CMPA    #$02      ; COMPARA CON EL CODIGO DE SETPOINT 2
        BEQ     SETPOINT2; BRINCA SI EL CODIGO Y EL VALOR DE V1 SON
                          IGUALES HACIA
                        ; LA SUBRUTINA INDICADA
                        ; SI NO SON IGUALES SE EJECUTA LA SIGUIENTE
                          INSTRUCCION
        STAB    PORTA     ; ENVIA EL ACUMULADOR B HACIA EL PUERTO A
        STAB    PORTB     ; ENVIA EL ACUMULADOR B HACIA EL PUERTO B
        BRA     ESPERA    ; BRINCO INCONDICIONAL HACIA LA SUBRUTINA
                          ESPERA

;*********************************************************************
; ESTA SUBRUTINA ALMACENA EL VALOR DE V2 EN EL SETPOINT ASIGNADO POR  *
;EL CODIGO DE LA VARIABLE V1                                          *
**********************************************************************

SETPOINT1       LDAA    V2    ; CARGA EL VALOR DE V1 EN EL ACUMULADOR A
```

```
                STAA    SP1   ; Y LO ENVIA HACIA LA VARIABLE SP1
                BRA     FULL  ; BRINCO HACIA LA SUBRUTINA FULL
SETPOINT2       LDAA    V2    ; CARGA EL VALOR DE V1 EN EL ACUMULADOR A
                STAA    SP2   ; Y LO ENVIA HACIA LA VARIABLE SP2
                BRA     FULL  ; BRINCO HACIA LA SUBRUTINA FULL

;*********************************************************************
; LA SIGUIENTE SUBRUTINA VERIFICA  EL ESTADO DE LOS SETPOINTS         *
; UNA VEZ QUE HAN SIDO SELECCIONADOS PERMITE CONTINUAR AL PROGRAMA    *
;*********************************************************************

FULL            LDAA    SP1     ; CARGA EL SETPOINT 1 EN A
                CMPA    #$00    ; LO COMPARA CON 0
                BEQ     ESPERA  ; BRINCA SI LA COMPARACION FUE IGUAL
                                  HACIA LA SUBRUTINA ESPERA
                                ; PARA ESPERAR QUE SEA ASIGNADO UN
                                  VALOR AL SETPOINT 1
                LDAA    SP2     ; CARGA EL SETPOINT 2 EN A
                CMPA    #$00    ; LO COMPARA CON 0
                BEQ     ESPERA  ; BRINCA SI LA COMPARACION FUE IGUAL
                                  HACIA LA SUBRUTINA ESPERA
                                ; PARA ESPERAR QUE SEA ASIGNADO UN
                                  VALOR AL SETPOINT 2
                RTS             ;RETORNO DE LA SUBRUTINA

;*********************************************************
; ASIGNACION DE BANDAS SUPERIOR E INFERIOR DE LOS SETPOINTS*
;*********************************************************
BANDA:
                LDAA    SP1         ; CARGA A CON EL VALOR DE SP1
                ADDA    #$0A        ; LE SUMA UN #$0A AL CONTENIDO DE A
                STAA    TOP_UP_1    ; ENVIA EL RESULTADO DE LA SUMA
                                      HACIA EL LIMITE SUPERIOR DEL SP1
                LDAB    SP1         ; CARGA B CON EL VALOR DE SP1
                SUBB    #$0A        ; LE RESTA UN #$0A AL CONTENIDO DE B
                STAB    TOP_DOWN_1  ; ENVIA EL RESULTADO DE LA RESTA
                                      HACIA INFERIOR DE SP1

                LDAA    SP2         ; CARGA A CON EL VALOR DE SP2
                ADDA    #$0A        ; LE SUMA UN #$0A AL CONTENIDO DE A
                STAA    TOP_UP_2    ; ENVIA EL RESULTADO DE LA SUMA
                                      HACIA EL LIMITE SUPERIOR DEL SP1
                LDAB    SP2         ; CARGA B CON EL VALOR DE SP1
                SUBB    #$0A        ; LE RESTA UN #$0A AL CONTENIDO DE B
                STAB    TOP_DOWN_2  ; ENVIA EL RESULTADO DE LA RESTA
                                      HACIA EL LIMITE INFERIOR DE SP1
                RTS                 ; RETORNO DE LA SUBRUTINA

;*********************************************************
;SUBRUTINA DE CONFIGURACION E INICIALIZACIÓN DEL CONVERTIDOR *
;*********************************************************

CONVERTIDOR:
        LDAA    #$80    ; PERMITE AL CONVERTIDOR FUNCIONAR NORMALMENTE
        STAA    ATDCTL2 ; SE DESHABILITAN INTERRUPCIONES Y SE CONFIGURA
                          AL ADC PARA FUNCIONAR NORMALMENTE
        BSR     RETARDO ; SALTO A SUBRUTINA RETARDO, RETRASO DE 100 uS
        LDAA    #$00    ; SELECCIONA CONVERSION CONTINUA EN MODO BGND
        STAA    ATDCTL3 ; IGNORA FREEZE EN ADCTL3
        LDAA    #$01    ; SELECCIONA LA VELOCIDAD DEL ADC A 2 A/D
                        ; CLOCKS
        STAA    ATDCTL4 ; SELECCIONA EL DIVISOR POR 4


        LDAA    #$17    ; INICIALIZA ATD: SCAN=0,MULT=1,PAD6 Y PAD7,
```

```
                        LIMPIA LAS BANDERAS
        STAA    ATDCTL5 ; Y EJECUTA4 CONVERSIONES EN UNA SECUENCIA DE
                          CONVERSION SENCILLA
WTCONV: BRCLR   ATDSTATH,#$80,  WTCONV  ; ESPERA A LA BANDERA DE
                          SECUENCIA COMPLETADA
        LDAA    ADR3H   ; CARGA EL RESULTADO DE LA CONVERSIÓN DE ADR3H
                          EN A
        STAA    CH7     ; COLOCA EL RESULTADO EN A DENTRO DE LA
                          VARIABLE CH7
        LDAA    ADR2H   ; CARGA EL RESULTADO DE LA CONVERSIÓN
                          DE ADR2H EN A
        STAA    CH6     ; Y DESPUES LO ENVIA EN LA VARIABLE CH6
        RTS             ; REGRESO DESDE SUBRUTINA

;---------------------------------------------------------*
; SUBRUTINA DELAY 100 uS                                  *
; RETRASO REQUERIDO PARA QUE EL ATD SE ESTABILICE (100 uS)*
;---------------------------------------------------------*

RETARDO
                LDAA    #$C8    ; CARGA EL ACUMULADOR CON "100
                                  uS, VALOR DEL RETRASO (DELAY)"
DELAY:          DECA            ; DECREMENTA EL ACUMULADOR
                BNE     DELAY   ; BRANCH SI NO ES IGUAL A ZERO
                RTS             ; REGRESO DESDE SUBRUTINA




;*********************************************
; LOOP DE CONTROL DE LAS VARIABLES ANALOGICAS *
;*********************************************

LOOP_CONTROL
CANAL_7         LDAA    CH7         ; CARGA EL VALOR DE CH7 EN A
                CMPA    TOP_UP_1    ; LO COMPARA CON EL LIMITE SUPERIOR
                                      DEL SP1
                BHS     LED_CH6     ; BRINCA HACIA LED_CH7 SI ES MAYOR O
                                      IGUAL SI ES MENOR SE EJECUTA LA
                                      SIGUIENTE INSTRUCCIÓN
                CMPA    TOP_DOWN_1 ; COMPARA A CON EL LIMITE INFERIOR
                BLS     LED_CH6     ; BRINCA SI ES MENOR O IGUAL HACIA
                                      LED_CH7
                LDAA    PORTDLC     ; CARGA EL VALOR DEL PUERTO DLC EN A
                ANDA    #$30        ; EJECUTA UN AND ENTRE EL VALOR DE A
                                      Y EL INDICADO POR LA INSTRUCCIÓN
                STAA    PORTDLC     ; Y ENVIA EL RESULTADO AL PUERTO
                                      PARA APAGAR EL LED DEL PUERTO 7
CANAL_6         LDAA    CH6         ; CARGA EL VALOR DE CH6 EN A
                CMPA    TOP_UP_2    ; LO COMPARA CON EL LIMITE SUPERIOR
                                      DEL SP2
                BHS     LED_CH5     ; BRINCA HACIA LED_CH6 SI ES MAYOR O
                                      IGUAL SI ES MENOR SE EJECUTA LA
                                      SIGUIENTE INSTRUCCIÓN
                CMPA    TOP_DOWN_2 ; COMPARA A CON EL LIMITE INFERIOR
                BLS     LED_CH5     ; BRINCA SI ES MENOR O IGUAL HACIA
                                      LED_CH6
                LDAA    PORTDLC     ; CARGA EL VALOR DEL PUERTO DLC EN A
                ANDA    #$50        ; EJECUTA UN AND ENTRE EL VALOR DE A
                                      Y EL INDICADO POR LA INSTRUCCIÓN
                STAA    PORTDLC     ; Y ENVIA EL RESULTADO AL PUERTO
                                      PARA APAGAR EL LED DEL PUERTO 6
```

```
                RTS
;-------------------------------------------------------------------*
; SUBRUTINAS PARA ENCENDER LOS LEDS EN LOS BITS 6 Y 5 DEL PUERTO DLC*
;-------------------------------------------------------------------*

LED_CH6         LDAA    PORTDLC ; CARGA EN A EL VALOR DEL PUERTO DLC
                ORAA    #$40    ; REALIZA UNA OPERACIÓN OR ENTRE A Y EL
                                  VALOR INDICADO DESPUES DE LA
                                  INSTRUCCIÓN
                STAA    PORTDLC ; ENVIA EL RESULTADO  HACIA EL PUERTO
                                  DLC PARA ENCENDER EL LED 6
                BRA     CANAL_6 ; BRINCA HACIA CANAL_6
LED_CH5         LDAA    PORTDLC ; CARGA EN A EL VALOR DEL PUERTO DLC
                ORAA    #$20    ; REALIZA UNA OPERACIÓN OR
                                  ENTRE A Y EL VALOR INDICADO DESPUES
                                  DE LA INSTRUCCIÓN
                STAA    PORTDLC ; ENVIA EL RESULTADO  HACIA EL PUERTO
                                  DLC PARA ENCENDER EL LED 5
                RTS             ; RETORNO DE LA SUBRUTINA
;*******************************************************************



;************************************
; SUBRUTINA DE CALCULO DE FRECUENCIA   *
;************************************

FRECUENCIA      BRCLR   TFLG1,$01,FRECUENCIA; SE ESPERA LA BANDERA
                                  DE INTERRUPCION
                LDD     TC0H    ; SE CARGA EN D LOS PULSOS QUE CONTIENE
                                  EL REGISTRO DECAPTURA
                STD     N2      ; SE ENVIA EL NUMERO DE PULSOS A N2
                SUBD    N1      ; SE LE RESTA EL NUMERO PULSOS DE LA
                                  SECUENCIA  ANTERIOR
                STD     N       ; SE ENVIA EL RESULTADO A N
                MOVW    N2,N1   ; ENVIA N2 A N1 PARA INICIALIZARLA EN
                                  LA SIGUIENTE SECUENCIA
                LDY     #$03    ; CARGA UN 250000
                LDD     #$D090  ; 100
                LDX     N       ; CARGAU N COMO DIVISOR
                EDIV            ; EJECUTA LA DIVISION
                STY     VAR3    ; ENVIA EL RESULTADO HACIA VAR3,A LA
                                  QUE SE ACCEDE PARA MOSTRAR EN DISPLAY

                LDAA    #$01    ; CARGA UN #01 EN A
                STAA    TFLG1   ; BORRA BANDERA DE CONTEO DE PULSOS
                RTS
;************************************************************
;SUBRUTINA DE SELECCION DE VARIABLES A MOSTRAR EN EL DISPLAY *
;************************************************************

SELECTOR        LDAB    PORTDLC      ; CARGA EN B EL CONTENIDO DEL
                                       PUERTO DLC
                BITB    #$01         ; Y REALIZA UN AND ENTRE EL
                                       VALOR INDICADO Y EL ACUMULADOR B
                BNE     DISPLAYCH7   ; SI N0 ES IGUAL A LA SUBRUTINA
                                       INDICADA SI ES IGUAL, EJECUTA LA
                                       SIGUIENTE INSTRUCCION
                BITB    #$02         ; REALIZA UN AND ENTRE EL VALOR
                                       INDICADO Y B
                BNE     DISPLAYCH6   ; SI NO ES IGUAL BRINCA A LA
                                       SUBRUTINA INDICADA
                                     ; SI ES IGUAL, EJECUTA A LA
```

```
                                        SIGUIENTE INSTRUCCIÓN
                BITB    #$04          ; REALIZA UN AND ENTRE
                BNE     DISPLAYFREC

                CLR     PORTA         ; BORRA EL PUERTO A
                CLR     PORTB         ; BORRA EL PUERTO B
                RTS

DISPLAYCH7      CLR     PORTA         ; BORRA EL PUERTO A
                CLR     PORTB         ; BORRA EL PUERTO B
                LDAB    CH7           ; ENVIA EL CONTENIDO DEL CH7
                STAB    PORTB         ; AL PUERTO B
                LDAA    #$00
                STAA    PORTA         ; LIMPIA EL PUERTO A
                RTS


DISPLAYCH6      CLR     PORTA   ; BORRA EL PUERTO A
                CLR     PORTB   ; BORRA EL PUERTO B
                LDAB    CH6     ; ENVIA ELCONTENIDO DE CH6
                STAB    PORTB   ; AL PUERTO B
                LDAA    #$00    ;
                STAA    PORTA   ; LIMPIA EL PUERTO A
                RTS

DISPLAYFREC     CLR     PORTA
                CLR     PORTB
                LDD     VAR3    ; CARGA EN A EL VALOR DE VAR3
                STD     DATOHEX ; ENVIA EL VALOR HACIA DATOHEX
                JSR     HEX_BCD ; BRINCA A CONVERTIR A BCD
                LDAA    $0952   ; CARGA EN A EL BIT MENOS SIGNIFICATIVO
                STAA    PORTB   ; Y LO ENVIA AL PUERTO B
                LDAA    $0951   ; CARGA EN A EL BIT MAS SIGNIFICATIVO
                STAA    PORTA   ; Y LO ENVIA AL PUERTO A
                RTS             ; RETORNO DE LA SUBRUTINA

;*****************************************************
; SUBRUTINA DE ATENCION A INTERRUPCION DEL PUERTO SERIE*
;*****************************************************
SCI0            LDAB    SC0SR1  ; CARGA EN B EL CONTENIDO DEL REGISTRO
                BITB    #$20    ; PRUEBA SI SE ACTIVO EL BIT FIN
                                  DERECEPCIÓN
                BEQ     SCI0    ; BRINCA SI NO SE A TERMINADO LA
                                  SECUENCIA DE RECEPCION
                                ; SI SE TERMINO DE RECIBIR EJECUTA LA
                                  SIGUIENTE INSTRUCCION
                LDAB    CUENTA  ; ALMACENA EN B EL CONTENIDO DE CUENTA
                LDAA    SC0DRL  ; CARGA EN A EL CONTENIDO DEL REGISTRO
                                  DE DATOS RECIBIDOS
                STAA    DATO    ; Y LO ENVIA HACIA LA VARIABLE DATO
                DEC     CUENTA  ; DECREMENTA LA VARIABLE CUENTA
                LDAB    CUENTA  ; CARGA B CON EL CONTENIDO DE CUENTA
                CMPB    #$00    ; LO COMPARA CONTRA 0
                BNE     UNO     ; SI NO ES IGUAL BRINCA A LA SUBRUTINA
                                  UNO, SI ES IGUAL EJECUTA LA SIGUIENTE
                                  INSTRUCCIÓN
                LDAA    #$02    ; CARGA EN A UN #$02
                STAA    CUENTA  ; Y POSTERIORMENTE LO ENVIA A CUENTA
                LDAA    DATO    ; CARGA EN A EL CONTENIDO DE LA
                                  VARIABLE DATO
                STAA    V2      ; ENVIA EL CONTENIDO DE DATO A LA
                                  VARIABLE V2
                RTI             ; RETORNO DE INTERRUPCIÓN

UNO             LDAA    DATO    ; CARGA EN A EL CONTENIDO DE LA
```

```
                                VARIABLE DATO
                STAA    V1      ; ENVIA EL CONTENIDO DE A HACIA LA
                                  VARIABLE V1
                RTI             ; RETORNO DE INTERRUPCION



;*********************************
;SUBRUTINA PARA CONVERTIR A BCD  *
;*********************************

HEX_BCD:        MOVW    #$0,DATOBCD     ; INICIALIZA RESULTADO BCD
                clr     DATOBCD+2
                ldd     DATOHEX         ; LEE DATO HEXADECIMAL
                cpd     #$09            ; SI ES MENOR QUE 9 NO
                                          CONCIERTE
                bhi     CONVIERTE       ; SI ES MAYOR QUE 9 CONVIERTE
                stab    DATOBCD+2       ; ES MENOR QUE NUEVE, COLOCA EL
                                          RESULTADO
                bra     FIN_BCD         ; TERMINA

CONVIERTE:      ldx     #0AH            ; PREPARA DIVISION ENTRE BASE
                                          10
                idiv                    ; PREPARA DIVISION
                stab    DATOBCD+2       ; PRIMER RESIDUO, COLOCA EN
RESULTADO
                xgdx                    ; CAMBIA COCIENTE A D (A:B)
                cpd     #09H            ; SI ES MAYOR A 9 EFECTUA LA
                                          SEGUNDA DIVISIÓN
                bhi     DIV_2
                lslb                    ; COCIENTE ES MENOR QUE 9
                lslb                    ; PREPARA EL COCIENTE
                lslb
                lslb
                addb    DATOBCD+2       ; JUNTA DIGITOS DE PESO Y 1
                stab    DATOBCD+2
                bra     FIN_BCD         ; TERMINA

DIV_2:          ldx     #$0A            ; PREPARA DIVISIÓN ENTRE BASE
                                          10
                idiv                    ; EJECUTA LA DIVISION
                lslb                    ; PREPARA EL SEGUNDO RESIDUO
                lslb
                lslb
                lslb
                addb    DATOBCD+2       ; JUNTA DIGITOS DE PESO 0 Y 1
                stab    DATOBCD+2


                xgdx                    ; CAMBIA COCIENTE A D
                cpd     #09H            ; SI ES MENOR A 9 EFECTUA LA
                                          TERCERA DIVISION
                bhi     DIV_3
                stab    DATOBCD+1       ; COCIENTE ES MENOR A 9,COLOCA
                                          DIGITO PESO 2
                bra     FIN_BCD         ; TERMINA

DIV_3:          ldx     #$0A            ; PREPARA TERCERA DIVISION
                idiv                    ; EJECUTA DIVISION
                stab    DATOBCD+1       ; COLOCA DIGITO PESO 2
                xgdx                    ; CAMBIA COCIENTE A D
                cpd     #09H            ; SI ES MAYOR A 9 EFECTUA
                                          CUARTA DIVISION
```

```
                bhi     DIV_4
                lslb                    ; COCIENTE ES MENOR A 2
                lslb                    ; PREPARA DIGITO PESO 3
                lslb
                lslb
                addb    DATOBCD+1       ; COMPLETA DIGITO DE PESO 2 Y 3
                stab    DATOBCD+1
                bra     FIN_BCD         ; TERMINA

DIV_4:          ldx     #$0a            ; PREPARA CUARTA DIVISION
                idiv                    ; EJECUTA LA DIVISION
                lslb                    ; PREPARA DIGITO DE PESO 3
                lslb
                lslb
                lslb
                addb    DATOBCD+1       ; COMPLETA DIGITO DE PSO 2 Y 3
                stab    DATOBCD+1
                xgdx                    ; CAMBIA ULTIMO COCIENTE A D
                stab    DATOBCD         ; COLOCA DIGITO DE PESO 4
FIN_BCD:        RTS
;*****************************************************************


                ORG     $0B16   ; DEFINE EL ORIGEN PARA EL VECTOR DE
                                  INTERRUPCION
                DC.W    SCI0    ; DEL PUERTO SERIE

                ORG     $950    ; DEFINE EL ORIGEN A PARTIR DE LA
                                  LOCALIDAD $0950

DATOBCD:        ds.b    3; DEFINE LOCALIDAD DE MEMORIA DE 3 BYTE
DATOHEX:        ds.b    2; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE
CH6             DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           CANAL 6
CH7             DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           CANAL 7
TOP_UP_1        DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LIMITE SUPERIOR DE SP1
TOP_DOWN_1      DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LIMITE INFERIOR DE SP1
TOP_UP_2        DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LIMITE SUPERIOR DE SP2
TOP_DOWN_2      DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LIMITE INFERIOR DE SP2
VAR3            DS.B    2; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE VAR3
SP1             DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE SP1
SP2             DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE SP2
N2              DS.W    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE N2
N1              DS.W    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE N1
N               DS.W    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE N
CUENTA          DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE CUENTA
V1              DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE V1
V2              DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE V2
DATO            DS.B    1; DEFINE LOCALIDAD DE MEMORIA DE 1 BYTE PARA
                           LA VARIABLE DATO
```

Modulo Generador de Frecuencia

El Generador de Frecuencia es un microcontrolador programado en modo autónomo con el canal 0 del PWM como salida de frecuencia. Básicamente se programa el pwm del micro con una resolución a 8 bits y se genera una frecuencia de salida variable con la lectura de el canal analógico 7 y almacenando a esta lectura en los registros de periodo y duración de ciclo, obteniendo una forma de onda cuadrada de tamaño simétrico en cada uno de sus fases (ONN-OFF), este canal se conecta externamente a un potenciómetro para modificar al valor de frecuencia y así obtener el valor deseado.

El micro esta ciclado indefinidamente por lo que se puede variar la frecuencia en cualquier momento.

A continuación se muestra el programa empleado para cada parte del proyecto:

```
;***********************************************
; GENERADOD DE FRECUENCIA VARIABLE DE 1 A 10 KHZ*
;***********************************************

#include hc12.inc

; INICIO DEL PROGRAMA
        org EE_START

; CONFIGURACION DE REGISTROS DEL PWM

                CLR     COPCTL
                LDS     #$0C00
                CLR     CH7
                MOVB    #$1C,PWCLK ; SELECCIONO LOS DOS RELOJES PARA
                                     CANAL 0,1,2 .SE DIVIDE EL RELOJ
                                     ENTRE 120 6.5 HZ
                MOVB    #$FF,PWPOL ; SELECCIONO POLARIDAD POSITIVA Y
                                     FUENTES DE RELOJ A,B PARA TODOS
                                     LOS CANALES
                MOVB    #$08,PWCTL ; CONFIGURO EL PWM PARA FUNCIONAR EN
                                     EL CENTRO DEL PERIODO
                MOVB    #$01,PWEN  ; HABILITO LOS CANALES 0,1,2

MAIN:           MOVB    CH7,PWPER0 ; DURACION DEL PERIODO PARA LOGRAR
                                     1KHZ DE PERIODO

                LDD     CH7        ; CARGA CH7 EN D
                LDX     #$02       ; CARGA EN X UN 2 COMO DIVISOR
                IDIV               ; EJECUTA LA DIVISION
                STX     VAR1       ; ENVIA EL RESULTADO A LA VARIABLE
                                     VAR1
                MOVB    VAR1,PWDTY0 ; FACE A .7500ms

                BSR     INIT       ; SALTO A INIT SUPRUTINA PARA
                                     INICIALIZAR EL ADC
                JSR     CONVERT    ; SALTO A CONVERT SUBRUTINA PARA
                                   CONVERSION
                BRA     MAIN
; CONFIGURA EL CONVERTIDOR

INIT:
                LDAA    #$80    ; PERMITE AL CONVERTIDOR FUNCIONAR
                                  NORMALMENTE
```

```
                STAA    ATDCTL2 ; SE DESHABILITAN INTERRUPCIONES Y SE
                                  CONFIGURA AL ADC PARA FUNCIONAR
                                  NORMALMENTE
                BSR     DELAY   ; SALTO A SUBRUTINA DELAY, RETRASO DE
                                  100 uS
                LDAA    #$00    ; SELECCIONA CONVERSION CONTINUA EN
                                  MODO BGND
                STAA    ATDCTL3 ; IGNORA FREEZE EN ADCTL3
                LDAA    #$01    ; SELECCIONA LA VELOCIDAD DEL ADC A 2
                                  A/D CLOCKS
                STAA    ATDCTL4 ; SELECCIONA EL DIVISOR POR 4
                RTS             ; REGRESO DE SUBRUTINA

; ********************************************************************
; CONFIGURA E INICIA LA CONVERSION Y PONE EL RESULTADO EN UNA LOCALIDAD
; DE MEMORIA
; CONFIGURA E INICIA LA CONVERSION
; SEÑAL ANALOGA DE ENTRADA EN PUERTO AD6
; CONVERT: USA UN SOLO CANAL
; EL RESULTADO PODRA SER ENCONTRADO EN ADR2H
;*******************************************************************
CONVERT:
                LDAA    #$17    ; INICIALIZA ATD: SCAN=0,MULT=1,PAD7.
                                  ESCRIBE LIMPIAR BANDERAS
                STAA    ATDCTL5 ; 4 CONVERSIONES EN UNA SECUENCIA DE
                                  CONVERSION SENCILLA
WTCONV:         BRCLR   ATDSTATH,#$80,  WTCONV  ; ESPERA A LA BANDERA
                                  DE SECUENCIA COMPLETADA
                LDAA    ADR3H   ; CARGA EN A EL CONTENIDO DEL REGISTRO
                                   ADR3H
                STAA    CH7     ; Y LO ENVIA A LA VARIABLE CH7
                RTS             ; RETORNO DESDE SUBRUTINA
;*********************************************************
; SUBRUTINA DELAY 100 Us                                  *
; RETRASO REQUERIDO PARA QUE EL ATD SE ESTABILICE (100 uS)*
;*********************************************************

                LDAA    #$C8    ; CARGA EL ACUMULADOR CON "100 uS,
                                  VALOR DEL RETRASO (DELAY)"
DELAY:          DECA            ; DECREMENTA EL ACUMULADOR
                BNE     DELAY   ; BRANCH SI NO ES IGUAL A ZERO
                RTS
                BRA     MAIN    ; BRINCO A INICIO DEL PROGRAMA (MAIN)

                ORG     $0950
CH7             DS.B    1; RESERVA UNA LOCALIDAD DE 1 BYTE PARA CH7
VAR1            DS.B    1; RESERVA UNA LOCALIDAD DE 1 BYTE PARA VAR1
```

CAPITULO 8

CONCLUSIONES Y RECOMENDACIONES

En el desarrollo de esta tesis se fueron cubriendo las partes fundamentales para la comprensión del funcionamiento del microcontrolador, trato de hacerse un seguimiento cronológico avanzando desde las cuestiones más básicas hasta profundizar en el manejo de los diferentes periféricos del microcontrolador. Esto permite que el seguimiento de este manual didáctico reduzca el tiempo necesario empleado en el aprendizaje del uso y programación del microcontrolador MC68HC12B32.

Las practicas desarrolladas para cada tema fueron desarrolladas tratando de aclarecer la mayoría de las dudas que se pudieran tener dentro del proceso de aprendizaje del microcontrolador por lo cual el seguimiento de las mismas, desarrollara una mayor habilidad en el manejo del microcontrolador así como preparara al usuario para elaborar aplicaciones mas desarrolladas conforme vaya avanzando con este manual didáctico.

Debido a la naturaleza practica de este manual didáctico, se recomienda tratar de realizar la mayoría de las prácticas con sus circuitos de interfaz, ya que esto nos permitirá aclarar la mayoría de las dudas que surjan al utilizar cada uno de los periféricos o módulos del microcontrolador y dejara mas en claro el objetivo de cada practica, a la vez que tratara de ampliar el panorama al usuario acerca del uso para el que puede emplear este dispositivo.

Al tratarse de algo práctico también es recomendable tratar de invertir mas tiempo desarrollando aplicaciones o cubriendo más ejercicios con el microcontrolador para tratar de ver la mayoría de las situaciones posibles en las que se puede ver envuelto al momento del desarrollo de una aplicación practica.

Bibliografía

Motorota: CPU12 Reference Manual, Abril del 2002, Revisión 3
Motorota: MC68HC12B Family Data Sheet, Julio del 2003, Revisión 8

Microcontroller Technology The 68HC11 and 68HC12
Autor: Peter Spasov
Editorial: Pearson Prentice Hall

Quinta Edición
ISBN: 0-13-112984-8

The M68HC11 Microcontroller
Aplications in Control, Instrumentation, and Communication
Autor: Michael Kheir
Editorial: Pearson
ISBN:0-13-205550-3

Software

MiniIDE Version 1.17
www.mqtek.com/miniide

Electronics Workbench Multisim Power Pro Version 8
www.electronicsworkbench.com

Listado de Figuras

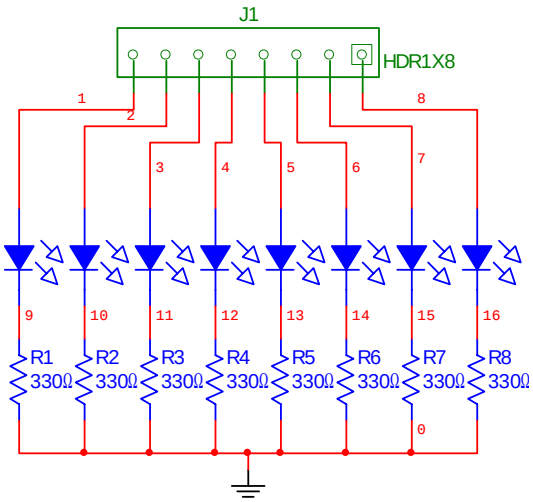
Listado de Tablas

Apéndice A Diagramas Esquemáticos

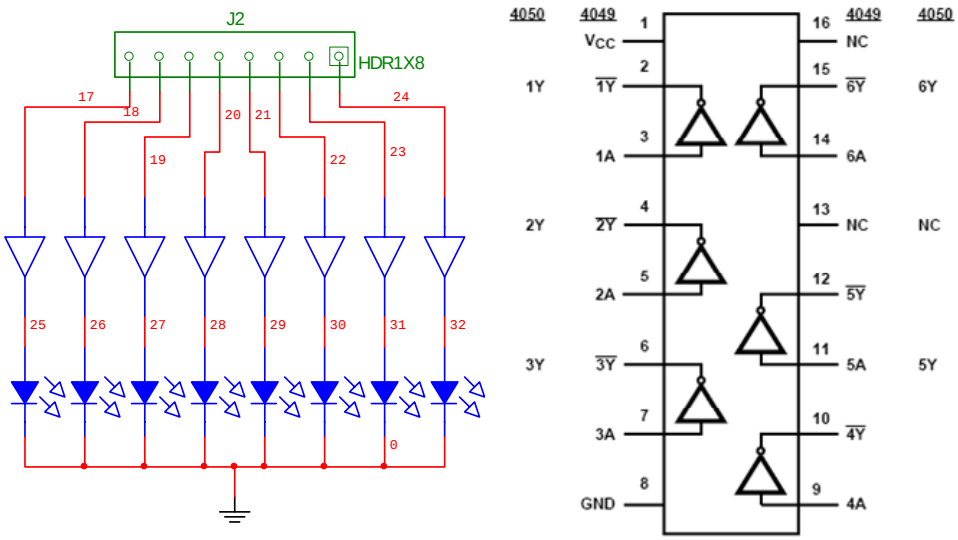
Circuito 1

El siguiente circuito es la forma más básica para poder observar el valor de salida en un puerto. Esta formado únicamente por un led por cada terminal de salida (Ver Figura A) y una resistencia limitadora de corriente, se puede adaptar el circuito para mostrar menos bits según el tamaño del puerto. También puede utilizarse un Buffer CMOS como

los circuitos CD4050B ó CD4049BE para eliminar el uso de las resistencias y reducir la corriente de entrega del puerto, sus conexiones se muestran en la figura B.



A)



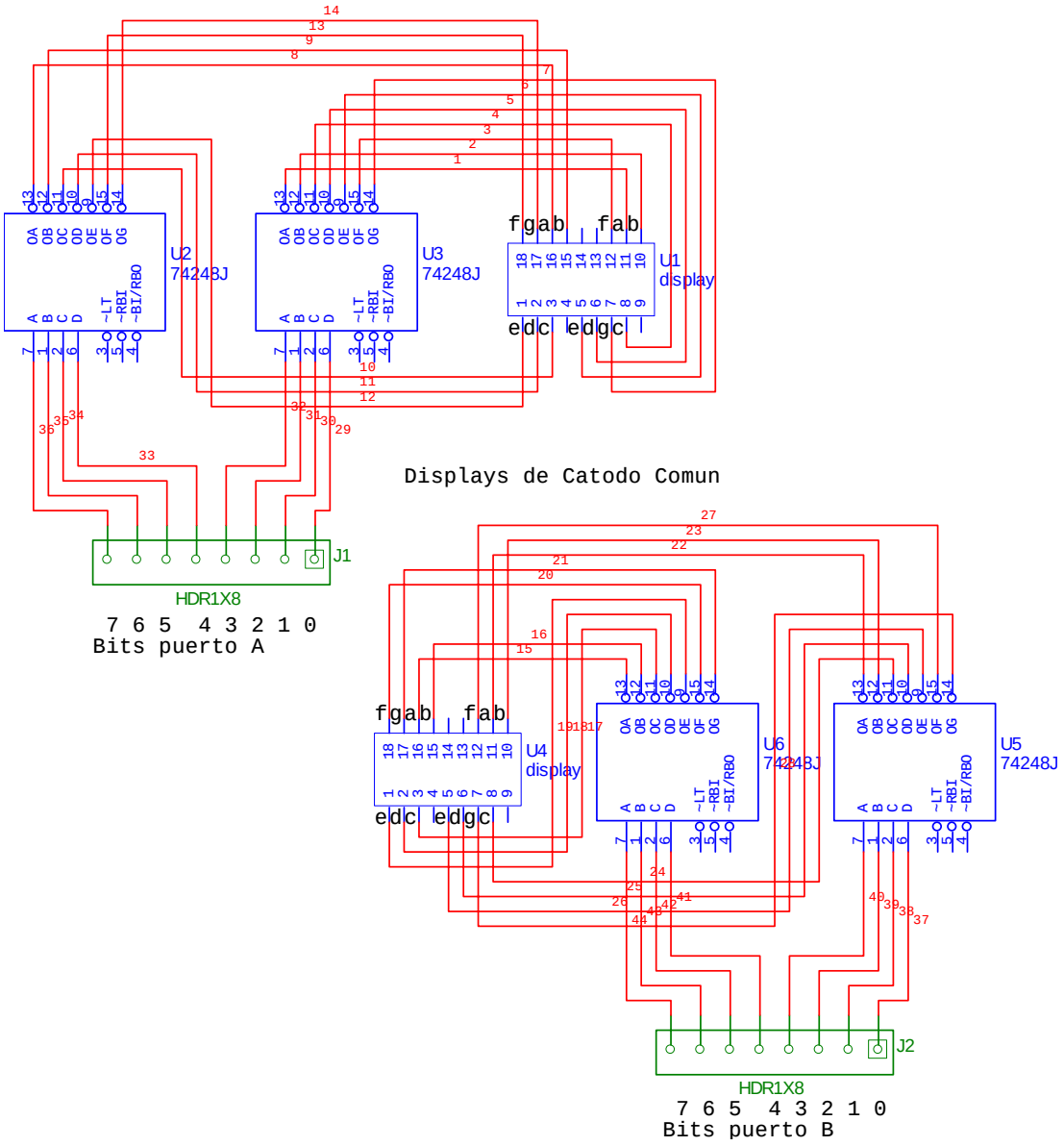
B) CD4050E ,CD4049E

Circuito 2

El siguiente circuito, esta formado por dos displays dobles de 7 segmentos de tipo cátodo común, y 4 circuitos 74LS48 los cuales son utilizados para manejar los displays.

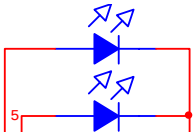
Cata display puede ser manejado por cualquier puerto de 8 bits, por lo que para manejar el circuito completo se necesitan dos puertos de salida los cuales pueden ser A y B por ejemplo, tomando al puerto A como el byte de la parte alta y al puerto B como el

byte de la parte baja. Se recomienda poner los display juntos para una mejor visualización.



Circuito 3

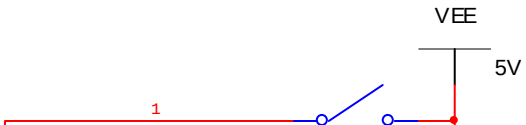
Este circuito trabaja con un registro de corrimiento de entrada serial y salida en paralelo de 8 bits, tiene match que retienen el ultimo valor de entrada y por sus características de CMOS podemos conectarlo directamente a los leds para visualizar el



dato de entrada, o incluso se pueden sustituir los leds y conectar directamente a un circuito 74LS48,49 para manejar un display.

Circuito 4

Este circuito sirve para ingresar datos binarios a un puerto y esta formado por un dip swuitch de 8 bits y una serie de resistencias. Es necesario conectar las líneas que no

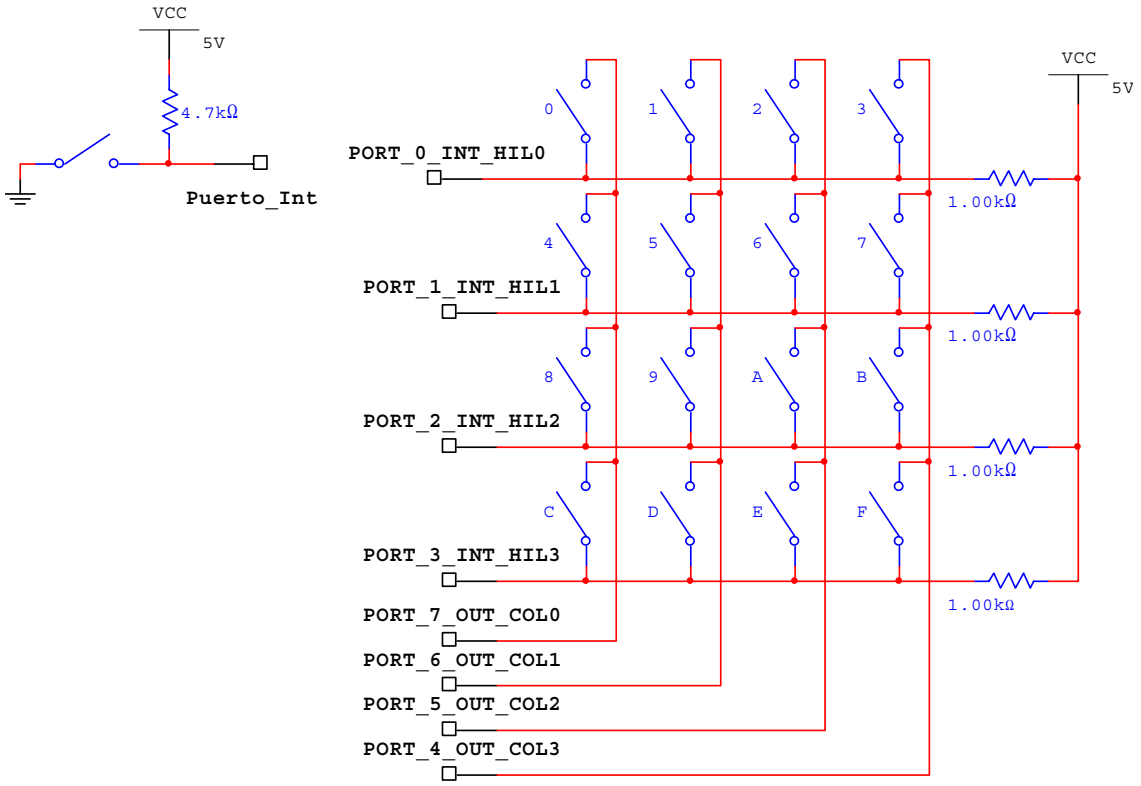


están activas (0 lógico) a tierra a través de una resistencia de 1 K para evitar la entrada de ruido.

Circuito 5

Este teclado esta formado por una red de microswitchs normalmente abiertos conectados a Vcc mediante una resistencia que limita la corriente para proteger al switch.

Puede utilizarse una conexión más en la terminal del switch conectado al puerto para que al activarse una tecla se genere una interrupción.



Teclado Matricial con Interrupción

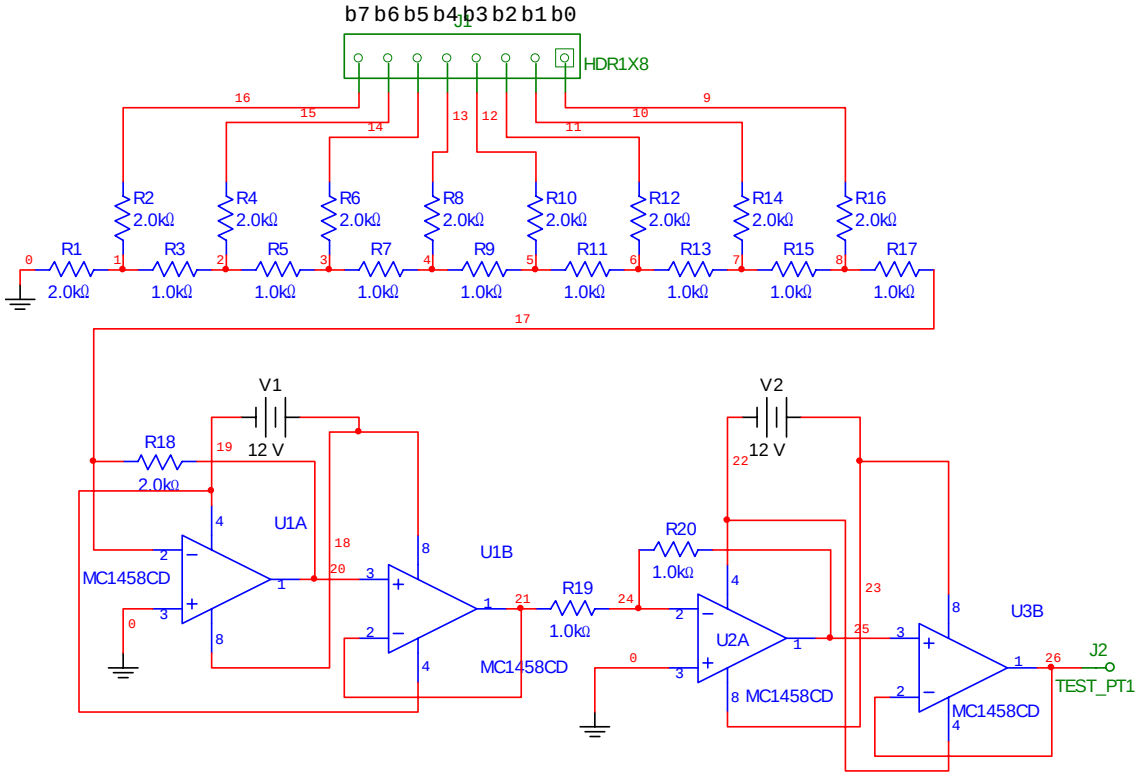
Circuito 6

El siguiente circuito muestra un convertidor de digital a analógico, con formato de entrada en código BCD con una resolución de 8 bits del tipo R2R.

Para calcular el tamaño del salto en voltaje se dividen el valor del voltaje entre la resolución total del convertidor, por ejemplo si se utiliza dentro de un rango de 0 a 5 volts se obtiene lo siguiente:

$$\frac{5}{256} = .019mV$$

Este mismo circuito se puede adaptar para aumentar el valor de la resolución únicamente se van agregando las 2 resistencias necesarias por cada bit que se agregue y se realiza el mismo procedimiento anterior para obtener el valor del salto en voltaje.



CONVERTIDOR DIGITAL ANALOGO TIPO R2R DE 8 BITS

Apéndice B Instrucciones del Microcontrolador

Instrucciones de Carga y Almacenamiento de Datos.

Instrucciones de Carga

LDAA: Carga un dato de 8 bits en el acumulador A.

```
Ejemplo     LDAA #$FF    ; A=FF (Direccionamiento Inmediato)
            LDAA $1000   ; Introduce en A el contenido de la dirección $1000
                           (Direccionamiento Directo)
```

LDAB: Carga un dato de 8 bits en el acumulador B.

LDD: Carga un dato de 8 o 16 bits en el doble acumulador D que se compone por el acumulador A y el acumulador B.

```
Ejemplo     LDD #$55FF       ; D=55FF, A=55 Y B=FF.
            LDD #$50         ; D=0050, A=00 Y B=50.
```

LDS: Carga un dato de 16 bits en el puntero de pila. Primero introduce en el byte más significativo el contenido de la memoria y después introduce en el byte menos significativo el siguiente byte.

LDX: Carga un dato de 16 bits en el registro índice X.

```
Ejemplo     LDX $1000        ; Introduce el contenido de la
                               dirección $1000 en el registro índice X.
```

LDY: Carga un dato de 16 bits en el registro índice Y.

LEAS: Carga el puntero de pila con la dirección efectiva especificada por el programa. Esta instrucción puede indicarse mediante cualquier operando con direccionamiento indexado.

```
Ejemplo:    LEAS 4,Y+        ; Primero S es cargado con el valor
                               de Y, después Y es incrementada con 4.
```

LEAX: Carga el registro índice X con la dirección efectiva especificada por el programa. Esta dirección puede indicarse mediante cualquier operando mediante direccionamiento indexado.

```
Ejemplo:    LEAX 4,Y+        ; Primero X es cargado con el valor de Y, después
                               Y es incrementada con 4.
```

LEAY: Carga el registro índice Y con la dirección efectiva especificada por el

programa. Esta dirección puede indicarse mediante cualquier operando a través de direccionamiento indexado.

```
Ejemplo:    LEAY 4,X+   ;Primero Y se carga con el valor del
                        registro X y después es incrementada con 4.
```

Instrucciones de Almacenamiento

STAA: Envía el contenido del acumulador A hacia una localidad de memoria.

```
Ejemplo     STAA $00FF ; Envía el contenido del acumulador A hacia la
            dirección $00FF mediante direccionamiento directo.
```

STAB: Envía el contenido del acumulador B hacia una localidad de memoria.

```
Ejemplo     STAB $1230 ; Envía el contenido del acumulador B hacia la
                       dirección $1230 mediante direccionamiento
                       extendido.
```

STD: Envía el contenido del doble acumulador D formado por los acumuladores A y B hacia una localidad de memoria de 16 bits.

STS: Guarda el contenido del puntero de pila en una memoria. El byte mas significativo es guardado en la dirección especificada, mientras el byte menos significativo es colocado en la siguiente localidad de memoria.

STX: Guarda el contenido del registro índice X en una memoria. El byte más significativo es guardado en la dirección especificada, mientras el byte menos significativo es colocado en la siguiente localidad de memoria.

STY: Guarda el contenido del registro índice Y en una memoria. El byte de mayor pero es guardado en la dirección especificada, mientras el byte de menor pero es colocado en la siguiente localidad de memoria.

Instrucciones de Transferencia e Intercambio de Datos.

Las instrucciones de transferencia copian el contenido de un registro o acumulador dentro de otro registro u acumulador. El contenido de las fuentes no cambia después de la operación.

Instrucciones de Transferencia

TAB: Transfiere el contenido del acumulador A hacia el acumulador B. El contenido del acumulador A no es afectado por la operación mas sin embargo el del registro B si resulta afectado. Los contenidos de estos registros son de 8 bits.

TAP: Transfiere el estado de los bits [7:0] del acumulador A, a la posición del bit correspondiente del registro CCR. El contenido del registro A no se altera con esta operación.

TBA: Transfiere el contenido del acumulador B hacia el acumulador A. El contenido del acumulador B no es afectado por la operación, pero el acumulador A contiene los nuevos datos enviados del acumulador B hacia el después de la operación.

TFR: Transfiere el contenido de un registro fuente hacia un registro destino especificado en la instrucción. Puede Transferir registros de 8 y 16 bytes, afectándose primero el byte más significativo de un registro.

TPA: Transfiere el contenido del registro CCR a su correspondiente posición de cada bit dentro del acumulador A. Después de la operación el contenido del registro CCD resulta inalterado.

TSX: Este es un mnemónico alterno que transfiere el valor del Stack Pointer (Puntero de pila) hacia el registro índice X. El Puntero de pila (Stack pointer) no resulta afectado por esta instrucción, es decir permanece inalterado. Después de ejecutar la instrucción TSX, X apunta hacia la ultima dirección almacenada en el Stack pointer.

TSY: Este es un mnemónico alterno que transfiere el valor del stack pointer al registro índice Y. El Puntero de pila (Stack pointer) no resulta afectado por esta instrucción, es decir permanece inalterado. Después de ejecutar la instrucción TSY, Y apunta hacia la ultima dirección almacenada en el Stack pointer.

TXS: Este es un mnemónico alterno que transfiere el contenido del registro indice X al Puntero de pila (Stack pointer). El contenido de X resulta inalterado.

TYS: Este es un mnemónico alterno que transfiere el contenido del registro indice Y al Puntero de pila (Stack pointer). El contenido de Y resulta inalterado.

EXG: Intercambia el contenido de los registros especificados por la instrucción.

XGDX: Intercambia el contenido del doble acumulador D (acumuladores A y B) con el registro índice X. Para compatibilidad con el 68HC11 esta instrucción puede cambiarse por EXG D, X por el ensamblador.

XGDY: Intercambia el contenido del doble acumulador D (acumuladores A y B) con el registro índice Y. Para compatibilidad con el 68HC11 esta instrucción puede cambiarse por EXG D, Y por el ensamblador.

SEX: Esta instrucción es un mnemónico alterno para la instrucción TFR r1,r2, donde r1 es un registro de 8 bits y r 2 es un registro de 16 bits. El resultado en r2 es una representación de 16 bits del numero 2 complemento de r1. El contenido de r1 resulta inalterado en todos los casos excepto en SEX A, D (donde D es A y B).

Instrucciones para Copiar "Move".

Las instrucciones Move copian bytes o palabras desde una fuente hacia un destino. Se permiten 6 combinaciones de direccionamiento inmediato, extendido e indexado (IMM ⇒EXT, IMM ⇒IDX, EXT ⇒EXT, EXT ⇒IDX, IDX ⇒EXT, IDX ⇒IDX) para especificar la fuente y la dirección de destino.

MOVB: Copia un dato de 1 byte (8 bits) de una fuente a un destino.

Ejemplo: MOVB #$FF, PORTA

MOVW: Copia un dato de 2 byte (16 bits) de una fuente a un destino.

Ejemplo: MOVW ¿?

Instrucciones de Suma y Resta

Las sumas con signo y sin signo pueden ser realizadas entre registros y registros y memoria. También pueden ser realizados cálculos con acarreo ya que hay instrucciones que suman un bit de acarreo al registro CCR para facilitar operaciones de mayor precisión.

Las restas con signo y sin signo ser realizadas entre registros y registros y memoria. También pueden ser realizados cálculos con acarreo ya que hay instrucciones que restan un bit de acarreo al registro CCR para facilitar operaciones de mayor precisión.

Instrucciones de Suma

ABA: Suma el contenido del acumulador A al acumulador B, el resultado es colocado en A. El contenido de B no cambia. El bit H del registro estatus es modificado por esta instrucción, lo que puede servir de ayuda para operaciones de calculo BCD. Observar la instrucción DAA para mas información.

ABX: Suma el contenido del acumulador B al registro índice X.

ABY: Suma los 8 bits sin signo del acumulador B al registro índice Y considerando el posible acarreo del bit de menor peso del registro Y. El resultado es puesto en Y, el contenido del acumulador B permanece inalterable.

ADCA: Suma al acumulador A a una localidad de memoria y el contenido del acarreo. Esta instrucción modifica el bit H del registro de estatus por lo que puede ser utilizada para operaciones aritméticas en BCD. Observar la instrucción DAA para más información.

ADCB: Suma al acumulador B una localidad de memoria y el contenido del acarreo. Esta instrucción modifica el bit H del registro de estatus por lo que puede ser utilizada para operaciones aritméticas en BCD. Observar la instrucción DAA para más información.

ADDA: Suma el contenido de una localidad de memoria al acumulador A. El resultado se almacena en A. Esta instrucción modifica el bit H del registro de estatus por lo que puede ser utilizada para operaciones aritméticas en BCD. Observar la instrucción DAA para más información

ADDB: Suma el contenido de una localidad de memoria al acumulador B. El resultado se almacena en B. Esta instrucción modifica el bit H del registro de estatus por lo que puede ser utilizada para operaciones aritméticas en BCD. Observar la instrucción DAA para más información

ADDD: Suma el contenido de una localidad de memoria y el siguiente valor, al doble acumulador D. El resultado se agrega en D. Recordemos que el acumulador A forma los 8 bits de mayor peso del acumulador D, y el acumulador B forma los 8 bits de menor peso.

Instrucciones de Resta

SBA: Substrae el contenido del acumulador B desde el contenido del acumulador A. El resultado se coloca en A, el contenido del acumulador B no se altera. Para las substracciones el bit C del registro de estado representa el préstamo.

SBCA: Substrae el contenido de una localidad de memoria (un dato), y el valor del bit C del registro desde el acumulador A. El resultado es colocado en A.

Para las substracciones el bit C del registro de estado representa el préstamo.

SBCB: Substrae el contenido de una localidad de memoria (un dato), y el valor del bit C del registro desde el acumulador B. El resultado es colocado en B. Para las substracciones el bit C del registro estado representa el préstamo.

SUBA: Substrae el contenido de una localidad de memoria de el acumulador A. el resultado es colocado en el acumulador A. Para las substracciones el bit C del registro estado representa el préstamo.

SUBB: Substrae el contenido de una localidad de memoria del acumulador B. El resultado es colocado en el acumulador B. Para las substracciones el bit C del registro de estado representa el préstamo.

SUBD: Substrae el contenido de una localidad de memoria y la siguiente del acumulador D. El resultado se coloca en D. Para las substracciones el bit C del registro de estado representa el préstamo.

Instrucciones de Decremento e Incremento

Las instrucciones de decremento e incremento son operaciones optimizadas de suma y resta de 8 y 16 bits. Generalmente se les emplea para generar contadores o retardos.

Instrucciones de Decremento

DEC: Decrementa uno de una dirección de memoria.
DECA: Decrementa en uno el contenido del acumulador A.
DECB: Decrementa en uno el contenido del acumulador B.
DES: Decrementa en uno el contenido del puntero de pila (SP).
DEX: Decrementa en uno el contenido del registro índice X.
DEY: Decrementa en uno el contenido del registro índice Y.

Instrucciones de Incremento

INC: Incrementa en uno el contenido de una memoria.
INCA: Incrementa en uno el contenido del acumulador A.
INCB: Incrementa en uno el contenido del acumulador B.
INS: Incrementa en uno el contenido del puntero de Pila (SP).
INX: Incrementa en uno el contenido del registro índice X.
INY: Incrementa en uno el contenido del registro índice Y.

Instrucciones de Comparación y Prueba de Bits.

Instrucciones de Comparación

CBA: Compara el contenido del acumulador A con el acumulador B y activa el bit correspondiente del CCR. Esta instrucción puede ser usada para operaciones aritméticas, lógicas o saltos condicionales (BRANCH).

CMPA: Compara el contenido del acumulador A con el contenido de una memoria y activa el bit correspondiente del CCR. El contenido de la memoria y el acumulador no cambian.

Esta instrucción puede ser usada para operaciones aritméticas, lógicas o saltos condicionales (BRANCH).

CMPB: Compara el contenido del acumulador B con el contenido de una memoria y activa el bit correspondiente del CCR. El contenido de la memoria y el acumulador no cambian.

Esta instrucción puede ser usada para operaciones aritméticas, lógicas o saltos condicionales (BRANCH).

CBD: Compara el contenido del doble acumulador D con el valor de 16 bits de la dirección especificado en la instrucción y activa el bit correspondiente del CCR. La comparación se logra internamente por una substracción de 16 bits de la memoria con el acumulador D, el contenido de ninguno de los dos cambia.

CPS: Compara el contenido del puntero de pila con el valor de 16 bits de la dirección especificada y activa el bit correspondiente del CCR. La comparación se logra internamente por una substracción de 16 bits de la memoria con el acumulador SP, el contenido de ninguno de los dos cambia.

CPX: Compara el contenido del registro índice X con el valor de 16 bits de la dirección especificada y activa el bit correspondiente del CCR. La comparación se logra internamente por una substracción de 16 bits de la memoria con el registro índice X, el contenido de ninguno de los dos cambia.

CPY: Compara el contenido del registro índice Y con el valor de 16 bits de la dirección especificada y activa el bit correspondiente del CCR. La comparación se logra internamente por una substracción de 16 bits de la memoria con el registro índice Y, sin modificar el contenido de ninguno de los dos.

Instrucciones de Prueba (Test) de Bits

TST: Substrae $00 al contenido de una localidad de memoria y activa el bit correspondiente del CCR. La substracción se logra internamente sin modificar la memoria. La instrucción TST provee de información limitada cuando se prueban valores sin signo, por lo que no resulta util la ejecución de instrucciones como BLO y BLS , mientras que BHI puede ser utilizada después de TST, esta efectúa la misma operación que BNE la cual es preferida. Pero después de utilizarse para valores con signo, todas las instrucciones de bifurcación pueden ser utilizadas.

TSTA: Substrae $00 al contenido del acumulador A y activa el bit correspondiente del CCR. La substracción se logra internamente sin modificar la memoria. La instrucción TSTA provee de información limitada cuando se prueban valores sin signo, por lo que no resulta útil la ejecución de instrucciones como BLO y BLS, mientras que BHI puede ser utilizada después de TSTA, esta efectúa la misma operación que BNE la cual es preferida. Pero después de utilizarse para valores con signo, todas las instrucciones de bifurcación pueden ser utilizadas.

TSTB: Substrae $00 al contenido del acumulador B y activa el bit correspondiente del CCR. La substracción se logra internamente sin modificar la memoria. La instrucción TSTA provee de información limitada cuando se prueban valores sin signo, por lo que no resulta útil la ejecución de instrucciones como BLO y BLS, mientras que BHI puede ser utilizada después de TSTA, esta efectúa la misma operación que BNE la cual es preferida. Pero después de utilizarse para valores con signo, todas las instrucciones de bifurcación pueden ser utilizadas.

Instrucciones de Lógica Booleana

ANDA: Efectúa la operación lógica AND entre el contenido de una memoria y el contenido del acumulador A, el resultado es colocado en A. Después de que la operación es efectuada cada bit de A es el lógico AND de la memoria antes de que se ejecutara la instrucción.

ANDB: Efectúa la operación lógica AND entre el contenido de una memoria y el

contenido del acumulador B, el resultado es colocado en B. Después de que la operación es efectuada cada bit de B es el lógico AND de la memoria antes de que se ejecutara la instrucción.

ANDCC: Efectúa la operación lógica AND entre el contenido de un operando de mascara y el contenido del CCR, el resultado es colocado en CCR. Después de que la operación es efectuada cada bit de CCR es el lógico AND de los correspondientes bits de la mascara. Para limpiar los bits de CCR, hay que limpiar los correspondientes bits de la mascara.

Los bits del CCR que corresponden a cada uno de los bits de la mascara no cambian por una operación ANDCC.

EORA: Efectúa la operación lógica OR Exclusivo y el contenido del acumulador A, el resultado se coloca en A. Cada bit del acumulador A es la lógica exclusiva OR del correspondiente bit de la memoria y A antes de que se realizara la operación.

EORB: Efectúa la operación lógica OR Exclusivo y el contenido del acumulador B, el resultado se coloca en B. Cada bit del acumulador B es la lógica exclusiva OR del correspondiente bit de la memoria y B antes de que se realizara la operación.

ORAA: Efectúa la operación OR entre el contenido del acumulador A y el contenido de la memoria, el resultado se coloca en A. Cada bit de A después de la operación es la lógica OR entre el correspondiente bit de la memoria y el acumulador A antes de la operación.

ORAB: Efectúa la operación OR entre el contenido del acumulador B y el contenido de la memoria, el resultado se coloca en B. Cada bit de A después de la operación es la lógica OR entre el correspondiente bit de la memoria y el acumulador B antes de la operación.

ORCC: Efectúa la operación lógica OR entre el contenido de una localidad de memoria y el contenido del CCR, el resultado es colocado en CCR. Después de que la operación es efectuada cada bit de CCR es la lógica OR de los correspondientes bits de la memoria y de CCR antes de la operación.

Los bits del CCR que corresponden a 0s de la mascara no cambian por una operación ORCC.

Instrucciones para Borrar, Complemento y Segundo Complemento

Las instrucciones de borrar (clear), complemento y Negate, efectúan una operación binaria específica en un valor de un acumulador o en memoria.

CLC: Borra el contenido del bit C del registro CCR.
CLI: Borra el bit I del registro CCR.
CLR: Borra el una localidad de memoria especificada en la instrucción.
CLRA: Borra el contenido del acumulador A.
CLRB: Borra el contenido del acumulador B.
CLV: Borra el bit V del registro CCR.
COM: Realiza el uno complemento a una localidad de memoria.
COMA: Realiza el uno complemento al acumulador A.
COMB: Realiza el uno complemento al acumulador B.
NEG: Realiza el dos complemento a una localidad de memoria.
NEGA: Realiza el dos complemento al acumulador A.
NEGB: Realiza el dos complemento al acumulador A.

Instrucciones de Multiplicación

EMUL: Un valor de 16 bits sin signo es multiplicado por otro valor de 16 bits sin signo para producir in valor de 32 bits sin signo. El primer operando se carga en el acumulador D y el segundo en el registro índice Y. Los 16 bits mas altos de el resultado de 32 bits son colocados en Y y los 16 bits de menor peso son colocados en D.
EMULS: Un valor de 16 bits con signo es multiplicado por otro valor de 16 bits con signo para producir in valor de 32 bits sin signo. El primer operando se carga en el acumulador D y el segundo en el registro índice Y. Los 16 bits mas altos de el resultado de 32 bits son colocados en Y, y los 16 bits de menor peso son colocados en D.

MUL: Multiplica un valor de 8 bits sin signo en el acumulador A por otro valor de 8 bits sin signo dentro del acumulador B. el resultado de 16 bits es colocado en el registro D. Los 8 bits de mayor peso se colocan en A y los 8 de menor peso se colocan en B.

Instrucciones de División

EDIV: Divide un valor de 32 bits sin signo del dividendo, los 16 bits de mayor peso se colocan en el registro Y, y los otros restantes en el acumulador D por un valor de 16 bits del divisor colocados en el registro X. Produce un cociente de 16 bits que se coloca en Y, y un residuo de 16 bits que se coloca en D.
EDIVS: Divide un valor de 32 bits con signo del dividendo, los 16 bits de mayor peso se colocan en el registro Y, y los otros restantes en el acumulador D por un valor de 16 bits del divisor colocado en el registro X. Produce un cociente de 16 bits que se coloca en Y, y un residuo de 16 bits que se coloca en D.
FDIV: Divide un valor de 32 bits con signo del dividendo por un valor de 16 bits del divisor. Produce un cociente y un residuo de 16 bits.
IDIV: Divide un valor de 16 bits sin signo del dividendo colocados en el acumulador D por un valor de 16 bits del divisor colocado en el registro X. Produce un cociente de 16 bits en X y un residuo de 16 bits en D.
IDIVS: Efectúa una división con un numero entero de 16 bits con signo en el doble acumulador D como numerador entre un valor de 16 bits con signo colocados en el registro índice X como denominador. Produce un cociente de 16 bits en X y un residuo de 16 bits en D.

Instrucciones de Prueba de Bits y Manipulación de Operaciones.

Las instrucciones de prueba de bits y manipulación de operaciones usan un valor de mascara para probar o cambiar el valor de un bit de forma individual colocado en algún acumulador o en memoria. Un valor de mascara es un byte cuyos bits en nivel 1 corresponden a los bits que se quieren poner a cero de la localidad de memoria.

Instrucciones de prueba de bits como BITA o BITB operan sin modificar el valor de otro operando.

BCLR: Pone en ceros los bits especificados de una localidad de memoria.
BITA: Comprueba si determinados bits del acumulador A están activos o no. Esta instrucción efectúa una operación lógica AND entre el contenido del acumulador y una localidad de memoria. No se altera ni el contenido del acumulador A ni el de memoria.
BITB: Comprueba si determinados bits del acumulador B están activos o no. Esta instrucción efectúa una operación lógica AND entre el contenido del acumulador y una localidad de memoria. No se altera ni el contenido del acumulador A ni el de memoria.
BSET: Pone en unos los bits especificados de una localidad de memoria.

Instrucciones de Desplazamiento y Rotación.

Existen desplazamientos y rotación para todos los acumuladores y localidades de memoria.

Instrucciones de Desplazamientos Lógicos

LSL: Desplaza todos los bits de una localidad de memoria una posición hacia la izquierda. El bit 0 es cargado con 0, y el bit mas significativo de la memoria se almacena en el bit C del registro de estado.
LSLA: Desplaza todos los bits del acumulador A una posición hacia la izquierda. El bit 0 es cargado con 0, y el bit mas significativo de A se carga en el bit C del registro de estado.
LSLB: Desplaza todos los bits del acumulador B una posición hacia la izquierda. El bit 0 es cargado con 0, y el bit mas significativo de B se carga en el bit C del registro de estado.
LSLD: Desplaza todos los bits del acumulador D una posición hacia la izquierda. El bit 0 es cargado con 0, y el bit mas significativo de A se carga en el bit C del registro de estado.

LSR: Desplaza todos los bits de una localidad de memoria una posición hacia la derecha. El bit 7 es cargado con 0, y el bit menos significativo de la memoria se almacena en el bit C del registro de estado.

LSRA: Desplaza todos los bits del acumulador A una posición hacia la izquierda. El bit 7 es cargado con 0, y el bit menos significativo de la memoria se almacena en el bit C del registro de estado.

LSRB: Desplaza todos los bits del acumulador B una posición hacia la izquierda. El bit 7 es cargado con 0, y el bit menos significativo de la memoria se almacena en el bit C del registro de estado.

Instrucciones de Desplazamientos Aritméticos.

Estas instrucciones son similares a los corrimientos lógicos, solo que estas, no se incluye el bit 7 para de esta manera no afectar el signo del dato.

ASL: Efectúa la mismo que la instrucción LSL.
ASLA: Efectúa la mismo que la instrucción LSLA.
ASLB: Efectúa la mismo que la instrucción LSLB.
ASLD: Efectúa la mismo que la instrucción LSLD.
ASR: Desplaza todos los bits de una localidad de memoria hacia la derecha. El bit 7 permanece constante y el bit 0 se almacena en el bit C del registro de estado.
ASRA: Desplaza todos los bits del acumulador A hacia la derecha. El bit 7 permanece constante y el bit 0 se almacena en el bit C del registro de estado.
ASRB: Desplaza todos los bits del acumulador B hacia la derecha. El bit 7 permanece constante y el bit 0 se almacena en el bit C del registro de estado.

Instrucciones de Rotación.

ROL: Rota todos los bits de una localidad de memoria una posición hacia la izquierda. El bit 0 se carga con el bit C del registro de estado y el bit mas significativo de la memoria se almacena en el bit C del registro de estado.

ROLA: Rota todos los bits del acumulador B una posición hacia la izquierda. El bit 0 se carga con el bit C del registro de estado y el bit mas significativo de la memoria se almacena en el bit C del registro de estado.

ROLB: Rota todos los bits del acumulador A una posición hacia la izquierda. El bit 0 es cargado con 0, y el bit mas significativo de la memoria se almacena en el bit C del registro de estado.

ROR: Desplaza todos los bits de una localidad de memoria hacia la derecha. El bit 7 se carga con el bit C del registro de estado y el bit menos significativo de la memoria se almacena en el bit C del registro de estado.

RORA: Desplaza todos los bits del acumulador A hacia la derecha. El bit 7 se carga con el bit C del registro de estado y el bit menos significativo del acumulador A se almacena en el bit C del registro de estado.

RORB: Desplaza todos los bits del acumulador B hacia la derecha. El bit 7 se carga con el bit C del registro de estado y el bit menos significativo del acumulador B se almacena en el bit C del registro de estado.

Instrucciones de Control BRANCH.

Instrucciones de Salto Corto

Las instrucciones de Branch operan de la siguiente forma: Cuando una condición esperada es encontrada un corrimiento (Offset) de 8 bits con signo es agregado al contador de programa (PC) por lo que es programa se ejecuta a partir de la nueva dirección.

El rango del salto es de 8 bits, es decir desde -128 hasta 127 bits a partir de la instrucción BRANCH.

BRA: Bifurcación incondicional hacia una dirección especificada.
BRN: Bifurcación deshabilitada.

Bifurcaciones Simples

BCC: Bifurcar si el bit C (acarreo) del CCR esta en cero.
BCS: Bifurcar si el bit C (acarreo) del CCR esta en uno.
BEQ: Bifurcar si el resultado de una operación a sido 0 (Z=1).
BMI: Bifurcar si el resultado de una operación es negativo (N=1).
BNE: Bifurcar si el resultado de una operación no es igual (Z=0).
BPL: Bifurcar si el resultado de una operación es positivo (N=0).
BVC: Bifurcar si el bit V (overflow) del CCR esta en 0 (V=0).
BVS: Bifurcar si el bit V (overflow) del CCR esta en 1 (V=1).

Bifurcaciones Sin Signo

BHI. Bifurcar si el resultado de una operación es mayor.
BHS: Bifurcar si el resultado de una operación es mayor o igual.
BLO: Bifurcar si el resultado de una operación es menor.
BLS: Bifurcar si el resultado de una operación es menor o igual.

Bifurcaciones Con Signo

BGE: Bifurcar si el resultado de una operación es mayor o igual.
BGT: Bifurcar si el resultado de una operación es mayor que.
BLE: Bifurcar si el resultado de una operación es menor o igual.
BLT: Bifurcar si el resultado de una operación es menor.

Instrucciones de Salto Largo

Las instrucciones Long Branch operan de la siguiente forma: Cuando una condición esperada es encontrada un corrimiento (Offset) de 16 bits con signo es agregado al contador de programa (PC) por lo que es programa se ejecuta a partir de la nueva dirección.

El rango del salto es de 16 bits, es decir desde -32768 hasta 32767 bits a partir de la instrucción BRANCH.

LBRA: Bifurcación incondicional hacia una dirección especificada.
LBRN: Bifurcación deshabilitada.

Bifurcaciones Simples

LBCC: Bifurcación Larga si el bit C (acarreo) del CCR esta en cero.
LBCS: Bifurcación Larga si el bit C (acarreo) del CCR esta en uno.
LBEQ: Bifurcación Larga si el resultado de una operación a sido 0 (Z=1).
LBMI: Bifurcación Larga si el resultado de una operación es negativo (N=1).
LBNE: Bifurcación Larga si el resultado de una operación no es igual (Z=0).
LBPL: Bifurcación Larga si el resultado de una operación es positivo (N=0).
LBVC: Bifurcación Larga si el bit V (overflow) del CCR esta en 0 (V=0).
LBVS: Bifurcación Larga si el bit V (overflow) del CCR esta en 1 (V=1).

Bifurcaciones Sin Signo

LBHI. Bifurcación Larga si el resultado de una operación es mayor.
LBHS: Bifurcación Larga si el resultado de una operación es mayor o igual.
LBLO: Bifurcación Larga si el resultado de una operación es menor.
LBLS: Bifurcación Larga si el resultado de una operación es menor o igual.

Bifurcaciones Con Signo

LBGE: Bifurcar si el resultado de una operación es mayor o igual.
LBGT: Bifurcar si el resultado de una operación es mayor que.
LBLE: Bifurcar si el resultado de una operación es menor o igual.
LBLT: Bifurcar si el resultado de una operación es menor.

Instrucciones de Condición de Bits.

Las bifurcaciones de Prueba (condición) de bit prueban un bit en específico de una localidad de memoria. Un operando de mascara se utiliza en esta instrucción para probar la condición del bit seleccionado.

Si todos los bits de la localidad de memoria corresponden a los bits activos de la mascara se ejecuta la bifurcación.

El rango del salto es de 8 bits, es decir desde -128 hasta 127 bits a partir de la instrucción BRANCH.

BRCLR: Bifurca si los bits seleccionados por el operando mascara están nivel 0 lógico.
BRSET: Bifurca si los bits seleccionados por el operando mascara están en nivel 1 lógico.

Instrucciones de Ciclo Primitivas.

Estas instrucciones pueden ser utilizadas para bifurcaciones mediante contadores. La instrucción prueba un valor colocado en un acumulador (A, B, D, X,Y, o SP) y bifurca si es 0 o diferente de 0 dependiendo de la condición de la bifurcación.

Estas instrucciones tienen un rango de ejecución de 9 bits, desde -256 hasta 255 bits desde la dirección donde se encuentra la instrucción hasta la nueva dirección.

DBEQ: Decrementa un contador y bifurca si es igual a 0.
DBNE: Decrementa un contador y bifurca si es diferente de 0.
IBEQ: Incrementa el contador y Bifurca si es igual a 0.
IBNE: Incrementa el contador y Bifurca si es igual a diferente de 0.
TBEQ: Evalúa el contador y bifurca si es igual a 0.
TBNE: Evalúa el contador y bifurca si es diferente de 0.

Instrucciones de Jump y Subrutinas.

La instrucción Jump (JMP) carga en el contador del programa (PC) un valor dentro de los 64Kbytes de memoria del microcontrolador, dirección en la cual se continúa ejecutando el programa. La dirección provista puede ser de 16 bits o puede determinarse por varias formas de direccionamiento indexado.

Las instrucciones de subrutina (BSR) permiten el proceso de transferencia de control a un segmento de código que efectúa una tarea determinada. Al ejecutarse una subrutina se guarda la dirección, y después principia la ejecución del programa en la dirección provista.

Las subrutina dentro del mapa de 64Kbytes de memoria son finalizadas con un retorno desde subrutina (RTS), después de este retorno se continua la ejecución de la siguiente instrucción al BSR o JSR.

La instrucción de llamado a subrutina CALL es utilizada para expandir memoria. La instrucción CALL almacena el valor del registro PPAGE y la dirección de retorno, entonces escribe un valor nuevo de PPAGE para seleccionar un nuevo mapa de memoria donde reside la subrutina.

El retorno de una llamada a subrutina RTC es usado para terminar una subrutina en modo expandido. Después de un RTC, el programa ejecuta la siguiente dirección a la instrucción CALL.

BSR: Bifurcación hacia una subrutina.
CALL: Llamado a una subrutina en memoria expandida
JMP: Salto a una dirección.
JSR: Salto a una subrutina.
RTC: Retorno desde un llamado a subrutina.
RTS: Retorno desde una a subrutina.

Instrucciones de Interrupción

Las instrucciones de interrupción facilitan la transferencia del control a una subrutina que evalúa una tarea crítica.

RTI: Retorno desde interrupción.
SWI: Interrupción pos software.

Instrucciones de Código de Condición

Las instrucciones del código de condición son formas especiales instrucciones matemáticas y transferencia de datos que pueden ser utilizadas para cambiar el CCR. A continuación se muestran estas instrucciones.

ANDCC: Realiza una operación lógica ADN entre el CCR y una localidad de memoria.
CLC: Borra el contenido del bit C (C=0).
CLI: Borra el contenido del bit I (I=0).
CLV: Borra el contenido del bit V (V=0).
ORCC: Realiza una operación lógica OR entre el CCR y una localidad de memoria.
PSHC: Coloca el contenido del CCR en el puntero de pila.
PULC: Retira el CCR del puntero de pila.
SEC. Activa el bit C (C=1).
SEI: Activa el bit I (I=1).
SEV: Activa el bit V (V=1).
TAP. Transfiere el acumulador A al CCR.
TPA: Transfiere el contenido del CCR al acumulador A.

Instrucciones Stop y Wait

Estas dos instrucciones ponen el CPU del microcontrolador en un estado de bajo consumo de energía

La instrucción STOP almacena el contenido de la dirección de retorno y el contenido de los registros del CPU y de los acumuladores. Después detiene todos los relojes del sistema.

La instrucción WAIT almacena el contenido de la dirección de retorno y el contenido de los registros del CPU y de los acumuladores. Después espera la solicitud de una rutina de servicio de interrupción. Algunas veces la señal de reloj continúa corriendo.

STOP: Detiene el CPU.
WAIT: Espera una interrupción.