Universidad Nacional Autónoma de México

Escuela Nacional de Estudios Profesionales "ARAGON"

PROGRAMACION DE SISTEMAS BASADOS EN MICROPROCESADORES

T E S I S

Que para obtener el título de

INGENIERO EN COMPUTACION

p r e s e n t a

ROBERTO MARTINEZ GONZALEZ

México, D. F. *1989*

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

TABLA DE CONTENIDO

PRESENTACION.

El presente trabajo muestra una metodología para el desarrollo de programas en microprocesadores de 8 bits. La metodología está basada en las técnicas de ingeniería de software.

El documento está dividido en las siguientes partes:

CONCEPTOS GENERALES SOBRE MICROPROCESADORES. Esta sección describe un conjunto de conceptos fundamentales para el entendimiento de la filosofía del funcionamiento de microprocesadores. Trata aspectos, fundamentalmente, del hardware.

TECNICAS DE PROGRAMACION PARA MICROPROCESADORES. En este capítulo se explican los conceptos y términos más importantes del software. Y aparece una breve crítica de las tendencias con mayor difunción en el desarrollo de programas.

ANALISIS DE UNA APLICACION PRACTICA. Se presenta, con base a un sistema real, el desarrollo e implementación de los conceptos estudiados en los acpítulos anteriores.

1. INTRODUCCION.

En la época actual, se está viviendo en México, así como en la mayor parte del mundo, la era de la **microcomputación**, es decir, la aplicación de los recursos computacionales reducidos a un tamaño físico muy pequeño. Es la época de los **Microprocesadores** y de las **Microcomputadoras**. Estos son términos que comunmente se utilizan como sinonimos y que realmente son conceptos diferentes.

Con el objeto de entender los conceptos de **Microprocesador** y de **Microcomputadora** se partirá del concepto de lo què es una **computadora digital**.

Las computadoras digitales, escencialemente, están conformadas por dispositivos electrónicos interconectados entre si denominados **HARDWARE (HW)** y que a través de un conjunto de órdenes que estos dispositivos realizan al pie de la letra y cuantas veces sea necesario llamadas **SOFTWARE (SW)** permiten le ejecución de un sinómero de funciones encaminadas hacia la realización de una actividad concreta.

En términos generales las computadoras digitales aceptan información del exterior a través de sus dispositivos de entrada (teclados, unidades de disco o cinta, lectoras opticas, etc.), procesan esta información de acuerdo a la secuencia de un conjunto de órdenes **(SW)** previamente almacenadas; proporcionando los resultados generados por el proceso a través de los dispositivos de salida (impresoras, displays, unidades de disco o cinta, etc.).

En la figura 1.1 se muestra la estructura general de una computadora digital.

Como se puede observar la máquina esta constituida por una Unidad de Entrada (**INPUT UNIT-IU**), una Unidad Central de Proceso (**CENTRAL PROCESSOR UNIT-CPU**), una Unidad de Memoria Principal (**MAIN MEMORY UNIT-MMU**) y una Unidad de Salida (**OUTPUT UNIT-OU**).

1.1. CPU CENTRAL PROCESSOR UNIT.

La CPU o procesador central es el elemento en el cual se realizan todas las operaciones indicadas por los programas sobre los datos. (Los programas y datos deben ser previamente colocados en la memoria principal por medio de los dispositivos de entrada).

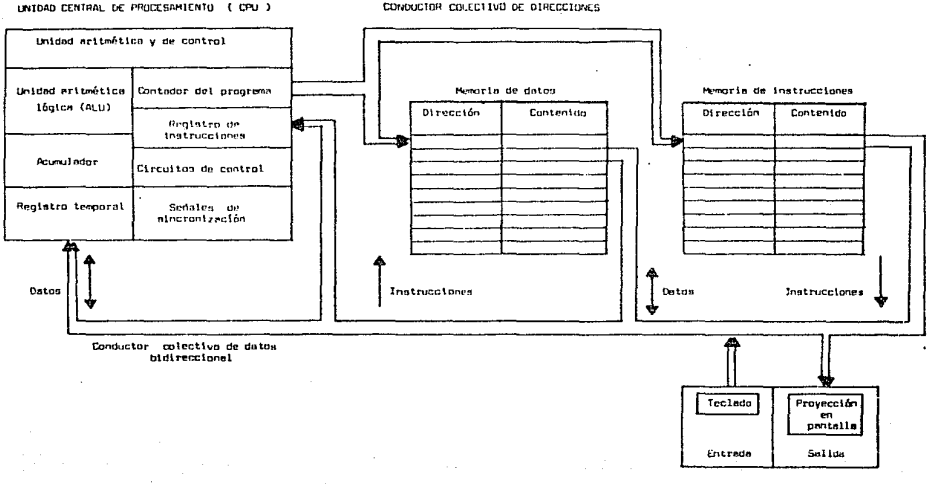


FIGURA 1.1. ESTRUCTURA GENERAL DE UNA COMPUTADORA DIGITAL.

1.2. INPUT / OUTPUT UNITS.

Los dispositivos de Entrada y Salida (**I/O**) son la parte de la computadora con la que el usuario interactua directamente. Es a través de estas unidades que el usuario puede introducir información a la Unidad de Memoria o recuperar los resultados totales o parciales de algún proceso.

El siguiente diagrama (figura 1.2) muestra la manera de conectar un dispositivo de I/O a la computadora:

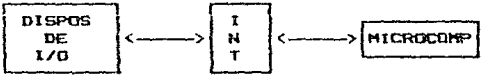


FIGURA 1.2. DIAGRAMA A BLOQUES DE LA CONEXION DE UN DISPOSITIVO PERIFERICO A LA COMPUTADORA.

El bloque etiquetado como DISP DE I/O (**Dispositivo de I/O**) representa a todos aquellos elementos que están directamente en contacto con el usuario o con el fenómeno y también se les llama **Periféricos.**

La **interfase** (INT) es un elemento que hace compatibles las señales generadas por el Dispositivo Periférico y las señales que puede "interpretar" la computadora (MICROCOMP) y viceversa.

Dentro de la interfase existen varios elementos, entre los que se encuentran unos registros que son de donde la CPU lee/escribe los datos generados/recibidos por el dispositivo. A estos registros se les denomina **PUERTOS,** los cuales son seleccionados (o **direccionados**) por el microprocesador a través del bus de direcciones (el concepto de bus será tratado más adelante en este trabajo).

1.3. UNIDAD DE MEMORIA PRINCIPAL.

La **Unidad de Memoria Principal** (Main Memoy Unit – MMU) en una microcomputadora es usada para almacenar programas y datos.

Los programas son almacenados en un conjunto de paquetes de **BITS** (bit = digito binario), a cada uno de estos paquetes se le denomina **PALABRA** o Contenido de Localidad de Memoria; y la cantidad de bits en cada una se llama **LONGITUD DE PALABRA**; esta puede variar entre 4 y 32 BITS, según el microprocesador utilizado.

Se puede pensar en la MMU como una matriz de N renglones y M columnas, donde los renglones corresponden a las direcciones y las columnas al contenido de dichas direcciones.

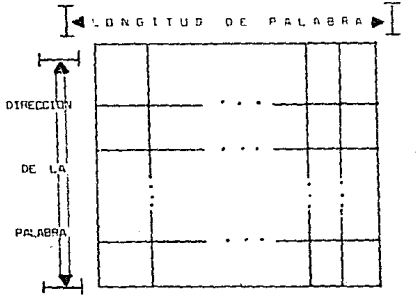


FIGURA 1.3. REPRESETACION DE LA MEMORIA PRINCIPAL DE UNA COMPUTADORA DIGITAL.

En esta figura existen N localidades con una longitud de palabras de M BITS.

DIRECCION DE MEMORIA : es el nombre único que se le da a cada uno de los diferentes renglones en que se divide a la MMU.

PALABRA Y CONTENIDO DE UNA LOCALIDAD DE MEMORIA: son terminos sinònimos.

LONGITUD DE PALABRA: es la cantidad de bits que contiene cada palabra.

Hay que hacer notar que las instrucciones y los datos se almacenan ambos como palabras de bits de tal manera que el procesador no puede distinguir autònomamente si el contenido de una direcciòn de memoria es un dato o una instrucciòn. Por esta razòn es que se debe tener cuidado de organizar adecuadamente la memoria del sistema: un àrea para datos y otra àrea diferente para programas.

Establecido lo anterior, se està en condiciones de definir los conceptos **MICROPROCESADOR** Y **MICROCOMPUTADORA.**

Una **MICROCOMPUTADORA (MC)** es una computadora digital cuya CPU (Central Preccessing Unit) esta estructurada en un sòlo circuito integrado (anteriormente era en varios) denominado **MICROPROCESADOR (MP)**, asi pues, se concluye que un microprocesador es una parte (la màs importante) del sistema, que en este caso es la microcomputadora.

En este trabajo se hablarà de los microprocesadores y de su programaciòn debiendose entender a este como el elemento principal de una microcomputadora y no como un elemento aislado, ya que por si solo no seria de utilidad alguna.

2. CONCEPTOS GENERALES SOBRE MICROPROCESADORES.

Dentro del àmbito de la tecnologìa se han logrado grandes avances en el desarrollo de los SEMICONDUCTORES, a partir de los cuales se fabrican los CIRCUITOS INTEGRADOS, los que a su vez contienen arreglos normalizados de circuitos lògicos, a estos elementos se les denomina comunmente "**CHIPS**".

Los chips comenzaron conteniendo ùnicamente algunas compuertas lògicas y se les denominò de baja escala de integraciòn (**Small Scale Integration - SSI**), posteriormente se estructuraron arreglos de compuertas para formar decodificadores, registros, etc. a esta etapa se le llamò de mediana escala de integraciòn (**Medium Scale Integration MSI**) posteriormente a estos aparecen los chips de alta escala de integraciòn (**Large Scale Integration LSI**) en los cuales se estructuran ALU's memorias de tipo RAM, etc. (aquì aparecen algunos de los primeros microprocesadores).

Asì la tecnologìa sigue avanzando hasta llegar a los circuitos de muy alta escala de integraciòn (**Very Large Scale Integration VLSI**), a la cual pertenecen los microprocesadores, microcontroladores e inclusive microcomputadoras enteras en un solo chip. Es decir, en el mismo circuito integrado estàn contenidos los puertos de E/S la CPU y la Unidad de Memoria Principal.

La limitante principal en cuanto a la complejidad a la que se puede llegar en la integraciòn de circuitos digitales en un solo chip son las conexiones externas del mismo, ya que como su àrea es muy reducida no puede tener demasiadas terminales de conexiòn para recibir o enviar señales.

Es gracias a la evoluciòn en la tecnologìa de semiconductores y a la visiòn de un grupo de diseñadores de INTEL que en 1971 aparece el primer microprocesador, el 4004, el cual se desarrollò para un fabricante japonès de calculadoras de escritorio.

Este nuevo elemento causò una verdadera revoluciòn en la industria electrònica ya que los ingenieros de diseño acostumbrados a pensar en tèrminos ùnicamente de hardware (dispositivos electrònicos) tienen que pensar ahora tambièn en tèrminos de software (programas).

Al principio no se visualizaba la trascendencia que tendrìa el uso de los MICROPROCESADORES, sin embargo, ràpidamente se asimilaron las ventajas que traerìan estos nuevos elementos, entre las cuales caben destacar las siguientes:

- Sistemas de procesamiento de informaciòn màs flexibles.

- Introducción de nuevas opciones en los sistemas ya establecidos.

- Modularidad en el diseño.

- Reducción de espacio y componentes.

- Menos consumo de potencia.

- Gran facilidad en la reparación, el mantenimiento y en el diagnóstico de averías.

- Reducción de costo.

Se debe resaltar el hecho de que los microprocesadores no sólo se utilizan en las microcomputadoras, sino en una gran cantidad de aplicaciones como son: dispositivos periféricos, automóviles, relojes, termómetros, sistemas de alarma, videojuegos, lavadoras, básculas, sistemas de comunicación, etc.

2.1. ARQUITECTURA INTERNA DE LOS MICROPROCESADORES.

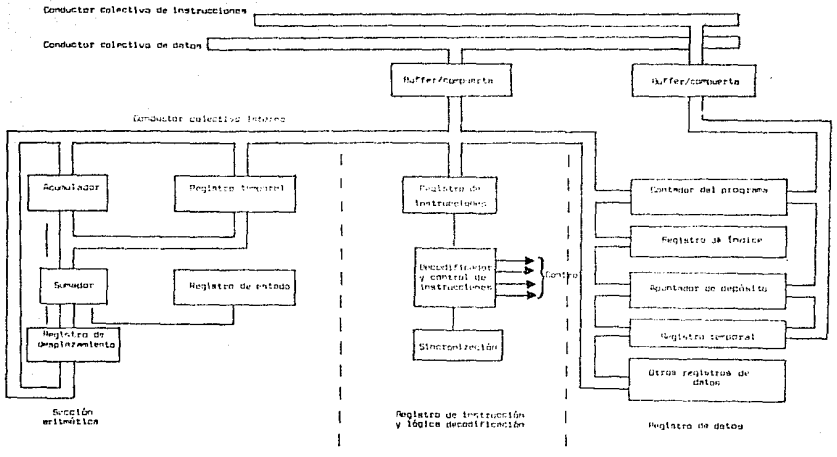
Se denomina arquitectura interna de un microporcesador a la organización de sus elementos integrales. Esta es definida por el diseñador del mismo y no puede ser modificada por el usuario.

Cada fabricante ha desarrollado una arquitectura interna diferente de sus microprocesadores y aunque pueden realizar generalmente las mismas actividades la manera en que las llevan a cabo es diferente. Sin embargo, es factible llegar a un modelo generalizado de la arquitectura interna de un microprocesador de 8 bits.

A los microprocesadores se les clasifica de 4, 8, 12, 16, 32, etc. bits, indicando este número la cantidad de bits del dato que el procesador puede manejar simultáneamente (en paralelo).

En la figura 2.1 se muestra la arquitectura interna típica de un microprocesador de 8 bits.

Como se puede observar contiene una unidad para procesar operaciones (**ARITMETHIC AND LOGIC UNIT - ALU**), un decodificador de instrucciones, un conjunto de registros para almacenamiento de datos temporales (**REGISTROS INTERNOS**) y una unidad de control y temporización. Todas estas unidades están unidas internamente a través de un bus común (un bus es un conjunto de conductores a través de los cuales se transmiten las señales eléctricas) denominado **BUS INTERNO**. Este no es el mismo que los buses de datos,



FIUGURA 2.1. ARQUITECTURA INTERNA DE UN MICROPROCESADOR DE * 8 * BITS.

direcciones y control los cuales son externos y se tratarán más adelante.

2.1.1. ALU. Unidad Aritmética y Lógica (Aritmethic and Logic Unit - ALU). Esta unidad esta constituida por una red de circuitos lógicos que permite realizar básicamente las siguientes tareas:

- Sumas binarias.
- Operaciones Logicas (AND, OR Y NOT).
- Transferencias y desplazamientos.

2.1.2. AMU. Unidad de Memoria Auxiliar (Auxiliar Memory Unit - AMU). A esta unidad se le suele denominar también conjunto de Registros Internos del Procesador. Su objetivo es el de presentar un medio de alojamiento temporal en donde se puedan manipular rápida y directamente los datos e instrucciones que se van a utilizar y ejecutar inmediatamente.

No se debe confundir la unidad de memoria principal (MMU) con la auxiliar (AMU); la primera es para uso del programador o usuario y la segunda para uso del propio procesador. Sin embargo, a través de técnicas que se tratarán más adelante es posible tener acceso a estos registros internos.

Los microprocesadores de cada fabricante, generalmente, tienen una cantidad diferente de registros internos, sin embargo en general, todos contienen los siguientes registros fundamentales:

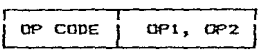
- Contador del programa (PC)
- Registro de instrucción (IR)
- Acumulador (ACC)
- Indice (I)
- De condiciones o de banderas (CR)
- Apuntador de Pila (SP)
- Próposito general.

Antes de describir cada uno de estoa elementos se presenta la explicación del término instrucción.

Una **instrucción** es un patrón de Bits que se alojan en la memoria principal y sirve para indicarle a la CPU la

secuencia de actividades a realizar. Este código binario o patrón de bits debe estar disponible en las entradas de datos del microprocesador en el preciso momento en el que éste espera una instrucción para interpretarla.

El formato general que debe tener una instrucción para que pueda ser procesada en la CPU es el siguiente:



Donde el OP CODE (código de operación) indica las operaciones que se deben realizar sobre los operandos (OP1 y OP2). Una instrucción puede tener 0, 1 ó 2 operandos dependiendo del tipo de instrucción.

a) **CONTADOR DEL PROGRAMA** (Program Counter - PC). En este registro se almacena la dirección de la siguiente instrucción a ser ejecutada. Se incrementa automáticamente de modo tal que siempre contiene la dirección de memoria cuyo contenido se interpretará como la siguiente instrucción a ejecutar.

A menos que se encuentre con una instrucción de salto el programa en memoria se ejecutará tal como se encuentra almacenado, es decir, **secuencialmente.** La longitud del PC es igual a la longitud del bus de direcciones, ya que ambos manejan precisamente direcciones.

Cuando se ejecuta una instrucción de salto o bifurcación el contenido del PC es cambiado por la dirección donde se encuentra la nueva instrucción a ejecutar.

En la siguiente figura se ejemplifica el funcionamiento del PC:

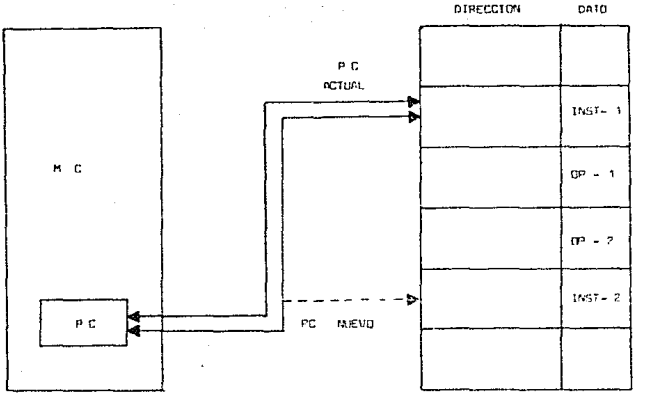


FIGURA 2.2. FUNCIONAMIENTO DEL PC.

b) REGISTRO DE INSTRUCCION (INSTRUCTION REGISTER IR). En el registro de instrucción unicamente se carga el **Código de Operación** (Operation Code- OP CODE), de la instrucción a ejecutarse y no toda la instrucción propiamente dicha.

INSTRUCCION DE UN MICROPROCESADOR

Código de Operación	Operando(s)
Se carga al IR	**Se carga en el ACC ó en los Registros Auxiliares.**

Una vez cargado el IR, su contenido sirve para generar las señales de temporización y secuenciación necesarias para la ejecución de la instrucción.

c) REGISTRO ACUMULADOR (ACCUMULATOR – ACC). En todas las oparaciones aritméticas, lógicas, de corrimiento y transferencia, un operando esta contenido en el acumulador.

Por lo general es el registro màs utilizado; por lo tanto, en los microprocesadores que tienen un sòlo acumulador se necesitan realizar mayor cantidad de operaciones para una tarea determinada a diferencia de los microprocesadores con varios acumuladores.

d) REGISTRO INDICE (INDEX REGISTER - IR). Dependiendo del microprocesador pueden existir uno o màs registros de este tipo.

Si se quiere hacer referencia a una determinada dirección no es necesario expresarla enteramente en la instrucción. Basta con que en ésta aparezca el desplazamiento al cual se le sumarà el contenido del registro índice (dirección base) para obtener la dirección efectiva.

DIRECCION BASE + DESPLAZAMIENTO = DIRECCION EFECTIVA

Se utiliza para poder realizar las instrucciones en el modo de direccionamiento indexado, el cual consiste en hacer referencia a una localidad de memoria en dos partes (los modos de direccionamiento se explican màs adelante en este trabajo).

e) REGISTRO DE CONDICION (CONDITION REGISTER - CR). También llamado **registro de banderas.**

Este registro interno es uno de los màs importantes, ya que es el que permite conocer el estado del microprocesador después de ejecutada cada instrucción y por lo tanto es posible tomar decisiones.

Cada fabricante codifica el CR de diferente manera pero para fines prácticos aqui se presenta como ejemplo el CR del microprocesador 6809:

b0	b1	b2	b3	b4	b5	b6	b7
E	F	H	I	N	Z	Y	C

Se tomarà como ejemplo el bit 5 (z=zero) el cual representa a la bandera "cero".

Si el contenido de este registro originalmente era $00 y después de la operación aritmética es $02, se concluye entonces que el resutado de dicha operación resultò igual a cero, ya que el bit 5 se activò (se colocò a "1" logico).

$00
0 0 0 0 0 0 0 0

Antes de ejecutar la instrucción

$02
0 0 0 0 0 0 1 0

Después de ejecutar la instrucción

Hay que notar que en este registro no se guarda el resultado de la operación sino que nos informa las características del resultado y por lo tanto del microprocesador después de una operación.

Una vez conocidas las características (estado o status) del procesador se puede tomar una desición.

f) **APUNTADOR DE PILA** (STACK POINTER - SP). Antes de describir al SP, es necesario explicar lo que es una "Pila" ó "Stack".

Una **Pila o Stack** es una estructura de datos del tipo LIFO (Last-In First Out), esto quiere decir que el primer dato que entra a la pila es el último que sale y el último dato que es depositado es el primero en salir.

Con la siguiente analogía se puede comprender mejor la estructura del STACK. Imaginemos un recipiente común, un vaso por ejemplo, dentro del cual se depositarán un conjunto de monedas.

Para poder recuperar la moneda número dos es necesario sacar antes las monedas 4 y 3, en ese orden.

Ahora, si se introdujera otra moneda esta ocuparía el lugar número 5, es decir, que hasta este momento el siguiente lugar disponible es el 5.

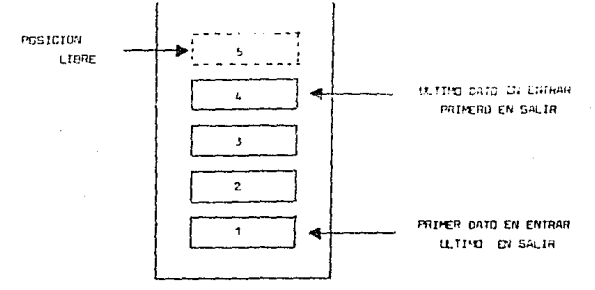


FIGURA 2.3. ILUSTRACION DEL FUNCIONAMIENTO DE UN STACK.

El SP es un registro que indica el lugar (dirección de memoria) que ocuparía la moneda (dato), dentro del vaso (memoria principal).

Concluyendo, el SP apunta a la siguiente localidad de memoria libre dentro del Stack.

El Area de Stack dentro de las microcomputadoras es un área de memoria principal definida por el usuario o por el sistema.

g) **REGISTROS DE PROPOSITO GENERAL.** La cantidad de estos registros varía de cero en adelante, dependiendo del microprocesador de que se trate.

Su aplicación es la que el programador quiera dar, es decir, puede almacenar resultados, direcciones, etc.

Un uso adecuado de estos registros redunda en programas más eficientes y por lo tanto el sistema en general también lo será.

2.1.3. UNIDAD DE CONTROL (CONTROL UNIT - CU). Esta unidad es la que controla las actividades de todas las demás partes antes mencionadas, por ejemplo, una suma no se realiza si no fue ordenada por la CU. Pero la CU no actua autonómamente, sino que para poder marcar la secuencia de antividades que debe controlar, antes se le debió indicar (**PROGRAMAR**) qué es lo que va a ordenar que se realice; esto es a lo que se le denominan **PROGRAMACION** o SOFTWARE y lo realiza el ser humano.

Por otro lado, la velocidad a la que se realizan las actividades definidas por el programa y controladas por la CU esta dada por la **Unidad de Temporizacion** (TIMING UNIT - TU) la cual esta constantemente generando una señal a una determinada frecuencia que es a la que trabajará la computadora.

2.2. CICLOS DEL PROCESADOR.

Como se mencionó anteriormente, la CPU es la parte del microprocesador que coordina todas las actividades que se deben realizar para que la información de entrada se procese adecuadamente a partir de la ejecución de un programa almacenado.

A su vez la CU está en función directa de las señales que la activan, es decir, las señales de "reloj" que son generadas por la unidad de temporización (TU).

En el momento de energizar a un microprocesador, èste inmediatamente toma el contenido de la localidad de memoria, apuntada por el PC, como una instrucciòn y realiza todas las operaciones indicadas por ella, posteriormente vuelve a realizar esta actividad repetidas veces.

Cuando se pone a funcionar a una microprocesador la ùnica actividad que realiza de una manera general es tomar un contenido de memoria interprètarlo como una instrucciòn y ejecutar la misma, esto sucede en forma repetititva hasta que se desenergiza o se interrumpe al microprocesador. La velocidad a la que se repiten estos ciclos esta determinada por la TU.

De lo anterior se concluye que la actividad que realizan en comòn todos las microprocesadores es:

- Tomar instrucciones y operandos de la MMU y

- Ejecutarlas

Estas dos actividades se realizan invariablemente no importando la complejidad del programa a ejecutar.

Cada instrucciòn nocesita diferentes cantidades de tiempo para ejecutarse totalmente. Este lapso de tiempo depende de la arquitectura interna del microprocesador y se mide en ciclos de reloj, un **ciclo de reloj o estado** T es un perfodo completo generado por la unidad de temporizaciòn.

A las operaciones que deben realizarce para ejecutar una instrucciòn completa se les denomina **CICLO DE INSTRUCCION.**

Un ciclo de instrucciòn esta a su vez dividido en los denominados ciclos de màquina, los cuales se constituyen de uno o varios ciclos T.

Los ciclos de màquina realizan sòlo unas cuantas acciones bàsicas, pero la combinacion adecuada de ciclos de màquina genera instrucciones completas, las cuales a su vez, en conjunto, forman programas. Y estos van desde los que realizan una comparaciòn lògica hasta los que controlan y dan "vida" a los robots.

En general existen 6 ciclos de màquina bàsicos, estos son los siguientes:

- Ciclo para la obtenciòn de la instrucciòn.

- Ciclo para realizar una lectura/escritura (R/W) de un operando ò resultado en la MMU.

- Ciclo de lectura/escritura (R/W) para dispositivos de entrada/salida (I/O).

- Ciclo para el manejo del BUS.

- Ciclo para la manipulación de interrupciones.

- Ciclo para indicar la terminación de actividades al microprocesador.

Cada ciclo de instrucción esta conformado de uno a varios ciclos de máquina, dependiendo de la complejidad de la propia instrucción; esto es, una instrucción complicada tardará más tiempo en ejecutarse que alguna más simple.

Para que una instrucción sea ejecutada totalmente tiene que pasar por 3 etapas o fases, las cuales son:

Fase 1) **FETCH**

Fase 2) **DEFER**

Fase 3) **EXECUTE**

A estas 3 fases se les conoce también como ciclos, pero para no confundirlos con los ciclos antes mencionados, en este trabajo se denominarán fases.

En la **fase FETCH**, se obtiene el código de operación de la instrucción a ejecutar. Durante esta primera fase se suceden varios ciclos de máquina:

1) Se copia el contenido de la localidad de memoria indicada por el PC al registro de instrucción (IR).

2) El PC se incrementa una cantidad adecuada de bytes, de modo tal que siempre apunte a la localidad de memoria donde se encuentra la siguiente instrucción a ejecutar.

En la segunda **fase (DEFER)** el contenido del IR es pasado por el decodificador donde se identifica el tipo de instrucción de la que se trata. En otras palabras, en esta fase se seleccionan los ciclos de máquina que se realizarán durante la siguiente fase.

Finalmente en la **fase (EXECUTE)** se realiza la ejecución propiamente dicha de la instrucción, es decir, se ejecutan los ciclos de máquina que le corresponden a la instrucción.

Una vez terminada la fase EXECUTE, automáticamente se inicia la fase FETCH de la siguiente instrucción.

2.3. BUSES DE LOS MICROPROCESADORES.

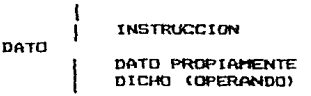
El microprocesador se comunica con el mundo externo (en el ambiente de los microprocesadores se denomina mundo externo a todo lo que no sea el propio chip del microprocesador) a través de 3 canales o buses.

- **El bus de datos.**
- **El bus de direcciones.**
- **El bus de control.**

2.3.1. BUS DE DATOS. Es un conjunto de conductores a través de los cuales se transfiere la información desde y hacia el microprocesador.

La longitud en bits de este Bus, es por lo general, igual a la longitud de palabra de la memoria principal (MMU).

Las instrucciones (códigos de operación) se transfieren por este bus debido a que en el momento de la transferencia el contenido de una localidad de memoria es considerada como un dato.



Al conjunto de bits que se transmiten desde la MMU hacia el microprocesador se les denominará datos, aunque realmente estos bits conformen una instrucción. La diferencia entre si son datos o instrucciones lo que se está transportando lo determina la unidad de control a través de diferentes señales.

2.3.2. BUS DE DIRECCIONES. Es el canal por el cual el microprocesador selecciona la localidad de memoria o el puerto de E/S con el cual va a realizar un intercambio de información.

Los bits alojados en el área de datos se llamarán siempre datos, aunque esten residentes en la memoria o se esten tranfiriendo.

2.3.3. BUS DE CONTROL. Por el bus de control el microprocesador transmite y recibe las señales adecuadas para la sincronización con el mundo externo (HANDSHAKE). Es

decir, si va a colocar un dato sobre la memoria debe indicarle al Chip de memoria correspondiente que le va a enviar un dato y a su vez el circuito debe contestarle que esta listo y en este momento se realiza la transferencia.

El bus de direcciones en lo microprocesadores de 8 Bits es, generalmente, de 16 bits. Con esta cantidad de bits se pueden direccionar directamente hasta 65,536 palabras de memoria (2**16= 64K ó 65 536 palabras).

A través de este mismo bus se seleccionan los puertos de I/O, aunque para ello sólo se utilice una parte de éste. Generalmente se utilizan las 8 líneas de menor peso para direccionar hasta 256 puertos de E/S (2**8 = 256).

Hasta este momento se ha mencionado el bus de direcciones, pero no se ha aclarado su concepto.

El bus de direcciones es un conjunto de conductores a través de los cuales circulan señales binarias (bits), sin embargo, estas señales binarias se deben interpretar como números de localidades de la Memoria Principal (MMU). Por ejemplo si se tienen 4 líneas en el bus de direcciones (A0, A1, A2 y A3), siendo A0 el bit menos significativo (LSB), las cuales tienen los siguientes valores:

(A3	, A2	, A1	, A0)
0	0	1	0

Entonces se accederá (se establecerá el contacto) con la localidad de memoria que le corresponda el número binario 0010. Una vez accesada el microprocesador intercambiará información con esta dirección.

Hay que aclarar que a diferencia del bus de datos, que es bidireccional (tanto la memoria como el microprocesador transmiten y reciben información) el bus de direcciones es unidereccional, esto es, **sólo el microprocesador genera direcciones.**

Como se puede observar en la figura (1.4) el microprocesador esta conectado a la MMU a través de un decodificador.

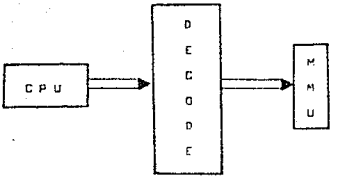


FIGURA 2.4. CONEXION DEL MICROPROCESADOR CON LA UNIDAD DE MEMORIA PRINCIPAL.

2.4. ENTRADA/SALIDA EN LOS MICROPROCESADORES.

La velocidad a la que trabajan los microprocesadores es generalmente mayor a la velocidad a la cual la mayoría de los periféricos envían o reciben información. Por otro lado, las señales que manejan los microprocesadores generalmente no son compatibles con las generadas por los periféricos, luego entonces, se hace necesario que exista un elemento intermedio entre el periférico y el mismo procesador. A este elemento se le denomina **INTERFASE.** Y es con este elemento con el que el microprocesador está realmente en contacto en las operaciones de entrada y salida.

Como se mencionó anteriormente, dentro de las interfases existen unos **registros de propósito especial denominados PUERTOS de entrada y salida.**

Generalmente los puertos son de la misma longitud del bus de datos del microprocesador utilizado ya que en éste el CPU leerá o escribirá información.

Al proceso de leer y/o escribir datos en los puertos de entrada y salida ó inclusive con la unidad de Memoria Principal, se le denomina **TRANSFERENCIA DE INFORMACION.**

2.4.1. MODOS DE TRANSFERENCIA DE INFORMACION

La transferencia de datos entre el microprocesador y cualquier elemento periférico o la memoria principal se puede enmarcar dentro de los siguientes modos:

- Transferencia de datos bajo el control de programa.

- Interrupciones.

- Accesos directos a Memoria.

a) **TRANSFERENCIA DE DATOS BAJO CONTROL DE PROGRAMA.** En este modo de transferencia el programa colocado en la MMU es el que dirige todas las operaciones de lectura de periféricos y escritura en los mismos. Es decir, el microprocesador (comandado por las instrucciones del programa) es el único elemento que esta en posibilidades de habilitar algún periférico, indicando esto que autónomamente ningún periférico habilitará al microprocesador para realizar un intercambio de información.

Dentro de este modo de transferencia existen algunas variantes, dos de las cuales se explican a continuación:

MEMORIA MAPEADA. En este método de intercambio de información los puertos de entrada y salida son tratados como localidades de memoria (esto se lleva a cabo a través de una adecuada interconexión de la interfase con el microprocesador) y por consiguiente es posible manejar y utilizar estos puertos con las instrucciones existentes (las cuales son generalmente muy variadas) para el interambio de información entre memoria principal y procesador.

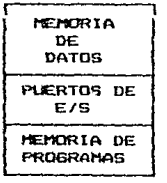


FIGURA 2.5. ORGANIZACION DE LA MEMORIA PRINCIPAL DE UNA MICROCOMPUTADORA QUE UTILIZA MEMORIA MAPEADA.

En la figura anterior se muestra la organización de una Unidad de Memoria Principal de una microcomputadora cuyo CPU utiliza el método de **memoria mapeada** para el intercambio de información con periféricos.

*** E/S AISLADA.** En la configuración aislada de E/S el microprocesador tiene instrucciones adicionales para el manejo de periféricos y cada una de estas instrucciones esta asociada con la dirección de un puerto de E/S. Cuando el microprocesador recibe una instrucción de E/S automáticamente habilita algún componente externo y deshabilita el rango de direcciones de la MMU.

En este método el microprocesador utiliza el mismo sistema de buses para seleccionar localidades de memoria y para puertos de E/S.

La distinción entre una transferencia con la memoria y una transferencia con registros de E/S se hace a través de líneas especificas del bus de control, una para lectura y otra para escritura.

2.5. INTERRUPCIONES.

Una vez que se le alimentan las señales de polarización y de tiempo adecuadas al microprocesador, éste repite continuadamente las 3 fases anteriormente descritas (**Fetch, Defer y Execute**).

Este proceso se detiene hasta el momento que un elemento externo al microprocesador lo interrumpe. Puede ser por ejemplo un dispositivo periférico que contiene una cierta información que le debe transmitir al propio microprocesador. A esta afectación en el modo de operación de una microprocesador se le conoce como "**INTERRUPCION**".

Esto es, una interrupción es una señal generada por un dispositivo externo con la intención de que el microprocesador deje de hacer lo que esta haciendo para atender al elemento que lo interrumpe.

Si no existieran las interrupciones el microprocesador debería estar preguntando cada cierta cantidad de tiempo si algún dispositivo externo desea ser atendido (**Método Polling**).

Para hacer más claro el concepto de las interrupciones citare una analogía.

Cuando una persona espera una llamada telefónica lo que hace es esperar a que suene el timbre de éste para descolgar el auricular y atender la llamada y no estar descolgando continuamente el auricular para saber si ya es que lo llamaron.

Hay que hacer notar que para que el microprocesador esté en posibilidad de detectar una interrupción, de alguna forma debe enterarse de su existencia. Esto se verifica gracias a que cada vez que se ejecuta un ciclo de

instrucción también se pruba el estado de la línea de interrupción, la cual estará activa si es que algún periférico desea atención o desactivada en caso contrario.

Una vez que la interrupción es reconocida por el microprocesador èste debe atenderla para lo cual se suceden las siguientes actividades:

1) Se detiene la ejecución del programa que este en ejecución en ese momento.

2) Se ejecuta un programa especial que cumple con las necesidades del dispositivo externo que interrumpió.

3) Continua ejecutando el programa original a partir del punto donde ocurrió la interrupción.

A el primer paso de esta secuencia se le denomina **"Reconocimiento de la Interrupción"**. Durante este paso, el microprocesador guarda el contenido de los registros Program Counter (PC) y Condition Register (CR) en una área de memoria RAM denominada STACK (PILA) y carga en el PC la dirección a partir de la cual se encuentra la rutina de servicio de la interrupción.

Una vez ejecutado el paso 2 los contenidos originales del PC y del CR son restaurados de modo tal que sigan en el mismo punto del programa que se estaba ejecutando en el momento de la interrupción.

Durante el segundo paso se ejecuta el programa llamado "Rutina de Servicio de la Interrupcción". Por lo general estas rutinas inician salvando o guardando el contenido de algunos o todos los otros registros, los cuales no son salvados automàticamente durante el primer paso.

Desde Software (es decir a través de una instrucción) es posible indicar en què momento, dentro del desarrollo del programa, no se desea atender interrupciones en el momento en que se sucedan. De lo cual se desprende el hecho de que existe un tipo de interrupciones que pueden ser pasadas por alto, es decir, que pueden ser **Enmascarables**.

Pero existen otro tipo de interrupciones las cuales no se pueden deshabilitar, esto es, que no importa la lògica del programa siempre que se active la línea de interrupción **no-enmascarable**, el microprocesador interrumpira el programa que esta ejecutando y atenderá a dicha interrupción.

En conclución, según su importancia existen dos tipos de interrupciones.

2.5.1. Las Enmascarables (en algún momento pueden ser ignoradas)

2.5.2. Las No-Enmascarables (siempre serán Atendidas).

Generalmente las interrupciones no-enmascarables se suceden cuando existe un corte en la alimentación de potencia del microprocesador.

Puede suceder que en un mismo instante dos dispositivos externos interrumpan al microprocesador para ser atendidos, pero el microprocesador sólo puede entender uno a la vez, por lo tanto deben asignarse prioridades, por ejemplo: se tienen dos dispositivos, el A y el B. El B envía al microprocesador información que varía muy rápidamente y que por lo tanto debe ser atendido de igual forma, entoces el dispositivo B tiene una prioridad mayor que el A. Si en un mismo instante ambos dispositivos interrumpen al microprocesador el que será atendido será el dispositivo B.

Por otro lado, si en un mismo instante interrumpe el dispositivo A, será atendido, si no ha interrumpido otro con mayor prioridad, pero si durante la ejecución de su rutina de servicio se reconoce una interrupción de un dispositivo con mayor prioridad esta rutina se interrumpe y se atenderá la rutina de servicio del otro dispositivo de mayor prioridad.

Para establecer las prioridades entre los diferentes dispositivos se deben agregar arreglos de circuitos lógicos externos al microprocesador y además agregar las instrucciones de habilitación y deshabilitación de interrupciones, dentro de las rutinas de servicio, dependiendo de las prioridades.

Dependiendo de la forma en que el microprocesador reconoce al dispositivo que interrumpe, existen 2 tipos de interrupciones:

a) **VECTORIZADAS.**- En este tipo el dispositivo que solicita atención proporciona la direccion (o parte de ella), de memoria donde se encuentra su propia rutina de servicio.

b) **NO-VECTORIZADAS.**- En estas, cada dispositivo tiene su propia línea de interrupción y por lo tanto, dependiendo de la línea activa, el microprocesador ejecuta su correspondiente rutina de servicio.

2.6. ACCESO DIRECTO A MEMORIA.

En este modo de transferencia (**Direct Memory Access**) los datos son manejados entre el periférico y la MMU, sin la intervención del microprocesador, la única participación de la CPU es para indicar la dirección de memoria a partir de

la cual se va a realizar el acceso y la cantidad de localidades involucradas.

Este modo de transferencia es utilizado para intercambiar datos con periféricos de alta velocidad, como discos magnéticos.

Para realizar transferencias en este modo se necesita una circuiteria adicional (HW), que controle la secuencia del acceso ya que en este caso la CPU se "desconecta" de los buses, para dejar el mando al controlador de DMA's.

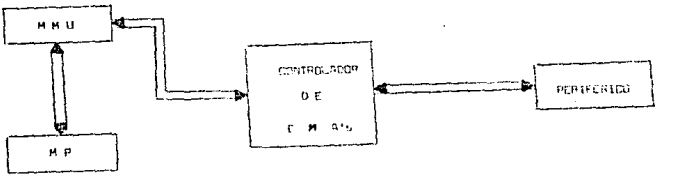


FIGURA 2.6. DIAGRAMA A BLOQUES DE LA INTERCONEXION DE UN CONTROLADOR DE DMA'S.

3. TECNICAS DE PROGRAMACION PARA MICROPROCESADORES.

En los siguientes párrafos y hasta el final del presente trabajo se tratará más a fondo la parte de un sistema basado en un microprocesador correspondiente a la programación. La cual considero que es la más importante, no sólo en los sistemas basados en microprocesadores, sino en general en cualquier sistema basado en una computadora digital. Ya que para una determinada configuración de HARDWARE es factible generar una gran cantidad de aplicaciones diferentes en base a la extraordinaria flexibilidad y número de programas que ejecuta éste.

Esta sección inicia con un conjunto de conceptos básicos de SOFTWARE, algunos de los cuales ya se trataron en secciones anteriores, sin embargo, se amplian para crear una secuencia ordenada que sea propicia para abordar el tema principal de este trabajo, es decir, la descripción de una **metodología** para el desarrollo de SOFTWARE para **microprocedadores.**

3.1. CONCEPTOS SOBRE PROGRAMACION DE MICROPROCESADORES.

3.1.1. METODOS DE DIRECCIONAMIENTO. Existen diferentes modos o maneras de indicarle al MICROPROCESADOR en què localidad de memoria se encuentra el operando u operandos a procesar según el código de operación. A estos diferentes modos se les llama **MODOS DE DIRECCIONAMIENTO.**

Al igual que otras características, ya mencionadas, cada MICROPROCESADOR tiene sus propios modos de direccionamiento, aunque, de igual forma, se pueden seleccionar los modos más generalizados que aparecen en todos los MICROPROCESADORES de 8 Bits; y estos son los siguientes:

a) **MODO IMPLICITO:** En este modo de direccionamiento la instrucción sòlo cuenta con código de operacion, y no contiene operandos.

b) **INMEDIATO:** En el direccionamiento inmediato el operando es el valor real que se va manipular, se utiliza para el manejo de constantes.

c) **ABSOLUTO:** En este modo, el operando es la dirección donde se encuentra el dato a procesar.

d) **INDEXADO:** En el modo de direccionamiento indexado el dato que se procesará está contenido en la dirección de memoria indicada por el contenido de un registro especial denominado **registro índice.**

3.1.2.PROGRAMA. Es una serie de instrucciones organizada secuencialmente, que comanda al CPU (MICROPROCESADOR en este caso) a realizar funciones más complejas que las que se logran con las simples instrucciones.

Todos los MICROPROCESADORES ejecutan las instrucciones en la forma en que se encuentran almacenadas, es decir, **secuencialmente**, a menos que una de las mismas instrucciones cambie la secuencia de ejecución o detenga la CPU. Esto es, el procesador obtiene la siguiente instrucción de la dirección inmediata posterior a la actual a menos que la instrucción actual especifique directamente la direción en donde se encuentra la siguiente instrucción a ejecutar.

Finalmente, todos los programas se traducen a un conjunto de números binarios. Por jemplo: el conjunto de BITS que se encuentra a continuación suma el contenido de la localidad $100 más el contenido de la localidad $101 y almacena el resultado en la localidad $605, este programa es válido para el MICROPROCESADOR 6809.

1011	0110	
0000	0001	
0000	0000	
1011	1011	Representación
0000	0001	en lenguaje de
0000	0001	máquina.
1011	0111	
0000	0110	
0000	0101	

A esta representación se le denomina **LENGUAJE DE MAQUINA**, y es el que directamente ejecuta el procesador.

A pesar de que este es el único lenguaje que "entiende" el MICROPROCESADOR no es el más conveniente para ser utilizado por los seres humanos por varias razones:

- El programa es difícil de entender y de depurar.
- El programa por si mismo no da ninguna información al programador acerca de la tarea que realiza.
- El manejo de una gran cantidad de BITS incrementa enormemente la probabilidad de cometer de errores.

Por las razones anteriores se optó utilizar un sistema de numeración que redujera la cantidad de digitos y que hiciera la conversión entre ambos sistemas fuera inmediata.

Los sistemas que más han proliferado en este aspecto son el octal y el **hexadecimal**. Y a su vez el hexadecimal es el de mayor aplicación y el utilizado en este trabajo.

La presentación del mismo programa de la figura anterior en código hexadecimal, es la siguiente:

```
B6      La traducción del sistema
01      binario al hexadecimal es
00      inmediata, ya que 4 bits
BB      se traducen en un número
01      hexadecimal.
01
B7
06
05
```

En esta representación el programa es un poco más manejable para el programador, pero el precesador únicamente entiende los codigos binarios, por lo tanto para que este programa sea ejecutado, tiene antes que ser traducido a su correspondiente notación binaria.

3.1.3. CARGADOR HEXADECIMAL. De esta tarea de traducción se encarga el denominado **CARGADOR HEXADECIMAL**, el cual acepta como entrada números hexadecimales y los convierte en números binarios, además coloca estos números binarios en la memoria principal de la computadora.

El CARGADOR HEXADECIMAL es a su vez otro programa estandarizado que lo proporciona el fabricante como parte de otro mas grande denominado MONITOR.

Por otra parte, si el traducir de números binarios a números hexadecimales facilita la tarea al programador, asignarle un nombre especial a cada número hexadecimal que representa una instrucción facilitaría aún más la programación.

Por lo general a cada código de operación se le asigna un nombre que ayude a recordar la operación que realiza, así por ejemplo, el número hexadecimal BB representa la suma (Addition en inglés), por lo tanto un nombre adecuado es ADD.

A cada uno de estos nombres asignados a los códigos de operación (claves en hexadecimal de cada una de las instrucciones) se les denomina **MNEMONICOS**. Por ejemplo, para el programa tratado, su representación en mnemónicos es:

```
LDA  $0100
ADDA $0101
STA  $0605
```

El cual indica que cargue (LOAD) el contenido de la dirección $0100 (el símbolo de pesos implica que el número asociado está expresado en hexadecimal) al acumulador A, a este valor le sume (ADDITION) el contenido de la dirección $0101 y que el resultado, que quedó en el acumulador A, se almacene (STORAGE) en la dirección $0605.

A la representación de programas en MNEMONICOS se le llama representación en **lenguaje ENSAMBLADOR.**

3.1.4. ENSAMBLADOR. Como ya se mencionó, el único lenguaje en el que el MICROPROCESADOR puede ejecutar un programa es en el lenguaje de máquina, por lo tanto, el programa en lenguaje ensamblador debe ser traducido a una conjunto de números binarios. El encargado de esta tarea es un programa denominado **ENSAMBLADOR.**

La palabra ENSAMBLADOR, se utiliza para nombrar dos cosas diferentes, una es el **LENGUAJE ENSAMBLADOR** y otra es el **PROGRAMA ENSAMBLADOR**, el cual traduce un programa en lenguaje ensamblador a una programa en lenguaje de máquina.

A los programas escritos en lenguaje ensamblador (y en general en lenguajes de más alto nivel) se les denomina **PROGRAMA FUENTE** y al mismo programa, pero en representación binaria se les denomina **PROGRAMA OBJETO.**

Además de traducir de lenguaje ensamblador a lenguaje de máquina, los programas ensambladores, o ensambladores, generalmente presentan una serie de ventajas adicionales:

- Permiten al usuario asignar nombres a las direcciones de memoria, a los puertos de E/S y a las propias instrucciones (etiquetas).
- Convierten direcciones o datos de diferentes sistemas de numeración (por ejemplo hexadecimal o decimal) a binario y convierten caracteres a su correspondiente código ASCII, (o al código que utilize para la representacion interna de textos).
- Permite al usuario asignar áreas de memoria como almacenamiento de datos.

Algunas desventajas del lenguaje ensamblador son:

- Se debe tener un conocimiento detallado del procesador utilizado.
- Muy poca portabilidad.
- Etc.

Una representación más simple y más entendible por cualquier persona sería:

SUMA = NUMERO1 + NUMERO2

Donde SUMA, NUMERO1 y NUMERO2 representan las localidades de memoria $0605, $0100 y $0101 respectivamente.

A esta última representación se le denomina **LENGUAJE DE ALTO NIVEL.**

Cada proposición (**proposición** es el equivalente a una instrucción en lenguaje de bajo nivel) desarrolla una cierta función y ésta corresponde, generalmente, a muchas instrucciones en lenguaje ensamblador.

El programa que traduce los programas de lenguajes de alto nivel a programas en lenguaje de máquina se le llama **COMPILADOR.** Un compilador es a su vez un programa que acepta como entrada un programa fuente en lenguaje de alto nivel y genera como salida el correspondiente programa objeto.

Estos lenguajes de alto nivel presentan una serie de ventajas y desventajas respecto a los lenguajes de bajo nivel (se denominan lenguajes de bajo nivel al ensamblador y al lenguaje de máquina):

VENTAJAS:

- Una descripción más conveniente de la tarea a realizar.
- Codificación más eficiente de programas.
- Fácilmente documentables.
- Sintáxis estandárizada (normalizada).
- Independencia de la estructura del procesador y en general de la computadora.
- Portabilidad.
- Disponibilidad de librerías y de combinación con otros programas.
- Fácil depuración de programas.

DESVENTAJAS:

- Reglas sintácticas especiales.
- Requiere SOFTWARE y HARDWARE adicional.

- Orientados a áreas de conocimiento especificas.

- Programas objeto ineficientes.

- Dificultad para alojar el programa objeto en memoria.

En general y concluyendo, cada uno de los diferentes lenguajes tienen sus ventajas y desventajas, intuyendo de esto que cada uno es el apropiado para diferentes aplicaciones.

A continuación se mencionan algunas de las caracteristicas más importantes para cada nivel:

- **Lenguaje de MÁquina:** Este lenguaje no presenta en realidad gran ventaja ya que el costo de los ensambladores es bastante accesible.

- **Lenguaje ensamblador:**

  . Adecuado para programas cortos.

  . Para aplicaciones donde el tamaño de la memoria es un factor importante.

  . Aplicacion de control en tiempo real.

  . Procesamiento limitado de datos.

  . MÁs operaciones de E/S que de cálculo.

- **Lenguajes de Alto Nivel:**

  . Programas muy largos.

  . Aplicaciones que requieren gran cantidad de memoria.

  . MÁs operaciones de cálculo que de E/S.

  . Compatibilidad con aplicaciones similares en otras computadoras.

  . Programas que requieren de depuración y cambios continuamente.

En el caso particular de este trabajo, el lenguaje utilizado, es por sus propias caracterisicas, el Lenguaje Ensamblador.

Otro concepto que es necesario conocer es el de **programa monitor.**

3.1.5. PROGRAMA MONITOR. En el momento de proporcionarle energía al MICROPROCESADOR, este inicia la ejecución de la instrucción indicada por el PC (este a su vez se cargó alatoriamente con una dirección que el usuario desconoce). Para evitar que esta situación quede fuera de control los fabricantes de computadoras proporcionan un método especial para cargar al PC con una dirección donde se encuentra un programa de inicialización.

Este programa se denomina **PROGRAMA MONITOR** o simplemente **MONITOR**, el cual se comunica con el usuario a través de un dispositivo de E/S, generalmente es un display y un teclado. El usuario da órdenes al monitor a través de comandos en un lenguaje especial, mientras que el monitor responde con mensajes o ejecución de acciones.

De entrada, el monitor le indica al ususarioque se encuentra listo para ejecutar el comando que se le indique a través de alguna marca especial (PROMPT), en este momento el usuario tiene el control del sistema y por medio de los comandos el monitor controlará todas las acciones de la computdora.

Las tareas que generalmente realiza un programa monitor son las siguientes:

- Cargar un programa objeto en memoria principal.
- Ejecutar programas.
- Accesar y modificar los registros internos del MICROPROCESADOR.
- Alojar datos en la MMU.

El programa monitor reside en la memoria principal pero en un tipo de ésta no volatil, es decir, que aunque se desenergize la información almacenada no se pierde.

3.2. TECNICAS DE PROGRAMACION PARA MICROPROCESADORES.

(INGENIERIA DE SOFTWARE)

En esta sección se abordará el estudio de un procedimiento sistemático para el desarrollo de Software.

En general se puede afirmar que existen dos tipos de software:

- **El software desarrollado para uso personal y**
- **El producto de software (uso general).**

El segundo tipo se diferencia del primero por varias características:

- Tiene gran cantidad de usuarios,

- involucra a varios programadores y

- requiere a varias personas para su mantenimiento.

En la mayoría de los casos las personas que desarrollan el producto de SOFTWARE no son las mismas que le dan mantenimiento. El desarrollo y mantenimiento de SOFTWARE requiere de un enfoque más ordenado y sistemático que el necesario para el desarrollo de SOFTWARE para uso personal.

Este enfoque ordenado y sistemático para la creación y mantenimiento de programas es denominado **INGENIERIA DE SOFTWARE ó INGENIERIA DE PROGRAMACION.**

Así pues, se puede definir a la **ingeniería de software (IS) como la disciplina tecnológica y administrativa dedicada a la producción sistemática de productos de programación, que son desarrollados y modificados a un tiempo y presupuesto dados.**

Los conceptos de IS nos dicen que los programas (el SOFTWARE) deben ser tratados como un producto de ingeniería y que para su desarrollo se deben aplicar las técnicas y procedimientos utilizados en las otras ingenierías. Sin embargo, existe una diferencia básica entre los productos desarrollados por las otras ingenierías y los desarrollados por la IS, esta es, que en la Ingeniería de programación se trata con productos que no son tangibles y por lo tanto no están sujetos a leyes físicas de la programación. Este es un factor muy importante para poder afirmar que un determinado producto de SOFTWARE es bueno o malo. Esto no es factible ya que siempre estará sujeto a subjetibidades, sin embargo, las técnicas de IS están enfocadas a crearles atributos reconocidos a los programas para tener criterios de calificación.

La IS se diferencia de las técnicas de programación tradicionales en que en ésta se utilizan técnicas ingenieriles para especificar, diseñar, validar y mantener un producto; además se preocupa de aspectos administrativos que queden generalemente fuera de la programación tradicional.

Algunas veces se menciona que las técnicas de la ingeniería de software son únicamente aplicables a los grandes proyectos, sin embargo, todos sus conceptos permanecen constantes y son validos para proyectos de cualquier tamaño, aún para los muy pequeños.

El término **PRODUCTO DE PROGRAMACION ó PRODUCTO DE SOFTWARE** no se refiere unicamente al programa como un código fuente, es más que esto, se refiere al código fuente más todos los manuales y documentación asociados.

3.2.1. CARACTERISTICAS DE UN PRODUCTO DE PROGRAMACION.

Todo producto de programación debe tener al menos los siguientes atributos fundamentales de calidad:

a) **UTILIDAD.** Esta es la característica de calidad más importante en cualquier producto de SOFTWARE, ya que se refiere al grado de satisfacción que tiene el producto para una aplicación determinada.

b) **CONFIABILIDAD.** La confiabilidad es la capacidad de un programa para desempeñar una función requerida bajo ciertas condiciones, durante un tiempo especifico.

c) **PORTABILIDAD.** Facilidad con la que un producto de programación puede ser transferido de un sistema de cómputo a otro ó de un ambiente a otro.

d) **EFICIENCIA.** Grado con el que un producto de programación efectua sus funciones mediante un mínimo de recursos computacionales.

e) **EXACTITUD.** Especificación cualitativa de ausencia de errores.

f) **ERROR.** Discrepancia entre una condición o valor calculado y la condición real especificada o valor correcto teórico.

g) **SOLIDEZ.** Grado con el que un producto de software puede continuar operando correctamente a pesar de la introducción de datos inválidos.

h) **CORRECCION.** Grado en el que un producto de software está libre de defectos de diseño y codificación, esto es, libre de fallas.

i) **COSTO.** Un producto de SOFTWARE debe ser costeable en su desarrollo, mantenimiento y uso. Debe requerir en su utilización menos recursos humanos o industriales, o menos tiempo que el utilizado antes de su implantación.

3.2.2. CLASIFICAION POR LINEAS DE CODIGO. Por otro lado, según la cantidad de líneas en el código fuente se pueden clasificar a los productos de programación en:

a) **TRIVIALES.** Contienen aproximadamente 500 líneas de código y es un sólo programador el que desarrolla el SOFTWARE.

b) **PEQUEÑOS.** Para estos proyectos se necesita un programador dedicado de uno a seis meses y genera programas de 1000 a 7000 líneas de código fuente. Ejemplos de estos proyectos son paquetes desarrollados para aplicaciones técnicas muy orientadas a equipos generalmente no muy comerciales.

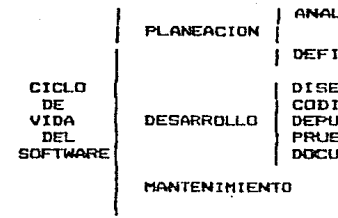
c) **MEDIANOS.** Un proyecto mediano requiere de dos a cinco programadores, trabajando de uno a dos años en la generación de 10,000 a 50,000 líneas de código fuente.

d) **GRANDES.** Un proyecto grande necesita de 5 a 20 programadores que trabajen durante 2 a 3 años para generar de 50,000 a 100,000 líneas de código. Ejemplos de estos proyectos se consideran a los compiladores de gran tamaño, paquetes de bases de datos, sistemas gráficos, etc.

e) **MUY GRANDES.** En un proyecto muy grande intervienen de 100 a 1000 programadores, durante un período de 4 a 5 años, para desarrollar cerca de 1'000,000 de líneas de código. Por lo general estos proyectos comprenden sistemas de tiempo real, sistemas de telecomunicaciones y multiusuarios, grandes sistemas operativos, grandes bases de datos, etc.

f) **EXTREMADAMENTE GRANDES.** Un proyecto de esta magnitud reune aproximadamente a 5000 programadores durante 10 años para producir aproximadamente 10',000,000 de líneas de código. Este tipo de proyectos no son muy comunes, generalmente son aplicaciones de tipo militar, como control de tráfico aereo ó sistemas de proyectiles de defensa.

3.2.3. En el siguiente diagrama se pueden apreciar las fases en las que se divide el ciclo de vida de un producto de software dentro del marco de la Ingeniería de Software.



3.2.3.1 FASE DE PLANEACION.

Uno de los principales propositos de esta fase es aclarar los objetivos, necesidades y restricciones del producto.

La falta de planeaciòn es la causa principal de retrasos en programaciòn, poca calidad y altos costos de mantenimiento en los desarrollos de productos de programaciòn.

Se debe iniciar la planeciòn formulando planes preliminares. Los planes preliminares se modificaràn segùn vayan evolucionando los productos de programaciòn; **la planeaciòn para el cambio es uno de los aspectos clave con los que se logra el èxito.**

La fase de planeaciòn esta dividida en dos subfases:

a) El anàlisis y definiciòn de requerimientos y

b) Definiciòn del problema.

3.2.3.1.1 ANALISIS Y DEIFNICION DE REQUERIMIENTOS. El objeto de la definiciòn de requisitos es especificar total y consistentemente los requerimientos tècnicos del producto de una manera concisa y sin ambigüedades.

En teoria, la especificaciòn de los requisitos establecerà **"què es el producto"** sin implicar **"como es èste"**. El diseño del producto se refiere a proporcionar los detalles de "còmo" va a desarrollar el "què".

El analisis y definiciòn de los requisitos intentan satisfacer los siguientes aspectos:

- Proporcionar una base para definir el flujo y estructura de la información que se utilizarAn durante la fase de desarrollo.

- Describir el producto de programación por la definición detallada de las interfases, proporcionando una descripción de funciones independiente.

- Establecer y mantener la comunicación con el cliente o usuario de modo tal que los dos asapectos anteriores se cumplan.

El siguiente formato sirve como guía para generar el documento que se derive de esta fase del ciclo de vida de un producto de SOFTWARE:

1.- INTRODUCION.

2.- DESCRIPCION DE LA INFORMACION.

A) Diagrama de flujo de datos.
B) Representación de la estructura de datos.
C) Diccionario de datos.
D) Descripción de las interfases del sistema
E) Interfases internas.

3.- DESCRIPCION FUNCIONAL

A) Funciones.
B) Narración de los procesos.
C) Criterios de Diseño.

4.- CRITERIO DE VALIDACION

A) Bandas de Perfomance.
B) Tipos de pruebas.
C) Respuestas esperadas por el programa.
D) Consideraciones especiales.

3.2.3.1.2 DEFINICION DEL PROBLEMA. Dentro de esta etapa se dan varias divisiones lògicas del problema, con la finalidad de lograr una definición más adecuada. Estas divisiones se presentan a continuación:

a) Entradas.

b) Salidas.

c) Sección de proceso.

d) Manipulación de errores.

e) Factores humanos e integración del operador.

a) **ENTRADAS.** Para las entradas se debe de definir:

1. Forma en la que se reicibirán.

2. Momento en que la entrada estará disponible y momento ó forma de indicarle al MICROPROCESADOR en que está disponible.

3. Por cuánto tiempo estará disponible.

4. Forma en que se dispondrá de ésta una vez dentro de la memoria (estructura de los datos).

5. Cómo cambiarla y cómo notificar al procesador de este cambio.

b) SALIDAS.

1. Forma de las salidas.

2. Momento en que estarán disponibles y momento o forma de indicarle al procesador que está disponible.

3. Tiempo que permanecerá disponible.

4. Un procedimiento para cambiarla y uno para indicar al periférico de este cambio.

5. Si son varias salidas, la secuencia que seguirán.

c) **SECCION DE PROCESO.**

Entre la obtención de las entradas y el envío de las salidas esta la sección de proceso, la cual se definirá a partir de los siguientes criterios:

1. Procedimiento Básico (algoritmo) para transformar la(s) entrada (s) en salida (S).

2. Tiempo de duración del proceso.

3. Cantidad de memoria que necesita, tanto el programa mismo como los datos.

4. Què programa estándar y tablas de datos serán utilizados.

5. Diferentes casos especialeds que puedan existir y su adecuada manipulación

6. Respuesta a los posibles errores.

d) MANEJO DE ERRORES.

Un factor importante en el diseño de Software es la capacidad del diseñador para preveer la cantidad de errores màs comunes y su recuperecion o recuperaciòn de la secuencia del programa. Para la definiciòn de una adecuada manipulaciòn de errores se debe tener en cuenta los siguientes paràmetros:

1. Errores que puedan ocurrir.
2. Errores màs frecuentes.
3. Errores inmediatamente detectables y errrores detectables despuès de una condiciòn o tiempo (Estatus de errores).
4. Procedimiento para una recuperaciòn que necesite poca pèrdida de tiempo.
5. Detecciòn de errores que involucren la ejecuciòn de procedimientos especiales.

e) FACTORES HUMANOS E INTERACCION DEL OPERADOR.

Este es un factor muy importante que se debe tomar en cuenta para el diseño de software, ya que en la mayoria de los casos son personas las que tienen que proporcionar/obtener informaciòn del sistema, por tal razòn se deben tener en cuenta las siguientes observaciones:

1. Procedimientos de entrada màs obvios para los seres humanos.
2. Determinaciòn fàcil y clara del inicio, continuidad y terminaciòn del proceso de carga de informaciòn al sistema.
3. Informaciòn adecuada para un error al operador.
4. Retroalimentaciòn al operador, para indicarle que los datos estan correctos.
5. Legibilidad de los mensajes.
6. Facilidad para operar el sistema.

7. El sistema debe ser operado casi por cualquier persona sin necesidad de que esta sea una experta en el propio sistema.

8. Facilidad para reinicializar el sistema en caso de que así se desee.

3.2.3.2 FASE DE DESARROLLO.

3.2.3.2.1 DISEÑO. El diseño se refiere a la identificación de los componentes de programación (funciones, flujos de datos y almacenamiento), especificando las relaciones entre ellos, la estructura de la programación y manteniendo un registro de las desiciones.

El diseño se divide en estructural y detallado. El **diseño estructural** comprende la identificación de los componentes de programación, su desacoplamiento y descomposición en módulos de procesamiento y estructrucras de datos conceptuales, y la especificación de las interconexiones entre componentes. El **diseno detallado** se refiere a detalles de "cómo": cómo acoplar módulos de procesamiento y cómo instrumentar los algoritmos para las estructuras de datos y sus interconexiones.

El diseño del programa es la etapa en la cuál es formulada la definición del problema como un programa de computadora.

Existen diversas técnicas, todas muy difundidas, para el diseño sistematizado de programas, entre las más destacadas están el diseño TOP-DOWN, la programación modular, la programación estructurada, los diagramas de flujo, etc.

Cada una de estas técnicas tiene sus ventajas y desventajas, razón por la cual no se puede afirmar que alguna sea mejor que otra, si no en realidad depende de la aplicación en particular y principalmente del estilo de programación del diseñador.

En este trabajo se seleccionan las dos últimas técnicas anteriormente mencionadas, es decir, **la programación estructuradas y diagramas de flujo**, por considerar el autor que son las más orientadas a una mejor organización y documentación de programas.

El utilizar diagramas de flujo para diseñar programas tiene varias ventajas, la más importante es que permite dividir el proyecto completo en varias subtareas, lo que implica desagregar el problema completo en pequeñas partes más simples, aunque tiene también algunas desventajas, entre las que las destacan las siguientes: sólo muestra la organizacion del proceso y no se ocupan de la organización

de los datos ni de la organización de los modulos de E/S; el número o tipo de interconexiones pueden ser muy fàciles de manejar a nivel gràfico, pero no a un nivel de codificación, en otras palabras, en muchos casos la codificación no es inmediata.

Por las razones antes expuestas los diagramas de flujo se utilizan, en este trabajo, sólo para describir la estructura del sistema en forma general **(Diseño Estructural)** y no para la especificación de los algoritmos.

La otra tècnica escogida es la programación estructurada, y será con esta tècnica con la que se realize la codificación de los programas **(Diseño Detallado)**.

Aunque esta sección no pretende ser un tratado sobre programación estructurada, se presentan y critican las caracteristicas de esta tècnica de diseño de programas.

La filosofia principal de la programación estructurada es la de diseñar programas a partir de un conjunto breve de estructuras.

La caracteristica particular y común de cada una de estas estructuras es que todas ellas tienen una sola entrada y una sola salida.

Las estructuras bàsicas o figuras bàsicas de la programación estructurada son las siguientes:

A) SECUENCIA ORDINARIA: Esta es la estructura màs simple, además de ser la màs común. En esta estructura las instrucciones se van ejecutando tal como aparecen, es decir, en forma consecutiva.

B) ESTRUCTURA CONDICIONAL: Dentro de esta estructura existen dos caminos diferentes a seguir por la secuencia, sin embargo sólo existe un único camino de salida. En esta estructura se pregunta por el estado de una condición, si el resultado de la condición es verdadero se sigue un camino en caso contrario se continua por el alterno. En esta estructura el BLOQUE puede no contener ninguna instrucción.

C) ESTRUCTURA DE CICLO: A través de esta estructura es posible repetir la ejecución de las instrucciones o estructuras contenidas dentro del cuerpo de la misma, hasta que el resultado de la condicion de prueba sea falso.

Dentro de esta estructura existen dos variantes:

i) La condición se prubra **ANTES** de ejecutar el bloque contenido dentro del cuerpo del ciclo.

ii) La condición se prueba **DESPUES** de haber ejecutado el conjunto de instrucciones contenido en el cuerpo de la estructura. Lo que quiere decir que en esta estructura el bloque correspondiente de instrucciones (o de estructuras) se ejecuta al menos una vez.

D) ESTRUCTURA "CASE": Dentro de esta figura básica existen una serie de caminos diferentes, pero sólo una es recorrida por la secuencia del programa dependiendo del valor de un elemento denominado INDICE.

En esta estructura, si el indice satisface la condición I, se ejecuta el I-ésimo bloque, por ejemplo, si cumple la condición 7 ejecuta el bloque número 7.

Las características más sobresalientes de la programación estructurada son:

- Sólo las cuatro estructuras básicas, o una combinación de estas, son permitidas.
- Las estructuras pueden ser anidades a cualquier nivel de complejidad.
- Cada estructura tiene una sola entrada y un solo punto de salida.

Por otra parte, las ventajas y desventajas de esta técnica son expuestas a continuacion:

VENTAJAS:

a) La secuencia de operación es simple de seguir, lo que permite un depuramiento y una serie de pruebas más fáciles de aplicar.

b) La terminologia esta estandárizada.

c) Dada la utilización de sus estructuras, los programas se autodocumentan y son fáciles de leer.

d) Dado que no permite bifurcaciones (saltos) incondicionales son fáciles de documantar.

e) Fuerza a un estilo de programación más disciplinado.

f) Programas mas organizados.

DESVENTAJAS:

a) Utilizan más memoria que los programas no estructurados.

b) Limita a sòlo 3 estructuras bàsicas, lo cual redunda en una mayor inversiòn de tiempo para la creaciòn de programas.

3.2.2.2 CODIFICACION. El proceso de codifiaciòn consiste en traducir un algaritmo determinado en proposiciones de un lenguaje especifico; el resultado de la codificaciòn es el **PROGRAMA FUENTE** escrito en el lenguaje seleccionado.

Es importante codificar los algoritmos con un buen estilo, esto es, proporcionar un còdigo fuente fàcil de comprender, sencillo y elegante (estructurado).

Se ha reconocido que un buen estilo de codificaciòn puede alivianar muchas de las deficiencias de un lenguaje de programaciòn primitivo, mientras que un estilo pobre puede frustrar los porpòsitos de un ecxelente lenguaje de programaciòn.

El lenguaje de programaciòn es un vèhiculo para la comunicaciòn entre el ser humano y las computadoras. El proceso de codificaciòn es una actividad humana. Como tal las caracterìsticas sicologicas de un lenguaje tienen un importante impacto sobre la calidad de la comunicaciòn. El proceso de codificaciòn puede ser visto tambièn como un paso en la metodologìa de la Ingenierìa de Software. Las caracterìsticas ingenieriles de un lenguaje tienen un importante impacto sobre los acontecimientos sucedidos en el desarrollo de un producto de Software. Finalmente, las caracterìsticas tecnicas de un lenguaje de programaciòn pueden influir en la calidad del diseño.

No hay un conjunto ùnico de reglas que se puedan aplicar en todas las situaciones, sin embargo, existen principios generales que son ampliamente aplicables:

3.2.2.1.ACCIONES A SEGUIR PARA UN BUEN ESTILO DE CODIFICACION.

A) Emplear sòlo unas cuantas estructuras estàndares de control.

B) Utilizar la intruccion "GO TO" de manera disciplinada.

C) Introducir tipos de datos definidos para el problema.

D) Cubrir las estructuras de datos bajo las funciones de acceso.

E) Ahislar las dependencias con el equipo particular en unas cuantas rutinas.

F) Proporcionar prólogos estándar de documentación para cada subprograma.

G) Examinar cuidadosamente las rutinas que tengan menos de cinco o más de 25 proposiciones.

H) Utilizar sangrías, paréntesis, espacios, líneas en blanco y márgenes alrededor de los bloques de comentarios para mejorar la legibilidad.

3.2.2.3 DEPURACION Y PRUEBA. El proceso de depuración y prueba es tal vez el que mayor cantidad de tiempo consume al diseñador, dentro de la etapa de desarrollo de software.

Este proceso y su complejidad esta en función directa de la técnica de diseño adoptada para el desarrollo, esto es, un porgrama escrito en un lenguaje estructurado y auxiliado con diagramas de flujo es más fácil de depurar y probar que uno escrito en forma no sistemática.

En esta parte se describen algunas tecnicas de depuración de software y algunas otras de prueba, para los datos que manejará el software.

3.2.2.3.1 DEPURACION.

Las herramientas de **depuración** tienen principalmente, dos funciones:

i) Definir la sección dentro de la cual se encuentra el error, esta sección debe ser lo mas compacta posible.

ii) Identificar qué es lo que hace el microprocesador en cada una de las secuencias del programa cuando este se ejecuta correctamente y de esta forma identificar los errores más obvios cuando el programa no se ejecuta correctamente.

Unas de las principales y más difundidas tecnicas de depuración son las siguientes:

A) **EJECUCION PASO A PASO.** Esta técnica permite ejecutar sólo una instrucción cada vez con la intención de conocer el estado de los datos y resultados parciales del programa cada vez que se termina de procesar cada una de éstas.

La ejecución paso a paso se utiliza básicamente en la depuración de la LOGICA de reglamentos cortos de programas, ya que si se intentara aplicar en otros aspectos, tales como iteraciones, la ejecución que comunmente toma algunos milisegundos al procesador al usuario le llevaría varios días.

B) **PUNTOS DE INTERRUPCION (Break points).** Un punto de interrupción es un punto dentro del programa en el cual se desea que el procesador se detenga en el momento de llegar a este y no continuará la ejecución hasta que el usuario lo indique a través de una orden especifica.

Esta técnica se utiliza para verificar segmentos completos de programa, por ejemplo, rutinas de inicialización o de almacenamiento de registros internos.

Los puntos de interrupción son el concepto más amplio de la ejecución paso a paso, y además se complementan ya que un segmento delimitado por puntos de interrupción puede ser ejecutado paso a paso.

C) **SEGUIMIENTO (Trace).** Por medio de esta técnica de depuración es factible ver los resultados parciales cada vez que una instrucción se ejecuta. Los resultados parciales pueden ser el vaciado del contenido de los registros internos o el contenido de alguna localidad de memoria.

Cada vez que se ejecuta una instrucción se despliegan los resultados indicados, e inmediatamente se ejecuta la siguiente instrucción generando a su vez los correspondientes resultados intermedios. De lo anterior se desprende que la cantidad de información generada es muy grande por lo que es conveniente limitar las variables a las que se les va a hacer el seguimiento.

D) **DEPURACION MANUAL.** El realizar un depuramiento manual es recomendable en muchas ocaciones ya que a través de éste se pueden localizar errores de lógica, principalmente. Al aplicar este método se debe hacer de una manera sistemática, porque de lo contrario sus resultados serían lo opuesto. Cuando se realiza una depuración manual es recomendable dirigir los esfuerzos sobre los siguientes puntos:

- Asegurarse de que todas las variables a ser utilizadas dentro de un ciclo se inicializen adecuadamente.

- Verificar las condiciones de los saltos condicionales, es decir, tomar un caso particular y probarlo sobre la condición a ser satisfecha.

- Probar los puntos extremos, el inicial y el final, de los ciclos que es donde comunmente se encuentran conflictos.

3.2.2.3.2 PRUEBA.

El proceso de prueba complementa al proceso de depuración, éste consiste en someter al programa a la ejecución del algoritmo con datos seleccionados.

El factor más importante del proceso de prueba es el hecho de seleccionar los datos más representativos y que abarquen la mayor cantidad de casos posibles.

Cuando el volumen de datos a procesar es pequeño este proceso no es muy relevante, no así cuando el volumen de datos es grande.

En la mayoría de los sistemas basado en MICROPROCESADORES la cantidad de datos a procesar es grande y por lo tanto el proceso de prueba se vuelve muy importante.

La siguiente secuencia presenta una alternativa para la adecuada selección de datos de prueba:

- Formar clases de datos (por ejemplo: positivos y negativos; mayores que un determinado valor; múltiplos de diez, etc.).
- Identificar en el mayor grado posible los casos especiales, los triviales y los ambiguos.
- Seleccionar de cada una de las diferentes clases sólo algunos datos, los más representativos de la misma.

3.2.2.4 DOCUMENTACION. El proceso de desarrollar software implica algo más que construir un programa que realice una tarea determinada. Además de los propios programas un producto de software debe de incluir la documentación necesaria para permitir el adecuado uso, mantenimiento y perfeccionamiento del mismo.

Existen varios puntos a considerar en el momento del diseño y codificación de programas para que la documentación sea la mas óptima posible.

3.2.2.4.1 AUTODOCUMENTACION. Todos los programas se deben documentar a si mismos, es decir, en el momento de ser leídos deben dar un panorama del tipo de trabajo que desarrollan, por lo cual se debe tener cuidado en los siguientes aspectos:

- Nombres de las variables y etiquetas: éstas deben ser lo más obvias posibles.
- Utilizar acrónimos.

- No utilizar nombres muy semejantes para conceptos diferentes.

3.2.2.4.2 COMENTARIOS. Los comentarios son la forma más obvia de documentacar programas, sin embargo, se deben considerar las siguientes reglas para no abusar o utilizarlos inadecuadamente:

- No comentar que es lo que hace una instrucción como tal, sino explicar què efecto tiene sobre el proceso en particular:

Incorrecto ------ INCA Incrementa el acumulador en 1

Correcto ------ INCA Incrementa el contador de intentos fallidos.

- Hacer los comentarios lo más específicos y claros posibles.

- Comentar todos los puntos importantes aunque estos sean muy claros; y comentar los puntos que aunque no sean muy importantes estèn confusos.

- No comentar lo obvio.

- Colocar los comentarios sobre el renglòn ò renglones donde se encuentra la secuencia que se desea explicar.

- Mantener como comentarios las modificaciones hechas al programa, incluyendo una breve explicacion del cambio, la fecha y el autor.

- Mantener Mapas de memoria. Un mapa de memoria es un documento en el cual se enlista la organización y distribución de la memoria del sistema. Estos permiten saber en dònde comienzan y terminan las àreas para programas, para datos del usuario, stack del sistema, y los propios programas del sistema. Es comùn representar los mapas de memoria en forma de tablas.

- Otros dos elementos importantes en la documentaciòn de software son los diagramas de flujo y los programas estructurados. Los diagramas de flujo ayudan a la documentaciòn porque en una mirada se puede conocer de una manera general el trabajo que desarrolla un determinado programa, y parcialmente la forma particular en què lo hace. La programaciòn estructurada por sus propias caracterìsticas, permite saber dònde comienza un bloque y dònde termina; ademàs de ilustrar gràficamente (gracias a la identaciòn) y textualmente la intenciòn de las propias instrucciones.

3.2.2.4.3 LIBRERIAS. El documentar subrutinas y programas de una manera estàndar permite la creaciòn de las denominadas **librerias,** que no son màs que un conjunto de programas que realizan actividades muy usuales, (tales como retardos, desplegados de encabezados, etc.) y que por lo tanto se pueden utilizar en diferentes programas con cambios mìnimos.

A continuaciòn se presenta una estructura para documantar programas, procedimientos y subrutinas :

NOMBRE:
DESCRIPCION:
ENTRADAS:
SALIDAS:
VARIABLES:
CONSTANTES:
AFECTA A:
FECHA DE CREACION:
AUTOR:
OBSERVACIONES:
DIRECCION DE INICIO:
DIRECCION DE TERMINACION:
CANTIDAD DE BYTES:

Esta estructura permite especificar la tarea que desarrolla la subrutina, además de indicar a què parte de la memoria o registros internos afecta; también contempla el hecho de poder llamar a otra subrutina. (Para ilustrar prácticamente esta estructura ver el capitulo siguiente de este documento)

GUIA DEL USUARIO. La guía del usuario es el documento más importante de la documentaciòn. Este debe explicar las caracteristicas del sistema y su uso, proporcionar tambièn ejemplos que clasifiquen el texto.

Al escribir una guía usuario se debe ser claro y objetivo, además de tener en cuenta el punto de vista de los demás.

Esta guía debe de ser orientada para dos tipos de usuarios: **PRINCIPIANTES,** los cuales deben ser "llevados de la mano"; y los usuarios **EXPERIMENTADOS,** los cuales siempre van a tratar de explotar al màximo el sistema. Es por estos dos tipos de enfoques que hay que abarcar ambos aspectos de una manera organizada, es decir, comenzar de lo màs simple hasta lo mas complicado, ahondando racionalmente en el tema no importando el grado de dificultad.

3.2.2.5 FASE DE MANTENIMIENTO. Las actividades de mantenimiento implican mejorar los productos de software, adaptarlos a nuevos ambientes y corregir problemas.

La corrección de problemas implica la modificación y revalidación del software para corregir los errores, algunos de los cuales requieren atención inmediata, otros se pueden corregir con base a un calendario pèriodico y los menos se conocen, pero nunca se corrigen.

Las actividades de mantenimiento consumen gran parte del presupuesto total del ciclo de vida. No es raro que el mantenimiento del software utilize el 70% de los costos totales de ciclo de vida del producto de programación.

Al su vez, de este 70% el 60% es utilizado para mejoras, es decir, el 42% del presupuesto total, y el otro 40% para adaptaciones y correcciones. Algunos autores sugieren que el modelo apropiado del ciclo de vida para el desarrollo de software es:

DESARROLLO ——— EVOLUCION ——— EVOLUCION ———....

Este aspecto hace evidente que el objetivo principal del desarrollo de software debe ser la producción de sistemas de programación que faciliten su propio mantenimiento.

A) MANTENIMIENTO CORRECTIVO.

La primer actividad que se da en la fase de mantenimiento es la de adecuar el producto de software a las necesidades especificas y particulares de la aplicación, ya que durante las etapas de diseño se pudieron pasar por alto algunos detalles, tales como la validación de datos de entrada, adecuados mensajes de menúes, etc. A esta actividad desarrollada sobre el producto de software se le denomina **MANTENIMIENTO CORRECTIVO.**

B) MANTENIMIENTO ADAPTIVO.

Este tipo de mantemíniento se da debido a la alta velocidad de cambio de los productos computacionales. De manera muy periódica aparecen, por ejemplo, nuevos dispositivos periféricos, sistemas operativos, circuitos integrados, etc. Razón por la cual se hace necesario adaptar los programas a los nuevos elementos agregados al sistema.

C) MANTENIMIENTO PERFECTIVO.

El mantenimiento perfectivo es la actividad que se realiza sobre un producto de software el cual esta funcionando adecuadamente, sin embargo, si se le hicieran agregados, modificaciones o en general se perfeccionará el producto su utilidad se incrementaría. Generalmente esta actividad es

la que consume la mayor cantidad de esfuerzo del invertido en el mantenimiento del software.

D) MANTENIMIENTO PREVENTIVO.

La cuarta actividad del mantenimiento ocurre cuando el software es modificado para preveer futuras expansiones. Este actividad no es muy común dentro de lo que comprende el entorno de la creación de software.

Cada una de las actividades descritas anteriormente forman parte de la fase de mantenimiento, dentro del ciclo de vida del software, sin embargo, estas actividades no siguen una secuencia preestablecida, si no más bien, se suceden en diferentes momentos, dependiendo del entorno sobre el cual se aplique el producto.

4. ANALISIS DE UNA APLICACION PRACTICA

MICROPROCESADOR 6809

El micrprocesador seleccionado para este trabajo es el 6809 de Motorola, este es un MICROPROCESADOR de 8 Bits con caracterìsticas adecuadas para aplicar tècnicas de programaciòn estructurada. Contiene uno de los màs completos conjuntos de instrucciones para MICROPROCESADORES de 8 Bits

Dentro de sus caracterìsticas màs sobresalientes estàn las siguientes:

- Dos registros acumuladores de 8 Bits que pueden ser concatenados para formar uno de 16 bits.
- Direccionamiento de pàgina directa.
- 10 diferentes modos de direccionamiento.
- Multiplicaciòn de datos enteros no signados de 8 bits.
- Aritmètica de 16 Bits.

4.1. ARQUITECTURA INTERNA. En la figura **4.1** se muestra la arquitectura interna de este MICROPROCESADOR.

A continuaciòn se explican las funciones especìficas de cada uno de los registros internos:

- **ACUMULADORES (A, B y D).** Son registros de propòsito general denominados acumuladores y se utilizan para el desarrollo de operaciones aritmèticas y para la mayorìa de las operaciones entre la CPU y/o la memoria y perifèricos.

Los registros A y B son de 8 bits cada uno y pueden ser concatenados para formar el registro D; el cual resulta ser un acumulador de 16 bits, quedando el byte màs significativo **(Most significative Byte-MSB)** en el registro A.

- **REGISTRO DE PAGINA DIRECTA (Direct Page DP).** Este registro Tene la finalidad de hacer màs potente el modo de direccionamiento directo ya que durante la ejecuciòn de una instrucciòn en este modo los 8 bits màs significativos de la direcciòn del operando se encuentran en este registro y en el formato de la instrucciòn sòlo se encuentra el byte menos significativo **(Last Significative Byte-LSB).**

- **REGISTROS INDICES (IX, IY).** Los registros ìndice son utilizados en el modo de direccionamiento indexado. Estos registros son de 16 bits de longitud y generalmente contienen una direcciòn base a la cual se le va a sumar un

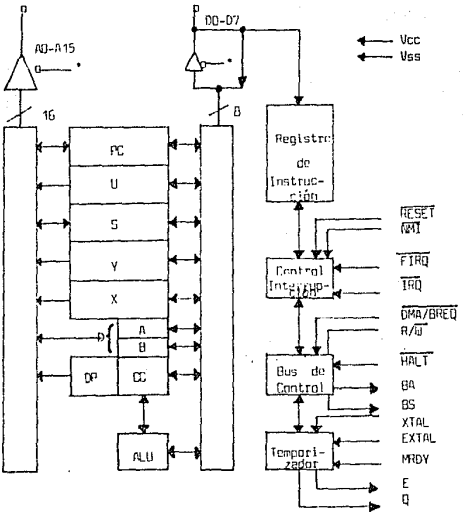


FIGURA 1.1 ARQUITECTURA INTERNA DEL MICROPROCESADOR 6809.

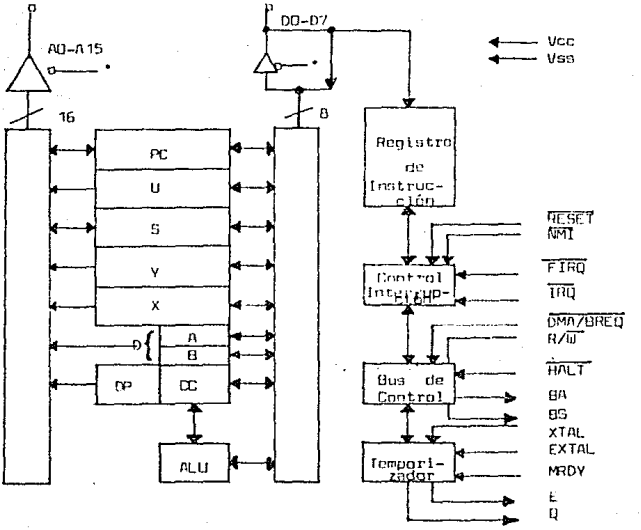


FIGURA 4.1 ARQUITECTURA INTERNA DEL MICROPROCESADOR 6809.

desplazamiento para obtener la dirección efectiva del dato a procesar. Durante la ejecución de algunas instrucciones el contenido de estos registros pueden ser incrementados o decrementados.

En general, cualquiera de los registros de 16 bits del microprocesador (registros apuntadores X, Y, U, y S) a excepción del PC pueden ser utilizados como registros **índice.**

- **APUNTADORES DE PILA (Stack Pointers U y S).** Estos registros son de 16 bits. El apuntador del stack del sistema (S) es utilizado automáticamente por el MICROPROCESADOR durante las llamadas a subrutinas y durante las interrupciones.

El apuntador del stack del usuario (U) es controlado exclusivamente por el programador.

Estos registros se utilizan generalmente para pasar parámetros hacia y desde diferentes subrutinas lo cual facilita las técnicas de programación estructurada.

- **CONTADOR DEL PROGRAMA (Program Counter PC).** Este registro de 16 bits de longitud es utilizado por el MICROPROCESADOR para apuntar a la localidad de memoria donde se encuentra la siguiente instrucción a ejecutar.

En algunas situaciones excepcionales el PC se puede utilizar como registro índice.

- **REGISTRO DE CONDICION (Condition Code Register CC).** El registro CC es de 8 bits de longitud y se utiliza para conocer el estado el procesador en cualquir momento. Esto es, cada vez que se ejecuta una instrucción el estado del procesador queda representado de una manera codificada en los 8 bits de este registro.

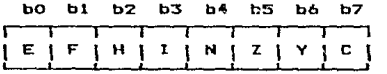


FIGURA 4.2 CODIFICACION DEL REGISTRO DE CONDICIONES.

De la figura anterior:

- **Bit 0 (C) acarreo.** Si este bit esta en un nivel alto (es decir en uno "1" lógico) indica que durante la última operación existió un acarreo dentro de la ALU. La bandera (C) también es utilizada para indicar que existió un

"prêstamo" en alguna de las siguientes instrucciones: CMP, NEG, SUB, SBC; en el caso de restas contiene el complemento del "prêstamo" (acarreo durante una resta aritmêtica).

- Bit 1 (V) sobreflujo. Esta bandera es la de sobreflujo (overflow) y es colocada a "1" lògico cuando una operaciòn aritmêtica entre los nûmeros no signados en complento a dos causa un overflow.

- Bits 2 (Z) cero. El bit 2 representa la bandera cero, èsta se activa (se pone a "1" lògico) cuando el resultado de la ûltima instrucciòn diò por resultado un cero. No hay que confundir:

a) Cuando el resultado es cero la bandera se pone en "1" y

b) Cuando el resultado es diferente de cero la bandera toma el valor "0".

- Bits 3 (N) negativo. Esta bandera contiene exactamente el valor del bit mas significativo (MSB) del resultado de la ûltima operaciòn. Asî un resultado negativo, en aritmêtica de complemento a 2, colocarà un "1" lògico en el bit 3 del registro de condiciones CC.

- Bits 4 (I) IRQ. El bit 4 del registro CC se utiliza para habilitar o deshabilitar las interrupciones enmascarables. Si este bit esta en "1" lògico el MICROPROCESADOR no atenderà la interrupciòn solicitada por el dispositivo que esta conectado a esta linea; en caso contrario, la interrupciòn serà atendida.

- Bits 5 (H) semiacarreo. Este bit representa el valor del acarreo del bit 3 dentro del ALU. La bandera H es utilizada por la instrucciòn DAA (ajuste adecimal, el cual se da cuando se realizan operaciones con nûmeros codificados en BCC); para realizar el ajuste a decimal de la ûltima operaciòn de suma aritmêtica realizada. En las operaciones de restas aritmêticas el valor de èsta bandera no esta definido.

- Bits 6 (F) FIRQ. Es semejante al bit 4, la bandera F se utiliza para habilitar/deshabilitar las probables interrupciones del dispositivo externo conectado al MICROPROCESADOR a traves de la linea FIRQ.

Cuando la bandera F està en un nivel bajo ("0" lògico), la interrupciòn serà atendida, en caso contrario serà ignorada por el MICROPROCESADOR.

- Bit 7 (E). La bandera E se activa una vez que todos los registros internos del MICROPROCESADOR fueron guardados (salvados) en una àrea de memoria llamada stack.

4.2. TERMINALES DEL MICROPROCESADOR. Para conocer adecuadamente un MICROPROCESADOR es necesario entender su arquitectura interna, pero además es importante también estar familiarizado con la distribución de pines ó ternimales; en otras palabras saber para què sirve cada una de las terminales del Circuito Integrado.

A continuación se da una descripción de la distribución de las 40 terminales de que dispone el MICROPROCESADOR 6809.

- Polarización (Vss, Vcc). Este MICROPROCESADOR posee únicamente dos pines (el 7 y el 1) para alimentarle de la potencia que necesita, estas señales son **Vcc**, la cual deber ser de +5.0 V con una tolerancia de ±5% mientras que Vss es la tierra.

- Bus de direcciones (A0 - A15). 16 terminales del chip son utilizadas como el bus de direcciones y son salidas de 3 estados; salidas porque el MICROPROCESADOR siempre es el que genera las direcciones y de 3 estados porque además de poder estar en cualquiera de los dos estados lògicos, pueden estar en un tercer estado el llamado de **alta impedancia**, lo cual significa que dichas lìneas estan virtualmente desconectadas del MICROPROCESADOR.

Con estas 16 terminales del bus de direcciones se pueden accesar hasta 64K direcciones de memoria, de manera directa (2**16 = 65 536 = 64K).

Cuando el MICROPROCESADOR no està utilizando estas 16 lìneas para transferir direcciones, las 16 terminales del bus estarán en un nivel alto, es decir, en "1" lògico. La señal R/W estará a "1" y la señal BS será "0"; a este estado se le denomina **"acces dummy"** o ciclo VMA.

Cuando la señal de Bus Disponible (**Bus Available - BA**) esta activa estas 16 lìneas se colocan automàticamente en el tercer estado.

- Bus de Datos (D0-D7). Estas 8 terminales son bidireccionales, es decir, que tanto el MICROPROCESADOR puede generar datos como también los puede recibir de la memoria o de un periférico.

- Señal de lectura/escritura (Read/Write - R/W). La señal R/W es una salida que indica la dirección ("de dónde vienen o hacia donde va") de los datos que circulan por el bus.

R/W = "1" significa que el MICROPROCESADOR esta tomando información de alguna localidad de memoria o de algún periférico, es decir, el sentido de los datos es hacia el MICROPROCESADOR. Si **R/W = "0"** los datos circulan desde el MICROPROCESADOR hacia algún dispositivo periférico.

- **RESET (Restaurar).** Cuando esta terminal del circuito integrado es activada, el MICROPROCESADOR automàticamente salta a la localidad de memoria $FFFE y $FFFF para ejecutar la instrucciòn contenida en dicha localidad. Por lo general la instrucciòn colocada en las direcciones $FFFE y $FFFF deben ser un salto hacia otra parte de la memoria, donde se localize la rutina de inicializaciòn del sistema.

- **Estado del bus y bus disponible (BS,BA).** Cuando la señal BA (BUS AVAILABLE - bus disponible) esta acitvada ("1" lògico) indica que los buses del MICROPROCESADOR estaràn colocados en el tercer estado (en alta impedancia), es decir, no estaràn disponibles.

Cuando la señal BS (Bus Status - Estado del Bus) es decodificada junto con la señal **BA**, representan el estado del MICROPROCESADOR. Ambas señales, **BA** y **BS**, son salidas.

- **Interrupciòn No Enmascarble (NMI).** Una transiciòn negativa sobre esta linea de entrada al MICROPROCESADOR generarà una secuencia de interrupciòn no enmascarable, es decir, que no podrà ser deshabilitada por programa y ademàs tiene prioridad sobre las otras dos interrupciones (IRQ y **FIRQ**).

En el momento de activarse esta linea el MICROPROCESADOR termina de ejecutar la instrucciòn que està ejecutando y en ese momento salva el estado del sistema en el stack del mismo; posteriormente ejecuta la rutina de atenciòn a la interrupciòn.

- **Peticiòn de Interrupciòn Ràpida (FIRQ).** Un nivel bajo en este pin de entrada iniciarà una secuencia de interrupciòn ràpida, siempre y cuando el bit (F) del registro de condiciòn (**CCR**) este en "1" lògico.

Esta interrupciòn tiene prioridad sobre la otra interrupciòn enmascarable, la IRQ; y es ademàs màs ràpida ya que sàlo guarda en el stack del sistema los contenidos de los registros (CC)y (SP).

- **Peticiòn de Interrupciòn (IRQ).** Si aparece un "0" lògico en esta terminal del MICROPROCESADOR, se iniciarà una secuencia de peticiòn de interrupciòn, siempre y cuando el bit (I) del registro de condiciones este en "0" lògico. Esta interrupciòn es màs lenta que la **FIRQ** debido que guarda en el stack del sistema el estado compelto del MICROPROCESADOR.

- **Señales XTAL, EXTAL, y E, Q.** Las dos primeras se utilizan para alimentar las señales de tiempo que marcaràn la velocidad a la que trabajarà el MICROPROCESADOR. Las segundas son señales de sincronizaciòn que tienen diferentes aplicaciones en la conexion del MICROPROCESADOR con el mundo

exterior, estas indican, dependiendo su nivel lógico, en qué instante de tiempo son válidos los datos que aparecen en las buses del MICROPROCESADOR.

- **MRDY.** Esta señal se utiliza en conjunto con las señales **E** y **Q.** Cuando esta terminal esta en un nivel bajo las señales de salida **E** y **Q** se hacen más largas, lo que implica que los datos sobre las buses del MICROPROCESADOR esten más tiempo disponibles para que puedan ser utilizados por un periférico.

- **Señal DMA/BRFQ.** Cuando esta señal de entrada se encuentra activa ("0" lógico), el MICROPROCESADOR deja las buses de direcciones y de datos disponibles para que los maneje otro dispositivo externo, éste por lo regular es un controlador de **DMA's (Accesos Directos a Memoria)**, el cual se utiliza para transferir datos de la memoria principal del sistema a un dispositivo prifèrico, pero sin pasar por el MICROPROCESADOR; también suele utilizarse para el refresco de memorias dinámicas.

A partir de el instante en que esta señal se activa el MICROPROCESADOR deja disponibles sus buses, pero sólo por un determinado tiempo (14 ciclos del bus) al término de los cuales el MICROPROCESADOR toma de nuevo el control sobre dichos buses.

4.3. MODOS DE DIRECCIONAMIENTO DEL MICROPROCESADOR 6809.

Como ya se mencionó, un modo de direccionamiento es una forma de indicarle al MICROPROCESADOR en qué dirección de memoria se encuentra el operando al cual se le debe aplicar la operación indicada por el Código de Operación. Es decir, para una misma intrucción existen (en la mayoría de las veces) varias formas de indicar el operando.

En particular, el MICROPROCESADOR 6809 tiene 59 instrucciones básicas y 13 modos de direccionamiento diferente. En total combinando las instrucciones básicas con los modos de direccionamiento, dan 1464 instrucciones posibles.

En el Apendice "A" se presenta la lista de instrucciones de este microprocesador.

A continuación se explican cada uno de los diferentes modos de direccionamiento:

a) INHERENTE o IMPLICITO. En este modo de direccionamiento, el código de operación contiene implícitamente el operando, el cual es siempre es un registro interno, y por

lo tanto las instrucciones en este modo de direccionamiento tienen vacío el campo del operando.

Las instrucciones en este modo utilizan una sola localidad de memoria para alojarse, es decir, un byte.

Ejemplo de estas instrucciones son: ABX, DAA, SWI, etc. (las instrucciones que permiten este modo de direccionamiento no permiten ningún otro modo diferente).

b) INMEDIATO. En el direccionamiento inmediato el campo del operando es ocupado por el valor efectivo a procesar y no representa una dirección.

Un ejemplo de este modo es:

```
LDA  #$4C
```

Esta instrucción indica cargar (LOAD) el número hexadecimal 4C al registro acumulador A.

Los símbolos # y $ indican, direccionamiento inmediato y número hexadecimal, respectivamente.

c) EXTENDIDO. En este modo de direccionamiento el campo del operando contiene la dirección de la localidad de memoria donde se encuentra el dato a operar.

Un ejemplo de esta instrucción es:

```
STY  $032F
```

La cual indica almacenar (STORAGE) el contenido del registro Y en la localidad de memoria $032F

c) EXTENDIDO INDIRECTO. En este modo el operando indica a partir de cuál localidad de memoria se encuentra la dirección del dato a procesar.

Ejemplo:

```
LDA ($FA14)

($FA14) = $2701

($2701) = $EF
```

En este ejemplo la instrucción indica que el contenido de las direcciones $FA14 y $FA15 contienen a su vez la dirección de la localidad de memoria en donde, al ejecutarse, el acumulador (A) quedará cargado con el contenido de la localidad $2701, es decir (A) = $EF.

e) DIRECCIONAMIENTO RELATIVO. El direccionamiento relativo se utiliza unicamente en las instrucciones de salto.

La dirección de la próxima instrucción a ejecutarse se calcula sumando el contenido del PC más el desplazamiento indicado, esta suma se deposita en el PC:

(PC) = (PC) + desplazamiento

y por lo tanto, ejecutará la instrucción contenida en esa localidad de memoria. Todo esto, siempre y cuando la condición de la prueba sea afirmativa; en caso contrario se ejecutará la instrucción que le sigue secuencialmente.

f) CONTADOR DE PROGRAMA RELATIVO. Este modo se puede utilizar para crear programas reubicables dentro de la memoria, ya que la dirección de un operando se calcula a partir de la suma del contenido actual del PC más un cierto desplazamiento y el contenido del PC no se afecta. Se puede utilizar para programas reubicales ya que los datos se pueden especificar en direcciones que están a una cierta distancia en relación con una instrucción y no en una direcciones efectiva.

g) DIRECTO. El direccionamiento directo o de página directa es semejante al direccionamiento extendido, pero con la excepción de que en el directo sólo se especifican los 8 bits menos significativos de la localidad de memoria donde se encuentra el operando y el byte mas significativo de dicha dirección se toma del contenido del registro **DP (Direct Page).**

Por ejemplo, si el contenido del (DP) = $01, la instrucción STA $00, guardaría el contenido del acumulador (A) en la localidad de memoria $0100.

Hay que notar que el contenido de el registro de **DP** se une con el valor indicado por la instrucción y no se suma, como se pudiera pensar.

h) DE REGISTRO. En este modo de direccionamiento los operandos siempre son algunos de los registros internos del MICROPROCESADOR.

La ventaja de este modo de direccionamiento es que las instrucciones ocupan únicamente 2 bytes:

* **El byte 1:** Donde se aloja el codigo de operacion.

* **El byte 2:** Donde se encuentran codificados los registros.

A este segundo byte se el denomina **POSTBYTE.**

i) **DIRECCIOMANIENTO INDEXADO.** En todas las variantes del direccionamiento indexado alguno de los registros internos de 16 bits de MICROPROCESADOR es utilizado para calcular la dirección efectiva de memoria donde se encuentra el operando.

En las instrucciones con este modo de direccionamiento, el postbyte define:

* La variante o tipo bÀsico de indexación.

* El registro a ser utilizado.

La caracteristica principal de las instrucciones en este modo de direccionamiento es que la dirección de la localidad de memoria donde se encuentra el operando se calcula sumando el contenido del registro apuntador màs un desplazamiento.

El registro apuntador puede ser cualquiera de la registros de 16 bits del MICROPROCESADOR, es decir: **X, Y, S, U** y en algunos casos el PC.

Por otra parte el desplazamiento se puede especificar de diferentes maneras, en total 5, y son las variantes de este modo de direccionamiento las que se explican a continuación:

- **Desplazamiento Cero.** En este modo el desplazamiento vale cero, por lo tanto la dirección del operando es la apuntada por registro indicado en la propia instrucción.

```
LDD 0,X

LDD ,X
```

Aambas instrucciones realizan la misma operación.

- **Desplazamiento Constante.** En esta variante la dirección efectiva se obtiene al sumar el contenido del registro apuntador especificado, con el nùmero o constante indicada explicitamente en la instrucción.

Este numero debe estar expresado en complemento a 2's.

```
LDA 23, Y
STX -2, S
```

En este modo el contenido del registro apuntador no es alterado.

- **Desplazamiento por Acumulador.** En este modo el valor de la dirección se obtiene al sumar el contenido del registro apuentador màs el contenido de cualquiera de los 3 acumuladores.

```
STA  A,Y
LDX  B,X
LDB  D,V
```

- **Auto Incremento/Decremento.** En esta variante la dirección del operando es la apuntada por el registro índice correspondiente, pero después de ejecutar la instrucción el contenido de este registro se incrementa/decrementa en 1 o 2 unidades, dependiendo de lo indicado por la instrucción.

```
1)      STA,    X+
2)      LDB,    X++
3)      LDB,    --U
4)      STA,    - S
```

En el primer ejemplo, el contenido del registro "A" se guardará en la dirección apuntada por el registro "X", posteriormente el contenido de "X" se incrementará automáticamente una unidad.

En el ejemplo 3 el contenido de la dirección indicada por "U" se cargará en el registro "B" y posteriormente el contenido de "U" se decrementará automaticamente en dos unidades.

- **Indexado Indirecto.** En este modo el valor de la dirección del operando se encuentra en la localidad de memoria indicada por el contenido del registro índice más el desplazamiento. Esto es, en la localidad de memoria obtenida por la suma del contenido del registro índice más el desplazamiento, se encuentra a su vez otra dirección (realmente en ésta y la posterior consecutiva) la cual apunta al operando.

Si antes de la ejecución se tiene:

```
(A) = $FF    ;    ($110) = $25
(X) = $0100  ;    ($111) = $6C  ;  ($256C) = $35
```

Y la instrucción es : **LDA ($10,X)**

Al final de la ejecución se tendrá:

```
(A) = $35
```

Por ejemplo, la instrucción: **LDA 5,PCR**

Cargará en el acumulador "A" el contenido de la localidad que se encuentra a 4 palabras de memoria después de donde se encuentra contenida la propia instrucción.

4.4. DESCRIPCION DE LAS INSTRUCCIONES DEL MICROPROCESADOR 6809.

En esta sección se describirán algunas de las instrucciones más representativas de este microprocesador.

Antes de describir las instrucciones es conveniente establecer algunas convenciones respecto a la nomenclatura utilizada:

r Cualquier de los siguientes registros A ó B.

R Cuarquiera de los siguientes registros X, Y, S, PC, U.

% Indica un número representado en binario.

$ Indica que el número que le sigue esta expresado en hexadecimal.

$M Dirección de memoria expresada totalmente (4 números hexadecimales o 16 bits).

$m Byte menos significativo (8 bits de una dirección).

( ) Cuando un registro o la dirección de una localidad de memoria se pone entre paréntesis se debe interpretar como el contenido de la dirección o del registro.

Cuando una instrucción se pueda expresar en varios modos de direccionamiento los códigos de operación correspondientes a cada modo (para la misma instrucción) aparecen en el siguiente formato:

MNEMONICO	MODO OP. CODE	MODO OP. CODE	. . .
			. . .
.	.	.	.
.	.	.	.
.	.	.	.

En la primer columna llmada MNEMONICO aparece el mnemónico asociado a dicha instrucción y en las siguientes columnas aparecen los códigos de operación correspondientes a cada modo permitido para esa instrucción.

En cada una de las instrucciones se mostrarà la forma en que dicha instrucciòn afecta al registro de condiciones (CC):

(BLANCO) La operaciòn no afecta a la bandera.

X El estado del bit es afectado segùn el resultado de la operaciòn.

1 La bandera correspondiente se activa.

O La bandera no se activa.

INSTRUCCIONES.

*** ABA** (A) ———> (A) + (B)

Suma el contenido del acumulador B al contenido del acumulador A, dejando el resultado en el registro A. Unicamente admite el modo de direccionamiento inherente.

Ejemplo:

Antes de ejecutar las instrucciones	**Instrucciòn**	**Despuès de ejecutar las instrucciones**
(A) --- $C2	ABA	(A) ---- $61
(B) --- $5F		(B) ---- $5F

*** ABX** (X) ———> + (B)

Suma el contenido del registro acumulador B a el contenido de registro indice X. El contenido del registro B es interprètado como un nùmero no-signado.

Esta instrucciòn no afecta a el registro de condiciones.

Ejemplo:

Antes de ejecutar las instrucciones	**Instrucciòn**	**Despuès de ejecutar las instrucciones**
(X) --- $10228	ABX	(X) --- $10CB
(B) --- $A0		(B) --- $A0

*** ADC.** Suma el contenido de las direcciones especificada con el contenido del acumulador A o B.

Mnemonico	**Inmediato**	**Dirc**	**Extendido**	**Indx/Indirc**
ADCA	89	99	B9	A9
ADCB	C9	D9	F9	E9

Ejemplo 1: Modo de direccionamiento Inmediato.

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(B) --- $50 (CC) --- $0000	ADCB #$9E	(B) --- $EE (CC) --- $0000

Ejemplo 2: Modo de direcciomamiento Directo.

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(D) --- $FF (A) --- $00 ($FF31) --- $8C	ADCA $ 31	(DP) --- $FF (A) --- $8C ($FF31) --- $8D

Ejemplo 3:

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(X) --- $8801 ($8801) --- $01 (A) --- $A7 (CC) --- $	ADCA ,X	(X) --- $8801 ($8801) --- $01 (A) --- $A8 (CC) --- $

$ ADD. Suma el contenido de la dirección especificada con el contenido del acumulador D.

Mnemonico	Inmediato	Dirc	Extendido	Indx/Indirc
ADDA	8B	9B	BB	AB
ADDA	CB	DB	FB	EB
ADDA	C3	D3	F3	E3

Ejemplo 1: Modo de direccionamiento Indirecto.

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(X) --- $8801 ($8801) --- $54 ($3F54) --- $00 (A) --- $FF (CC) --- %	ADDA, (X)	(X) --- $8801 ($8801) --- $3F54 ($3F54) --- $00 (A) --- $FF (CC) --- %

Ejemplo 2 : Modo de direccionamiento Inmediato.

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(A) --- $23 (B) --- $FA (C) --- %0000	ADDD #$0158	(A) --- $FB (B) --- $7B (CC) --- %0000

* **AND.** Realiza la operacion AND con alguno de los acumuladores o con el registro de condiciones y el operando indicado.

Mnemonico	Inmediato	Dirc	Extendido	Indx/Indirc
ANDA	84	94	B4	A4
ANDB	C4	D4	F4	E4
ANDCC	1C	--	--	--

Las instrucciones ANDCC solo permite direccionamiento inmediato.

Ejemplo 1:

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(A) --- %01011010 ($0807) --- %000000 (CC) --- %0000000	ANDA $0807	(A) --- %00000000 ($0807) --- %000000 (CC) ---% 000000

Ejemplo 2:

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(CC) --- %01110110	ANDCC #%11010111	(CC) --- %01010110

* **ASL.** Desplaza el contenido del operando un lugar hacia la izquierda colocando el primer bit a cero logico ("0")

Mnemonico	Inmediato	Dirc	Extendido	Indx/Indirc
ASL	--	08	78	38
ASLA	48	--	--	--
ASLB	58	--	--	--

Ejemplo:

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(A) --- %11010111	ASLA	(A) --- %1010110
(CC) --- %00000001		(CC) --- %0000000

* SALTOS O BIFURCACIONES.

El único modo de direccionamiento que aceptan los saltos es el direccionamiento relativo.

Todos los saltos utilizados en este procesador tienen la siguiente estructura:

DIRECCION DE SALTO = DESPLAZAMIENTO + CONTENIDO DEL PC.

Si la condición que se prueba es verdadera el control del programa pasa a la instrucción almacenada en la localidad de memoria que indica en la segunda parte de la instrucción de salto más el contenido del PC. En caso contrario el control continua en la instrucción siguiente a la del salto.

Ninguna instrucción de este grupo afecta las banderas de estado, unicamente prueba su valor.

En la siguiente tabla se muestran los diferentes tipos de condiciones disponibles para el MICROPROCESADOR 6809.

Mnemónico	Código oper.	Salta si se cumple la condición
BCC	4	C = 0
BCS	25	C = 1
BNE	26	Z = 0
BEQ	27	Z = 1
BPL	2A	N = 0
BMI	2B	N = 1
BVC	28	V = 0
BVS	29	V = 1

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(A) --- $02	LDA $0076	(A) --- $00
(CC) ----$00	ANDA #$74	(CC) --- $04
($0076) --- $35	BEQ $0300	
	LDB #$04	

En este ejemplo, si el contenido de la localidad $0076 es $8B se ejecutará la rutina almacenada a partir de la direccion $0300+(PC), en caso contrario la ejecución continuará secuencialemente.

*** BIT.** Esta instrucción realiza la operación lógica AND entre el contenido de alguno de los acumuladores y el otro operando; las banderas se afectan según el resultado, sin embargo, no se altera el contenido del acumulador.

BITA realiza la misma operación lógica que **ANDA**, solo que con la primera el contenido del acumulador A no se altera.

Mnemonico	Inmediato	Dirc	Extendido	Indx/Indirc
BITA	85	95	B5	A5
BITB	C5	C5	F5	E5

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(A) --- $5B	BITA $0807	(A) --- $5B
($0807) --- $00		($0807) --- $00
(CC) --- $00		(CC) --- $04

*** BSR.** Esta instrucción se utiliza para invocar subrutinas. Ejecuta un salto incondicional a la localidad de memoria que obtiene al sumar el contenido actual del PC más el desplazamiento indicado en el operando de la propia instrucción. Además almacena en el stack del sistema el contenido del PC.

*** RTS.** Con esta instrucción se regresa de alguna subrutina. El PC es cargado con los dos bytes que se encuentran en la parte superior del stack del sistema.

*** CMP.** Realiza la comparación entre el registro especificado y el contenido del segundo operando. Lo que realmente sucede es que se resta el contenido de la localidad especificada del contenido del registro indicado en la instrucción. Ninguno de los dos operandos es afectado, solo las banderas de estado se activan correspondientemente.

Para esta instrucción se pueden utilizar registros de 8 y 16 bits.

Mnemonico	Inmediato	Dirc	Extendido	Indx/Indirc
CMPA	81	91	B1	A1
CMPB	C1	D1	F1	E1
CMPD	1083	1093	10B3	10A3
CMPS	118C	119C	11BC	11AC
CMPU	1183	1193	11B3	11A3
CMPX	8C	9C	BC	AC
CMPY	108C	109C	10BC	10AC

Antes de ejecutar las instrucciones	Instrucción	Después de ejecutar las instrucciones
(D) --- $0000	LDD $1132	(D) --- $0000
($1132) --- $0264	CMPD #$0264	($1132) --- $0264
(CC) --- $00	BEQ $001A	(CC) --- $04

Para este ejemplo, solo si el contenido de la localidad $0000 es $0264 el control pasará a la instrucción alojada en la dirección $001A.

4.5. ANALISIS DE UNA APLICACION PRACTICA.

En esta sección se aplicarán las técnicas de la Ingeniería de SOFTWARE para el desarrollo de un producto de programación, que según lo expuesto, se localizaría en la categoría de **proyecto pequeño**, sin querer decir esto que se dejen de aprovechar las características de la Ingeniería de Software para su desarrollo.

En el capítulo anterior se expuso el ciclo de vida de un producto de programación de una manera en que cada fase y subfase estaban bien delimitadas, sin embargo, en una aplicación real estas delimitaciones no son totalmente palpables. Por esta razón se utilizan diferentes formatos para describir, hasta donde es posible, el ciclo de vida del producto estudiado. En otras palabras, no se especificará paso a paso el procedimiento seguido para llegar a la conclusión de una etapa, sino más bien se presentará el documento que se desprende de cada una.

4.5.1. DEFINICION DEL SISTEMA.

4.5.1.1. DEFINICION DEL PROBLEMA. Desarrollar un paquete de programas que auxilien en el desarrollo de aplicaciones encaminadas a la explotación de las características de la microcomputadora MP9 modelo I.

4.5.1.2. JUSTIFICACIONES DEL SISTEMA. Este sistema ayudará a ejercitar ó conocer la programación en lenguajes de bajo nivel a la comunidad estudiantil de ingeniería de la ENEP **Aragón;** mostrará un concepto actual y nuevo en el desarrollo de SOFTWARE, lo que redundará en una elevación en el nivel académico de esta institución; mostrará con un ejemplo real los conceptos tratados en los capítulos anteriores; proporcionará un conjunto de programas (cuya presentación será como un paquete terminado) que son de gran utilidad cuando se desarrollan sistemas en configuraciones básicas. Por otra parte, presentar un sistema con el cual sustentar la tésis de grado para el autor.

4.5.2. METAS DEL SISTEMA. Se pretende intruducir al usuario en el manejo del MICROPROCESADOR y sus periféricos, así como en la adecuada explotación de estos sistemas a través de la versatilidad que implica que sean programables, es decir, que el usuario piense en un MICROPROCESADOR configurado como un elemento de auxilio, pero no de una sola aplicación sino de una gran cantidad de éstas gracias a que es programable en muy alto grado.

4.5.3. RESTRICCIONES DEL SISTEMA. El sistema esta restringido básicamente por la poca capacidad de memoria RAM instalada (1 KB) con la que cuenta el hardware; el paquete no residirá en memoria no volátil, sino que se deberá cargar cada vez que se encienda el equipo y se desee utilizar dicho conjunto de programas.

Esta restricción es intencional por parte del autor con la esperanza de que este trabajo sea mantenido y modificado, el cual es el objetivo principal de la ingeniería de SOFTWARE, tema central del presente trabajo.

4.5.4. FUNCIONES QUE PROPORCIONA EL SISTEMA.

El paquete desarrollará varias funciones, éstas se clasifican en:

I.- MANEJO DE PERIFERICOS:

a) Escritura en pantalla.
b) Envío de señales a los LED's.
c) Lectura de los switches.

II.- CALCULOS ARITMETICOS:

a) Sumas.
b) Restas.
c) Multiplicaciones.
d) Divisiones.

III.- UTILERIAS:

a) Búsqueda de un byte en una tabla.
b) Ordenamiento de una tabla.
c) Comparación entre dos tablas.
d) Obtención del máximo de una tabla.
d) Obtención del mínimo de una tabla.

4.5.5. CARACTERISTICAS DEL USUARIO.

Las personas que deseen explotar este paquete deben tener conocimientos bàsicos acerca de:

1. Manejo de microcomputadoras,
2. Dominio de los sistemas de numeraciòn binario y hexadecimal,
3. Teoría bàsica sobre electrònica digital y convertidores CAD's y CDA's,
4. Nociones sobre programaciòn en lenguaje de màquina y lenguaje ensamblador y
5. Estar familiarizado con la operaciòn de la microcomputadora MP9-I.

Las caracterìsticas antes descritas son las ideales que debe poseer todo usuario para aprovechar al màximo el sistema, sin embargo, es factible que usuarios que no satisfagan las 3 primeras caracterìsticas utilizen exitosamente el sistema.

4.5.6. AMBIENTE DE DESARROLLO.

El medio ambiente sobre el cual corre el sistema es la microcomputadora MP9-I con las siguientes caracterìsticas:

- Monitor.
- Teclado.
- Tablilla maestra donde se encuentran contenidas:

  a) La memoria RAM.
  b) El MICROPROCESADOR.
  c) Los convertidores (CAD y CDA).
  d) Los switches.
  e) Los Led's.

En lo que se refiere al software, el sistema debe ser inicializado y estar trabajando bajo el Monitor CIUNAM 1.0 (ò versiones posteriores).

4.5.7. ESTRATEGIA DE SOLUCION.

El sistema se desarrollò estructurando los procedimientos de lo general a lo particular, es decir, se define el sistema de una manera general, posteriormente se describen los mòdulos que lo conforman y por àltimo se especifican los procedimientos y programas que conforman cada mòdulo.

La técnica de programación que se utilizó es la **programación estructurada** y la técnica de documentación de procesos es la de **diagramas de flujo**.

4.5.8. FUENTES DE INFORMACION.

Las principales fuentes de información utilizadas para la planeación, desarrollo y mantenimiento del sistema fueron:

- Manuales del MICROPROCESADOR 6809 y del PIA 6821.
- Manual del usuario de la MC MP9-I.
- Bibliografía sobre Ing. de SOFTWARE.
- Bibliografía sobre Prog. en lenguaje ensamblador.
- Bibliografía sobre algoritmos y estructuras de datos.
- Programas en lenguaje ensamblador desarrollados en el pasado por el autor de este trabajo.
- Revistas técnicas.
- Programas fuentes del Monitor CIUNAM 1.0.

4.5.9. CRITERIOS DE ACEPTACION DEL SISTEMA.

Antes del desarrollo del presente trabajo, el autor desarrolló diversos programas para la explotación del Hardware descrito. Probó varios modelos del sitema MISA, hasta obtener la versión final. Los criterios básicos para su liberación fueron:

- El sistema funciona como un paquete terminado con todas las opciones y procedimientos funcionado tal y como se planeó.

- El sistema cumple con un 95% de eficiencia respecto a su definición original.

- El software está libre de errores.

4.5.10. DEFINICION DE REQUISITOS.

A) **INTRODUCCION.** El sistema se dividirá en tres módulos funcionales, cada uno de los cuales operará independientemente de los demás. Sin embargo, todos los módulos se pueden comunicar entre sí para intercambiar y compartir datos, los cuales serán comunes para todos los módulos.

La forma de introducir los datos al sistema será, básicamente, clocando los bytes en las correspondientes localidades de memoria por medio del programa monitor.

La obtención de los datos se realizará accesando las localidades de memoria reservadas precisamente para los resultados ó accesando las localidades en dónde se encontrarán los datos fuentes, si es que se desea alguna consulta, agregado ó modificación.

Además del procedimiento anterior, existirá otra forma de leer ó escribir información en la memoria del sistema, esta será a través del primer módulo del sistema (MANEJO DE PERIFERICOS), el cual permitirá consultar, modificar ó agregar datos al sistema (ver guía del usuario para más detalles).

Una vez alojados los datos en la memoria del sistema, a éstos se les podrán aplicar diferentes procesos:

* Los proporcionados por el sistema MISA ó

* Los definidos por el usuario.

El sistema MISA se debe planear y estructurar de manera tal que los datos almacenados en la microcomputadora puedan ser procesados por programas del usuario ó por los que el sistema tiene disponibles.

Los programas proporcionados por el sistema MISA permitirán realizar procesos del tipo aritmético tales como sumas, restas, multiplicaciones y divisiones con números hexadecimales sin signo (CALCULOS ARITMETICOS); el otro tipo de procesos estarán orientados a las búsquedas y comparaciones entre bytes y/o tablas (UTILERIAS).

4.5.11. DESCRIPCION DE LA INFORMACION.

La memoria del sistema está dividida en diferentes áreas:

a) De datos del usuario.

b) De programas del sistema.

c) De datos temporales del sistema.

d) De resultados.

e) De programas del usuario.

Cada una de estas diferentes áreas estan perfectamente delimitadas y por lo tanto deben ser respetadas por el usuario para conseguir un adecuado funcionamiento del sistema. En otras palabras, mientras el usuario este utilizando el sistema MISA nunca deberá accesar las áreas b) y c). Además de que en el área a) deberá colocar únicamente

la información correspondiente, sin rebasar los límites de la misma.

El área **e)** puede ser administrada por el usuario de la manera que sea más conveniente para su aplicación, respetando siempre los límites definidos.

El área para **datos temporales** del sistema es un conjunto de localidades de memoria contiguas que son administradas por el sistema. Estas se utilizan para colocar resultados parciales de los diferentes procesos y para almacenar datos que son constantes y comunes para algunos módulos. Tiene una estructura de stack (pila), siendo muy dinámica la información almacenada en esta área, razón por la cual los datos contenidos aquí no tienen ninguna validez para el usuario.

El área para los **programas del sistema** es un espacio de localidades sucesivas donde se alojan los programas del sistema. Esta área es la más delicada de todas ya que si se altera su contenido el sistema MISA no realizará las funciones para las que fué diseñado.

Existe un área distinta para alojar **programas propios** del usuario, con la finalidad de que los datos usados en el sistema MISA puedan también ser utilizados por otro u otros programas (de tamño limitado) independientes de éste.

4.5.12. DESCRIPCION FUNCIONAL.

Como ya se ha mencionado, el sistema MISA proporciona varias funciones, las cuales quedan enmarcadas en tres diferentes módulos:

1 Manejo de Periféricos.

a) **Escritura en Display.** La primera parte de este módulo permite escribir carcateres ASCII desplegables en la pantalla de la microcomputadora. Para lograrlo es necesario colocar los códigos ASCII del mensaje que se desea desplegar en las localidades de memoria especificadas. Para facilitar esta actividad en la **Guía del Usuario** se proporciona un formato especial y una tabla del código ASCII.

El formato que se menciona es un mapeo de la pantalla de la microcomputadora y por lo tanto en éste se distribuirá el texto a desplegar. Del mismo se obtendrán las direcciones de las localidades de memoria donde es necesario cargar los códigos ASCII del texto, para que éste se despliegue tal y como se plasmó en el formato.

b) **Envio de Señales a los LED's.** Esta parte del primer mòdulo permite enviar (escribir) datos en los LED's. Es a través de estas rutinas que el usuario puede enviar información hacia el exterior. Los datos siempre serán canalizados hacia los LED's. Esto se hace posible gracias a que el método de E/S que utiliza el procesador 6809 es el de MEMORIA MAPEADA.

c) **Lectura de los interruptores.** Es a través de esta parte del primer mòdulo que el usuario puede introducir información desde el exterior hacia el sistema (además del teclado). Los datos son leidos de los interruptores, cada determinado tiempo, el cual es definido por el usuario (FRECUENCIA DE LECTURA). Cada vez que es leido un byte es colocado en el àrea de memoria para datos, a partir de la dirección especificada.

El usuario debe tener cuidado de no exceder el Area especificada al intentar introducir màs datos que la capacidad permitida.

*** Cálculos Aritméticos.** En esta sección se pueden realizar las operaciones aritmeticas bàsicas (sumas, restas, multiplicaciones y divisiones) entre nùmeros enteros, sin signo y en base hexadecimal. La longitud de los operandos es de 8 ó 16 bits según se especifique en la operación correspondiente.

Para ejecutar las operaciones de esta sección el usuario debe proporcionar los operandos a procesar,

Una vez ejecutada la operación los resultados serán alojados en la memoria de resultados y en este momento el usuario puede disponer y accesar a èstos.

*** Utilerias.** En este modùlo se podràn realizar bùsquedas de datos dentro de una tabla, obtención del byte cuyo valor absoluto sea el màximo y/o el mínimo de una tabla y comparar dos tablas de la misma longitud.

4.5.13. FASE DE DISEÑO.

El sistema MISA està orientado para correr sobre un medio ambiente determinado, el cual (hasta el momento en que se concluyò este trabajo) no tiene ningùn dispositivo de almacenamiento externo, razòn por la cual se deberà cargar cada vez que la microcomputadora sea encendida.

El sistema MISA esta estructurado como se muestra en el siguiente diagrama:

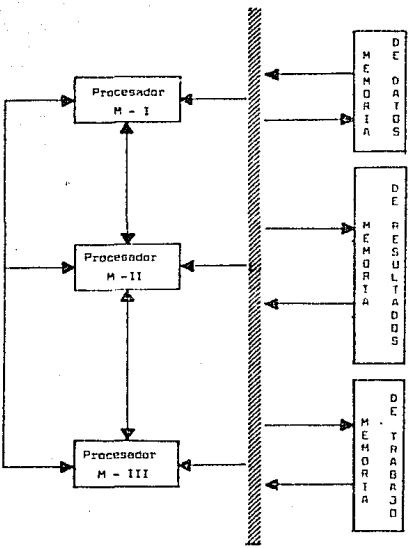


FIGURA 4.2. DIAGRAMA A BLOQUES DEL SISTEMA MISA.

Como se puede observar esta constituído por tres módulos funcionales (Manejo de Periféricos, CÁlculos Aritmeticos y Utilerías) y además comparte tres conjuntos de información (Memoria de Datos, Memoria de Resultados y Memoria de Trabajo).

- Módulo para el Manejo de Periféricos. Este módulo está dividido a su vez en tres (3) submódulos:

MANEJO DE PERIFERICOS

ESCRITURA EN PANTALLA.
ENVIO DE SEÑALES A LOS LED's.
LECTURA DE LOS SWITCHES.

A continuación se presentan los codigos fuente y objeto, las direcciones de inicio y fin de las rutinas y la cantidad de memoria utilizada por cada uno de los submódulos:

ESCRITURA EN PANTALLA.

NOMBRE: DESPLIEGA.

DESCRIPCION:

Programa para desplegar datos en la pantalla de la microcomputadora.

ESPECIFICACIONES:

a) Los bytes almacenados a partir de la dirección $ 0000 y hasta la dirección $ 01FF (512 bytes) serán desplegados en la pantalla.

ENTRADAS:

a) Bytes que conforman el texto a desplegar.

SALIDAS:

La salida es el desplegado del texto sobre la pantalla.

VARIABLES:

```
X          --> Apuntador asociado a los datos
Y          --> Apuntador asociado a el video
U          --> Contador de datos (max=512).
A          --> Datos a enviar.
```

CONSTANTES:

```
INC_DATOS --> Dirección de inicio de los
              datos. ($0000)
INC_RESUL --> Dirección de inicio del
              resultado ($B000)
CANT_DATOS --> Cantidad de datos ($01FF).
               (512 datos).
```

ETIQUETAS:

```
LOOP1 --> $____
```

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A,U,X,Y, CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirse al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa DESPLIEGA.

        LDY   #$INIC_DATOS
        LDX   #$INIC_RESUL
        LDU   #$0000
        CLRA

LOOP1 : LDA   ,Y+
        STA   ,X+
        LEAU $1,U
        CMPU #$CANT_DATOS
        BNE   LOOP1

        SWI
```

ENVIO DE SENALES A LOS LED's.

NOMBRE: ENVIA.

DESCRIPCION:

Programa para enviar los datos (bytes) contenidos a partir de la direeciòn $ 0000 y hasta la direcciòn $01FF a los LED's, a la frecuencia especificada por la magnitud del numero almacenado en la direcciòn $ 0000.

ESPECIFICACIONES:

a) Los bytes almacenados a partir de la direcciòn $ 0000 y hasta la direcciòn $ 01FD (510 bytes) seràn enviados a los LED's.

ENTRADAS:

a) Bytes que serán enviados a los LED's.
b) La frecuiencia a la que se desee sean enviados los datos. Esta se especificarà colocando un nùmero en las direcciones $01FE y $01FF (LSB y MSB respectivamente).

SALIDAS:

La salida es el desplegado de los datos en los LED's.

VARIABLES:

A --> Datos a enviar.
B --> Contador de datos.
Y --> Contador del retardo.
X --> Apuntador asociado a los datos.

CONSTANTES:

INIC_DATOS --> Direcciòn de inicio de datos ($0000)
PUERTOB --> Direcciòn donde se encuentra mapeado el dato de salida.
FREQ --> Direcciòn donde se encuentra el valor del rertardo entre el envio de un dato y otro. ($01FE y $01FF para el LSB y el MSB respectvamente). El valor màximo del rctardo es #$FFFF.

ETIQUETAS:

LOOP1 --> $____
LOOP2 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A,B,X,Y,CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirse al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa ENVIA.

        ;Programación del PIA.

        CLR  $8403          ;Accesa registro de dirección.

        LDA  #$FF           ;Se programa al registro B
        STA  $8402          ;como SALIDA.

        LDA  #$04           ;Se programa el registro de
        STA  $8403          ;control para poder accesar
                            ;el registro de datos.
        ;Inicialización.

        CLRA
        CLRB
        LDY  $FREQ
        LDX  #$INIC_DATOS

        ;Cuerpo del programa

LOOP1:  LDY  $FREQ          ;Frecuencia de desplegado.
        LDA  ,X+            ;Carga el dato y avanza el
                            ;apuntador

LOOP2:  LEAY -$1,Y          ;Rutina de retardo.
        CMPY #$0000
        BNE  LOOP2

        STA  $PUERTOB       ;Despliega el dato
        INCB
        CMPB #$CANT_DATOS   ;¿Se terminaron los datos?
        BNE  LOOP1          ;No: salta a LOOP1.

        SWI
```

LECTURA DE LOS SWITCHES.

NOMBRE: LECTURA.

DESCRIPCION:

Programa para leer datos de los switches a una determinada frecuencia.

ESPECIFICACIONES:

a) Los datos leídos se almacenarán a partir de la dirección $0000 y hasta la dirección $01FD.
b) La cantidad máxima de datos que se pueden leer es de $01FD (510 bytes).
c) La frecuencia de lectura se determinará a partir de la magnitud del dato almacenado en las direcciones $01FE y $01FF con el LSB y el MSB respectivamente.

ENTRADAS:

a) Datos tomados de los switches.
b) La frecuencia a la que se desee sean leídos los datos.

SALIDAS:

La salida serán los datos leídos, los cuales se almacenarán a partir de la dirección $0000.

VARIABLES:

```
A          --> Datos a leer.
B          --> Contador de datos.
Y          --> Contador del retardo.
X          --> Apuntador asociado a
               los datos leídos.
```

CONSTANTES:

```
INIC_RESUL --> Dirección de inicio de datos
               leídos ($0000).
PUERTOB --> Dirección donde se encuentra
            mapeado el dato de entrada.
FREQ    --> Dirección donde se encuentra
            el valor del rertardo entre la
            lectura de un dato y otro.
            ($01FE y $01FF para el LSB y el
            MSB respectvamente). El valor
            máximo del retardo es #$FFFF.
```

ETIQUETAS:

LOOP1 --> $____
LOOP2 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A,B,X,Y,CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirce al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa LECTURA.

        ;Programación del PIA.

        CLR   $8401           ;Accesa registro de dirección.
        CLR   $8400

        LDA   #$04            ;Se programa al registro A
        STA   $8401           ;como ENTRADA.

        ;Inicialización.

        CLRA
        CLRB
        LDY   $FREQ
        LDX   #$INIC_RESUL

        ;Cuerpo del programa

LOOP1:  LDY   $FREQ           ;Frecuencia de desplegado.
        LDA   $PUERTOA        ;Carga el dato y avanza el
                              ;apuntador
        STA   ,X+             ;Despliega el dato

LOOP2:  LEAY  -$1,Y           ;Rutina de retardo.
        CMPY  #$0000
        BNE   LOOP2

        INCB
        CMPB  #$CANT_DATOS    ;¿Se terminaron los datos?
        BNE   LOOP1           ;No: salta a LOOP1.

        SWI
```

- CAlculos Aritmeticos. Este mòdulo cuenta con cuatro funciones:

CALCULOS ARITMETICOS.

SUMAS.
RESTAS.
MULTIPLICACIONES.
DIVISIONES.

Los operandos sobre los cuales funcionan adecuadamente estos procedimientos deben tener las siguientes caracterìsticas:

a) Enteros.
b) No signados.
c) Expresados en hexadecimal.

A continuaciòn se presentan las descripciones, còdigos fuente y objeto, direcciones de inicio y fin de las rutinas y la cantidad de memoria utilizada por cada una de estas.

SUMAS.

NOMBRE: SUMA

DESCRIPCION:
Programa para sumar un conjunto de datos almacenados en una tabla.

ESPECIFICACIONES:

a) El tamaño de la tàbla es de 20 datos 8 bits (2 bytes) cada uno.

b) Los datos deben de estar almacenados en forma contigua a partir de la direcciòn $0000.

c) Los nùmeros se deben expresar en notaciòn hexadecimal, ser enteros y sisn signo.

d) La direcciòn a partir de la cual se almacenarà el resultado es $0380.

g) Si los datos a sumar son menos de 128, llenar el resto de localidades con $00.

ENTRADAS:

a) Los datos a sumar, almacenados en forma de tabla.

SALIDAS:

La única salida es el resultado de la suma de todos los datos contenidos en la tabla.

VARIABLES:

A --> Acumulador de sumandos.
X --> Apuntador asociado al resultado.
Y --> Apuntador asociado a los datos.

CONSTANTES:

INIC_DATOS --> Dirección de inicio de datos ($0000).
INCI_DATOS --> Dirección de inicio del resultado ($0380).
CANT_DATOS --> Canyidad de datos a sumar.

DUMMY --> Constante utilizada para contabilizar el acarreo en los bytes superiores al LSB del resultado. Su valor es $ 0000.

ETIQUETAS:

LOOP1 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A, X, Y, CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirce al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa SUMA.

        CLRA                ;Limpia los acumuladores.
        CLRB

        CLR  $0380          ;Limpia las localidades
        CLR  $0781          ;del resultado.

        LDX  $INC_RESUL     ;Apuntador del resultado
        LDY  $INC_DATOS     ;Apuntador de los datos

        ANDC #$FE           ; Pone en cero el bit de acarreo

LOOPI : LDA  ,X
        ADDA ,Y
        STA  ,X
        LEAX $1,X
        LDA  ,X
        ADCA #$DUMMY
        STA  ,X
        LDX  $INC_RESUL
        LEAY $1,Y
        CMPY #$CANT_DATOS
        BNE  LOOPI

        SWI
```

RESTAS.

NOMBRE: RESTAS.

DESCRIPCION:

Programa para obtener la diferencia entre dos números.

ESPECIFICACIONES:

a) El tamaño de los operandos (minuendo y sustraendo) es de 16 bits cada uno.

b) El minuendo se almacenará en las direcciones $0000 y $0001.

c) El sustraendo se acenará en las direcciones $0002 y $0003.

d) El resultado se acenará en las direcciones $0380 y $0381.

e) Los números se deben expresar en notación hexadecimal, ser enteros y sisn signo.

f) El MSB del primer dato debe estar contenido en la primer localidad de memoria, el LSB de éste en la siguiente.

g) Si el minuendo o el sustraendo tienen un sólo byte, el MSB se debe llenar con $00.

ENTRADAS:

a) Minuendo.
b) Sustraendo.

SALIDAS:

a) El resultado de restar el sustraendo del minuendo.
b) En el registro de condiciones (bit 0), si este bit es igual a "0" lógico después de la resta, significa que existió un préstamo.
c) Después de la ejecución el contenido del bit 3 del registro de condiciones indica el signo del resultado (1 -> negativo y 0-> positivo).

VARIABLES:

A	--> Acumulador para el miuendo.
X	--> Apuntador asociado al minuendo.
y	--> Apuntador asociado al sustraendo.
U	--> Apuntador asociado al resultado.

CONSTANTES:

INIC_MINUENDO	---> Dirección de inicio del resultado ($0000).
FIN_MINUENDO	--> Dirección de inicio del resultado ($0001).
INIC_SUSTRAEN	--> Dirección de inicio del sustraendo ($0002)
INIC_RESUL	--> Dirección de inicio del resultado ($0380).

ETIQUETAS:

LOOP1 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A,X,U,Y,CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martinez Gonzalez.

OBSERVACIONES: Referirce al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa RESTA.

        CLRA
        LDX   $INC_MINUENDO
        LDY   $INC_SUSTRAENDO
        LDU   $INIC_RESUL

        ANDC #$FE               ;Pone en cero el bit de acarrreo

LOOP1 : LDA   ,X                ;Resta el LSB
        SBCA  ,Y
        STA   ,U
        INCX                    ;Avanza los apuntadores a
        INCY                    ;los MSB
        INCU
        CMPX #$FIN_MINUENDO     ;¿Es el fin del minuendo?
        BNE   LOOP1             ;No: salta a LOOP1.

        SWI
```

MULTIPLICACIONES.

NOMBRE: MULTIPLICA.

DESCRIPCION:

Programa para multiplicar dos nùmeros de 16 bits cada uno.

ESPECIFICACIONES:

a) El tamaño de los operandos (multiplicando y multiplicador) serà de 16 bits cada uno.

b) Los datos deben de estar almacenados en forma contigua a partir de alguna direcciòn valida dentro de la memoria para datos .

c) Los nùmeros se deben expresar en notaciòn hexadecimal, ser enteros y sisn signo.

d) El MSB del primer dato debe estar contenido en la primer localidad de memoria, el LSB de èste en la siguiente, y asi sucesivamente para cada uno de los restantes datos de la tabla.

f) La direcciòn a partir de la cual se almacenarà el resultado (èsta debe estar localizada al menos a cuatro localidades de distancia antes del fin de la memoria de resultados).

ENTRADAS:

a) Direcciòn de inicio del multiplicando.
b) Direcciòn de inicio del multiplicador.
c) El resultado siempre se almacenarà a partir de la direcciòn: $ 0000.

SALIDAS:

La ùnica salida es el producto obtenido de multiplicar el contenido de la direcciòn del multiplicador por el contenido de la direcciòn del multiplicando.

VARIABLES:

D --> Acumulador del multiplicador.
Y --> Acumulador del multiplicando.
X --> Apuntador asociado al resultado.
Y --> Apuntador asociado al multiplicador.

CONSTANTES:

INIC_MULTIPLICANDO --> Dirección de inicio del multiplicando.
INIC_MULTIPLICADOR --> Dirección de inicio del multiplicador.
INIC_RESUL --> Dirección de inicio del resultado.

DUMMY --> Constante utilizada para contabilizar el acarreo en los bytes superiores al LSB del resultado. Su valor es $ 0000.

ETIQUETAS:

LOOP1 --> $____
LOOP2 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: D, X, Y, CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirce al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa MULTIPLICA.

        CLRD
        LDU   #$INIC_MULTIPLICADOR
        LDX   #$INC_RESUL
        STU   ,X              ;Considera el multiplicador
        LDY   $#0000
        ANDC  #$FE            ;Pone en cero el bit de acarreo

LOOP2 : LDD   ,X
        ADCD  $INIC_MULTIPLICANDO ;Suma el multiplicando
        STD   ,X
LOOP1 : LEAX  $2,X                ;Avanza al siguiente byte
        LDD   ,X                  ;del resultado.
        ADCD  #$DUMMY             ;Suma el acarreo
        STD   ,X
        CMPX  $FIN_RESUL          ;¿Barrio todo el resultado?
        BNE   LOOP1               ;No: salta a LOOP1
        LDX   $INIC_RESUL         ;Sí: apunta al inicio del
                                  ;resultado
        LEAY  $1,Y                ;Incrementa el contador del
                                  ;multiplicador.
        CMPY  $INIC_MULTIPLICADOR ;¿Contador=multiplicador?
        BNE   LOOP2               ;No: salta a LOOP2.

        SWI
```

DIVISIONES.

NOMBRE: DIVISION.

DESCRIPCION:

Programa para dividir dos datos de 16 bits.

ESPECIFICACIONES:

a) El tamaño de los operandos (dividendo y divisor) será de 16 bits cada uno.

b) Los números se deben expresar en notación hexadecimal, ser enteros y sisn signo.

c) La dirección a partir de la cual se almacenará el resultado es $0380.

d) El dividendo se almacenará a partir de la dirección $0000, primero el LSB.

e) El divisor se almacenará a partir de la dirección $0002, primero el LSB.

f) El resultado se almacenará en la dirección $0380.

g) El residuo se almacenará en la dirección $0381.

ENTRADAS:

a) Dividendo.
b) Divisor.

SALIDAS:

Las salidas son el cocsiente y el residuo que se obtinen de dividir el dividendo entre el divisor.

VARIABLES:

D --> Acumulador de 16 bits.

CONSTANTES:

RESUL --> Dirección del resultado ($0380).

RESIDUO --> Dirección del residuo ($03B1).

INIC_DIVIDENDO --> Dirección de inicio del dividendo ($0000).
INIC_DIVISOR --> Dirección de inicio del divisor ($0003).

LONGOTUD --> Es la mitad de la longitud del dividendo ($08).

ETIQUETAS:

LOOP1 --> $____
LOOP2 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: D,Y,CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirse al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa DIVISION.

        CLRD
        ANDC #$FE           ; Pone en cero el bit de acarrreo

        LDA  #$LONGITUD
        STA  $RESIDUO
        LDD  $DIVIDENDO

LOOP2 : ASLB
        ROLA
        CMPA  $DIVISOR
        BCS   LOOP1
        SUBA  $DIVISOR
        INCB
LOOP1 : DEC  $RESIDUO
        BNE  LOOP2
        STD  $RESUL

        SWI
```

- El módulo de utilerías esta orientado a la manipulación de datos y a su vez se divide en cinco (5) submódulos:

UTILERIAS.

BUSQUEDA DE UN BYTE EN UNA TABLA.
ORDENAMIENTO ASCENDENTE DE UNA TABLA.
COMPARACION DE DOS TABLAS.
OBTENCION DEL MAXIMO VALOR EN UNA TABLA.
OBTENCION DEL MINIMO VALOR EN UNA TABLA.

A continuación se presentan los codigos fuente y objeto, las direcciones de inicio y fin de las rutinas y la cantidad de memoria utilizada por cada uno de los submódulos:

BUSQUEDA DE UN BYTE EN UNA TABLA.

NOMBRE: BUSQUEDA.

DESCRIPCION:
Programa para determinar si un dato de ocho bits se encuentra dentro de una tabla de datos.

ESPECIFICACIONES:

a) Los bytes almacenados a partir de la dirección $ 0000 constituirán la tabla.

b) La longitud de la tabla es de 20 bytes.

c) La dirección del dato a buscar es ($0015)

ENTRADAS:

a) Datos que constituyen la tabla
b) Dato a buscar.

SALIDAS:
La salida es un $01 en la localida $0380 si es que el dato si se encuentra dentro de la tabla, o un $00 en la misma dirección en caso contrario.

VARIABLES:

```
B          --> Almacena el dato a comparar.
A          --> Bandera que indica la
               existencia del dato dentro
               de la tabla.
X          --> Apuntador asociado a la
               tabla.
```

CONSTANTES:

```
INIC_TABLA --> Dirección de inicio de
               la tabla ($0000).
FIN_TABLA  --> Dirección última de la tabla

ESTA       --> Dirección para almacenar
               el resultado.
```

ETIQUETAS:

```
LOOP1 --> $____
LOOP2 --> $____
```

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A, B, X, CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirse al manual del usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa BUSQUEDA.


        LDX   $INIC_TABLA
LOOP2:  LDB   ,X+           ;Toma un dato de la tabla y
                            ;Avanza el apuntador.
        CMPB  ,$DATO        ;Compara el dato.
        BEQ   LOOP1         ;¿Es igual? Sí: termina.
        CMPX  $FIN_TABLA    ;No: ¿Es el fin de la tabla?
        BNE   LOOP2         ;No: salta a LOOP2
        LDA   #$00
        STA   $ESTA         ;No está el dato
        JMP   FIN
LOOP1:  LDA   #$01
        STA   $ESTA         ;Sí está el dato.
  FIN:  SWI
```

ORDENAMIENTO ASCENDENTE DE UNA TABLA.

NOMBRE: SORT.

DESCRIPCION:

Programa para ordenar en forma ascendente una tabla de datos de 8 bits de longitud cada uno.

ESPECIFICACIONES:

a) Los bytes almacenados a partir de la dirección $ 0000 y hasta la dirección $0014 conformarán la tabla.

b) La longitud máxima de la tabla es de 20 bytes.

c) La tabla ordenada se almacenará en la misma área de memoria.

ENTRADAS:

a) La tabla de datos.

SALIDAS:

La salida se almacena sobre los datos originales, es decir, la tabla ordenada se almacenará a partir de la dirección $0000.

VARIABLES:

A	--> Dato menor.
B	--> Dato mayor.
X	--> Apuntador asociado a la tabla.

CONSTANTES:

INIC_TABLA --> Dirección de inicio de datos.

FIN_TABLA --> Dirección última de los datos.

ETIQUETAS:

LOOP1 --> $____
LOOP2 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A, B, X, CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martinez González.

OBSERVACIONES: Referirce al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa SORT.

        CLRA
        CLRB
        LDX  #INIC_TABLA

LOOP1:  CMPX #$FIN_TABLA      ;¿Es el fin de la tabla?
        BEQ  FIN              ;Si: salta al fin
        LDA  ,X+              ;Carga un dato y avanza
                              ; el apuntador.
        LDB  ,X               ;Carga el siguiente dato
        CMPA ,X
        BLE  LOOP2            ;Si es <= salta a LOOP2
        EXG  A,B              ;Si es mayor, intercambialos
        STA  ,X-              ;Reacomodalos en la tabla
        STB  ,X++
        JMP  LOOP1            ;Inicia nuevamente
LOOP2:  STB  ,X-              ;Si es <= No intrcambia
        STA  ,X++
        JMP  LOOP1            ;Inicia nuevamente.
  FIN:  SWI
```

COMPARACION ENTRE DOS TABLAS.

NOMBRE: COMPARA.

DESCRIPCION:

Programa para comparar dos tabalas de la misma longitud y determinar si son iguales.

ESPECIFICACIONES:

a) Los bytes almacenados a partir de la dirección $ 0000 y hasta la dirección $0014 conformarán la tabla 1.

b) Los bytes almacenados a partir de la dirección $ 0015 y hasta la dirección $0028 conformarán la tabla 2.

c) La longitud máxima de las tablas es de 20 bytescada una.

c) El resultado se almacenará en la localidad $0380, si ésta contiene un $01, indica que las tablas son iguales, en caso contrario contendrá un $00.

ENTRADAS:

a) Las tablas de datos.

SALIDAS:

La salida se lamacena en la localidad $0380.

VARIABLES:

```
A       --> Dato de la tabla 1
B       --> contadorr asociado a
            la longitud de las tablas.
X       --> Apuntador asociado a
            la tabla 1.
Y       --> Apuntador asociado a
            la tabla 2.
```

CONSTANTES:

```
INIC_TABLA1 --> Dirección de inicio de
                la tabla 1.
INIC_TABLA2 --> Dirección de inicio de
                la tabla 2.

RESUL --> Dirección del resultado.
```

ETIQUETAS:

LOOP1 --> $____
LOOP2 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A, B, X, CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirse al manual del usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa COMPARA.

        CLRA
        CLRB
        LDX   $INIC_TABLA1
        LDY   $INIC_TABLA2

        LDB   #$LONGITUD
LOOP2:  DECB
        CMPB  #$00              ;Se terminaron los datos
        BEQ   LOOP1             ;SI: salta a LOOP1
        LDA   ,X+               ;No: compara los datos
        CMPA   ,Y+              ;que ocupan la misma posi-
                                ;ción en ambas tablas y avan-
                                ;za los apuntadores de ambas.
        BEQ   LOOP2             ;Si son iguales salta a LOOP2
        LDA   #$00              ;SI son diferentes termina.
        STA   $RESUL
        JMP   FIN
LOOP1:  LDA   #$01
  FIN:  SWI
```

OBTENCION DEL MAXIMO DE UNA TABLA.

NOMBRE: MAXIMO.

DESCRIPCION:

Programa para obtener el máximo valor de una tabla.

ESPECIFICACIONES:

a) Los bytes almacenados a partir de la dirección $ 0000 y hasta la dirección $0014 conformarán la tabla.

b) La longitud máxima de la tabla es de 20 bytes.

c) El máximo valor se depositará en la dirección $0380.

ENTRADAS:

a) La tabla de datos.

SALIDAS:

La salida es el dato que tiene mayor valor absoluto dentro de la tabal.

VARIABLES:

A --> Dato menor.
B --> Dato mayor.
X --> Apuntador asociado a la tabla.

CONSTANTES:

INIC_TABLA --> Dirección de inicio de datos.

FIN_TABLA --> Dirección última de los datos.

RESUL --> Dato máximo.

ETIQUETAS:

LOOP1 --> $____
LOOP2 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A, B, X, CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martínez González.

OBSERVACIONES: Referirse al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa MAXIMO.

        CLRA
        CLRB
        LDX   $INIC_TABLA

LOOP1:  CMPX  #$FIN_TABLA      ;¿Es el fin de la tabla?
        BEQ   FIN              ;Si: salta al fin
        LDA   ,X+              ;Carga un dato y avanza
                               ; el apuntador.
        LDB   ,X               ;Carga el siguiente dato
        CMPA  ,X
        BLE   LOOP2            ;Si es <= salta a LOOP2
        EXG   A,B              ;Si es mayor, intercambialos
        STA   ,X-              ;Reacomodalos en la tabla
        STB   ,X++
        JMP   LOOP1            ;Inicia nuevamente
LOOP2:  STB   ,X-              ;Si es <= No intrcambia
        STA   ,X++
        JMP   LOOP1            ;Inicia nuevamente.
        LDA   $FIN_TABLA       ;Toma el dato mayor
        STA   $RESUL
  FIN:  SWI
```

OBTENCION DEL MINIMO DE UNA TABLA.

NOMBRE: MINIMO.

DESCRIPCION:

Programa para obtener el minimo valor de una tabla.

ESPECIFICACIONES:

a) Los bytes almacenados a partir de la dirección $ 0000 y hasta la dirección $0014 conformarán la tabla.

b) La longitud màxima de la tabla es de 20 bytes.

c) El minimo valor se depositarà en la dirección $0380.

ENTRADAS:

a) La tabla de datos.

SALIDAS:

La salida es el dato que tiene mayor valor absoluto dentro de la tabal.

VARIABLES:

A --> Dato menor.
B --> Dato mayor.
X --> Apuntador asociado a la tabla.

CONSTANTES:

INIC_TABLA --> Dirección de inicio de datos.

FIN_TABLA --> Dirección àltima de los datos.

RESUL --> Dato minimo.

ETIQUETAS:

LOOP1 --> $____
LOOP2 --> $____

AFECTA A:

a) Ninguna Rutina/Subrutina.
b) Registros Internos: A, B, X, CC.

FECHA DE CREACION: 12 de mayo de 1986.

AUTOR: Roberto Martinez González.

OBSERVACIONES: Referirce al manual dl usuario para conocer su forma de operación.

DIRECCION DE INICIO:

DIRECCION DE TERMINACION:

CANTIDAD DE BYTES:

CODIGO FUENTE:

```
        ;Programa MINIMO.

        CLRA
        CLRB
        LDX   $INIC_TABLA

LOOP1:  CMPX  #$FIN_TABLA       ;¿Es el fin de la tabla?
        BEQ   FIN               ;Si: salta al fin
        LDA   ,X+               ;Carga un dato y avanza
                                ; el apuntador.
        LDB   ,X                ;Carga el siguiente dato
        CMPA  ,X
        BLE   LOOP2             ;Si es <= salta a LOOP2
        EXG   A,B               ;Si es mayor, intercambialos
        STA   ,X-               ;Reacomodalos en la tabla
        STB   ,X++
        JMP   LOOP1             ;Inicia nuevamente
LOOP2:  STB   ,X-               ;Si es <= No intrcambia
        STA   ,X++
        JMP   LOOP1             ;Inicia nuevamente.
        LDA   $INIC_TABLA       ;Toma el dato mayor
        STA   $RESUL
  FIN:  SWI
```

4.6. MANUAL DE USUARIO.

La presente sección explica la forma general de operar el sistema MISA versión 1.0.

Para lograr una adecuada explotación del sistema, el usuario debe seguir los pasos que se presentan a continuación:

a) Definir el problema que se desea resover.

b) Analizar las funciones proporcionadas por el sistema MISA.

c) Evaluar la factibilidad de resolver el problema, total o parcialmente, con el adecuado uso del sistema MISA.

d) Si la desición del punto anterior es afimativa, pasar al siguiente inciso. En caso contrario, iniciar la planeación y definición de la alternativa a seguir para solucionar dicho problema.

e) Plasmar por escrito la definición del problema, incluyendo los siguientes aspectos:

- Datos de entrada
- Flujo del proceso
- Datos de salida

f) Dibujar, haciendo uso de diagramas de bloques, las rutinas adicionales al sistema MISA que se necesitan y el flujo de información que se requiere para enlazar los módulos.

g) Desarrollar los procedimientos previamente definidos. En la mayoría de los casos sólo fungirán como interfase entre el sistema MISA y la forma de representación de los resultados.

h) Haciendo uso de los comandos del programa Monitor. Cargar el sistema MISA en la memoria RAM de la máquina.

i) Realizar la ejecución de las rutinas a través de los comandos del monitor (vea la Guía de Operación de la Microcomputadora MP9-I) o a través de los procedimiento que haya definido.

j) Finalmente, es aconsejable registrar los resultados por escrito.

4.7. MANTENIMIENTO.

A continuación se muestra el formato de mantenimiento para cada una de las rutinas que conforman el sistema:

FECHA DE MODIFICACION. Es la fecha en que se terminó de realizar la modificación a la rutina.

NUMERO DE MODIFICAION. Aquí se pone el numero consecutivo de la modificación que corresponde, es decir, a la primer modificación le corresponde el número "1", a la segunda el número "2", y así consecuentemente.

GRADO DEL CAMBIO:

( ) MODIFICACION DE VARIABLES
( ) MODIFICACION DE CONSTANTES
( ) MODIFICACION DEL ALGORITMO

Aquí se especifica el grado o alcance de la modificación realizada.

DESCRIPCION DE LA MODIFICACION: En este punto se hace una breve descripción de las carcaterísticas de la modificación, agregado o de cualquier cambio en general a la rutina. Si alguna constante o variable es modificada, en este punto debe quedar registrado dicho cambio.

MOTIVOS DE LA MODIFICACION: En este inciso se describirá brevemente la razón por la que se efectuó la modificación.

Para cada cambio realizada a alguna de las rutinas del sistema deberá existir una hoja con todos y cada uno de los puntos mencionados anteriormente.

5. CONCLUSIONES.

A lo largo del desarrollo de este trabajo se llegó a las siguientes conclusiones:

1) El desarrollo de programas en lenguajes de bajo nivel proporcionan un excelente medio para:

a) Entender la filosofía de la programación de computadoras como un concepto, más que como la codificación de un algoritmo. El hecho de programar en lenguajes de este nivel implica un mayor esfuerzo para el programador, sin embargo, la disciplina que se deriva de éste es muy adecuada y benéfica, ya que el desarrollador cuenta con sólo un conjunto de instrucciones, relativamente, reducido y por lo tanto se requiere de una mayor inversión de recursos en los procesos de codificación y depuramiento, lo que en el plazo corto permite desarrollar programas en lenguajes de alto nivel de una manera más rápida, eficiente y productiva.

b) Al estar, este tipo de lenguajes, intimamente ligados al hardware de la computadora, permite asimilar claramente su funcionamiento.

2) La utilización de una metodología para el desarrollo de sistemas de información no está limitada a aplicaciones que se basen en lenguajes de alto nivel, ni tampoco se limita al tamaño del sistema generado.

3) La utilización de una metodología para el desarrollo de sistemas de información proporciona una base adecuada y fundamentada para mantener vigente a un sistema con un esfuerzo significativamente menor al requerido cuando no se utiliza una metodología.

4) El desarrollo de sistemas de información con técnicas de Ingeniería de Software trae varias consecuencias:

a) Se requiere un mayor esfuerzo en todas las fases del ciclo de vida de un sistema.

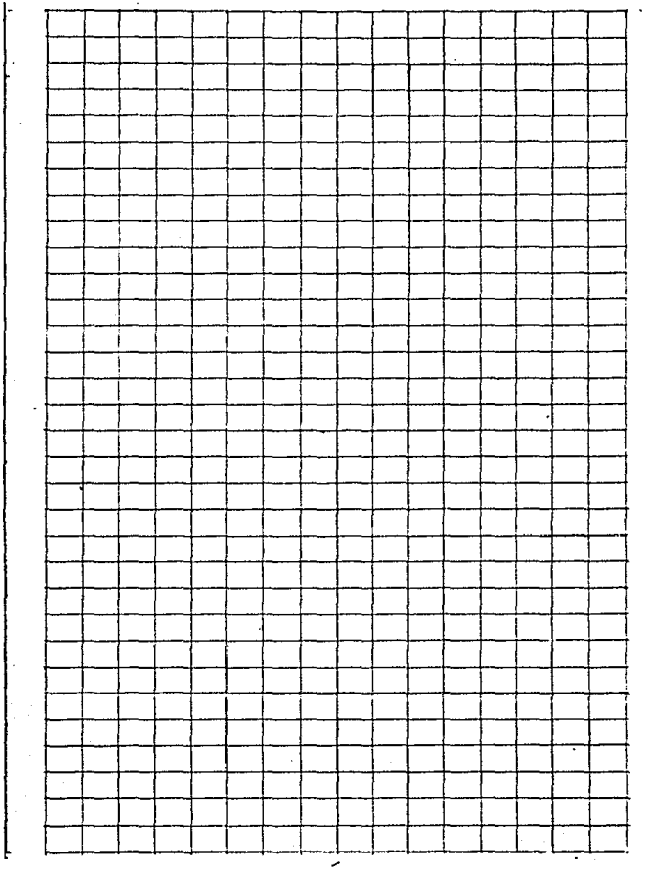
b) Quedan bien delimitadas las fases del proceso de creación de un producto de programación y por lo tanto esta actividad compleja se puede dividir en varias tareas menos complicadas.

c) Existe, en todo momento, una documentación de todos los aspectos, tanto objetivos como subjetivos, del ciclo de vida del sistema, lo que permite entender al diseñador original, al usuario final y a las personas, que sin ser diseñadoras, están involucradas en el mantenimiento del mismo.

d) Los costos para mantener la vigencia del sistema se bajan considerablemente.

e) Es factible hacer un presupuesto del costo del proyecto.

f) Por último, a todas las personas que apliquen estas técnicas les permitirá estar al día en el medio informático internacional.



BIBLIOGRAFIA

1. SOFTWARE PSYCOLOGHY (HUMAN FACTORS IN COMPUTER AND INFORMATION SYSTEMS).
   Ben Scheiderman.

2. SOFTWARE QUALITY EVALUATION.
   John Hanna.

3. SOFTWARE READILITY, PRINCIPLES AND PRACTICES
   Glenford J. Meyers.

4. THE ART OF SOFTWARE TESTING.
   Glenford J. Meyers.

5. STANDARIZED DEVELOPMENT OF COMPUTER SOFTWARE.
   Robert C. Tausworthe.

6. SOFTWARE METRICS.
   Tom Gilb.

7. SYSTEMATIC PROGRAMMING: AN INTRODUCTION.
   Wirth N.

8. SOFTWARE ENGENIEERING: A PRACTICE APPROACH.
   Myers W.

9. 6809 ASSEMBLY LANGUAGE PROGRAMMING.
   Lance A. Leventhal.

10. INTRODUCCION A LOS MICROPROCESADORES.
    Edward V. Ramirez.

11. HANDBOOK OF SOFTWARE QUALITY ASSURANCE.
    G. Gordonn Schulmeyer.

12. ELEMENTS OF SOFTWARE SCIENCE.
    Halsted, M.H.