ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

TESIS PREVIA LA OBTENCION DEL TITULO DE
INGENIERIA ELECTRONICA Y CONTROL

" SIMULADOR DEL MICROCONTROLADOR 8096 "

Realizada por: *Patricia Proaño Rosado*
Dirigida por: *Ing. Fernando Flores*

Quito, Agosto de 1990

CAPITULO I

INTRODUCCION

La creciente demanda del uso de los microcontroladores en los procesos industriales ha creado la necesidad de tener un programa que permita simular el funcionamiento del microcontrolador sin necesidad de implementar el hardware correspondiente.

Esta tesis tiene por objeto simular el microcontrolador 8096 en un computador personal IBM o compatible con la siguiente configuración:

- 640 Kb mínimo de memoria
- monitor monocromático o color
- disk drive
- disco duro para acelerar la simulación
- coprocesador matemático para obtener una mayor velocidad de ejecución (opcional)
- DOS versión 3.0 o mayor

En la actualidad este tipo de computador es de fácil acceso al usuario puesto que su uso es generalizado.

El lenguaje Quick Basic fue escogido para el desarrollo de

este simulador puesto que guarda las caracteristicas del basic normal el cual es uno de los lenguajes más difundidos en la actualidad.

Con este lenguaje podemos realizar una programación estructurada que permite una mayor comprensión del programa realizado, además nos presenta facilidades en la programación, compilación y depuración de dicho programa.

El microprocesador 8096 a simularse es un sistema digital áltamente integrado que contiene en un solo chip las siguientes unidades:

- CPU de 16 bit.
- ROM de 8 kbytes.
- RAM de 232 bytes.
- Multiplicador/Divisor en hardware de 16x16 en multiplicación y 32/16 en división.
- 6 modos de direccionamiento.
- Unidad de entrada salida de alta velocidad de 8 líneas:
  4 líneas que establecen como entrada o como salida especificamente.
  4 líneas que pueden programarse como entrada o como salida.

- Conversor análogo digital de 10 bits.
- Pórtico Serial que tiene modo de transmisión Full-Duplex.
- Entrada para 40 líneas de entrada/salida.
- 8 fuentes programables del sistema de interrupción.
- Salida por modulación de ancho de pulso.
- Reloj de tiempo real.

El simulador de este microprocesador permite depurar un programa sin necesidad de cambiar el archivo fuente, modificando el contenido de las localidades de memoria en las que se encuentra el programa, especialmente en procesos repetitivos, y de esta manera lograr los resultados deseados.

Es necesario la utilización del ensamblador CROSS 16, que nos proporciona un archivo hexadecimal en formato INTEL de 16 bits a partir del cual se procede ha desensamblarlo para luego ejecutarlo.

Para el desarrollo del simulador se ha estructurado el estudio de la siguiente manera:

En el capítulo II se hace una descripción del microcontrolador 8096 en el que se describen las características de su

arquitectura, modos de direccionamiento, interrupciones, software, set de instrucciones y aplicaciones.

En el capítulo III se describe la manera de simular el CPU, la memória , los registros, el conversor A/D, los Timers y el pórtico serial.

En el capítulo IV se explican el programa principal y los subprogramas que son:

- Programa de inicialización
- Programa de Menus
- Programa de desensamblado
- Programa de Conversor Análogo Digital
- Programa de Pórtico Serial

En el capítulo V se describen los ejemplos realizados y sus resultados obteniéndose de esto las conclusiones y recomendaciones.

CAPITULO II

DESCRIPCION DEL MICROCONTROLADOR 8096

Desde la introducción del primer microcontrolador 8048 en 1976 por la INTEL, la evolución de estos ha continuado, así en 1980 aparece la familia del 8051 ofreciendo muchas ventajas adicionales en comparación con el anterior, y en 1983 aparece una nueva generación de microcontroladores que indudablemente presentan características muy especiales que facilitan su utilización en procesos de control y que se lo conoce como el 8096.

El microcontrolador 8096 tiene las siguientes características:

- CPU de 16 bit.
- ROM de 8 kbytes.
- RAM de 232 bytes.
- Multiplicador/Divisor en hardware de 16x16 en multiplicación y 32/16 en división.
- 6 modos de direccionamiento.
- Unidad de entrada salida de alta velocidad de 8 líneas:
  4 líneas que establecen como entrada o como salida especificamente.
  4 líneas que pueden programarse como entrada o como

salida.

- Conversor análogo digital de 10 bits.
- Pórtico Serial que tiene modo de transmisión Full-Duplex.
- Entrada para 40 líneas de entrada/salida.
- 8 fuentes programables del sistema de interrupción.
- Salida por modulación de ancho de pulso.
- Reloj de tiempo real.

2.1 ARQUITECTURA

En la figura No. 1 se muestra el diagrama de bloques de este microprocesador. Para fines de estudio se lo puede considerar como constituido por algunas secciones tales como: la CPU, líneas de entrada/salida (I/O) de alta velocidad programables, Conversor análogo Digital, puerto serial de comunicaciones, modulador por ancho de pulsos utilizado para la conversión digital análoga. Existen otras secciones adicionales de soporte tales como el reloj de tiempo real, timers, RAM, ROM, etc.

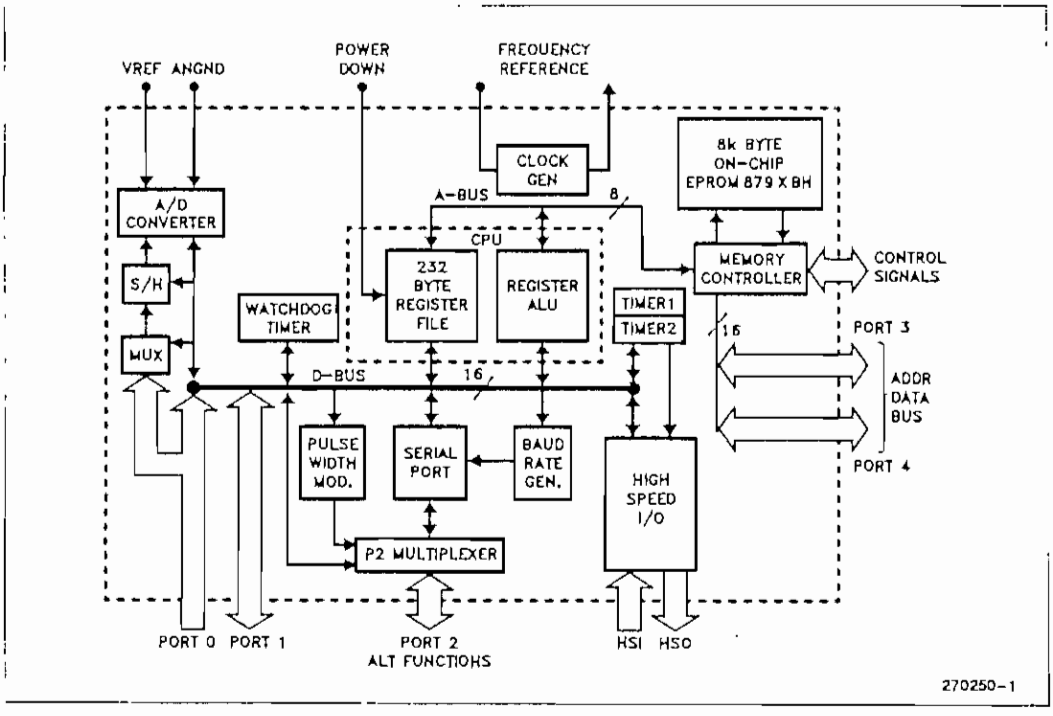


Fig. No. 1 Diagrama de bloques del microcontrolador 8096.

2.1.1 CPU

Esta compuesta por:

- Archivo de registros
- RALU
- Unidad de control

El archivo de registros contiene 232 bytes de RAM, que pueden ser accesados por bytes, palabras y dobles palabras.

La primera palabra en el archivo de registros está reservada para el stack pointer (SP). Las direcciones de acceso al archivo de registros y al SFR's son temporalmente almacenadas en dos registros de direccionamiento de 8 bit del hardware de CPU.

El diagrama de bloques de la RALU (Register/Arithmetic Logic unit) se muestra a continuación:

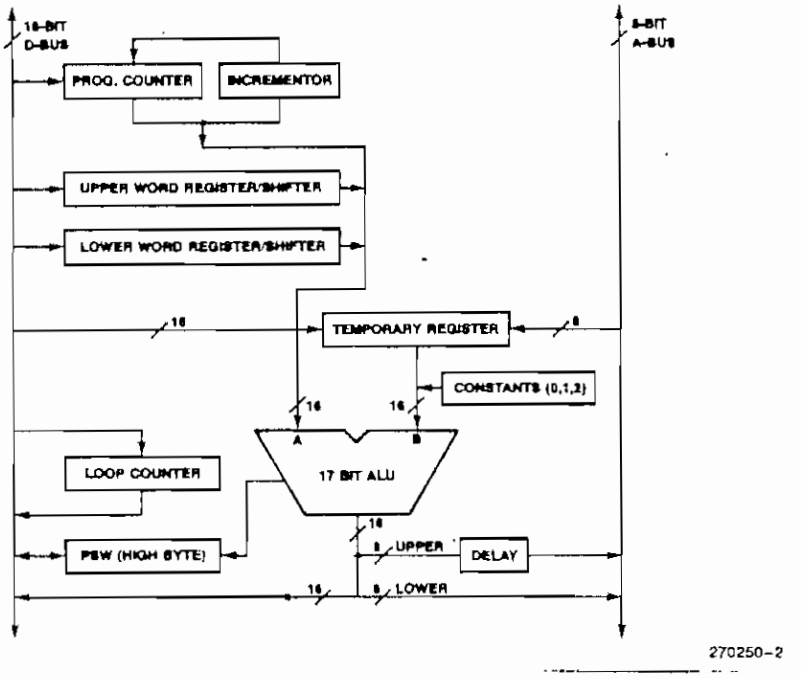


Fig. 2 Diagrama de bloques de la RALU.

La RALU es la unidad aritmética lógica que usa como acumulador a cualquiera de los 232 bytes del archivo de registros. Contiene la ALU de 17 bits (16 del registro más el carry) la palabra de estado de programa (PSW), contador del programa (PC), contador de lazo, tres registros temporales, dos de los cuales tienen la lógica de desplazamiento. Estos registros son utilizados para

operaciones que requieren rotaciones (desplazamientos) como por ejemplo multiplicación y división.

Un registro temporal almacena el segundo operando de una instrucción de dos operandos.

El DELAY mostrado en la fig. 2 es utilizado para convertir el bus de 16 bits en uno de 8 bit. Algunas constantes tales como 0,1 y 2 son guardadas en el RALU para aumentar la velocidad de los cálculos.

La unidad de control decodifica las instrucciones y genera la secuencia correcta de señales. Los accesos a la ROM interna o a la memoria externa se hace a través del controlador de memoria, una parte de este es el PC (contador de programa esclavo) que es un bypassed para la generación del contador de programa.

BUSES DE LA CPU

Una unidad de control y dos buses conectan el archivo de registros y el RALU. El bus A tiene 8 bit y es usado como un bus de datos y de direcciones (multiplexados), el cual esta conectado a la memoria del controlador. El bus D tiene 16 bit y transfiere datos solo entre el RALU , el archivo de registros o el SFR's.

2.1.2 MEMORIA

El espacio de memoria direccionable es de 64 Kbytes, el espacio libre es utilizado para programas o datos, hay localidades que tienen propósitos especificos como:

0000H hasta 00FFH

1FFEH hasta 2010H

La distribución de la memoria es la siguiente:

0000H – 00FFH se encuentra la memoria interna RAM, el archivo de registros, el stack pointer, el registro de funciones especiales.

0100H – 1FFEH se encuentra la memoria externa o periféricos de I/O.

1FFEH – 2000H está destinado para el pórtico 3 y el 4.

2000H – 2012H para los nueve vectores de interrupción.

2012H – 2080H en los cuales se encuentran los códigos de fábrica para verificación, en la 2080H esta el reset.

0BH TIMER1 (HI)		11
0AH TIMER1 (LO)	WATCHDOG	10
09H INT_PENDING	INT_PENDING	9
08H INT_MASK	INT_MASK	8
07H SBUF (RX)	SBUF (TX)	7
06H HSI_STATUS	HSO_COMMAND	6
05H HSI_TIME (HI)	HSO_TIME (HI)	5
04H HSI_TIME (LO)	HSO_TIME (LO)	4
03H AD_RESULT (HI)	HSI_MODE	3
02H AD_RESULT (LO)	AD_COMMAND	2
01H R0 (HI)	R0 (HI)	1
00H R0 (LO)	R0 (LO)	0

2.1.4 TIMER

El 8096 tiene dos timer de 16 bit, el Timer 1 es usado para sincronizar eventos a tiempo real mientras que el Timer 2 puede ser un reloj externo y sincroniza a señales externas.

El Timer 1 puede contar cada ocho estados de tiempo y solo puede borrarse con el reset, otra forma de cambiar el valor es escribiendo en la localidad 000CH pero en este caso se setean ambos Timers a 0FFFXH y no debe usarse en programas.

El Timer 2 puede ser incrementado por transiciones (una cuenta por cada transición). El funcionamiento del timer está determinado por el estado de la entrada de control 0, bit 7 (ioc-0.7), la máxima transición es cada ocho estados de tiempo. El Timer 2 puede ser borrado ejecuntando el reset, seteando el IOC0.1 disparo de entrada de alta velocidad del canal 0EH, pulsando T2RST o HSI.0.

Tanto el Timer 1 como el Timer 2 pueden generar interrupciones las cuales son controladas por IOC.2 Y EL IOC.3 respectivamente, las banderas son seteadas por el IOS1.6 y el IOS1.7 respectivamente.

2.1.5 ENTRADAS DE ALTA VELOCIDAD

La unidad de entrada de alta velocidad , puede ser usada para grabar el tiempo al cual ocurren los eventos de acuerdo al Timer 1, existen cuatro líneas las cuales pueden funcionar en cuatro modos y pueden grabar hasta 8 eventos. Las cuatro líneas pueden ser habilitadas o deshabilitadas individualmente.

Los modos son los siguientes:

- 00 8 transiciones positivas
- 01 cada transición positiva

- 10 cada transición negativa

- 11 cualquier transición positiva o negativa

2.1.6 SALIDAS DE ALTA VELOCIDAD (HSO)

Es utilizada para mandar eventos a tiempos especificos , estos eventos incluyen: comienzo de la conversión de A a D, seteo del Timer 2, seteo de 4 banderas de software, generación de interrupciones y elección de 6 líneas de salida.

HSIO CAM

El HSIO CAM es el centro de control del HSO siendo este un archivo de 8 registros. Cada registro esta formado por 23 bits 16 de los cuales especifican el tiempo en el cual la acción se va a realizar y los 7 restantes la naturaleza de la acción y la referencia de tiempo (Timer 1 o Timer 2).

El formato del registro de comando del HSO se muestra a continuación.(fig. 3)

- 10 cada transición negativa
- 11 cualquier transición positiva o negativa

2.1.6 SALIDAS DE ALTA VELOCIDAD (HSO)

Es utilizada para mandar eventos a tiempos especificos , estos eventos incluyen: comienzo de la conversión de A a D, seteo del Timer 2, seteo de 4 banderas de software, generación de interrupciones y elección de 6 líneas de salida.

HSIO CAM

El HSIO CAM es el centro de control del HSO siendo este un archivo de 8 registros. Cada registro esta formado por 23 bits 16 de los cuales especifican el tiempo en el cual la acción se va a realizar y los 7 restantes la naturaleza de la acción y la referencia de tiempo (Timer 1 o Timer 2).

El formato del registro de comando del HSO se muestra a continuación.(fig. 3)

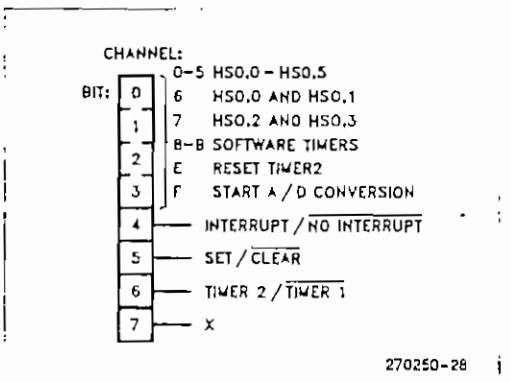


Fig.3 Formato del registro HSO

2.1.7 CONVERSOR ANALOGO DIGITAL

Cada conversión requiere 168 estados de tiempo independientemente del voltaje de entrada , el valor del voltaje de entrada debe estar entre 0 y VREF.

2.1.8 PUERTO DE COMUNICACIONES SERIALES

El puerto de comunicaciones seriales es compatible con el MCS-51 (microcontroladores de la familia del 8751). Este puerto de comunicaciones es full duplex , lo cual quiere decir que puede

transmitir y recibir simultáneamente. También la recepción es almacenada en un buffer lo que significa que puede empezar la recepción de un segundo byte antes de que el byte previo haya sido leido desde el registro de recepción. Los registros del puerto serial (SBUF) son accesados en la localidad 07H. Una instrucción de escritura a esta localidad accesa al registro de transmisión y una lectura accesa a un registro de recepción que esta fisicamente separado.

El puerto serial puede operar en cuatro modos diferentes como se describe a continuación:

MODO 0

En este modo se trabaja como con un registro de desplazamiento.

A través de TXD se emiten un grupo de 8 pulsos mientras que en RXD se transmiten o reciben datos, 8 bits de datos (el menos significativo primero) son transferidos. Este es el único modo en el que se utiliza RXD como una salida de datos. Aunque no es posible transmitir y recibir al mismo tiempo en este modo se puede utilizar compuertas externas para multiplexar en el tiempo.

MODO 1

Una estructura de 10 bits es transmitida a través de TXD y recibida a través de RXD: un bit de inicio, 8 bits de datos (el menos significativo primero), y un bit de parada. Si el bit PEN (habilitación de paridad) igual 1 entonces un bit de paridad par es transmitido en vez del octavo bit de datos. Este modo es uno de los más comunmente usados para terminales.

MODO 2

Una estructura de 11 bits es transmitida a través de TXD y recibida a través de RXD: un bit de inicio, 8 bits de datos (menos significativo primero), un noveno bit programado y un bit de parada.

MODO 3

Una estructura de 11 bit es tranmitida a través de TXD y recibida a través de RXD : un bit de inicio, 8 bits de datos(el menos significativo primero), un noveno bit programable y un bit de parada.

El MODO 2 y el MODO 3 son provistos para comunicaciones multiprocesador.

2.1.9 PUERTOS DE ENTRADA Y SALIDA (0,1,2,3,4)

Hay 5 puertos de entrada y salida de 8 bits cada uno en el 8096. Algunos de estos puertos son de entrada solamente otros de salida solamente, algunos bidireccionales y otros tienen funciones alternadas. Los puertos de entrada se conectan al bus internos a través de un buffer de entrada. Los puertos de salida se comunican con el exterior a través de un buffer de salida. Los puertos bidireccionales consisten de un registro interno, un buffer de salida y un buffer de entrada.

PUERTO 0

El puerto 0 es un puerto de entrada solamente y comparte sus pines con las entradas análogas del conversor A/D. Uno puede leer el puerto 0 digitalmente, escribiendo los bits de control apropiados en el registro de comando del A/D, seleccionar una de las líneas de este puerto como una entrada al conversor A/D. Mientras la conversión esta en proceso la impedancia de la línea seleccionada es más baja que la normal.

PUERTO 1

El puerto 1 es un puerto quasi-bidireccional.

PUERTO 2

El puerto 2 es un puerto multifuncional. Seis de sus pines son compartidos -con otras funciones en el 8096 como se muestra a continuación:

PORT	FUNCTION	ALTERNATE FUNCTION	CONTROLLED BY
P2.0	output	TXD (serial port transmit)	IOC1.5
P2.1	input	RXD (serial port receive)	N/A
P2.2	input	EXTINT (external interrupt)	IOC1.1
P2.3	input	T2CLK (Timer 2 input)	IOC0.7
P2.4	input	T2RST (Timer 2 reset)	IOC0.5
P2.5	output	PWM (pulse-width modulation)	IOC1.0
P2.6	quasi-bidirectional		
P2.7	quasi-bidirectional		

2.2.1 MODO DE DIRECCIONAMIENTO DIRECTO

El modo de direccionamiento directo nos permite accesar a los primeros 256 bytes del archivo de registros mediante un campo de 8 bit el cual nos da la dirección del registro en el cual esta el operando.

La instrucción y la dirección del registro deben seguir las reglas correspondientes dependiendo del tipo de operando.

Dependiendo de la instrucción más de tres registros pueden tomar parte en el cálculo.

2.2.2 MODO DE DIRECCIONAMIENTO INDIRECTO

El modo de direccionamiento indirecto nos permite accesar a todos los registros del 8096 incluyendo a los primeros 256 bytes mediante un campo de 8 bit el cual nos da la direccion del registro donde se encuentra una palabra variable la cual contiene la dirección del operando.

Al igual que el anterior tanto la instrucción como la dirección del registro deben seguir las reglas correspondientes dependiendo del tipo de operando.

Una instrucción puede contener solo un direccionamiento indirecto los otros direccionamientos, si los hay, deben ser directos.

2.2.3 MODO DE DIRECCIONAMIENTO INDIRECTO CON AUTOINCREMENTO

Este modo de direccionamiento es igual al anterior excepto la palabra variable la cual contiene la dirección del operando, ya que ésta se incremeta despues de que se ha utilizado la dirección en la cual se encuentra el operando.

Si el operando es un byte o Short Integers la palabra variable se incrementa en uno, si el operando es una palabra o Integers la palabra variable es decir la dirección se incrementa en dos.

2.2.4 MODO DE DIRECCIONAMIENTO INMEDIATO

En el modo de direccionamiento inmediato el operando es tomado directamente de la instrucción mediante un campo el cual puede ser de 8 bit o de 16 bit dependiendo del tipo de operando esto es, si el operando es un byte o Short Integer el campo es de 8 bit y si el operando es una palabra o Integer el campo es de 16 bit.

Una instrucción puede tener solo un direccionamiento inmediato si tiene otros direccionamientos estos deben ser directos.

2.2.5 MODO DE DIRECCIONAMIENTO SHORT INDEXADO

En el modo de direccionamiento Short Indexado en la instrucción existen dos campos de 8 bit, el primer campo contiene la dirección de un registro en el cual está la palabra variable, el segundo campo contiene un short integer el cual debe extenderse a long integer para así ser sumando a esta palabra variable obteniendo así la dirección del operando.

Mediante el segundo campo la palabra variable puede reducirse hasta 128 bytes e incrementarse hasta 127 bytes.

Una instrucción solo puede tener un direccionamiento Short Indexado, si existen otros direccionamientos estos deben ser directos.

2.2.6 MODO DE DIRECCIONAMIENTO LONG INDEXADO

Este direccionamiento Long Indexado es igual al anterior pero el segundo campo es de 16 bit y no necesita ser extendido

para sumarse a la palabra variable.

En una instrucción solo puede haber un direccionamiento long indexado, si existe otro direccionamiento debe ser directo.

A continuación hablaremos de algunos direccionamientos los cuales son casos especiales de los direccionamientos básicos.

2.2.7 DIRECCIONAMIENTO CON EL REGISTRO CERO

Los dos primeros bytes del archivo de registros de acuerdo al hardware de la 8096 son ceros fijos, es decir son una fuente fija de ceros para cálculos o comparaciones, estos registros pueden ser usados como la palabra variable en el modo de direc cionamiento long indexado, de este modo podemos seleccionar un registro en cualquier localidad de memoria directamente.

2.2.8 DIRECCIONAMIENTO DEL STACK POINTER

El sistema del Stack Pointer en la 8096 puede ser accesado directamente a través del registro 18H del archivo interno de registros, esto facilita el acceso de operandos en el stack, así por ejemplo puede usarse el stack pointer como la palabra variable en el modo de direccionamiento indirecto, al igual que en el

direcionamiento short indexado.

2.3 INTERRUPCIONES

Existen 8 fuentes de interrupción, las cuales tienen su propio vector de direccionamiento y son:

FUENTE DE INTERRUPCION	BYTE MAS SIGNIF.	BYTE MENOS SIGNIF.	PRIO.
Software	2011H	2010H	No Aplic.
Extint	200FH	200EH	7 (máx.)
Pórtico Serial	200DH	200CH	6
Timer's	200BH	200AH	5
HSI.0	2009H	2008H	4
Salida de alta velocidad	2007H	2006H	3
Datos disponibles en HSI	2005H	2004H	2
Fin de conversión A/D	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (min.)

2.3.1 CONTROL DE INTERRRUPCIONES

Un diagrama de bloques del sistema de interrupciones se muestra en la fig. No. 4. Cada una de las fuentes de interrrup-

ción es chequeada para detectar una transición de 0 a 1, si la transición ocurre el bit respectivo en el Interrupt pending Register, localizado en la dirección 0009H es seteado. El bit es borrado cuando el vector de interrupciones es cargado en el contador de programa y se ejecute la respectiva rutina de interrupción. El interrupt pending register puede ser seteado aun cuando la interrupción esté deshabilitada.

Se debe tener cuidado al escribir en el interrupt pending register y borrar las interrupciones. Si la interrupción ha sido atendida el bit es borrado. Esto es porque la 8096 traerá la siguiente instrucción dentro del flujo normal de ejecución del programa y se procede a ejecutar la interrupción. El efecto en el programa es que se tenga que añadir una instrucción NOP adicional.

La habilitación y deshabilitación de interrupciones individuales se hace a través de registro de interrupción con máscara localizado en la dirección 0008H. Si el bit en el registro de máscara es 1 entonces la interrupción es habilitada si no es deshabilitada. Aún si la interrupción es con máscara podría estar pendiente. Por lo tanto debe borrarse el bit pendiente antes de una interrupción sin máscara.

El registro de interrupción con máscara es el byte bajo del PSW. Todas las interrupciones pueden ser habilitadas o deshabilitadas por el usario con el "EI" y el "DI" sin alterar el contenido del registro de máscara.

2.3.2 PROGRAMACION DE LA PRIORIDAD DE LAS INTERRUPCIONES

Las interrupciones son seleccionadas si estan pendientes o

habilitadas de acuerdo a la mas alta prioridad. Las interrupcio nes indicadas anteriormente estan dadas en orden descendente de prioridad. El vector de interrupción entonces llama a la localidad indicada en el vector de localización, siendo esta la que comienza la rutina de servicio de interrupción (ISR).

La interrupción de menor prioridad debe ser ejecutada antes de la instrucción DI o PUSHF (Push flags). El PUSHF borra el PSW, al borrarse este deshabilita todas las interrupciones en dos formas, borrando el PSW.9, la interrupción habilitada pone un bit en la localidad de 0 a 7 de la PSW , la interrupción permite que se interrumpa el ISR para que pueda setearse el registro de máscara o la instrucción EI sea ejecutada. Otra forma es determinar cual interrupción esta habilitada o desabilitada sin interrumpir el ISR .

La dos últimas instrucciones del ISR son normalmente POPF, el cual guarda la PSW y luego RET el cual guarda el contador de programa entonces el programa puede continuar desde donde se quedó.

2.4 SOFTWARE

A continuación se hará referencia a ciertos términos que

posteriormente serán utilizados.

AX,BX,CX,DX son registros de 16 bit

AL, es el byte bajo de AX, y AH es el bytes alto

2.4.1 TIPOS DE OPERANDOS

La arquitectura del MSC-96 soporta una variedad de tipos de datos lo cual es muy útil para aplicaciones de control ,los cuales se describen a continuación.

BYTES

Son variables sin signo de 8 bits las cuales pueden tomar valores entre 0 y 255. Operaciones de relación y aritméticas pueden ser aplicadas a este tipo de operandos, pero el resultado debe ser interpretado en módulo aritmético 256. Operaciones lógicas bit a bit también son aplicables.

Los bits son enumerados de 0 a 7 siendo el 0 el menos significativo.

No existen restricciones en lo conserniente a la localización de los operandos en el espacio direccionable del MCS-96.

WORDS

Son variables sin signo de 16 bits las cuales pueden tomar valores entre 0 y 65535. Operaciones de relación y aritméticas pueden ser aplicadas a este tipo de operandos, pero el resultado debe ser interpretado en módulo arimético 65536. Operaciones lógicas bit a bit también son aplicables.

Los bits son enumerados de 0 a 15 siendo el 0 el menos significativo.

Esta formado de dos bytes, el menos significativo se sitúa en la dirección par y el más significativo byte en la siguiente dirección impar, si la dirección del menos significativo es impar no se garantiza que la operación se realice en una manera consistente.

SHORT-INTEGERS

Son variables con signo de 8 bits las cuales pueden tomar valores entre -128 y 127. Operaciones aritméticas pueden generar resultados fuera del rango de un SHORT-INTEGER lo cual seteará la bandera de overflow en la palabra de estado (PSW). El resultado numérico será equivalente al de una operación con variables tipo

BYTES.

No existen restricciones en lo conserniente a la localización de los operandos en el espacio direccionable del MCS-96.

INTEGERS

Son variables con signo de 16 bits las cuales pueden tomar valores entre -32768 y 32767. Operaciones aritméticas pueden generar resultados fuera del rango de un INTEGER lo cual seteará la bandera de overflow en la palabra de estado (PSW). El resultado numérico será equivalente al de una operación con variables tipo WORDS.

Tiene iguales restricciones que las varibles tipo WORDS.

BITS

Son operandos simples y pueden tomar valores Booleanos es decir verdadero o falso. La unión de bits forman BYTES o WORDS. El 8096 provee pruebas directas a cualquiera de los bits de un registro interno.

La arquitectura del MCS-96 requiere que los bits sean

direccionados como parte de BYTES o WORDS y no directamente como ocurre en la MCS-51.

DOUBLE-WORDS

Son variables sin signo de 32 bits las cuales pueden tomar valores entre 0 y 4294967295. La arquitectura del MCS-96 provee soporte directo para este tipo de operandos solamente para desplazamientos , divisiones de 32 a 16 bits y en multiplicaciones de 16 por 16 bits.

El operando debe residir en el registro de archivo cuya localidad debe ser divisible para cuatro, se direcciona a través del byte menos significativo.

Operaciones con este tipo de variables no pueden ser directamente soportadas, pero son fácilmente implementadas utilizando dos WORDS.

LONG-INTEGERS

Son variables con signo de 32 bits las cuales pueden tomar valores entre -2147483648 y 2147483647. Tiene las mismas características y restricciones que las DOUBLE-WORDS.

2.4.2 BANDERAS

Los 8 bits más significativos de la palabra de estado (PSW) son banderas que se setean como producto de la ejecuión de instrucciones las mismas que pueden ser probadas para saltos condicionales.

Estas banderas son:

Z (CERO).- Esta bandera se setea cuando la operación genera un resultado igual a cero.

N (NEGATIVO).- Esta bandera se setea cuando la operación genera un resultado negativo, note que la bandera N será seteada para dar un resultado algebraico correcto aún si el resultado está fuera de rango.

V (OVERFLOW).- Esta bandera se setea para indicar que el resultado de la operación esta fuera de rango de acuerdo al tipo de operando.

VT (OVERFLOW TRAP).- Esta bandera se setea en el momento que la bandera de overflow sea seteada, pero puede ser borrada solamente con intrucciones especificas tales como CLRVT,JVT. Esta bandera

permite revizar condiciones de overflow despues de una secuencia de operaciones aritméticas por lo cual es más eficiente que la bandera de overflow.

C (CARRY).- Esta bandera indica el estado del carry aritmético siendo el bit más significativo del ALU. También en él se guarda el bit correspondiente al carry de los desplazamientos.

En operaciones de substracción el CARRY es el complemento del BORROW ARITMETICO.

ST (STIKY BIT).- Esta bandera se setea cuando en una operación de desplazamiento hacia la derecha un 1 ha sido desplazado primero en el carry y entonces ha sido sacado fuera.

El ST queda indefinido después de una operación de multiplicación. El ST puede ser usado junto con el carry para controlar el redondeo despues de un desplazamiento a la derecha y en multiplicaciones con cantidades de 8 bits cuyo resultado es menor de 12 bits.

I (INTERRUPCION).- Esta bandera se setea cuando se habilitan las interrupciones.

2.5 SET DE INSTRUCCIONES

En esta sección se describe el set de instrucciones en el cual se utilizan ciertos términos que facilitan la compresión de las instrucciones y cuyo significado se explica a continuación.

- PC Contador de programa
- aa Nos indica el modo de direccionamiento es decir:

aa	Modo de direccionamiento
00	Directo
01	inmediato
10	indirecto
11	indexado

- breg Es un byte del archivo interno de registros, para diferenciar si es fuente o destino se antepone "S"o "D" respectivamente si es necesario.

- baop Es un byte operando el cual depende de los modos de direccionamiento.

- bitno Es un campo de tres bits que es parte del código de una

instrucción y selecciona uno de los ocho bits de un byte.

- wreg Es una palabra del archivo interno de registros, para diferenciar si es fuente o destino se antepone "S"o "D" respectivamente si es necesario.

- waop Es una palabra operando el cual depende de los modos de direccionamiento.

- lreg Un registro de 32 bit del archivo interno de registros.

- BEA Bytes extras que el código requiera debido al modo de direccionamiento.

- CEA Estados de tiempo extras debido al modo de direccionamiento.

- cadd Una dirección en el código del programa.

1. ADD (Dos operandos) Sumar Palabras

Operación: La suma de palabras de dos operandos es almacenada en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) + (SRC)

Formato assembler: DST SRC

ADD wreg, waop

Código Objeto: [011001aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		—

2. ADD (tres operandos) Sumar Palabras

Operación: La suma de palabras del segundo y tercer operandos es almacenada en el operando que se encuentra más a la izquierda.

(DEST) ← (SRC1) + (SRC2)

Formato assembler: DST SRC1 SRC2

ADD Dwreg, Swreg, waop

Código Objeto: [010001aa] [waop] [Swreg] [Dwreg]

Bytes : 3 + BEA

Estados: 5 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		–

3. ADDB (Dos operandos) Sumar Bytes

Operación: La suma de bytes de dos operandos es almacenada en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) + (SRC)

Formato assembler:

	DST	SRC
ADDB	breg	baop

Código Objeto: [011101aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		–

4. ADDB (tres operandos) Sumar Bytes

Operación: La suma de bytes del segundo y tercer operandos es almacenada en el operando que se

encuentra más a la izquierda.

(DEST) ← (SRC1) + (SRC2)

Formato assembler: DST SRC1 SRC2

ADD Dbreg, Sbreg, baop

Código Objeto: [010101aa] [baop] [Sbreg] [Dbreg]

Bytes : 3 + BEA

Estados: 5 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		—

5. ADDC Sumar Palabras con Carry

Operación: La suma de palabras de dos operandos y la bandera de carry es almacenada en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) + (SRC) + C

Formato assembler: DST SRC

ADDC wreg, waop

Código Objeto: [101001aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
	✓	✓	✓		—

6. ADDCB Sumar Bytes con Carry

Operación: La suma de bytes de dos operandos y la bandera de carry es almacenada en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) + (SRC) + C

Formato assembler:

	DST	SRC
ADDCB	breg,	baop

Código Objeto: [101101aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
	✓	✓	✓		—

7. AND (Dos operandos) And Lógico de Palabras

Operación: En el AND de palabras de dos operandos el resultado es 1 solo si los dos bits de igual posición son 1 si no cumple esta condición el resultado es 0.

El resultado es guardado en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) AND (SRC)

Formato assembler: DST SRC

AND wreg, *waop

Código Objeto: [011000aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	—	—

8. AND (Tres operandos) And Lógico de Palabras

Operación: En el AND de palabras del segundo y tercer operandos el resultado es 1 solo si los dos bits de igual posición son 1 si no cumple esta condición el resultado es 0 .

El resultado es guardado en el operando que se encuentra más a la izquierda.

(DEST) ← (SRC1) AND (SRC2)

Formato assembler: DST SRC1 SRC2

AND Dwreg, Swreg, waop

Código Objeto: [010000aa] [waop] [Swreg] [Dwreg]

Bytes : 3 + BEA

Estados: 5 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	—	—

9. ANDB (Dos operandos) And Lógico de Bytes

Operación: En el AND de bytes de dos operandos de resultado es 1 solo si los dos bits de igual posición son 1 si no cumple esta condición el resultado es 0 .

El resultado es guardado en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) AND (SRC)

Formato assembler: DST SRC

ANDB breg, baop

Código Objeto: [011100aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	—	—

10. ANDB (Tres operandos) And Lógico de Bytes

Operación: En el AND de bytes del segundo y tercer operandos el resultado es 1 solo si los dos bits de igual posición son 1 si no cumple esta condición el resultado es 0 .
El resultado es guardado en el operando que se encuentra más a la izquierda.

(DEST) ← (SRC1) AND (SRC2)

Formato assembler: DST SRC1 SRC2

ANDB Dbreg, Sbreg, baop

Código Objeto: [010100aa] [baop] [Sbreg] [Dbreg]

Código Objeto: [100010aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		_

17. CMPB Comparar Bytes

Operación: El byte operando que se encuentra más a la derecha es restado de el byte operando que se encuentra más a la izquierda.

Las banderas son alteradas pero no los operandos.La bandera de carry se setea como complento del Borrow.

(DEST) - (SRC)

Formato assembler:

	DST	SRC
CMPB	breg,	baop

Código Objeto: [100110aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		–

18. DEC Decrementar Palabra

Operación: El valor de la palabra operando es decrementada en uno.

(DEST) ← (DEST) - 1

Formato assembler: DEC wreg

Código Objeto: [00000101] [wreg]

Bytes : 2

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		–

19. DECB Decrementar Byte

Operación: El valor del byte operando es decrementada en uno.

(DEST) ← (DEST) - 1

Formato assembler: DECB breg

Código Objeto: [00010101] [breg]

Bytes : 2

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		–

20. DI Deshabilitar Interrupciones

Operación: Las interrupciones son deshabilitadas.

(PSW.9) ← 0

Formato assembler: DI

Código Objeto: [11111010]

Bytes : 1

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

22. DIVB Dividir Enteros Cortos

Operación: La instrucción divide el contenido de un operando entero para el contenido de una entero corto, usando signo aritmético. El byte menos significativo contendrá el cuociente y byte más singnificativa contendrá el residuo.

(menos sign DEST) ← ¿DEST) / (SRC)

(más signif DEST) ← ¿DEST) MOD (SRC)

Formato assembler:

	DST	SRC
DIVB	wreg,	baop

Código Objeto: [11111110] [100111aa] [baop] [wreg]

Bytes : 2 + BEA

Estados: 21 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	?		—

23. DIVU Dividir Palabras

Operación: La instrucción divide el contenido de un operando doble palabra para el contenido de una palabra , sin usar signo aritmético. La palabra menos significativa contendrá el cuociente y la palabra más singnificativa contendrá el residuo.

(menos sign DEST) ← ¿DEST) / (SRC)

(más signif DEST) ← ¿DEST) MOD (SRC)

Formato assembler:

	DST	SRC
DIVU	lreg,	waop

Código Objeto: [100011aa] [waop] [lreg]

Bytes : 2 + BEA

Estados: 25 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	✓		—

24. DIVUB Dividir Bytes

Operación: La instrucción divide el contenido de un palabra operando para el contenido de un byte, sin usar signo aritmético. El byte menos significativo contendrá el cuociente y el byte más singnificativa contendrá el residuo.

(menos sign DEST) ← (DEST) / (SRC)

(más signif DEST) ← (DEST) MOD (SRC)

Formato assembler: DST SRC

DIVUB wreg, baop

Código Objeto: [100111aa] [baop] [wreg]

Bytes : 2 + BEA

Estados: 17 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	✓		–

25. DJNZ Decrementar y saltar si no es cero

Operación: El valor del byte operando es decrementada en uno. Si el resultado no es gual a cero, la

Formato assembler: EXTB wreg

Código Objeto: [00010110] [wreg]

Bytes : 2

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	—	—

29. INC Incrementar Palabra

Operación: El valor de la palabra operando es incrementada en uno.

(DEST) ← (DEST) + 1

Formato assembler: INC wreg

Código Objeto: [00000111] [wreg]

Bytes : 2

Estados: 4

tuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si el bit es uno entonces se ejecuta la siguiente instrucción.

```
if (bit especificado) = 0 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JBC breg,bitno,cadd

Código Objeto: [00110bbb] [breg] [disp]

Bytes : 3

Estados: 5 si no salta

9 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

32. JBS Saltar si bit uno

Operación: El bit especificado es probado.Si el resultado es igual a uno la distancia desde el fin de la instrucción hasta el nivel deseado es

33. JC Saltar si la bandera de Carry es uno

Operación: Si la bandera de carry es uno la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de carry es cero entonces se ejecuta la siquiente instrucción.

```
if C = 1 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JC cadd

Código Objeto: [11011011] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

34. JE Saltar si es igual a

Operación: Si el resultado es igual a cero la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseada debe estar en el rango de -128 a +127.

Si el resultado no es cero entonces se ejecuta la siguiente instrucción.

```
if Z = 1 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JE cadd

Código Objeto: [11011111] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

35. JGE Saltar si es mayor o igual que

Operación: Si la bandera negativa es cero la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera negativa es uno entonces se ejecuta la siguiente instrucción.

```
if N = 0 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JGE cadd

Código Objeto: [11010110] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

36. JGT Saltar es mayor que

Operación: Si las banderas negativa y cero son borradas la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera negativa o la bandera cero es uno entonces se ejecuta la siguiente instrucción.

```
if N = 0 AND Z = 0 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JGT cadd

Código Objeto: [11010010] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	—

37. JH Saltar si es muy grande

Operación: Si la bandera de carry es uno pero la bandera de cero no lo es,la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de carry es cero o la bandera cero es uno entonces se ejecuta la siguiente instrucción.

```
if C = 1 AND Z = 0  then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JH cadd

Código Objeto: [11011001] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	—

38. JLE Saltar si es menor o igual que

Operación: Si la bandera negativa o la bandera cero es uno entonces la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la las dos banderas son cero entonces se ejecuta la siguiente instrucción.

```
if N = 1 OR Z = 1  then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JLE cadd

Código Objeto: [11011010] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

40. JNC Saltar si la bandera de Carry es cero

Operación: Si la bandera de carry es cero la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de carry es uno entonces se ejecuta la siguiente instrucción.

```
if C = 0 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JNC cadd

Código Objeto: [11010011] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	—

41. JNE Saltar si no es igual a

Operación: Si el resultado no es igual a cero la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si el resultado es cero entonces se ejecuta la siguiente instrucción.

```
if Z = 0 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JNE cadd

Código Objeto: [11010111] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

42. JNH Saltar si no es muy grande

Operación: Si la bandera de carry es cero o la bandera de cero (Z) es uno la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de carry es uno y la bandera cero no es uno entonces se ejecuta la siguiente instrucción.

```
if C = 0 OR Z = 1  then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JNH cadd

Código Objeto: [11010001] [disp]

Código Objeto: [11010000] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

44. JNV Saltar si la bandera de Overflow es cero

Operación: Si la bandera de overflow es cero la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de overflow es uno entonces se ejecuta la siguiente instrucción.

```
if V = 0 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JNV cadd

Código Objeto: [11010101] [disp]

Bytes : 2 .

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

45. JNVT Saltar si la bandera de VT es cero

Operación: Si la bandera de VT es cero la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de VT es uno entonces se ejecuta la siguiente instrucción.

```
if VT = 0 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JNVT cadd

Código Objeto: [11010100] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	0	–

46. JST Saltar si la bandera de ST es cero

Operación: Si la bandera de ST es uno la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de ST es cero entonces se ejecuta la siguiente instrucción.

```
if ST = 1 then
    PC ← PC + disp (extendido a 16 bits)
```

Formato assembler: JST cadd

Código Objeto: [11011000] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

47. JNC Saltar si la bandera de Overflow es uno

Operación: Si la bandera de overflow es uno la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de overflow es cero entonces se ejecuta la siguiente instrucción.

if V = 1 then

PC ← PC + disp (extendido a 16 bits)

Formato assembler: JV cadd

Código Objeto: [11011101] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

48. JVT Saltar si la bandera de VT es uno

Operación: Si la bandera de VT es uno la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador de programa efectuándose el salto.

El offset desde el fin de esta instrucción hasta el nivel deseado debe estar en el rango de -128 a +127.

Si la bandera de VT es cero entonces se ejecuta la siguiente instrucción.

if VT = 1 then

PC ← PC + disp (extendido a 16 bits)

Formato assembler: JVT cadd

Código Objeto: [11011100] [disp]

Bytes : 2

Estados: 4 si no salta

8 si salta

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	0	–

49. LCALL Llamada larga

Operación: El contenido del contador del programa es puesto en el stack. Entonces la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador del programa efectuándose la llamada. El operando puede tener cualquier dirección en el espacio de direccionamiento.

SP ← SP – 2

(SP) ← PC

PC ← PC + disp

Formato assembler: LCALL cadd

Código Objeto: [11101111] [disp-low] [disp-hi]

Bytes : 3

Estados: 13

16

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

50. LD Cargar palabras

Operación: El valor de la fuente (palabra más a la derecha del operando) es guardado en el destino (palabras más a la izquierda del operando).

(DEST) ← (SRC)

Formato assembler:

	DST	SRC
LD	wreg,	waop

Código Objeto: [101000aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

51. LDB Cargar bytes

Operación: El valor de la fuente (byte más a la derecha del operando) es guardado en el destino (byte más a la izquierda del operando).

(DEST) ← (SRC)

Formato assembler:

	DST	SRC
LDB	breg,	baop

Código Objeto: [101100aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

52. LDBSE Cargar enteros con enteros cortos

Operación: El valor de la fuente (byte más a la derecha del operando) es extendido y guardado en el destino (palabra más a la izquierda del operando).

```
(menos sign DEST) ← (SRC)
if (SRC) < 80H then
(más signif DEST) ← 0
else
(más signif DEST) ← 0FFH
fin_if
```

Formato assembler:

```
                DST   SRC
LDBSE    wreg,  baop
```

Código Objeto: [101111aa] [baop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	—

53. LDBZE Cargar palabras con bytes

Operación: El valor de la fuente (byte más a la derecha del operando) es extendido con cero y guardado en el destino (palabra más a la izquierda del operando).

(menos sign DEST) ← (SRC)

(más signif DEST) ← 0

Formato assembler: DST SRC

LDBZE wreg, baop

Código Objeto: [101011aa] [baop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

54. LJMP Salto largo

Operación: La distancia desde el final de la instrucción al lugar deseado,se suma al PC para que se realice el salto. El operando puede ser una dirección, en el espacio direccionable.

PC ← PC + disp

Formato assembler: LJMP cadd

Código Objeto: [11100111] [disp-low] [disp- hi]

Bytes : 3

Estados: 8

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	—

55. MUL (Dos operandos) Multiplicación de enteros

Operación: Los dos operandos enteros son multiplicados usando signo aritmético y el resultado de 32 bits es guardado en el destino (operando entero largo).

La bandera de ST queda indefinida después de esta instrucción.

(DEST) ← (DEST) * (SRC)

Formato assembler: DST SRC

MUL lreg, waop

Código Objeto: [11111110] [011011aa] [waop] [lreg]

Bytes : 3 + BEA

Estados: 29 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	?

56. MUL (tres operandos) Multiplicación de enteros

Operación: El segundo y tercer operandos enteros son multiplicados usando el signo aritmético y el resultado de 32 bits es almacenado en el destino (operando entero largo).
La bandera de ST queda indefinida después de ejecutar esta instrucción.

(DEST) ← (SRC1) * (SRC2)

Formato assembler:	DST	SRC1	SRC2
MUL	lreg,	wreg,	waop

Código Objeto: [1111110] [010011aa] [waop] [wreg] [lreg]

Bytes : 4 + BEA

Estados: 30 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	?

57. MULB (Dos operandos) Multiplicación de enteros cortos

Operación: Los dos operandos enteros cortos son multiplicados usando signo aritmético y el resultado de 16 bits es guardado en el destino (operando entero).
La bandera de ST queda indefinida después de esta instrucción.

(DEST) ← (DEST) * (SRC)

Formato assembler:

	DST	SRC
MULB	wreg	baop

Código Objeto: [1111110] [01111laa] [baop] [wreg]

Bytes : 3 + BEA

Estados: 21 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	?

58. MULB (tres operandos) Multiplicación de enteros cortos

Operación: El segundo y tercer operandos enteros cortos son multiplicados usando el signo aritmético y el resultado de 16 bits es almacenado en el destino (operando entero).

La bandera de ST queda indefinida después de ejecutar esta instrucción.

(DEST) ← (SRC1) * (SRC2)

Formato assembler:

	DST	SRC1	SRC2
MULB	wreg,	breg,	baop

Código Objeto: [1111110] [010111aa] [baop] [breg] [wreg]

Bytes : 4 + BEA

Estados: 22 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	?

59. MULU (Dos operandos) Multiplicación de palabras

Operación: Las dos palabras operandos son multiplicados sin signo aritmético y el resultado de 32 bits es guardado en destino (operando doble palabra).

La bandera de ST queda indefinida después de esta instrucción.

(DEST) ← (DEST) * (SRC)

Formato assembler: DST SRC

MULU lreg, waop

Código Objeto: [011011aa] [waop] [lreg]

Bytes : 3 + BEA

Estados: 25 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	?

60. MULU (tres operandos) Multiplicación de palabras

Operación: La segunda y tercera palabra operandos son multiplicados sin el signo aritmético y el resultado de 32 bits es almacenado en el destino (operando doble palabra).

La bandera de ST queda indefinida después de ejecutar esta instrucción.

(DEST) ← (SRC1) * (SRC2)

Formato assembler: DST SRC1 SRC2

MULU lreg, wreg, waop

Código Objeto: [0100111aa] [waop] [wreg] [lreg]

Bytes : 4 + BEA

Estados: 26 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	?

61. MULUB (Dos operandos) Multiplicación de bytes

Operación: Los dos bytes operandos son multiplicados sin signo aritmético y el resultado de 16 bits es guardado en el destino (palabra operando).

La bandera de ST queda indefinida después de esta instrucción.

(DEST) ← (DEST) * (SRC)

Formato assembler: DST SRC

MULUB wreg, baop

Código Objeto: [011111aa] [baop] [wreg]

Bytes : 2 + BEA

Estados: 17 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	?

62. MULUB (tres operandos) Multiplicación de bytes

Operación: El segundo y tercer bytes operandos son multiplicados sin el signo aritmético y el resultado de 16 bits es almacenado en el destino (palabra operando).

La bandera de ST queda indefinida después de ejecutar esta instrucción.

(DEST) ← (SRC1) * (SRC2)

Formato assembler: DST SRC1 SRC2

MULUB wreg, breg, baop

Código Objeto: [010111aa] [baop] [breg] [wreg]

Bytes : 3 + BEA

Estados: 18 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	?

63. NEG Negar un entero

Operación: El valor del operando entero es negado.

(DEST) ← -(DEST)

Formato assembler: NEG wreg

Código Objeto: [00000011] [wreg]

Bytes : 2

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		–

64. NEGB Negar un entero corto

Operación: El valor del operando entero corto es negado.

(DEST) ← -(DEST)

Formato assembler: NEGB breg

Código Objeto: [00010011] [breg]

Bytes : 2

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		—

65. NOP No operación

Operación: No se realiza nada y el control pasa a la próxima instrucción.

Formato assembler: NOP

Código Objeto: [11111101]

Bytes : 1

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

66. NORML Normalizar un entero largo

Operación: El operando entero largo es desplazado hacia la izquierda hasta que el bit más significativo sea 1. Si el bit más significativo es cero hasta los 31 desplazamientos , el proceso termina y la bandera de cero es 1. El número de desplazamientos es guardado en el segundo operando.

```
(COUNT) ← 0
do while (MSB(DEST) = 0 ) AND ( (COUNT) < 31)
(DEST) ← (DEST) * 2
(COUNT)← (COUNT) + 1
end_while
```

Formato assembler: NORML lreg,breg

Código Objeto: [00001111] [breg] [lreg]

Bytes : 3

Estados: 11 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	?	0	–	–	–

67. NOT Complemento de una palabra

Operación: El valor de la palabra operando es complementado es decir cada bit 1 es cambiado por 0 y cada bit 0 por 1.

(DEST) ← NOT(DEST)

Formato assembler: NOT wreg

Código Objeto: [00000010] [wreg]

Bytes : 2

Estados: 4 .

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	–	–

68. NOTB Complementar un byte

Operación: El valor del byte operando es complementado cada bit 1 es cambiado por 0 y cada bit 0 por 1.

(DEST) ← NOT(DEST)

Formato assembler: NOTB breg

Código Objeto: [00010010] [breg]

Bytes : 2

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	—	—

69. OR OR lógico de palabras

Operación: En el OR lógico de palabras el resultado es 1 si uno de los dos bits de igual posición es 1 si no cumple esta condición el resultado es 0 El resultado es guardado en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) OR (SRC)

Formato assembler: DST SRC

OR wreg, waop

Código Objeto: [100000aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	—	—

70. OR OR lógico de bytes

Operación: En el OR lógico de bytes el resultado es 1 si uno de los dos bits de igual posición es 1 si no cumple esta condición el resultado es 0 . El resultado es guardado en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) OR (SRC)

Formato assembler: ORB breg,baop

Código Objeto: [100100aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	–	–

71. POP Traer una palabra

Operación: La palabra que se encuentra en le stack es sacada y puesta en el operando destino.

(DEST) ← (SP)

SP ← SP + 2

Formato assembler: POP waop

Código Objeto: [110011aa] [waop]

Bytes : 1 + BEA

Estados: 12 + CEA

14 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

72. POPF Traer Banderas

Operación: La palabra que se encuentra en le stack es sacada y puesta en el PSW. Las llamadas de interrupción no ocurren inmediatamente sino en la siguiente instrucción.

(PSW) ← (SP)

SP ← SP + 2

Formato assembler: POPF

Código Objeto: [11110011]

Bytes : 1

Estados: 9

13

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓	✓	✓

73. PUSH Poner una palabra en el stack

Operación: El operando especificado es puesto en el stack.

SP ← SP - 2

(SP) ← (DEST)

Formato assembler: PUSH waop

Código Objeto: [110010aa] [waop]

Bytes : 1 + BEA

Estados: 8 + CEA

12 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	—

74. PUSHF Poner Banderas en el stack

Operación: El PSW es puesto en el stack, y en PSW se setea con ceros. Esto implica que todas las interrupciones son deshabilitadas es decir las llamadas de interrupción no ocurren inmediatamente después de esta instrucción.

SP ← SP - 2

(SP) ← PSW

PSW ← 0

SP ← SP − 2

(SP) ← (DEST)

Formato assembler: PUSH waop

Código Objeto: [11001Oaa] [waop]

Bytes : 1 + BEA

Estados: 8 + CEA

12 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	—

74. PUSHF Poner Banderas en el stack

Operación: El PSW es puesto en el stack, y en PSW se setea con ceros. Esto implica que todas las interrupciones son deshabilitadas es decir las llamadas de interrupción no ocurren inmediatamente después de esta instrucción.

SP ← SP − 2

(SP) ← PSW

PSW ← 0

77. SCALL Llamada corta

Operación: El contenido del contador del programa es puesto en el stack. Entonces la distancia desde el fin de la instrucción hasta el nivel deseado es sumada al contador del programa efectuándose la llamada. El offset desde el fin de la instrucción hasta el nivel deseado debe estar en el rango de -1024 a + 1023 inclusive.

SP ← SP − 2

(SP) ← PC

PC ← PC + disp (exten.a 16 bits)

Formato assembler: SCALL cadd

Código Objeto: [00101xxx] [disp-low]

donde xxx son los bit más sign. del desplazamiento.

Bytes : 2

Estados: 13

16

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

78. SETC Setear la bandera de Carry

Operación: La bandera de carry es puesta en 1.

C ← 1

Formato assembler: SETC

Código Objeto: [11111001]

Bytes : 1

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	1	–	–	–

79. SHL Desplazamiento a la izquierda de una palabra

Operación: La palabra operando (más a la izquierda) es desplazada hacia la izquierda cuantas veces este especificado por count operando (más a la derecha). El count puede ser especificado

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	–	–	–	–

78. SETC Setear la bandera de Carry

Operación: La bandera de carry es puesta en 1.

C ← 1

Formato assembler: SETC

Código Objeto: [11111001]

Bytes : 1

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
–	–	1	–	–	–

79. SHL Desplazamiento a la izquierda de una palabra

Operación: La palabra operando (más a la izquierda) es desplazada hacia la izquierda cuantas veces este especificado por count operando (más a la derecha). El count puede ser especificado

con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Los bits de la derecha son llenados con ceros. El último bit desplazado es guardado en la bandera de carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit más significativo de (DEST)
(DEST) ← (DEST) * 2
Temp ← Temp - 1
end_while
```

Formato assembler: SHL wreg,#count

o

SHL wreg,breg

Código Objeto: [00001001] [cnt/breg] [wreg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	?	✓	✓		—

80. SHLB Desplazamiento a la izquierda de un byte

Operación: La byte operando (más a la izquierda) es desplazada hacia la izquierda cuantas veces

este especificado por count operando (más a la derecha). El count puede ser especificado con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Los bits de la derecha son llenados con ceros. El último bit desplazado es guardado en la bandera de carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit más significativo de (DEST)
(DEST) ← (DEST) * 2
Temp ← Temp - 1
end_while
```

Formato assembler: SHLB breg,#count

o

SHLB breg,breg

Código Objeto: [00011001] [cnt/breg] [breg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	?	✓	✓		-

81. SHLL Desplazamiento a la izquierda de una doble palabra

Operación: La doble palabra operando (más a la izquierda) es desplazada hacia la izquierda cuantas veces este especificado por count operando (más a la derecha). El count puede ser especificado con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Los bits de la derecha son llenados con ceros. El último bit desplazado es guardado en la bandera de carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit más significativo de (DEST)
(DEST) ← (DEST) * 2
Temp ← Temp - 1
end_while
```

Formato assembler: SHLL lreg,#count

o

SHL1 lreg,breg

Código Objeto: [00001101] [cnt/breg] [lreg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	?	✓	✓		–

82. SHR Desplazamiento a la derecha de una palabra

Operación: La palabra operando (más a la izquierda) es desplazada hacia la derecha cuantas veces este especificado por count operando (más a la derecha). El count puede ser especificado con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Los bits de la izquierda son llenados con ceros. El último bit desplazado es guardado en la bandera de carry.

El ST es borrado cuando comienza la instrucción y setea con 1 si en cualquier momento del desplazamiento un 1 es desplazado a la

bandera del carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit menos significativo de (DEST)
(DEST) ← (DEST) / 2  división sin signo
Temp ← Temp - 1
end_while
```

Formato assembler: SHR wreg,#count

o

SHR wreg,breg

Código Objeto: [00001000] [cnt/breg] [wreg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	0	✓	0	–	✓

83. SHRA Desplazamiento aritmético a la derecha de una palabra

Operación: La palabra operando (más a la izquierda) es desplazada hacia la izquierda cuantas veces este especificado por count operando (más a

la derecha). El count puede ser especificado con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Si el bit original más significativo fue cero, ceros son desplazados y si fue uno, unos son desplazados.

El último bit desplazado es guardado en la bandera de carry.

El ST es borrado cuando comienza la instrucción y setea con 1 si en cualquier momento del desplazamiento un 1 es desplazado a la bandera del carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit más significativo de (DEST)
(DEST) ← (DEST) / 2  división con signo
Temp ← Temp - 1
end_while
```

Formato assembler: SHRA wreg,#count

o

SHRA wreg,breg

Código Objeto: [00001010] [cnt/breg] [wreg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	0	–	✓

84. SHRAB Desplazamiento aritmético a la derecha de una byte

Operación: El byte operando (más a la izquierda) es desplazada hacia la izquierda cuantas veces este especificado por count operando (más a la derecha). El count puede ser especificado con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Si el bit original más significativo fue cero, ceros son desplazados y si fue uno, unos son desplazados.

El último bit desplazado es guardado en la bandera de carry.

El ST es borrado cuando comienza la instrucción y setea con 1 si en cualquier momento del desplazamiento un 1 es desplazado a la bandera del carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit más significativo de (DEST)
(DEST) ← (DEST) / 2  división con signo
Temp ← Temp - 1
end_while
```

Formato assembler: SHRAB breg,#count

o

SHRAB breg,breg

Código Objeto: [00011010] [cnt/breg] [breg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

85. SHRAL Desplazamiento aritmético a la derecha de una doble palabra

Operación: La doble palabra (más a la izquierda) es desplazada hacia la izquierda cuantas veces este especificado por count operando (más a la derecha). El count puede ser especificado con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Si el bit original más significativo fue cero, ceros son desplazados y si fue uno, unos son desplazados.

El último bit desplazado es guardado en la bandera de carry.

El ST es borrado cuando comienza la instrucción y setea con 1 si en cualquier momento del desplazamiento un 1 es desplazado a la bandera del carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit más significativo de (DEST)
(DEST) ← (DEST) / 2  división con signo
Temp ← Temp - 1
end_while
```

Formato assembler: SHRAL lreg,#count

o

SHRAL lreg,breg

Código Objeto: [00001010] [cnt/breg] [lreg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

86. SHRB Desplazamiento a la derecha de una byte

Operación: La byte operando (más a la izquierda) es desplazada hacia la derecha cuantas veces este especificado por count operando (más a la derecha). El count puede ser especificado con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Los bits de la izquierda son llenados con ceros. El último bit desplazado es guardado en la bandera de carry.

El ST es borrado cuando comienza la instrucción y setea con 1 si en cualquier momento

del desplazamiento un 1 es desplazado a la bandera del carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit menos significativo de (DEST)
(DEST) ← (DEST) / 2  división sin signo
Temp ← Temp - 1
end_while
```

Formato assembler: SHRB breg,#count

o

SHRB breg,breg

Código Objeto: [00001000] [cnt/breg] [breg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	0	✓	0	–	✓

87. SHRL Desplazamiento a la derecha de una doble palabra

Operación: La doble palabra (más a la izquierda) es desplazada hacia la derecha cuantas veces este especificado por count operando (más a la derecha). El count puede ser especificado con un valor inmediato de 0 a 15 o con el valor del contenido de cualquier registro. Los bits de la izquierda son llenados con ceros. El último bit desplazado es guardado en la bandera de carry.

El ST es borrado cuando comienza la instrucción y setea con 1 si en cualquier momento del desplazamiento un 1 es desplazado a la bandera del carry.

```
Temp ← (COUNT)
do while Temp <> 0
C ← bit menos significativo de (DEST)
(DEST) ← (DEST) / 2  división sin signo
Temp ← Temp − 1
end_while
```

Formato assembler: SHRL lreg,#count

o

SHRL lreg,breg

Código Objeto: [00001000] [cnt/breg] [lreg]

Bytes : 3

Estados: 7 + No de desplazamientos

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

88. SJMP Salto corto

Operación: La distancia desde el fin de la instruc ción hasta el nivel deseado es sumada al contador del programa efectuándose el salto. El offset desde el fin de la instrucción hasta el nivel deseado debe estar en el rango de- 1024 a + 1023 inclusive.

PC ← PC + disp (exten.a 16 bits)

Formato assembler: SJMP cadd

Código Objeto: [00100xxx] [disp-low]

donde xxx son los bit más sign. del desplazamiento.

Bytes : 2

Estados: 8

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

89. SKIP Dos bytes sin operación

Operación: No se realiza nada y el control pasa a la próxima instrucción. Es como NOP de dos bytes el segundo byte puede ser cualquier valor simplemente es ignorado.

Formato assembler: SKIP breg

Código Objeto: [00000000] [breg]

Bytes : 2

Estados: 4

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

90. ST Guardar palabras

Operación: El valor de la fuente (palabra más a la izquierda del operando) es guardado en el destino (palabras más a la derecha del operando).

(DEST) ← (SRC)

Formato assembler: SRC DST

ST wreg, waop

Código Objeto: [110000aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
—	—	—	—	—	—

91. STB Guardar bytes

Operación: El valor de la fuente (byte más a la izquierda del operando) es guardado en el destino (byte más a la derecha del operando).

(DEST) ← (SRC)

Formato assembler: SRC DST

STB breg, baop

Código Objeto: [101100aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
-	-	-	-	-	-

92. SUB (Dos operandos) Restar Palabras

Operación: La palabra más a la derecha es restada de la palabra más a la izquierda y el resultado es guardado en el destino. La bandera de carry es seteada como complemento del Borrow.

(DEST) ← (DEST) - (SRC)

Formato assembler: DST SRC

SUB wreg, waop

Código Objeto: [011010aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		–

93. SUB (tres operandos) Restar Palabras

Operación: La palabra más a la derecha es restada de la segunda palabra operando y el resultado es guardado en el destino (palabra más a la izquierda).

La bandera de carry es seteada como complemento del Borrow.

(DEST) ← (SRC1) – (SRC2)

Formato assembler:

	DST	SRC1	SRC2
SUB	Dwreg,	Swreg,	waop

Código Objeto: [010010aa] [waop] [Swreg] [Dwreg]

Bytes : 3 + BEA

Estados: 5 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		–

94. SUBB (Dos operandos) Restar Bytes

Operación: El byte más a la derecha es restada del byte más a la izquierda y el resultado es guardado en el destino. La bandera de carry es seteada como complemento del Borrow.

(DEST) ← (DEST) - (SRC)

Formato assembler:

	DST	SRC
SUBB	breg	baop

Código Objeto: [011110aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		-

95. SUBB (tres operandos) Restar Bytes

Operación: El byte más a la derecha es restado de el segundo byte operando y el resultado es guardado en el destino (byte más a la izquierda).

La bandera de carry es seteada como complemento del Borrow.

(DEST) ← (SRC1) + (SRC2)

Formato assembler: DST SRC1 SRC2

SUBB Dbreg, Sbreg, baop

Código Objeto: [010110aa] [baop] [Sbreg] [Dbreg]

Bytes : 3 + BEA

Estados: 5 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓	✓		—

96. SUBC Restar Palabras con Carry

Operación: La palabra más a la derecha es restada de la palabra más a la izquierda y se resta el valor del carry guardando el resultado en destino.

La bandera de carry es seteada como complemento del Borrow.

(DEST) ← (DEST) - (SRC) - C

Formato assembler: DST SRC

SUBC wreg, waop

Código Objeto: [101010aa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
	✓	✓	✓		–

97. SUBCB Restar Bytes con Carry

Operación: El byte más a la derecha es restado del byte más a la izquierda y el valor del carry es restado siendo el resultado guardado en el destino. La bandera de carry es seteada como complemento del Borrow.

(DEST) ← (DEST) – (SRC) – C

Formato assembler: DST SRC

SUBCB breg, baop

Código Objeto: [101110aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	✓			–

98. XOR OR lógico exclusivo de palabras

Operación: En el OR lógico exclusivo de palabras el resultado es 1 si uno de los dos bits de igual posición es 1 pero no ambos si no cumple el resultado es 0 .

El resultado es guardado en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) XOR (SRC)

Formato assembler: DST SRC

XOR wreg, waop

Código Objeto: [10000laa] [waop] [wreg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	–	–

99. XOR OR lógico exclusivo de bytes

Operación: En el OR lógico de bytes el resultado es 1 si uno de los dos bits de igual posición es 1 pero no ambos si no cumple el resultado es 0 El resultado es guardado en el operando que se encuentra más a la izquierda.

(DEST) ← (DEST) XOR (SRC)

Formato assembler: XORB breg,baop

Código Objeto: [100101aa] [baop] [breg]

Bytes : 2 + BEA

Estados: 4 + CEA

Banderas que son Afectadas

Z	N	C	V	VT	ST
✓	✓	0	0	–	–

2.6. APLICACIONES

El MCS-96 es un microcontrolador construido en un simple chip y diseñado para ser usado en sofisticadas aplicaciones en tiempo real tales como control industrial, instrumentación y en computación.

Por su CPU de 16 bit, alta velocidad de procesamiento y alta velocidad de entrada y salida es ideal para controles complejos de motores y sistemas de control de ejes como por ejemplo motores de alta potencia de tres fases de AC y robots.

Por el conversor análogo digital de 10 bit tambin se lo utiliza en sistemas de adquisición de datos y en controladores de lazo cerrado análogos.

El chip es ideal en el área de instrumentación tal como el cromatógrafo de gases. De la misma forma se lo utiliza en componentes de aplicaciones aeroespaciales tales como guía de misiles y control.

A continuación se dará una tabla de aplicaciones en los diferentes campos.

INDUSTRIA

Control de motores

Robots

Procesos de control discretos y continuos

Copiadoras de color y blanco & negro

Discos Winchester

Impresoras de impacto y no impacto

Sistemas de Backup

TELECOMUNICACIONES

Modems

Tarjetas de control inteligentes

INDUSTRIA AUTOMOTRIZ

Control de ignición

Control de transmisión

Control de frenos

Control de emisión

Como otra de las aplicaciones mostraremos a continuación un ejemplo tomado del manual de la Intel del microcontrolador 8096 página 5-20.

Este programa permite encontrar el valor de la ordenada (Y), ingresando la absisa (X).

```
look:   LDB  AL, IN_VAL      ; Cargar un registro temporal
                             ; con el valor de entrada
        SHRB AL, #3          ; Divide el byte para 8
        ANDB AL, #11111110B ;Asegura que AL sea palabra direcc.
        LDBZE AX, AL

        LD   TABLE_LOW,TABLE [AX] ; tabla_low es cargado con
                                  ; un valor de la tabla para
                                  ; (X) múltiplo de 10H

        LD     TABLE_HIGH, (TABLE+2 [AX]); tabla_high es cargado
                                          ; con el siguiente valor
                                          ; de la tabla.

        SUB    TAB_DIF, TABLE_HIGH,TABLE_LOW
        ANDB   IN_DIFB, IN_VAL, #0FH      ; cuatro bits menos
                                          ; significativos
        LDBZE IN_DIF, IN_DIFB
        MUL'   OUT_DIF, IN_DIF, TAB_DIF
        SHRAL OUT_DIF, #4                 ; divide para 16
        ADD    OUT,OUT_DIF, TABLE_LOW     ; suma la diferencia
        SHR    OUT, #4                    ; redondea a 12 bits
        ADDC   OUT,zero                   ; redondea si C=1
```

CAPITULO III

CARACTERISTICAS DEL SIMULADOR A IMPLEMENTARSE

Las características del simulador del microcontrolador 8096, siendo este una herramienta para el desarrollo y depuración de programas, tienen por objeto brindar la mayor flexibilidad en el tratamiento de los datos e instrucciones de un determinado programa.

Siendo este el objetivo del simulador, se trabaja en una sola pantalla, la cual consta de cuatro áreas:

- Area de Memoria (ventana izquierda)
- Area de Ejecución de Instrucciones (ventana derecha)
- Area de menus y mensajes
- Area de PSW

AREA DE MEMORIA

En la ventana izquierda de la pantalla se presenta un bloque del contenido de 128 localidades de memoria, mientras se realiza la simulación, estas pueden ser las primeras 128 localidades o a partir de la localidad 2080H (puesto que en esta área de memoria se encuentra el programa a ejecutarse).

En la opción de Tabla se utiliza esta ventana para desplegar el contenido de la tabla. De igual manera en la opción de Setear se pueden observar los puntos de parada.

AREA DE EJECUCION DE INSTRUCCIONES

La ventana derecha se utiliza para observar el desarrollo de la ejecución del programa. En esta se despliegan los mnemónicos de 16 instrucciones presentándose en video invertido la instrucción a ejecutarse.

El simulador nos permite ingresar a esta ventana para escrolar los mnemónicos de todas las instrucciones del programa.

AREA DE MENUS Y MENSAJES

El simulador presenta un sistema de menus similar al del Lotus 1-2-3, eligiéndose este sistema por su versatilidad.

Una opción puede elegirse con la primera letra la cual se encuentra intensificada, con las flechas de dirección izquierda derecha o con el espaciador.

Cada opción presenta una ayuda que le permite al usuario conocer su función, si la opción contiene un submenú este

aparecerá en la línea superior.

AREA DE PSW

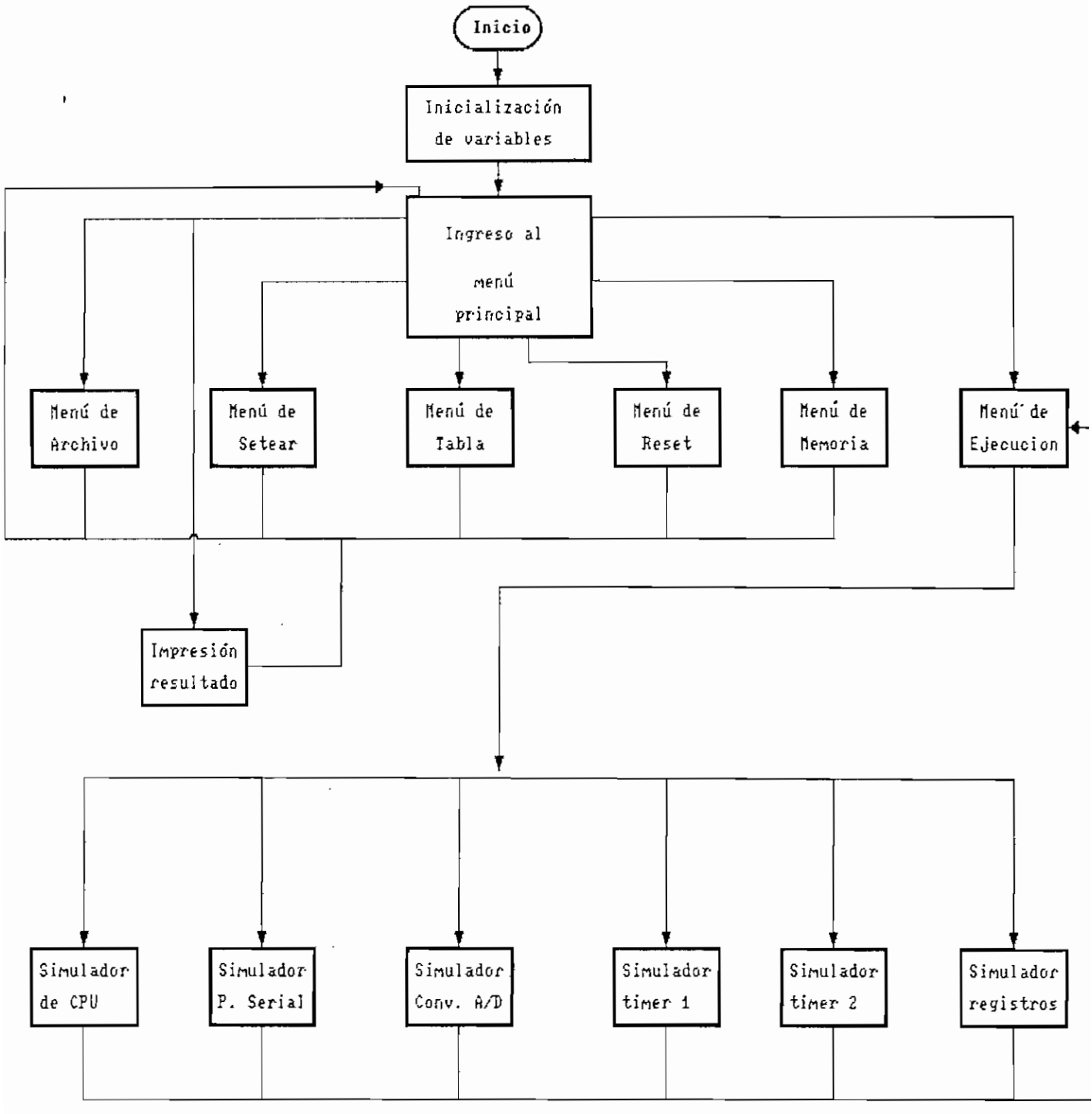
Esta área se encuentra entre la ventana izquierda y la derecha y nos muestra el contenido de la palabra de estado, la cual consta de las interrupciones y las banderas.

El simulador desensambla automáticamente cuando se ha producido un cambio en el contenido de la memoria, sin variar el archivo que contiene el programa en código hexadecimal, esta caracteritica permite variar ciertos parámetros del programa y ejecutarlo sin necesidad de volver a ensamblarlo, sobre todo en lo que hace relación a saltos o lazos de repetición .

Los puntos de parada "break point" permiten ejecutar un programa hasta una línea determinada y se puede revisar banderas, valores de determinadas localidades de memoria, registros especiales etc, y luego continuar la ejecución evaluando así el programa simulado.

En el simulador a más de la CPU, memoria, registros especiales, líneas de entrada y salida se simulan un conversor análogo digital, pórtico serial, timer1 y timer2.

3.1 DIAGRAMA DE BLOQUES



3.2 SIMULACION DE LA CPU

Para simular la CPU se programó en subrutinas todas las instrucciones del MCS-96 para lo cual se utiliza las operaciones lógicas y aritméticas que permite el lenguaje BASIC, es decir estas realizan el papel del ALU.

El archivo 8096.PRN contiene el set de instrucciones y está estructurado de la siguiente manera:

-Nemónico de la instrucción

-Código real

-Número de Bytes

-Código para la simulación

-Número de estados de tiempo

El Nemónico es el nombre de cada una de las intrucciones por ejemplo.

LD	Cargar una palabra
LDB	Cargar un Byte
ADD	Suma de palabras

El código real es el valor hexadecimal dado por el fabricante con el cual el microcontrolador identifica las instrucciones

por ejemplo.

```
LD      161 4 52 5
LDB     176 3 56 4
ADD      68 4 29 5
```

El número de bytes indica la cantidad de bytes de cada intrucción

```
LD      161 4 52 5
```

El código para la simulación es un número asignado a cada instrucción con el cual el simulador la localiza para su ejecución por ejemplo

```
LD    161 4 52 5
```

El número de estados de tiempo indica la cantidad de estados de tiempo que tarda el microcontrolador en ejecutar cada instrucción por ejemplo

```
LD    161 4 52 5
```

De acuerdo a la arquitectura del 8096 el contador de

programa se inicia con un valor de 2080H localidad en la cual debe encontrarse la primera instrucción a ejecutarse.

3.3 SIMULACION DE LA MEMORIA

La simulación de la memoria se realiza mediante dos arreglos de 32k cada uno, sus elementos representan un registro de 8 bits por ejemplo

Localidad	Vector
0000H	M(0)
0100H	M(256)

Cuando son palabras el byte menos significativo se encuentra en una localidad par y el byte más significativo en la inmediata superior,por ejemplo:

Localidad	Vector
0010H	byte menos significativo
0011H	byte más significativo

De igual manera si son dobles palabras, por ejemplo:

Localidad	Vector
0010H	byte menos significativo
0011H	segundo byte
0012H	tercer byte
0013H	byte más significativo

En el programa se da como dirección tanto de una palabra como de una doble palabra solo la dirección del byte menos significativo que como ya se indicó debe ser par, si es impar no se garantiza un resultado consistente.

3.4 SIMULACION DE LOS REGISTROS

La simulación de los registros se realiza de la misma manera que la memoria pues estos tiene un lugar específico en la memoria del microprocesador.

Los registros especiales son los siguientes:

Localidad	Nombre		Vector
	Lectura	Escritura	
19H	STACK POINTER	STACK POINTER	M(25)
18H	------	-----	M(24)

17H	-----	PWM_CONTROL	M(23)
16H	IOS1	IOC1	M(22)
15H	IOS0	IOC0	M(21)
14H	-----	-----	M(20)
13H	RESERVED	RESERVED	M(19)
12H	-----	-----	M(18)
11H	SP_STAT	SP_CON	M(17)
10H	IO PORT 2	IO PORT 2	M(16)
0FH	IO PORT 1	IO PORT 1	M(15)
0EH	IO PORT 0	BAUD_RATE	M(14)
0DH	TIMER2 (HI)	-----	M(13)
0CH	TIMER2 (LO)	RESERVED	M(12)
0BH	TIMER1 (HI)	-----	M(11)
0AH	TIMER1 (LO)	WATCHDOG	M(10)
09H	INT_PENDING	INT_PENDING	M(9)
08H	INT_MASK	INT_MASK	M(8)
07H	SBUF (RX)	SBUF (TX)	M(7)
06H	HSI_STATUS	HSO_COMMAND	M(6)
05H	HSI_TIME (HI)	HSO_TIME (HI)	M(5)
04H	HSI_TIME (LO)	HSO_TIME (LO)	M(4)
03H	AD_RESULT (HI)	HSI_MODE	M(3)
02H	AD_RESULT (LO)	AD_COMMAND	M(2)
01H	R0 (HI)	R0 (HI)	M(1)
00H	R0 (LO)	R0 (LO)	M(0)

En la opción de tabla se debe asignar como variables los nombres dados a las localidades de los registros utilizados en el programa a simularse,asi por ejemplo:

Localidad	Variable
01H	R0 (HI)
00H	R0 (LO)

Cuando son palabras la variable debe terminar en X con lo cual se denominará al byte menos significativo con L y al más significativo con H , así por ejemplo:

Localidad	Variable
0030H	AX

con lo cual se obtendrá :

Localidad	Variable
0030H	AL
0031H	AH

3.5 SIMULACION DEL CONVERSOR A/D

La conversión de análogo a digital se realiza por aproxima-

ciones sucesivas, para lo cual los datos ingresan mediante un archivo que contiene los valores muestreados de la función análoga.

Este archivo debe ser un archivo secuencial en código ascii el cual puede crearse con cualquier editor u hoja de cálculo así por ejemplo:

Si la función es SEN 2T tomando un $\delta T = 0,1$ entonces los datos muestreados serán

0

0,199

0,389

0,565

0,717

Las conversiones se realizan cada 168 estados de tiempo y se presentan en el canal escogido por el usuario. La simulación del conversor no se realiza como dos procesos paralelos debido a que el microcomputador no puede atenderlos simultáneamente por lo que cada vez que se contabilizan los 168 estados de tiempo que supuestamente debería demorarse la conversión se carga el simulador del conversor A/D.

3.6 SIMULADOR DE LOS TIMERS

Los timers en la MCS-96 son dos TIMER 1 y TIMER 2 .
El TIMER 1 se simula como un contador que inicia su conteo desde el momento en que comienza la ejecución y cuenta cada 8 estados de tiempo y el resultado de este conteo esta almacenado en la localidad 0AH (TIMER1 (LO)) y 0BH (TIMER1 (HI)).
El TIMER 2 puede funcionar como un contador o como un divisor programable. Para simular un contador se dan las señales a través del teclado en forma manual y para simular un divisor programable se inicia el conteo cuando el usuario lo inicializa contando máximo cada 8 estados de tiempo, esto también lo programa el usuario, el conteo esta almacenado en la localidad 0CH (TIMER2-(LO)) y 0DH (TIMER2 (HI)).

3.6.1 SIMULADOR DEL PORTICO SERIAL

Mediante la programación del registro correspondiente se escoge como base de tiempo el timer1 o el timer2.
Cuando se escoge el timer2 existen dos opciones,una manual y otra automática. En la automática al igual que con el timer1 la velocidad de Tx y Rx esta dada por el valor de la frecuencia la cual esta determinada por los registros correspondientes. En la manual la velocidad esta dada por pulsaciones en el teclado.

Al igual que para la simulación A/D. la entrada de datos se lo realiza por medio de un archivo, de iguales características que el conversor. Con la diferencia que contiene las palabras binarias por ejemplo.

Archivo Rx.prn

0 1 1 1 0 0 0 1

De acuerdo a los estados de tiempo que se demoran las instrucciones ejecutadas, el simulador relaciona con la velocidad de Rx para tomar un dato del archivo, para luego procesarlo. El simulador en la transmisión presenta en pantalla los datos transmitidos más no los guarda en archivo.

Como se desprende de esto la simulación no es a tiempo real por iguales razones que en el conversor A/D.

CAPITULO IV

PROGRAMAS

El simulador esta conformado por seis módulos:

- Programa Principal
- Inicialización
- Menú
- Conversor A/D
- Pórtico Serial
- Desensamblador

4.1 PROGRAMA PRINCIPAL

4.1.1 DESCRIPCION

Al iniciar el programa principal se ha cargado el archivo a simular y la tabla de variables.

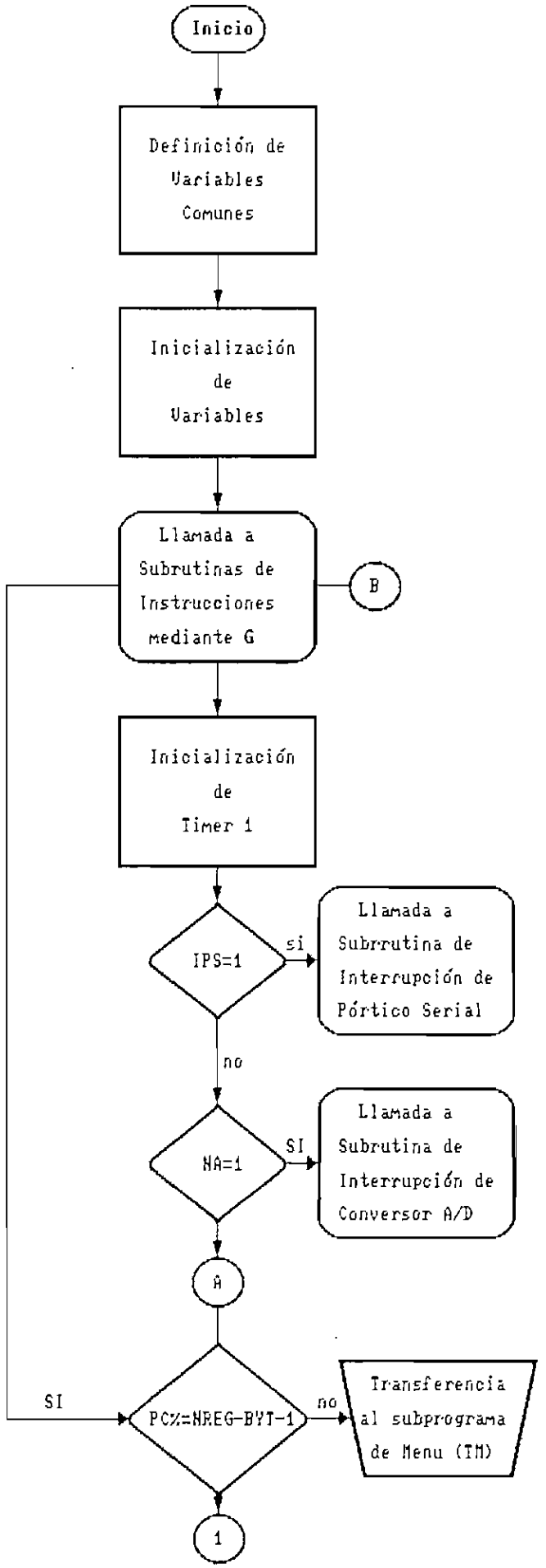
La ejecución del programa se inicia con el contador de programa en la localidad 2080H (puesto que al hacer un reset el contador de programa apunta a esta localidad siendo esta una característica del hardware del microcontrolador), el cual indica la localidad de memoria en la que se encuentra el código

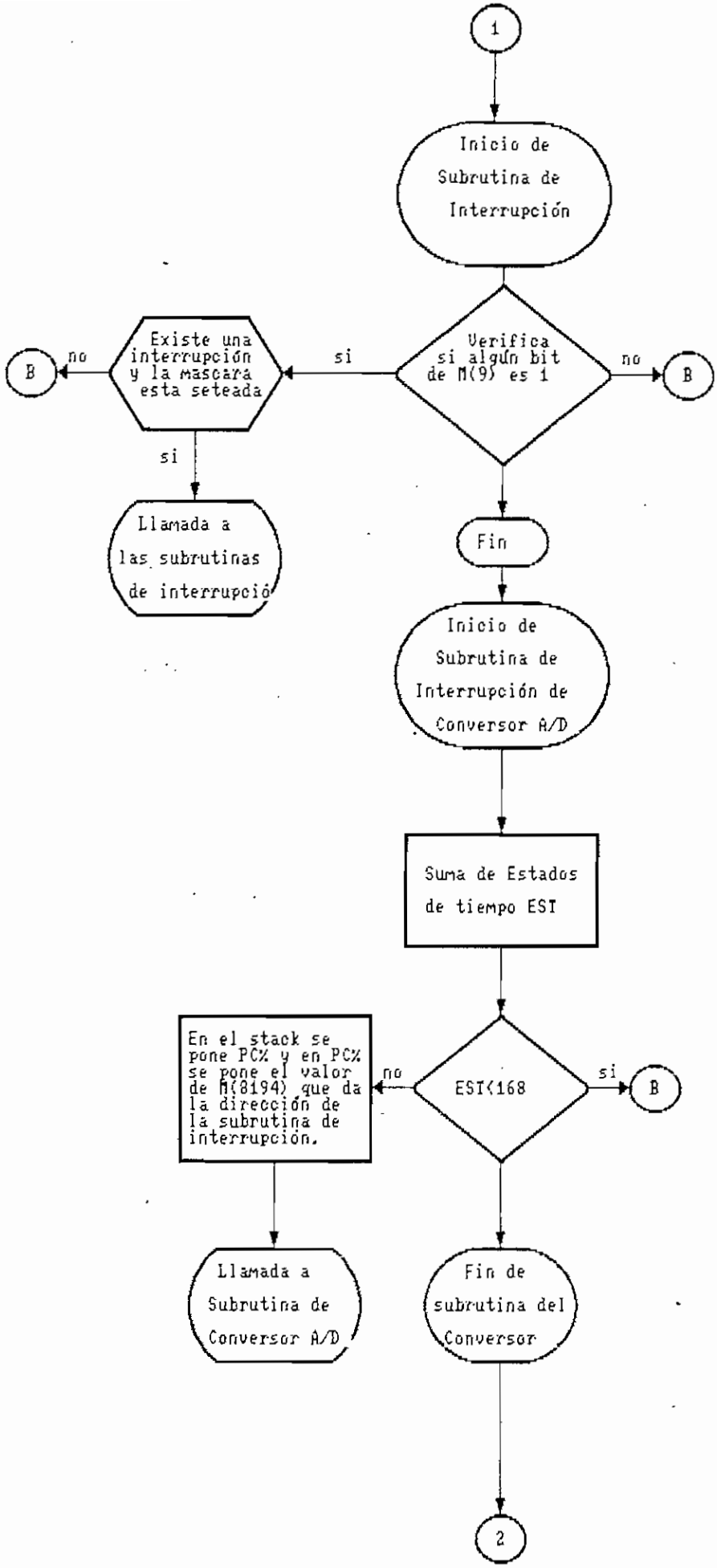
real de la instrucción, con este podemos encontrar el código de simulación, que nos permite determinar la subrutina de la instrucción deseada.

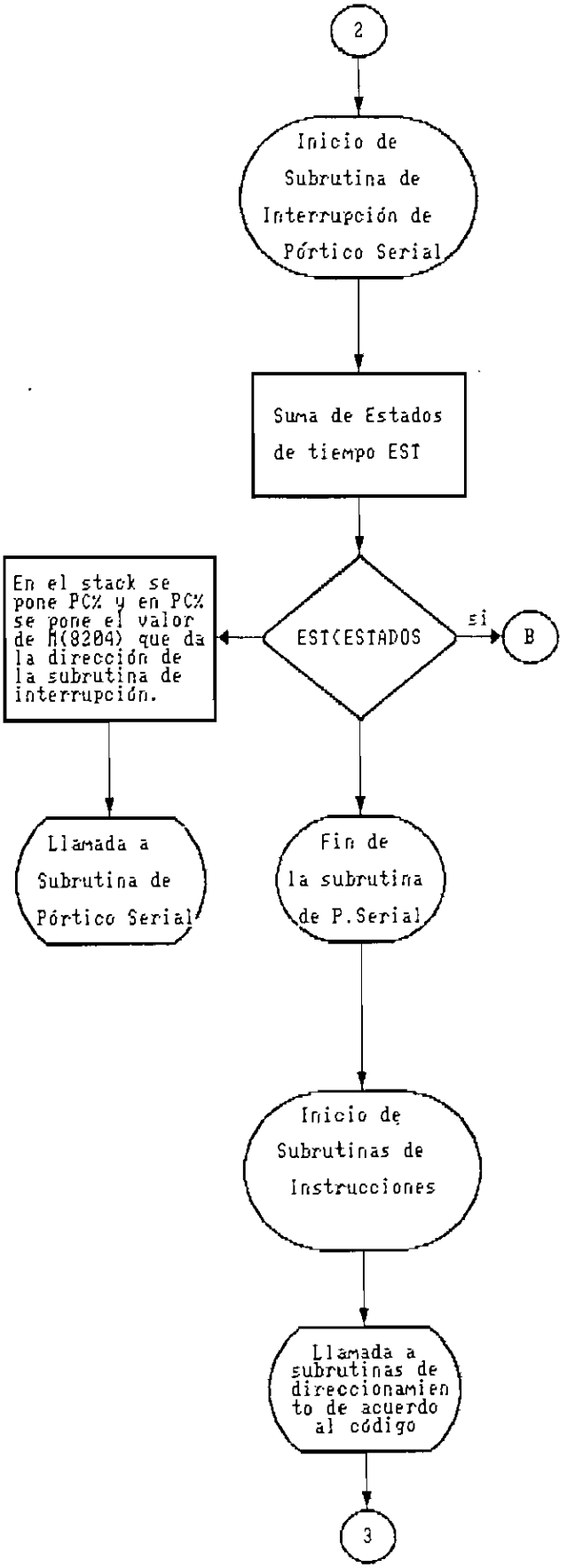
Si la instrucción tiene direccionamiento, mediante el código real se envía a la subrutina del direccionamiento correspondiente, en donde se cargan los operandos y se ejecuta la operación . Si no tiene direccionamiento se ejecuta la operación inmediatamente.

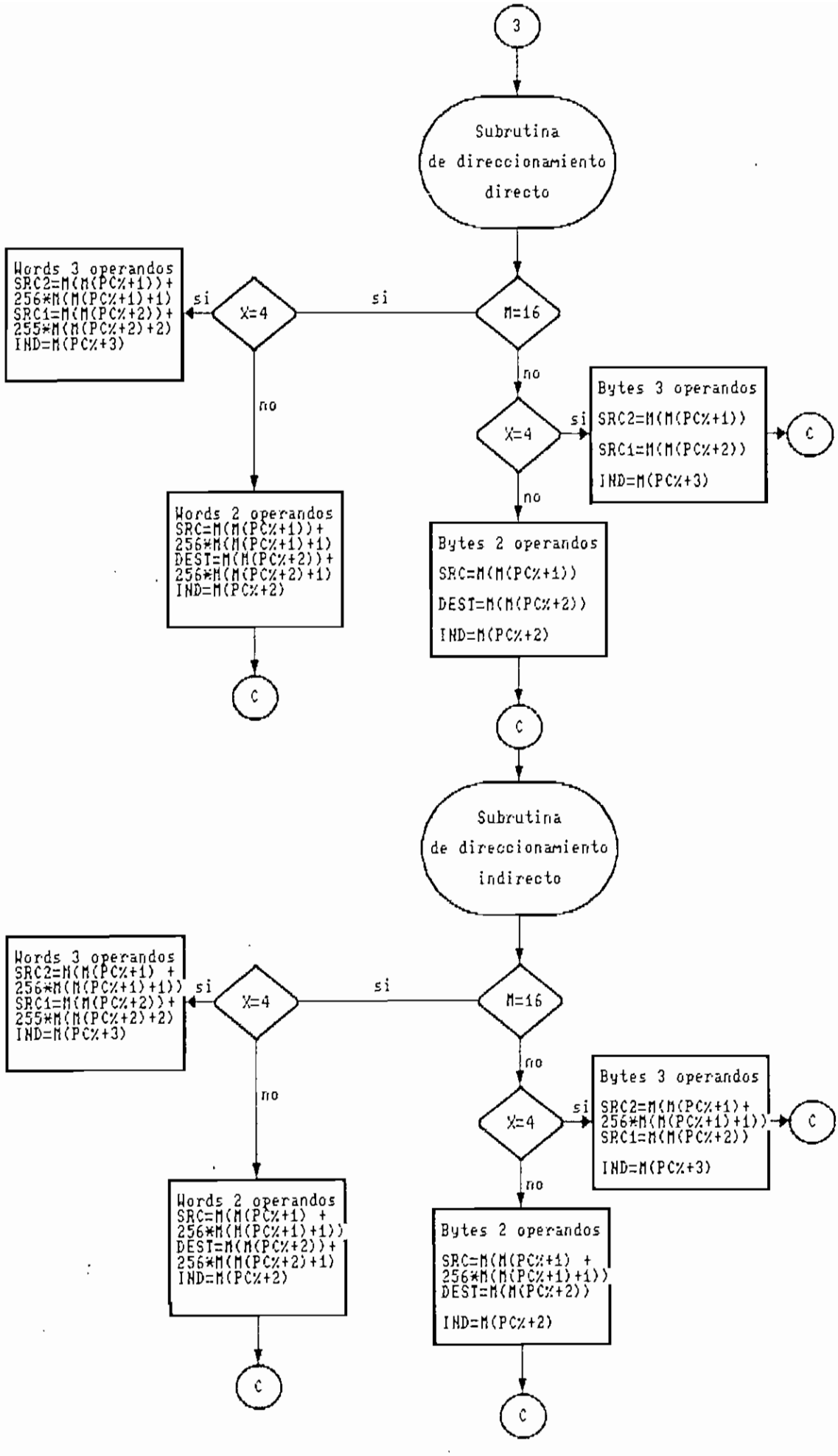
El contador de programa se incrementa de acuerdo al número de bytes de cada instrucción, de esta manera al terminar la ejecución de las subrutinas el contador de programa está apuntando a la siguiente instrucción, entonces se verifica si el valor del contador de programa es el break point y si es así se termina la ejecución, sino se repite el proceso.

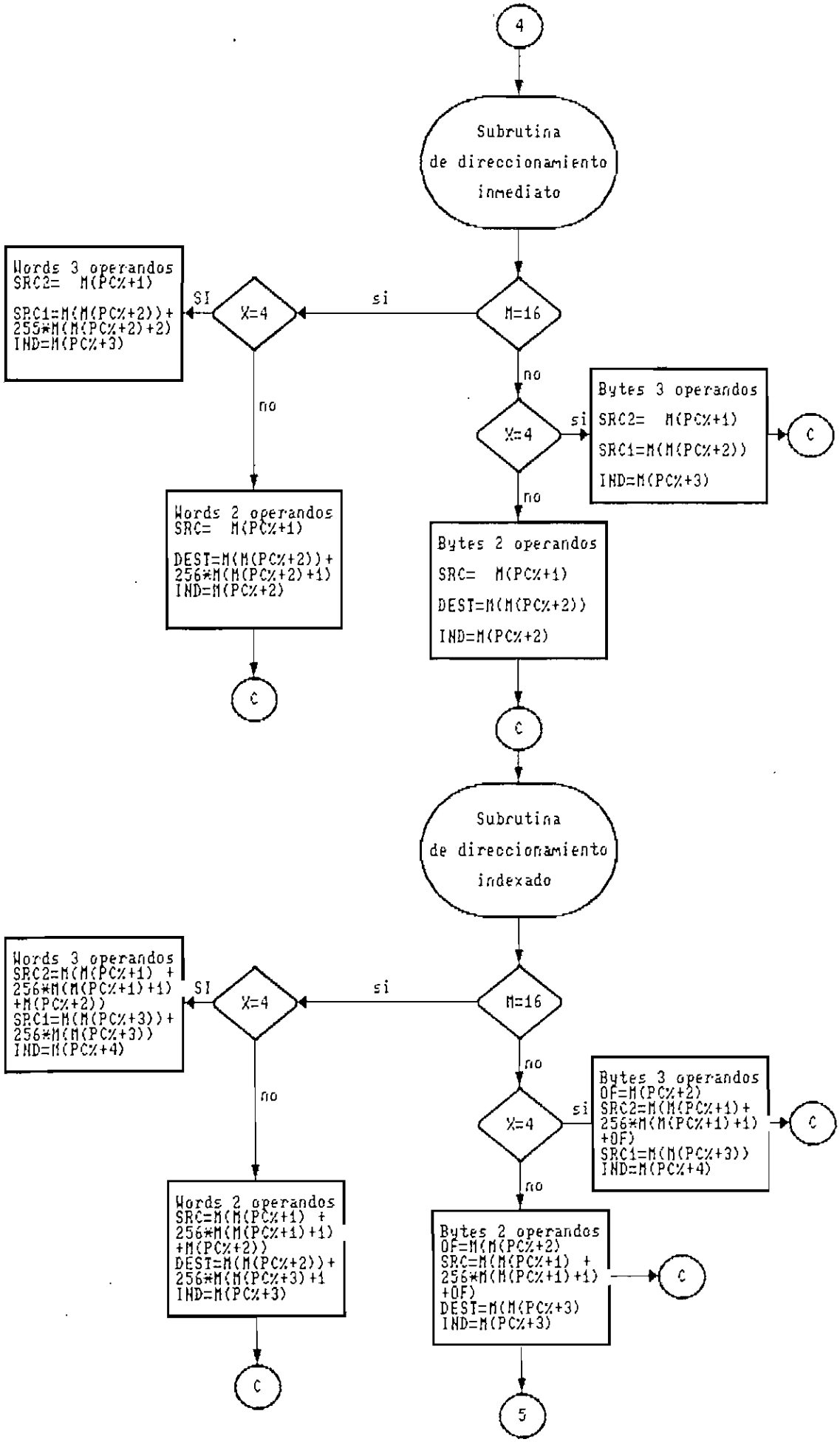
4.1.2 DIAGRAMA DE FLUJO DE PROGRAMA PRINCIPAL




1
Inicio de Subrutina de Interrupción
Verifica si algún bit de M(9) es 1
si
no
B
Existe una interrupción y la mascara esta seteada
no
B
si
Llamada a las subrutinas de interrupció
Fin
Inicio de Subrutina de Interrupción de Conversor A/D
Suma de Estados de tiempo EST
EST<168
no
si
B
En el stack se pone PC% y en PC% se pone el valor de M(8194) que da la dirección de la subrutina de interrupción.
Llamada a Subrutina de Conversor A/D
Fin de subrutina del Conversor
2


2
Inicio de Subrutina de Interrupción de Pórtico Serial
Suma de Estados de tiempo EST
EST<ESTADOS
si
B
En el stack se pone PC% y en PC% se pone el valor de M(8204) que da la dirección de la subrutina de interrupción.
Llamada a Subrutina de Pórtico Serial
Fin de la subrutina de P.Serial
Inicio de Subrutinas de Instrucciones
Llamada a subrutinas de direccionamiento de acuerdo al código
3


3
Subrutina de direccionamiento directo
M=16
si
X=4
si
Words 3 operandos
SRC2=M(M(PC%+1))+
256*M(M(PC%+1)+1)
SRC1=M(M(PC%+2))+
255*M(M(PC%+2)+2)
IND=M(PC%+3)
no
Words 2 operandos
SRC=M(M(PC%+1))+
256*M(M(PC%+1)+1)
DEST=M(M(PC%+2))+
256*M(M(PC%+2)+1)
IND=M(PC%+2)
C
no
X=4
si
Bytes 3 operandos
SRC2=M(M(PC%+1))
SRC1=M(M(PC%+2))
IND=M(PC%+3)
C
no
Bytes 2 operandos
SRC=M(M(PC%+1))
DEST=M(M(PC%+2))
IND=M(PC%+2)
C
Subrutina de direccionamiento indirecto
M=16
si
X=4
si
Words 3 operandos
SRC2=M(M(PC%+1) +
256*M(M(PC%+1)+1))
SRC1=M(M(PC%+2))+
255*M(M(PC%+2)+2)
IND=M(PC%+3)
no
Words 2 operandos
SRC=M(M(PC%+1) +
256*M(M(PC%+1)+1))
DEST=M(M(PC%+2))+
256*M(M(PC%+2)+1)
IND=M(PC%+2)
C
no
X=4
si
Bytes 3 operandos
SRC2=M(M(PC%+1)+
256*M(M(PC%+1)+1))
SRC1=M(M(PC%+2))
IND=M(PC%+3)
C
no
Bytes 2 operandos
SRC=M(M(PC%+1) +
256*M(M(PC%+1)+1))
DEST=M(M(PC%+2))
IND=M(PC%+2)
C


4
Subrutina de direccionamiento inmediato
M=16
si
X=4
SI
Words 3 operandos
SRC2= M(PC%+1)
SRC1=M(M(PC%+2))+ 255*M(M(PC%+2)+2)
IND=M(PC%+3)
no
Words 2 operandos
SRC= M(PC%+1)
DEST=M(M(PC%+2))+ 256*M(M(PC%+2)+1)
IND=M(PC%+2)
C
no
X=4
si
Bytes 3 operandos
SRC2= M(PC%+1)
SRC1=M(M(PC%+2))
IND=M(PC%+3)
C
no
Bytes 2 operandos
SRC= M(PC%+1)
DEST=M(M(PC%+2))
IND=M(PC%+2)
C
Subrutina de direccionamiento indexado
M=16
si
X=4
SI
Words 3 operandos
SRC2=M(M(PC%+1) + 256*M(M(PC%+1)+1) +M(PC%+2))
SRC1=M(M(PC%+3))+ 256*M(M(PC%+3))
IND=M(PC%+4)
no
Words 2 operandos
SRC=M(M(PC%+1) + 256*M(M(PC%+1)+1) +M(PC%+2))
DEST=M(M(PC%+2))+ 256*M(M(PC%+3)+1
IND=M(PC%+3)
C
no
X=4
si
Bytes 3 operandos
OF=M(PC%+2)
SRC2=M(M(PC%+1)+ 256*M(M(PC%+1)+1) +OF)
SRC1=M(M(PC%+3))
IND=M(PC%+4)
C
no
Bytes 2 operandos
OF=M(M(PC%+2)
SRC=M(M(PC%+1) + 256*M(M(PC%+1)+1) +OF)
DEST=M(M(PC%+3)
IND=M(PC%+3)
C
5


5
Inicio de subrutina de impresión
Mediante un for se imprime DASM$ Y M
Salida en pantalla de DASM$ Y M
B

4.2 SUBPROGRAMAS

4.2.1 DESCRIPCION

4.2.1.1 PROGRAMA DE INICIALIZACION (TI)

En este programa es donde se inicializan las variables, también se crea la pantalla de presentación, luego se carga el archivo 8096.PRN descrito anteriormente y finalmente presenta la pantalla de menú .Este programa ejecuta al programa de menú TM.

4.2.1.2 PROGRAMA DE MENU (TM)

En el programa de menú existen las siguientes opciones:

- Archivo
- Memoria
- Setear
- Reset
- Ejecutar
- Tabla
- Fin
- Dos
- Imprimir

- Hexadecimal

- Análogo

- Serial

ARCHIVO/ARCHIVO/HEXADECIMAL

Esta opción permite cargar el archivo en hexadecimal del programa a simularse, almacenando los datos en los arreglos que simulan la memoria y el subindice de estos está dado por el contador de programa que es un dato del archivo.

A continuación se muestra el archivo *.hex que produce el ensamblador.

```
:020000020000FC
:10208000A1000118012055082002FDFD3B02FDB0120A0D
:1020900002ICB003ID5420201EACIEIEC3IE2BIC930A0D
:0720A000172071032027DF680A0D
:00000001FF0A0D
```

en el que la primera línea representa el tipo de microcontrolador que se esta utilizando y otra información adicional del ensamblador, por esta razón esta línea no debe tomarse en cuenta.

A partir de la segunda linea se encuentran los datos dados en hexadecimal:

Cada linea se inicia con : (dos puntos).

Los dos primeros digitos de la izquierda significan el número de datos por linea en este caso es el 10H.

Los siguientes cuatro digitos determinan la localidad de memoria a partir de la cual se deben almacenar los datos en este ejemplo es la localidad 2080H

A continuación existe dos digitos más los cuales son siempre 00 y se los utiliza como separadores.

Finalmente se encuentran los datos que vienen en pares asi se ve en el ejemplo: A1 00 01 1B 01 20 55 0B 20 02 FD FD 3B 02 FD B0 12 0A 0D.

Luego viene un byte para especificar el check sum , luego un caracter de control correspondiente a line feed y finalmente el carry return.

De igual manera se procede con las siguientes lineas hasta la linea final que tiene la forma:

:00000001FF

MEMORIA/CONTENIDO

En el contenido se pregunta la dirección de la memoria que desea revisar la cual se muestra en pantalla en forma hexadecimal. La pantalla muestra 128 localidades de memoria entre las cuales esta la localidad deseada. Adicionalmente se pueden ver las siguientes localidades con las flechas arriba y abajo.

MEMORIA/MODIFICAR

En esta opción se pregunta la dirección de la memoria que desea modificar, también se pregunta el valor con el cual desea modificarla, con estos datos el programa modifica la localidad deseada con el valor dado.

MEMORIA/BUSQUEDA

Se pregunta el valor que desea encontrar , también se pregunta la dirección desde donde debe empezar la búsqueda, con estas dos variables se procede a localizar la localidad de memoria que corresponda a este valor imprimiendo el valor del subíndice en hexadecimal, oprimiendo cualquir tecla continua con la búsqueda y con ESC termina el proceso.

4.2.1.3 PROGRAMA DE CONVERSION A/D (TA)

Este programa comienza abriendo el archivo donde se encuentran los datos para la conversión, el procedimiento es similar a la opción de archivo. Se trae un dato cada conversión.

El canal escogido se encuentra en los 3 bits menos significativos del elemento M(2) del arreglo que representa la memoria. Una vez escogido el canal se procede a la conversión por aproximaciones sucesivas.

El conversor A/D es de 10 bits por lo cual el resultado obtenido se coloca de la siguiente manera:

Los dos bits menos significativos se guardan en los dos bits más significativos del elemento M(2) y los 8 bits más significativos en el elemento M(3).

Una vez realizada la conversión , regresa al programa principal y sigue la ejecución del programa hasta que pasen otros 168 estados de tiempo en los cuales el programa principal llama a este subprograma y el proceso comienza otra vez con el siguiente dato.

Como segundo paso se verifica el MODO en el que trabaja y este está indicado en los dos bits menos significativos del elemento M(17) del arreglo que simula la memoria.

De acuerdo al MODO y a la bandera de recepción o transmisión se realiza la ejecución.

En la recepción se lee un dato del archivo RXD.PRN cada tiempo de recepción, es decir dado por los baudios ya calculados, se guarda en el elemento M(7) del arreglo que simula la memoria.

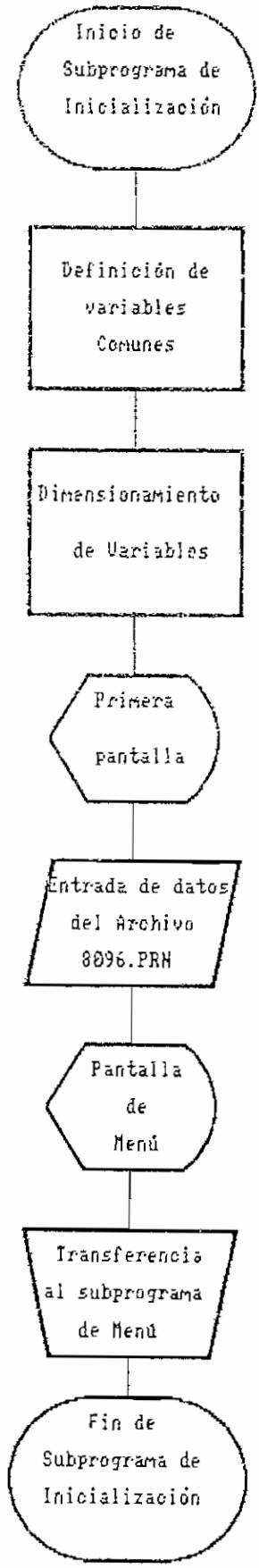
En la transmisión se toman los bits del elemento M(7) de uno en uno cada tiempo de transmisión desde el menos significativo y se imprimen en TXD que representa la salida TXD.

Cuando se termina la transmisión o recepción se setea la bandera correspondiente.

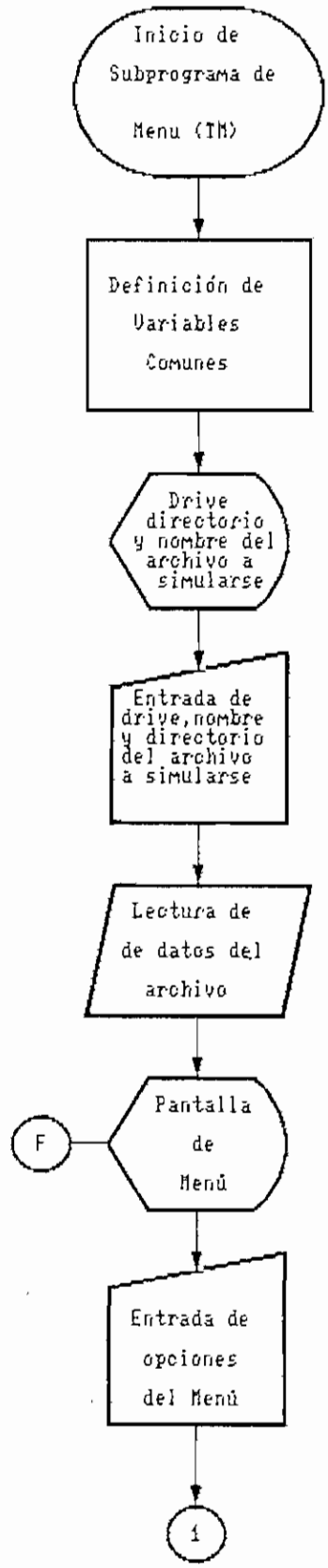
4.2.1.5 PROGRAMA DE DESENSAMBLADO (TD)

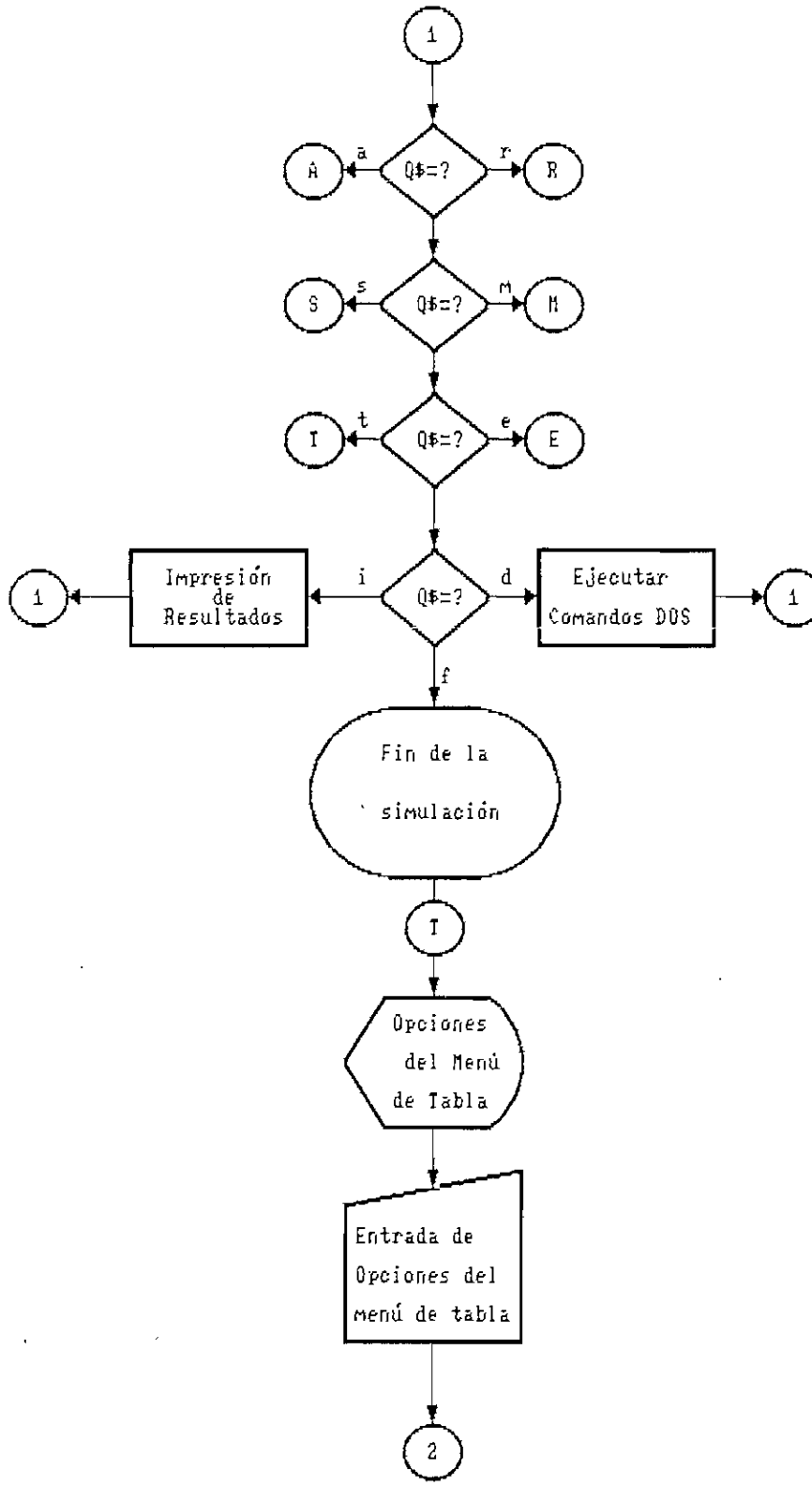
Este programa tiene una estructura similar al programa principal es decir realiza el mismo proceso de ejecución pero las subrutinas que simulan el ALU no realizan ninguna operación sino que conforman los nemónicos de cada instrucción del programa con lo cual se convierte el programa hexadecimal a nemónico.

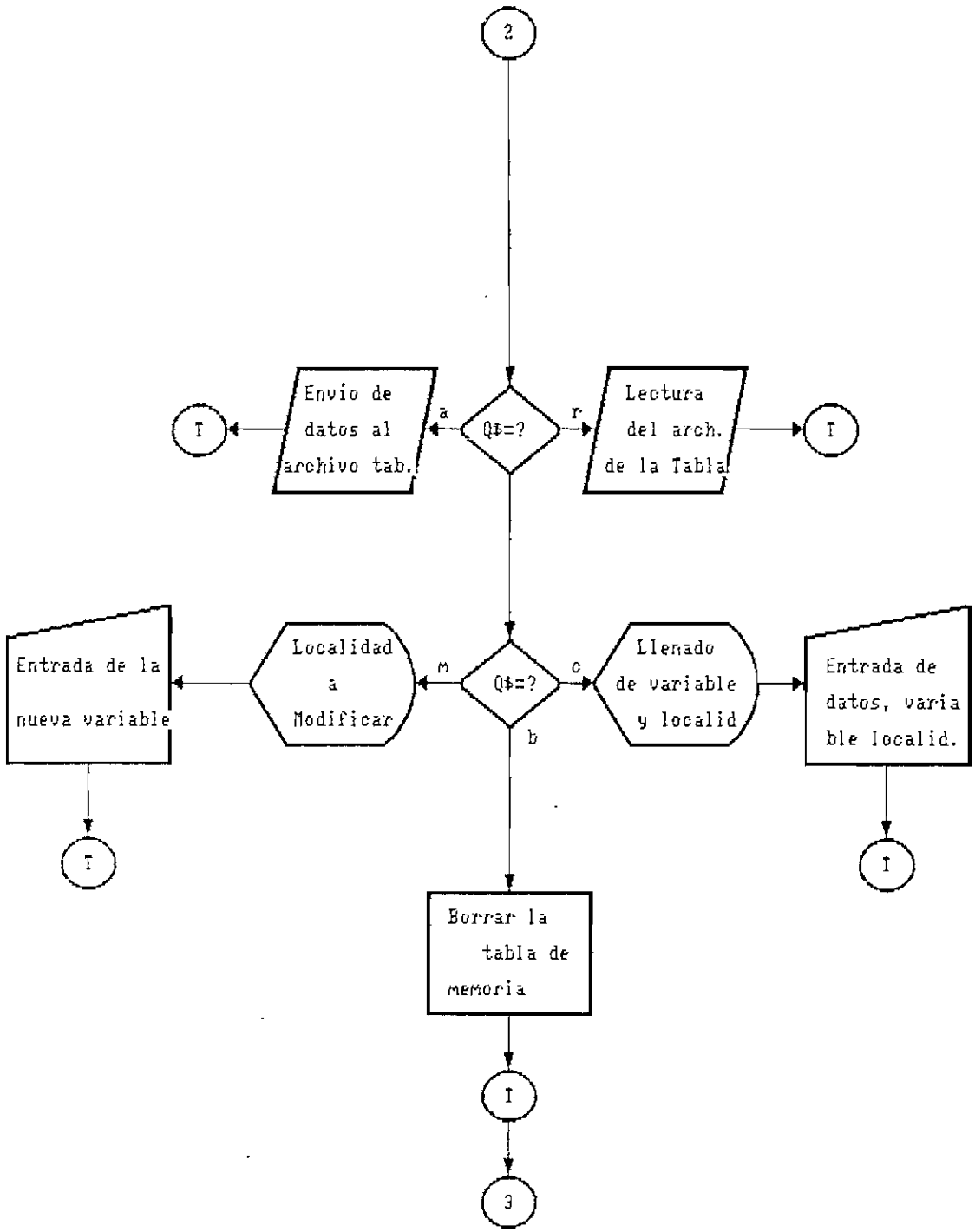
DIAGRAMA DE FLUJO DE SUBPROGRAMA DE INICIALIZACION (TI)

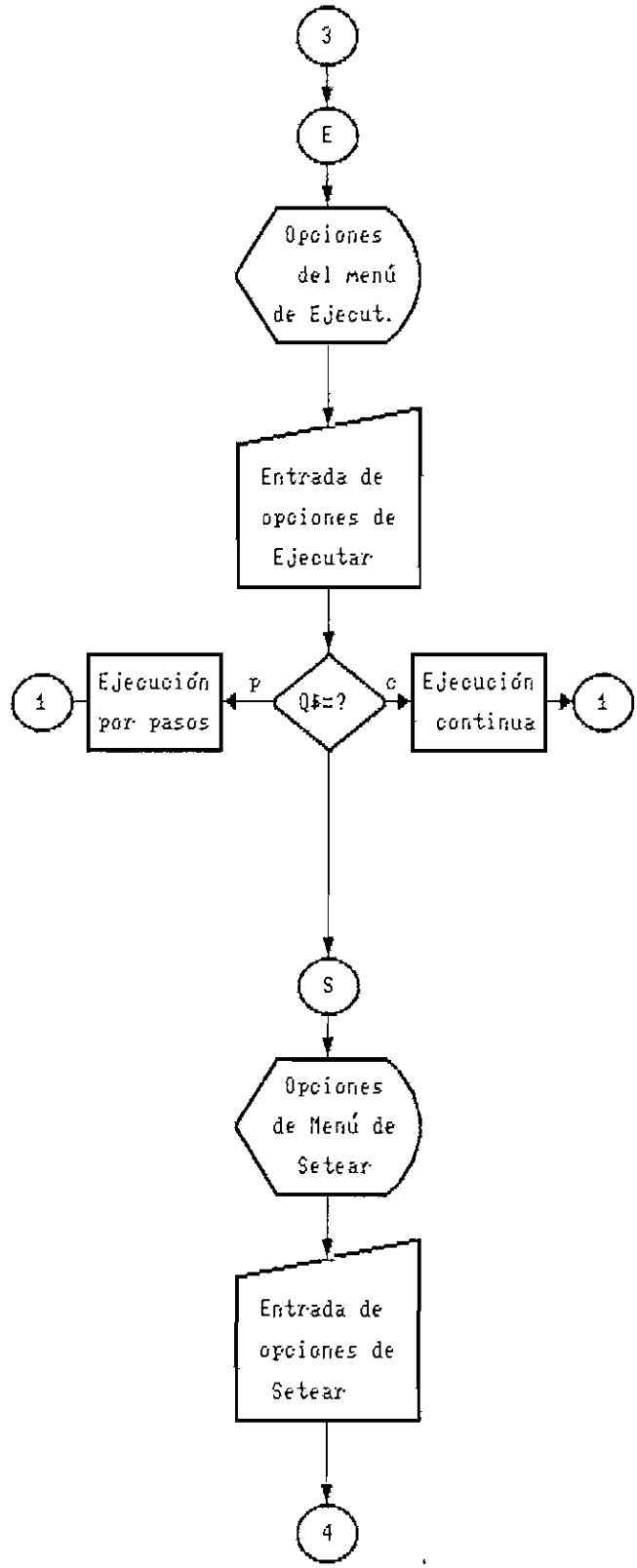


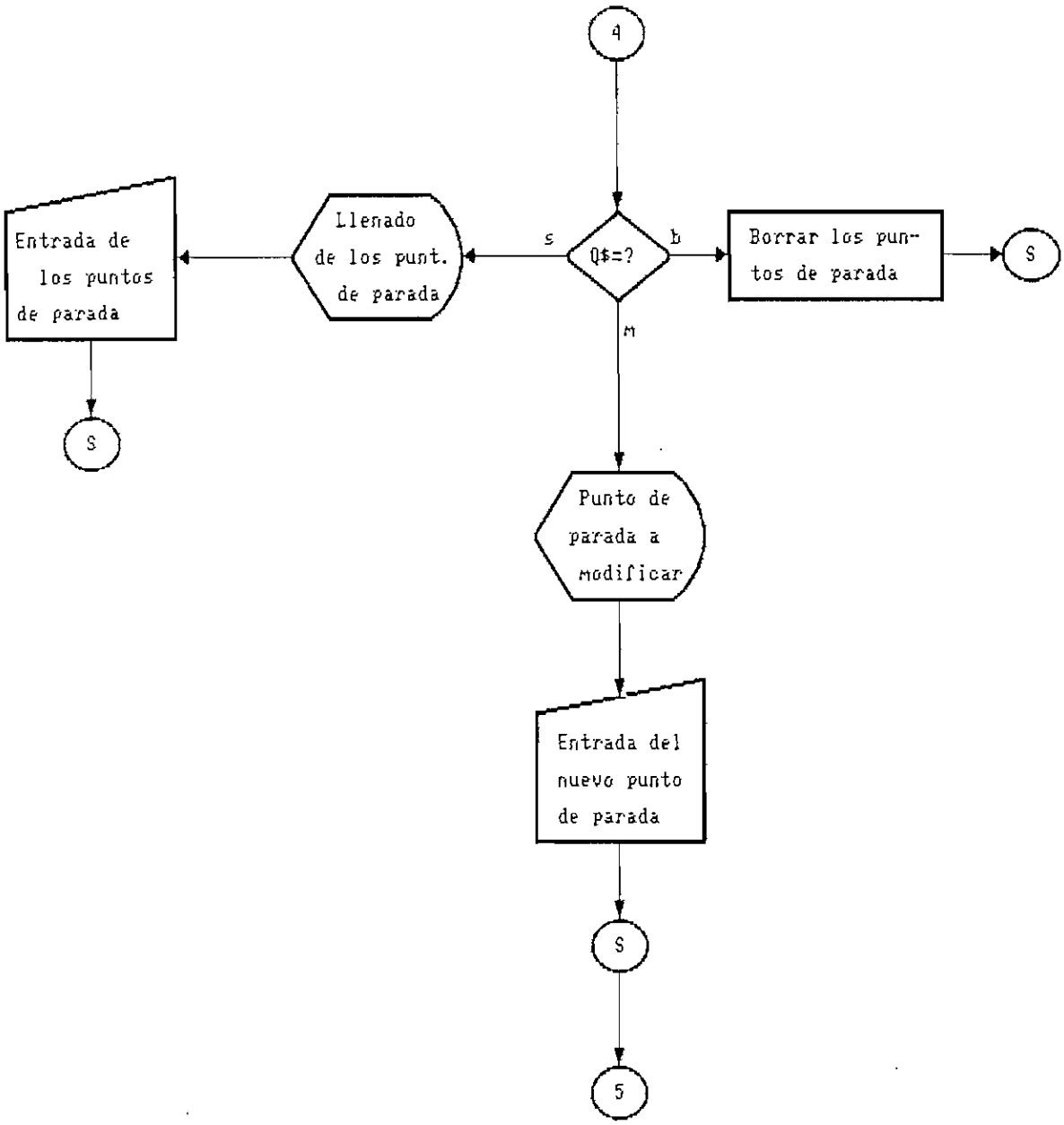
SUBPROGRAMA DE MENU (TM)

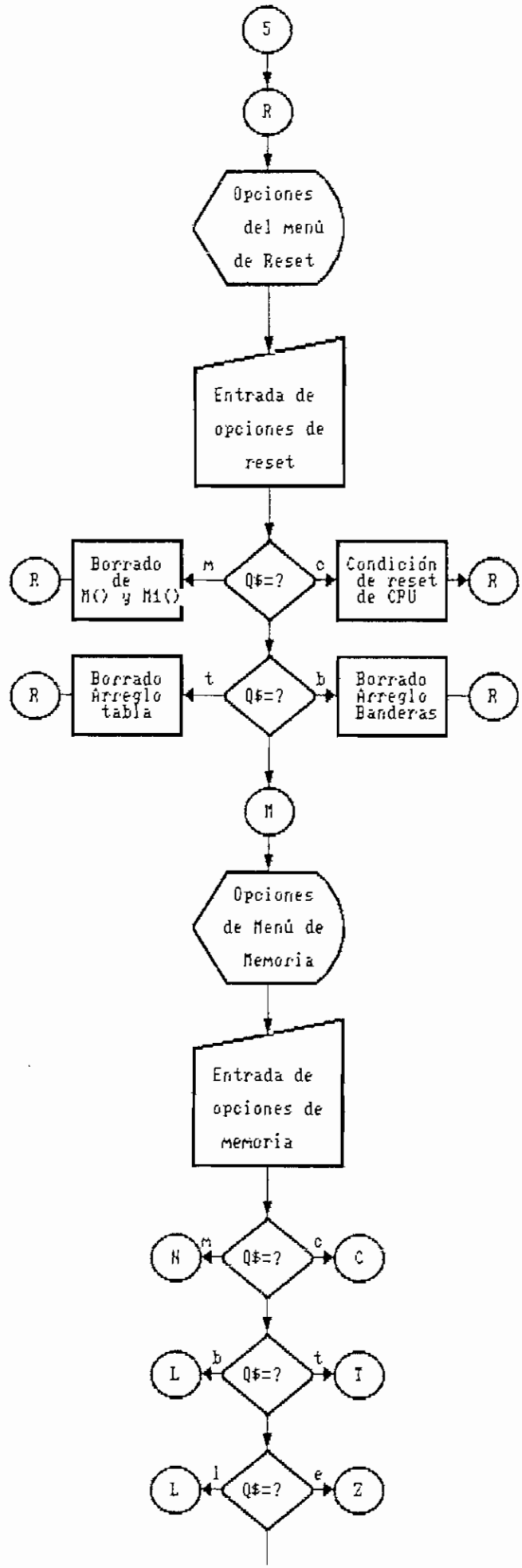



1
A
a
Q$=?
r
R
S
s
Q$=?
M
M
I
t
Q$=?
e
E
1
Impresión de Resultados
i
Q$=?
d
Ejecutar Comandos DOS
1
f
Fin de la simulación
I
Opciones del Menú de Tabla
Entrada de Opciones del menú de tabla
2

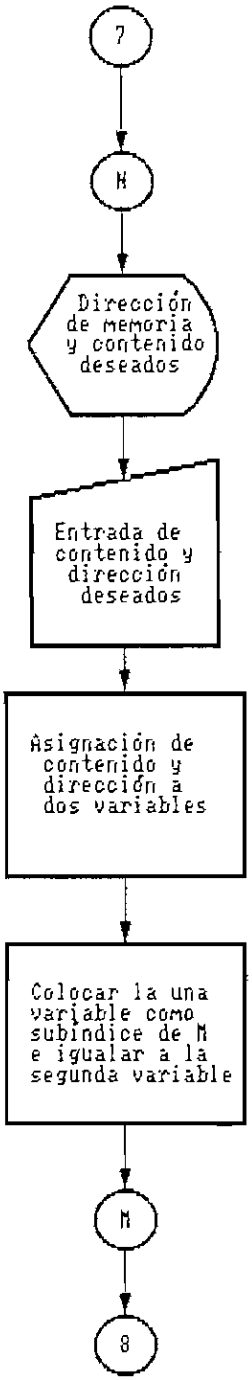

2
Envio de datos al archivo tab.
a
Q$=?
r
Lectura del arch. de la Tabla
T
T
Entrada de la nueva variable
Localidad a Modificar
m
Q$=?
c
Llenado de variable y localid
Entrada de datos, variable localid.
b
T
I
Borrar la tabla de memoria
I
3

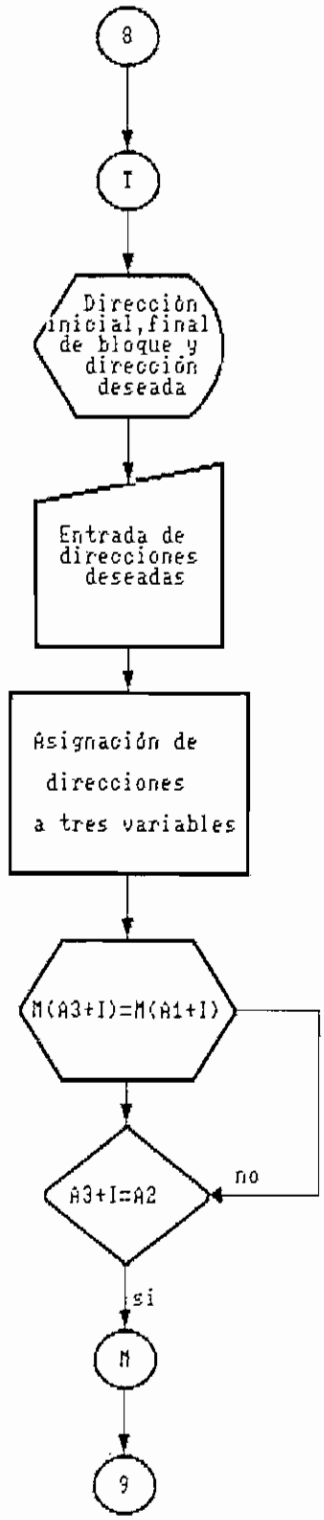

3
E
Opciones del menú de Ejecut.
Entrada de opciones de Ejecutar
1
Ejecución por pasos
p
Q$=?
c
Ejecución continua
1
S
Opciones de Menú de Setear
Entrada de opciones de Setear
4

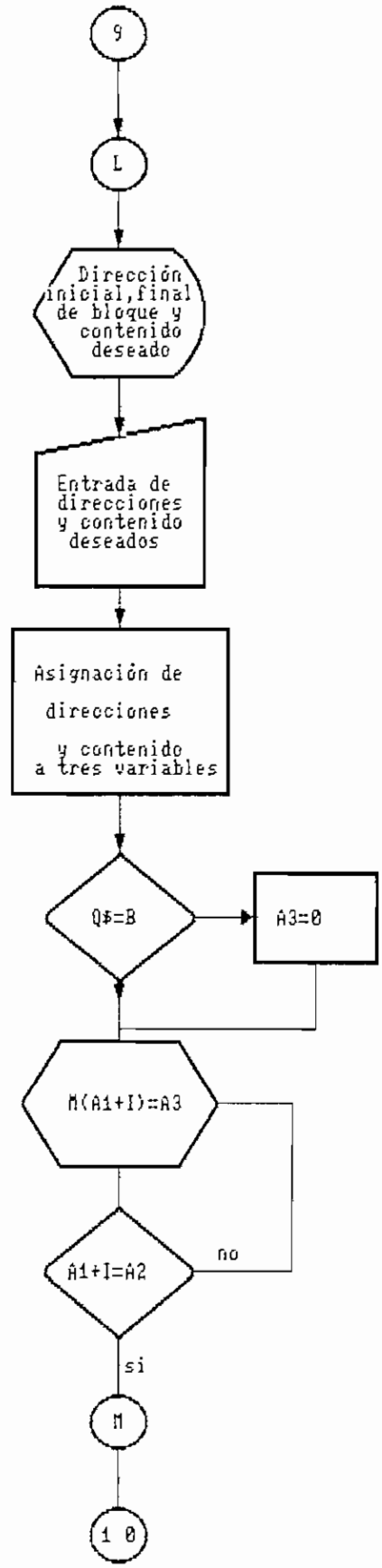

4
Entrada de los puntos de parada
Llenado de los punt. de parada
s
Q$=?
b
Borrar los puntos de parada
S
m
S
Punto de parada a modificar
Entrada del nuevo punto de parada
S
5

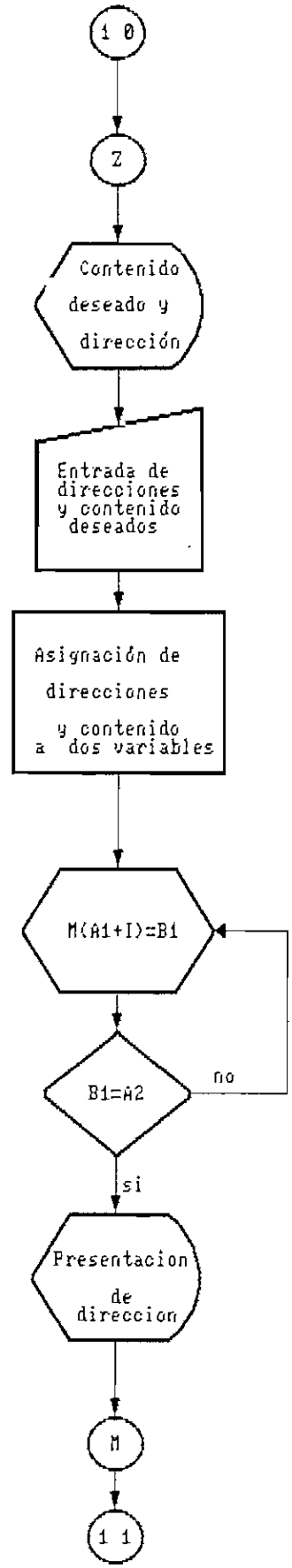

S
R
Opciones del menú de Reset
Entrada de opciones de reset
R
Borrado de M() y M1()
m
Q$=?
c
Condición de reset de CPU
R
R
Borrado Arreglo tabla
t
Q$=?
b
Borrado Arreglo Banderas
R
M
Opciones de Menú de Memoria
Entrada de opciones de memoria
N
m
Q$=?
c
C
L
b
Q$=?
t
T
L
l
Q$=?
e
Z

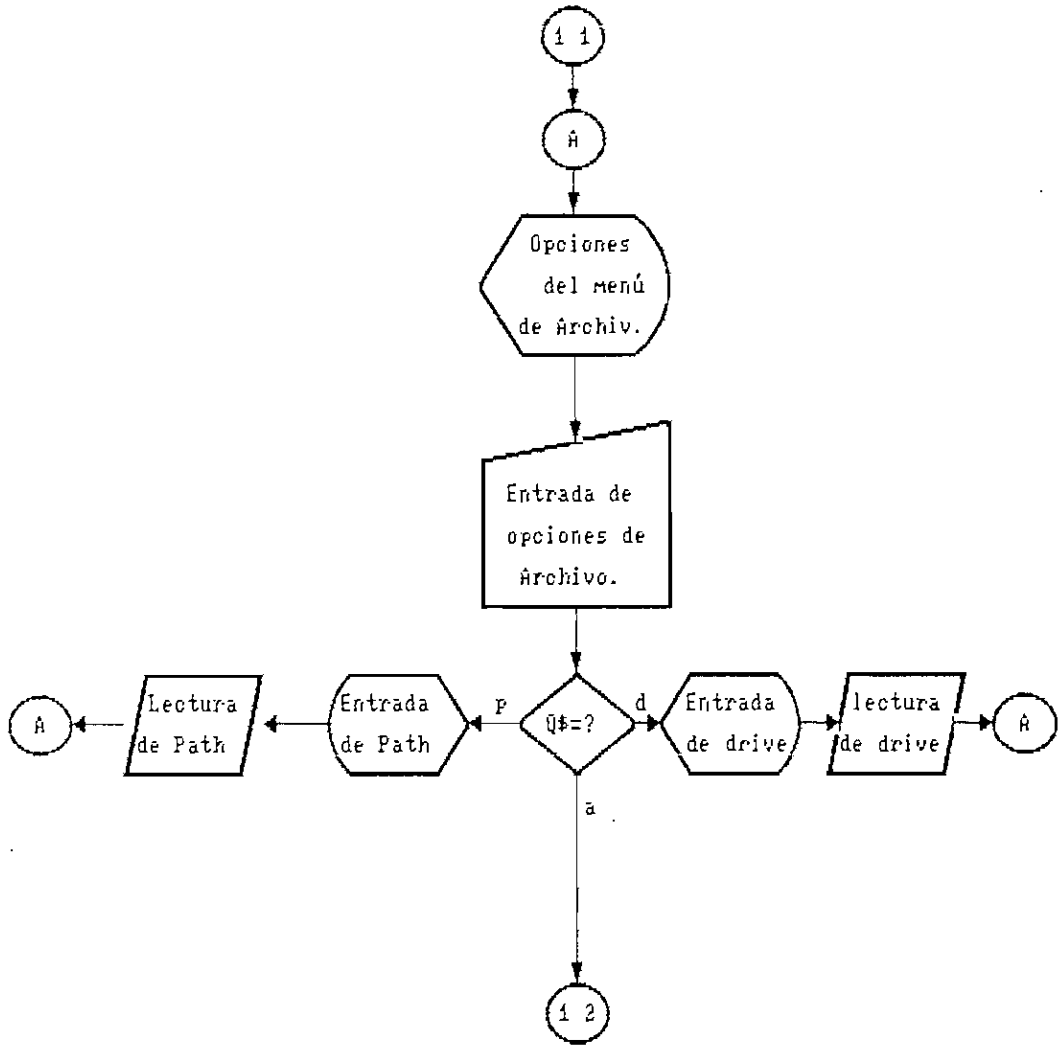

6
C
Dirección de memoria deseada
Entrada de dirección deseada
Asignación de dirección a una variable
Colocar la variable como un subindice de M.
Presenta la memoria deseada
M
7

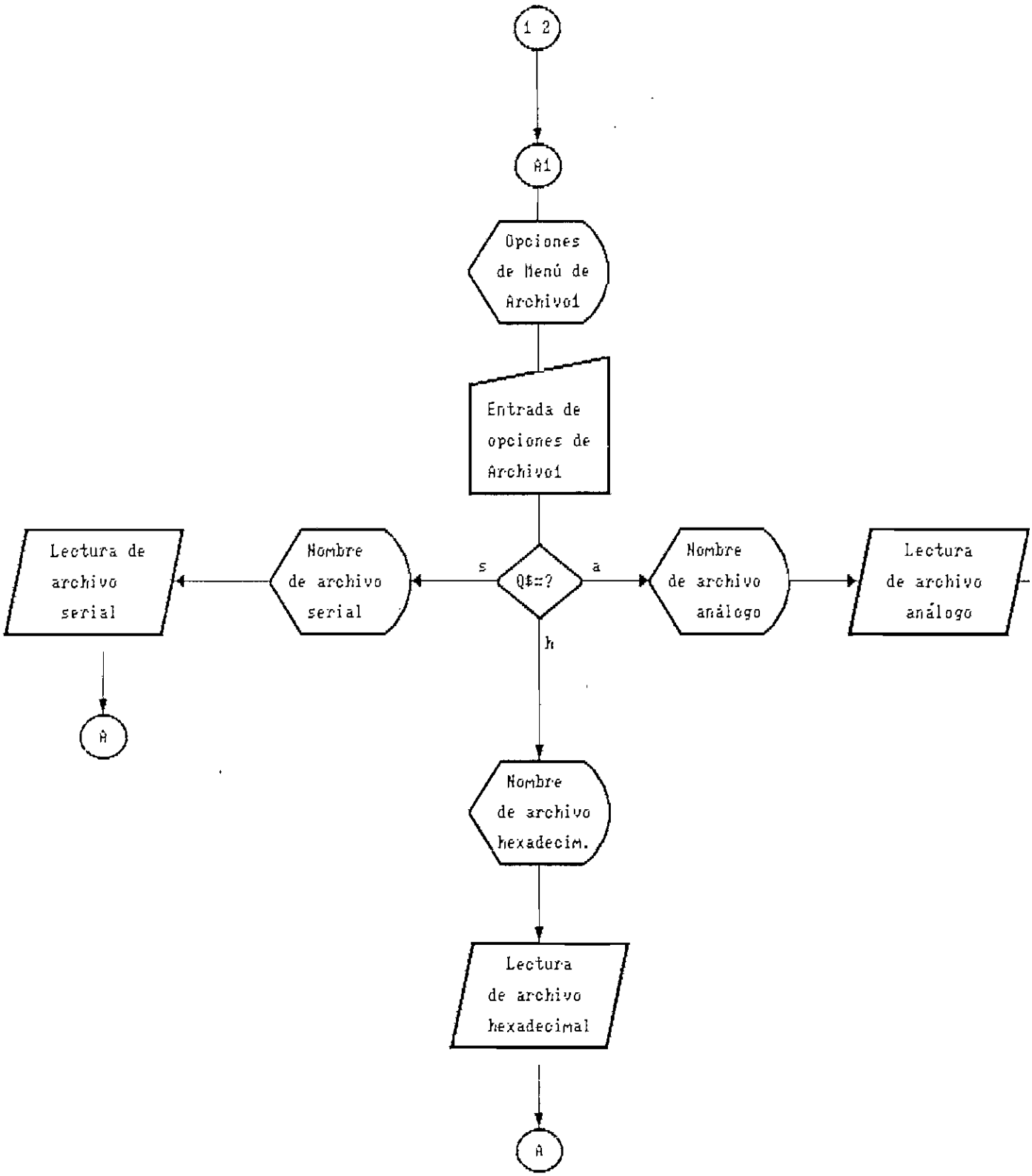

7
M
Dirección de memoria y contenido deseados
Entrada de contenido y dirección deseados
Asignación de contenido y dirección a dos variables
Colocar la una variable como subindice de M e igualar a la segunda variable
M
8


8
I
Dirección inicial,final de bloque y dirección deseada
Entrada de direcciones deseadas
Asignación de direcciones a tres variables
M(A3+I)=M(A1+I)
A3+I=A2
no
si
M
9

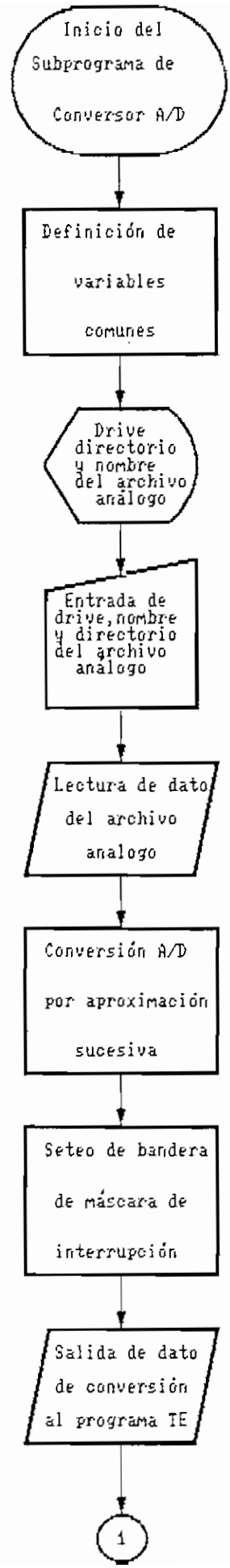

9
L
Dirección inicial,final de bloque y contenido deseado
Entrada de direcciones y contenido deseados
Asignación de direcciones y contenido a tres variables
Q$=B
A3=0
M(A1+I)=A3
A1+I=A2
no
si
M
1 0

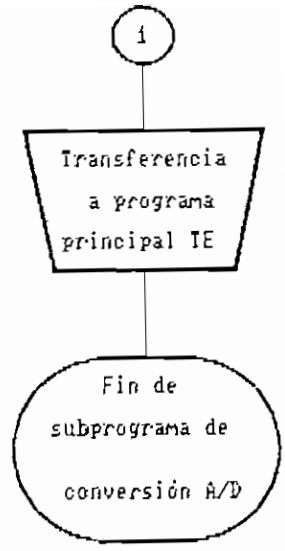

1 0
Z
Contenido deseado y dirección
Entrada de direcciones y contenido deseados
Asignación de direcciones y contenido a dos variables
M(A1+I)=B1
B1=A2
no
si
Presentacion de direccion
M
1 1


1 1
A
Opciones del menú de Archiv.
Entrada de opciones de Archivo.
Q$=?
p
d
a
Entrada de Path
Lectura de Path
A
Entrada de drive
lectura de drive
A
1 2

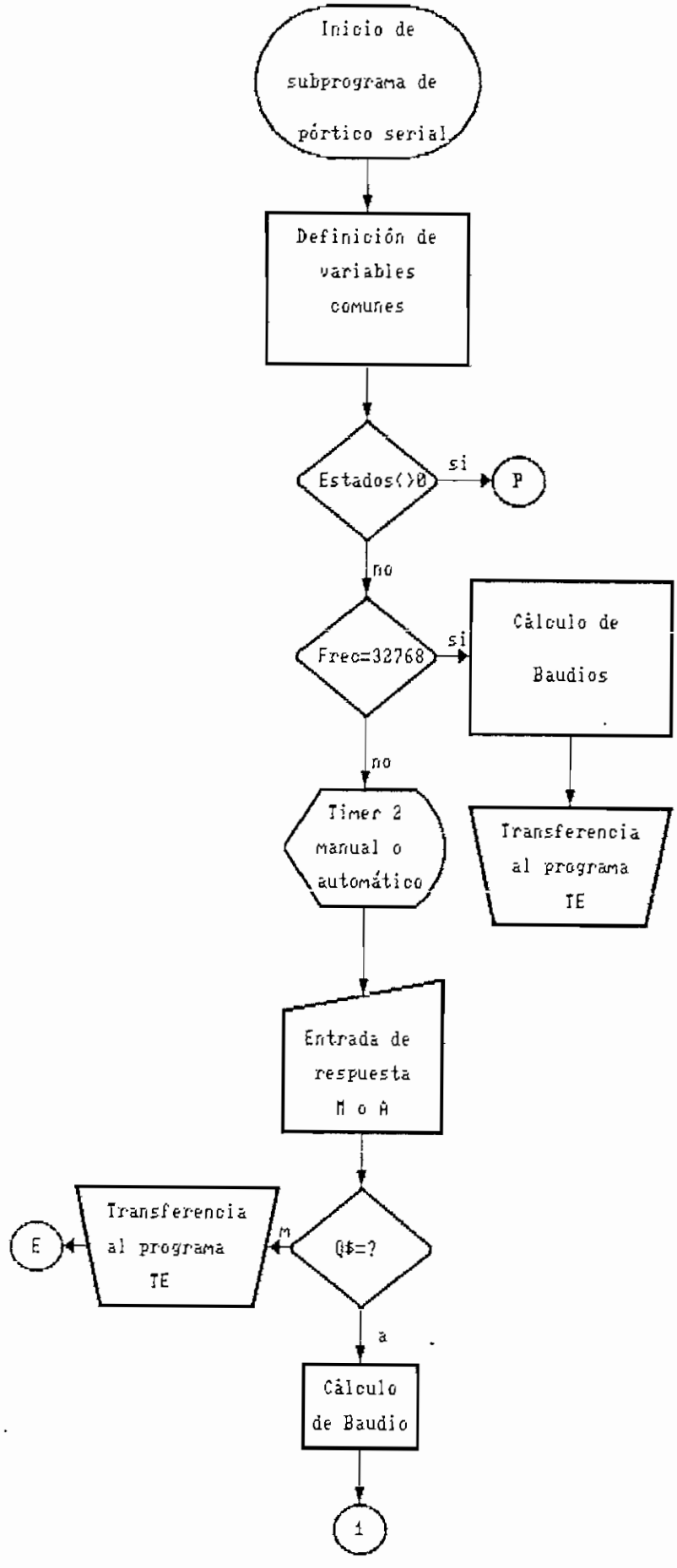

1 2
A1
Opciones de Menú de Archivo1
Entrada de opciones de Archivo1
Q$=?
s
a
h
Nombre de archivo serial
Lectura de archivo serial
Nombre de archivo análogo
Lectura de archivo análogo
Nombre de archivo hexadecim.
Lectura de archivo hexadecimal
A
A

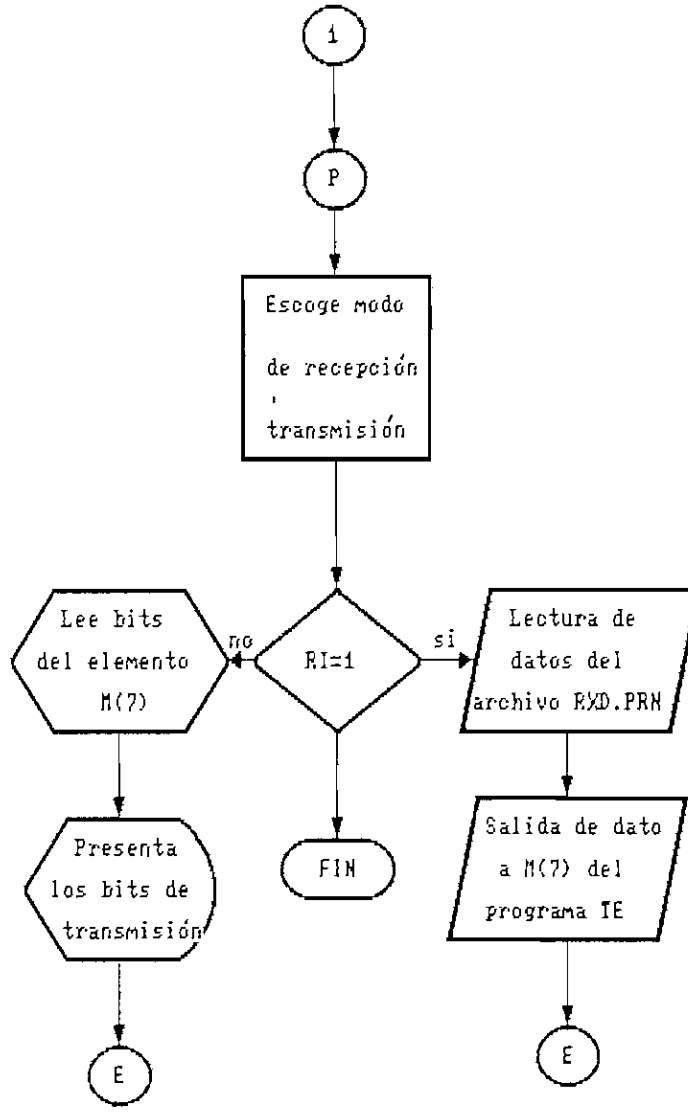
SUBPROGRAMA DE CONVERSION A/D



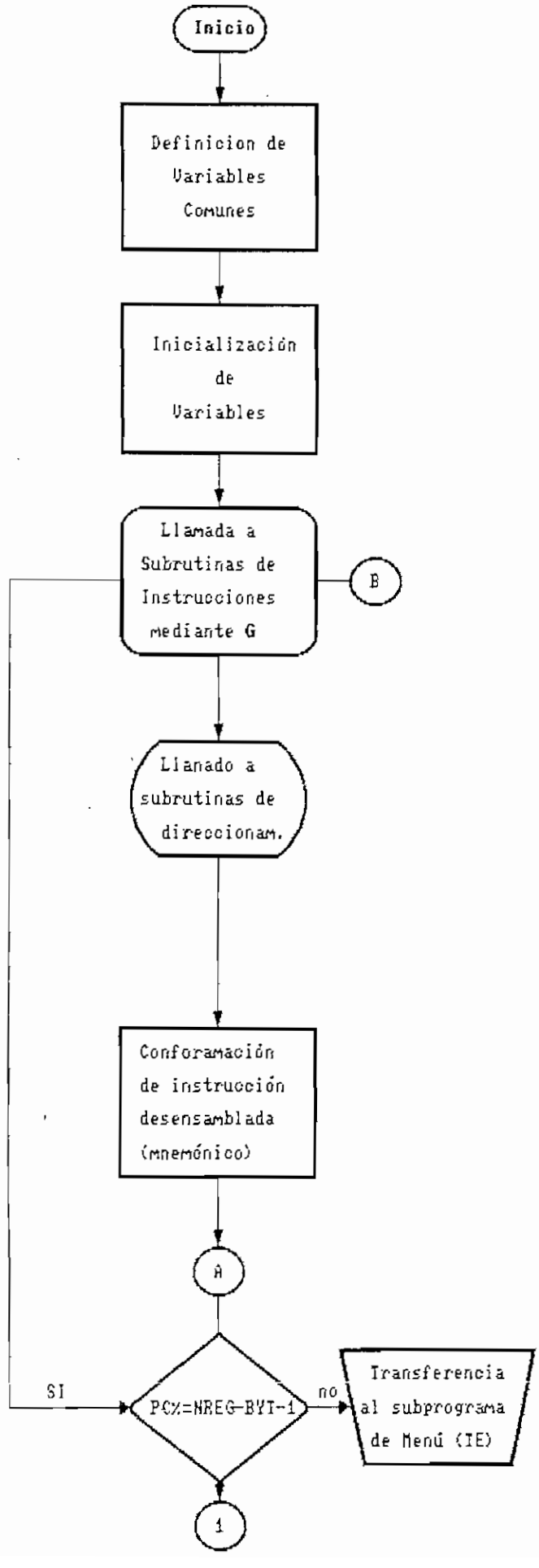

i
Transferencia a programa principal TE
Fin de subprograma de conversión A/D

4.1.2 SUPROGRAMA DEL PORTICO SERIAL (PSERIAL)




1
P
Escoge modo
de recepción
transmisión
Lee bits
del elemento
M(7)
no
RI=1
si
Lectura de
datos del
archivo RXD.PRN
Presenta
los bits de
transmisión
FIN
Salida de dato
a M(7) del
programa IE
E
E

1.2 DIAGRAMA DE FLUJO DE SUBPROGRAMA DE DESENSAMBLADO.



CAPITULO V

RESULTADOS Y CONCLUSIONES

5.1 EJEMPLOS DE APLICACION

En este capítulo se describen algunos ejemplos que demuestran el funcionamiento del Simulador.

Los programas que se han implementado son previamente ensamblados utilizando el Cross Assembler C16 en formato HOF "INT16".

5.1.1 INTERPOLACION

Este programa permite encontrar el valor de la ordenada (Y), ingresando la absisa (X).

El algoritmo utilizado se basa en la interpolación lineal, en donde los valores de la función están dados por una tabla para absisas múltiplos de 16, como por ejemplo:

```
look:   LDB  AL, IN_VAL     ; Cargar un registro temporal
                            ; con el valor de entrada

        SHRB AL, #3         ; Divide el byte para 8
        ANDB AL, #11111110B ;Asegura que AL sea palabra direcc.
        LDBZE AX, AL

        LD   TABLE_LOW,TABLE [AX] ; tabla_low es cargado con
                                  ; un valor de la tabla para
                                  ; (X) múltiplo de 10H

        LD     TABLE_HIGH, (TABLE+2 [AX]); tabla_high es cargado
                                          ; con el siguiente valor
                                          ; de la tabla.

        SUB   TAB_DIF, TABLE_HIGH,TABLE_LOW
        ANDB  IN_DIFB, IN_VAL, #0FH      ; cuatro bits menos
                                         ; significativos
        LDBZE IN_DIF, IN_DIFB
        MUL   OUT_DIF, IN_DIF, TAB_DIF
        SHRAL OUR_DIF, #4                ; divide para 16
        ADD   OUT,OUT_DIF, TABLE_LOW     ; suma la diferencia
        SHR   OUT, #4                    ; redondea a 12 bits
        ADDC  OUT,zero                   ; redondea si C=1
```

```
        ST    OUT, RESULT
        BR    [look]
        ORG   2100H

table:  DWL 0000H, 2000H,3400H,4C00H
        DWL 5D00H, 6A00H,7200H,7B00H
        DWL 7B00H, 7D00H,7600H,6D00H
        DWL 5D00H, 2000H,3400H,2200H
        DWL 1000H

        END
```

PROGRAMA ENSAMBLADO

```
0000001B    =                   SP:         equ     1BH
                        ; Segmento de datos
000000                  RESULT_TABLE:       ORG     0
000000                       RESULT:        DFS     2
000000                                      ORG     0
000000                         AX:          DFS     2
00000000    =                  AL:          equ     AX
00000001    =                  AH:          equ     AX+1
000002                         BX:          DFS     2
00000002    =                  BL:          equ     BX
00000003    =                  BU:          equ     BX+1
```

```
000004                          IN_VAL:         DFS     2
000006                        TABLE_LOW:        DFS     2
000008                       TABLE_HIGH:        DFS     2
00000A                          IN_DIF:         DFS     2
0000000A        =               IN_DIFB:        EQU     IN_DIF
00000C                          TAB_DIF:        DFS     2
00000E                             OUT:         DFS     2
000010                          OUT_DIF:        DFS     4

                            ; Segmento de códigos

000000                              ORG     0000H
000000 A1C00018         start:  LD  SP, #0C0H
000004 B00400           look:   LDB  AL, IN_VAL
000007 180300                   SHRB AL, #3
00000A 71FE00                   ANDB AL, #11111110B
00000D AC0000                   LDBZE AX, AL
000010 A301002106               LD   TABLE_LOW,TABLE [AX]
000015                          LD    TABLE_HIGH, (TABLE+2 [AX])
000015 48060B0C                 SUB   TAB_DIF, TABLE_HIGH,TABLE_
000019 510F040A                 ANDB  IN_DIFB, IN_VAL, #0FH
00001D AC0A0A                   LDBZE IN_DIF, IN_DIFB
000020 FE4C0C0A10               MUL   OUT_DIF, IN_DIF, TAB_DIF
000025 000000                   SHRAL OUR_DIF, #4
000028 4406100E                 ADD   OUT,OUT_DIF, TABLE_LOW
00002C 08040E                   SHR   OUT, #4
```

```
00002F 00                       ADDC  OUT,zero
000030 C0000E                   ST    OUT, RESULT
000033 E304                     BR    [look]
002100                          ORG   2100H
002100 00000020003400 table:    DWL 0000H, 2000H,3400H,4C00H
00210B 005D006A007200           DWL 5D00H, 6A00H,7200H,7B00H
002110 007B007D007600           DWL 7B00H, 7D00H,7600H,6D00H
00211B 005D0020003400           DWL 5D00H, 2000H,3400H,2200H
002120 0010                     DWL 1000H
000000                          END
00000001  AH                    00000000  AL
00000000  AX
00000002  BL                    00000003  BU
00000002  BX
0000000A  IN_DIF                0000000A  IN_DIFB
00000004  IN_VAL
00000004  LOOK                  0000000E  OUT
00000010  OUT_DIF
00000000  RESULT                00000000  RESULT_TABLE
0000001B  SP
00000000  START                 00002100  TABLE
00000008  TABLE_HIGH
00000006  TABLE_LOW             0000000C  TAB_DIF
```

```
00002F 00                       ADDC  OUT,zero
000030 C0000E                   ST    OUT, RESULT
000033 E304                     BR    [look]
002100                          ORG   2100H
002100 00000020003400 table:    DWL 0000H, 2000H,3400H,4C00H
002108 005D006A007200           DWL 5D00H, 6A00H,7200H,7B00H
002110 007B007D007600           DWL 7B00H, 7D00H,7600H,6D00H
002118 005D0020003400           DWL 5D00H, 2000H,3400H,2200H
002120 0010                     DWL 1000H
000000                          END
00000001  AH                    00000000  AL
00000000  AX
00000002  BL                    00000003  BU
00000002  BX
0000000A  IN_DIF                0000000A  IN_DIFB
00000004  IN_VAL
00000004  LOOK                  0000000E  OUT
00000010  OUT_DIF
00000000  RESULT                00000000  RESULT_TABLE
00000018  SP
00000000  START                 00002100  TABLE
00000008  TABLE_HIGH
00000006  TABLE_LOW             0000000C  TAB_DIF
```

8096/SIM

REGISTROS DE MEMORIA

```
2080 A1 00 01 18 B0 22 1C 18
2088 03 1C 71 FE 1C AC 1C 1C
2090 A3 1D 00 21 24 A3 1D 02
2098 21 26 48 24 26 2A 51 0F
20A0 22 28 AC 28 28 FE 4C 2A
20A8 28 30 0E 04 30 44 24 30
20B0 2C 0A 04 2C A4 00 2C C0
20B8 2E 2C 27 C8 00 00 00 00
20C0 00 00 00 00 00 00 00 00
20C8 00 00 00 00 00 00 00 00
20D0 00 00 00 00 00 00 00 00
20D8 00 00 00 00 00 00 00 00
20E0 00 00 00 00 00 00 00 00
20E8 00 00 00 00 00 00 00 00
20F0 00 00 00 00 00 00 00 00
20F8 00 00 00 00 00 00 00 00
```

PSW

```
I0 = 0
I1 = 0
I2 = 0
I3 = 0
I4 = 0
I5 = 0
I6 = 0
I7 = 0
ST = 0
 I = 0
 - = 0
 C = 0
VT = 0
 V = 0
 N = 0
 Z = 1
```

EJECUCION EJEM-5.HEX

```
2080   LD     SP,#100
2084   LDB    AL,IN-V
2087   SHRB   AL,#3
208A   ANDB   AL,#FE
208D   LDBZE  AL,AL
2090   LD     T-LOW,2100[AX]
2095   LD     T-HI,2102[AX]
209A   SUB    T-DIF,T-HI,T-LO
209E   ANDB   IN-D,IN-V,#F
20A2   LDBZE  IN-D,IN-D
20A6   MUL    OUT-D,IN-D,T-DI
20AA   SHRAL  OUT-D,#4
20AD   ADD    OUT,OUT-D,T-LOW
20B1   SHRA   OUT,#4
20B4   ADDC   OUT,SP
20B7   ST     OUT,RES
```

Ejecutando el programa
>Pasos Continuo

8096/SIM

REGISTROS DE MEMORIA

```
0000 00 00 00 00 00 00 00 00
0008 00 00 15 00 00 00 00 00
0010 00 00 00 00 00 00 00 00
0018 00 01 00 00 04 00 00 00
0020 00 00 20 00 00 34 00 4C
0028 00 00 00 18 40 03 40 03
0030 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00
0058 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00
0068 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00
```

PSW

```
I0 = 0
I1 = 0
I2 = 0
I3 = 0
I4 = 0
I5 = 0
I6 = 0
I7 = 0
ST = 0
 I = 0
 - = 0
 C = 0
VT = 0
 V = 0
 N = 0
 Z = 0
```

EJECUCION EJEM-5.HEX

```
2080   LD     SP,#100
2084   LDB    AL,IN-V
2087   SHRB   AL,#3
208A   ANDB   AL,#FE
208D   LDBZE  AL,AL
2090   LD     T-LOW,2100[AX]
2095   LD     T-HI,2102[AX]
209A   SUB    T-DIF,T-HI,T-LO
209E   ANDB   IN-D,IN-V,#F
20A2   LDBZE  IN-D,IN-D
20A6   MUL    OUT-D,IN-D,T-DI
20AA   SHRAL  OUT-D,#4
20AD   ADD    OUT,OUT-D,T-LOW
20B1   SHRA   OUT,#4
20B4   ADDC   OUT,SP
20B7   ST     OUT,RES
```

Cpu, Memoria, Banderas, Tabla
>Reset Archivo Memoria Fin Setear Ejecutar Tabla Dosc Imprimir

SIMULADOR DEL MICROCONTROLADOR 8096

ARCHIVO SIMULADO: EJEM-5.HEX

```
 2080    LD     SP,#100
 2084    LDB    AL,IN-V
 2087    SHRB   AL,#3
 208A    ANDB   AL,#FE
 208D    LDBZE  AL,AL
 2090    LD     T-LOW,2100[AX]
 2095    LD     T-HI,2102[AX]
 209A    SUB    T-DIF,T-HI,T-LOW
 209E    ANDB   IN-D,IN-V,#F
 20A2    LDBZE  IN-D,IN-D
 20A6    MUL    OUT-D,IN-D,T-DIF
 20AA    SHRAL  OUT-D,#4
 20AD    ADD    OUT,OUT-D,T-LOW
 20B1    SHRA   OUT,#4
 20B4    ADDC   OUT,SP
 20B7    ST     OUT,RES
 20BA    SJMP   7C8
```

REGISTROS ESPECIALES DE MEMORIA

```
0000   00 00 00 00 00 00 00 00 00 00 16 00 00 00 00 00 ................
0010   00 00 00 00 00 00 00 00 00 01 00 00 04 00 00 00 ................
```

CONTENIDO DE LOCALIDADES DE MEMORIA

```
0000   00 00 00 00 00 00 00 00 00 00 16 00 00 00 00 00 ................
0010   00 00 00 00 00 00 00 00 00 01 00 00 04 00 00 00 ................
0020   00 00 20 00 00 34 00 4C 00 00 00 18 40 03 40 03 .. ..4.L....@.@.
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

ESTADOS DE TIEMPO: 0

5.1.2. RAIZ CUADRADA

Este programa permite encontrar la raiz cuadrada exacta de un número en base a restas sucesivas.

El algoritmo verifica primero si el número es par o impar de esta manera se resta números pares o impares respectivamente. Se inicia las restas sucesivas con el 2 o el 3 según corresponda, y se verifica si el resultado es cero o negativo , si es negativo se incrementa el número a restarse en dos reiniciando el mismo proceso.

Si es cero se verifica si efectivamente es la raiz , si no lo es se incrementa el número a restarse en dos continuando con el proceso.

PROGRAMA FUENTE

```
; RAIZ CUADRADA EXACTA  MEDIANTE RESTAS SUCESIVAS

        CPU     "8096.TBL"      ; CPU TABLE
        HOF     "INT16"         ; HEX FORMAT

; MCS-96 INTERNAL REGISTERS
```

```
                        ; Code Segment
002080                          ORG     2080H
002080                  start:
002080 A1000118                 LD      SP,#100H
002084 A102001E                 LD      BX,#2
002088 89040026                 CMP     EX,#4
00208C DF20                     JE      RES1
00208E A103001E                 LD      BX,#3
002092 302602                   JBC     EL,0,par
002095 200A                     SJMP    IMPAR
002097                  par:
002097 A104001E                 LD      BX,#4
00209B 2004                     SJMP    IMPAR
00209D                  incr:
00209D 071E                     INC     BX
00209F 071E                     INC     BX
0020A1                  IMPAR:
0020A1 5C1E1E1C                 MULUB   AL,BL,BL
0020A5 881C26                   CMP     EX,AX
0020A8 DF04                     JE      RES1
0020AA DE02                     JLT     RES1
0020AC 27EF                     SJMP    INCR
0020AE                  res1:
000000                          END
```

8096/SIM

REGISTROS DE MEMORIA

```
2080 A1 00 01 18 A1 02 00 1E
2088 89 04 00 26 DF 20 A1 03
2090 00 1E 30 26 02 20 0A A1
2098 04 00 1E 20 04 07 1E 07
20A0 1E 5C 1E 1E 1C 88 1C 26
20A8 DF 04 DE 02 27 EF 00 00
20B0 00 00 00 00 00 00 00 00
20B8 00 00 00 00 00 00 00 00
20C0 00 00 00 00 00 00 00 00
20C8 00 00 00 00 00 00 00 00
20D0 00 00 00 00 00 00 00 00
20D8 00 00 00 00 00 00 00 00
20E0 00 00 00 00 00 00 00 00
20E8 00 00 00 00 00 00 00 00
20F0 00 00 00 00 00 00 00 00
20F8 00 00 00 00 00 00 00 00
```

```
PSW
I0 = 0
I1 = 0
I2 = 0
I3 = 0
I4 = 0
I5 = 0
I6 = 0
I7 = 0
ST = 0
 I = 0
 - = 0
 C = 0
VT = 0
 V = 0
 N = 0
 Z = 0
```

EJECUCION RAI.HEX

```
2080   LD     SP,#100
2084   LD     BX,#2
2088   CMP    EX,#4
208C   JE     20
208E   LD     BX,#3
2092   JBC    EL,0,2
2095   SJMP   A
2097   LD     BX,#4
209B   SJMP   4
209D   INC    BX
209F   INC    BX
20A1   MULUB  AL,BL,BL
20A5   CMP    EX,AX
20A8   JE     4
20AA   JLT    2
20AC   SJMP   7EF
```

Ejecutando el programa
>Pasos Continuo

8096/SIM

REGISTROS DE MEMORIA

```
0000 00 00 00 00 00 00 00 00
0008 00 00 1D 00 00 00 00 FF
0010 C1 00 00 00 00 00 00 00
0018 00 01 00 00 09 00 03 00
0020 00 00 00 00 00 00 09 00
0028 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00
0058 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00
0068 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00
```

```
PSW
I0 = 0
I1 = 0
I2 = 0
I3 = 0
I4 = 0
I5 = 0
I6 = 0
I7 = 0
ST = 0
 I = 0
 - = 0
 C = 0
VT = 0
 V = 0
 N = 0
 Z = 1
```

EJECUCION RAI.HEX

```
2080   LD     SP,#100
2084   LD     BX,#2
2088   CMP    EX,#4
208C   JE     20
208E   LD     BX,#3
2092   JBC    EL,0,2
2095   SJMP   A
2097   LD     BX,#4
209B   SJMP   4
209D   INC    BX
209F   INC    BX
20A1   MULUB  AL,BL,BL
20A5   CMP    EX,AX
20A8   JE     4
20AA   JLT    2
20AC   SJMP   7EF
```

Cpu, Memoria, Banderas, Tabla
>Reset Archivo Memoria Fin Setear Ejecutar Tabla Dosc Imprimir

```
                    SIMULADOR DEL MICROCONTROLADOR 8096
                    ===================================
ARCHIVO SIMULADO: RAI.HEX

 2080    LD     SP,#100
 2084    LD     BX,#2
 2088    CMP    EX,#4
 208C    JE     20
 208E    LD     BX,#3
 2092    JBC    EL,0,2
 2095    SJMP   A
 2097    LD     BX,#4
 2098    SJMP   4
 209D    INC    BX
 209F    INC    BX
 20A1    MULUB  AL,BL,BL
 20A5    CMP    EX,AX
 20A8    JE     4
 20AA    JLT    2
 20AC    SJMP   7EF


                    REGISTROS ESPECIALES DE MEMORIA
                    ===============================


0000    00 00 00 00 00 00 00 00 00 00 1E 00 00 00 00 FF ................
0010    C1 00 00 00 00 00 00 00 00 01 00 00 09 00 03 00 ⊥...............

                    CONTENIDO DE LOCALIDADES DE MEMORIA
                    ===================================
0000    00 00 00 00 00 00 00 00 00 00 1E 00 00 00 00 FF ................
0010    C1 00 00 00 00 00 00 00 00 01 00 00 09 00 03 00 ⊥...............
0020    00 00 00 00 00 00 09 00 00 00 00 00 00 00 00 00 ................
0030    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00D0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00E0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00F0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

ESTADOS DE TIEMPO:  0
```

```
0000001E    =       BL:             EQU     BX
0000001F    =       BU:             EQU     (BX+1)
00000018    =       SP:             EQU     18H
00000020    =       CL:             EQU     CX
00000021    =       CH:             EQU     (CX+1)

                        ; Code Segment

002080                          ORG     2080H
002080                  start:
002080 A1000118                 LD      SP,#100H
002084 4C201C24                 MULU    RES,AX,CX
002088 4C201E2C                 MULU    HX,BX,CX
00208C 4C221C30                 MULU    FX,AX,DX
002090 4C221E28                 MULU    RES2,BX,DX
002094 642C26                   ADD     RES1,HX
002097 DB11                     JC      SUM1
002099                  CAR1:
002099 643026                   ADD     RES1,FX
00209C A42E28                   ADDC    RES2,EX
00209F DB11                     JC      SUM2
0020A1                  CAR2:
0020A1 643228                   ADD     RES2,GX
0020A4 A500002A                 ADDC    RES3,#0
```

```
0020A8 2014                     SJMP  FIN
0020AA                  SUM1:
0020AA A500002B                 ADDC  RES2,#0
0020AE DB08                     JC    SUM3
0020B0 27E7                     SJMP  CAR1
0020B2                  SUM2:
0020B2 A500002A                 ADDC  RES3,#0
0020B6 27E9                     SJMP  CAR2
0020B8                  SUM3:
0020B8 A500002A                 ADDC RES3,#0
0020BC 27DB                     SJMP CAR1
0020BE                  FIN:
0000001D  AH                    0000001C  AL
0000001C  AX
0000001E  BL                    0000001F  BU
0000001E  BX
00002099  CAR1                  000020A1  CAR2
00000021  CH
00000020  CL                    00000020  CX
00000022  DX
0000002E  EX                    000020BE  FIN
00000030  FX
00000032  GX                    0000002C  HX
00000034  IX
```

8096/SIM

REGISTROS DE MEMORIA

```
2080 A1 00 01 18 4C 20 1C 24
2088 4C 20 1E 2C 4C 22 1C 30
2090 4C 22 1E 28 64 2C 26 DB
2098 11 64 30 26 A4 2E 28 DB
20A0 11 64 32 28 A5 00 00 2A
20A8 20 14 A5 00 00 28 DB 08
20B0 27 E7 A5 00 00 2A 27 E9
20B8 A5 00 00 2A 27 DB 00 00
20C0 00 00 00 00 00 00 00 00
20C8 00 00 00 00 00 00 00 00
20D0 00 00 00 00 00 00 00 00
20D8 00 00 00 00 00 00 00 00
20E0 00 00 00 00 00 00 00 00
20E8 00 00 00 00 00 00 00 00
20F0 00 00 00 00 00 00 00 00
20F8 00 00 00 00 00 00 00 00
```

```
PSW
I0 =  0
I1 =  0
I2 =  0
I3 =  0
I4 =  0
I5 =  0
I6 =  0
I7 =  0
ST =  0
 I =  0
 - =  0
 C =  0
VT =  0
 V =  0
 N =  0
 Z =  1
```

EJECUCION MUL.HEX

```
2080   LD     SP,#100
2084   MULU   RES,AX,CX
2088   MULU   HX,BX,CX
208C   MULU   FX,AX,DX
2090   MULU   RES2,BX,DX
2094   ADD    RES1,HX
2097   JC     11
2099   ADD    RES1,FX
209C   ADDC   RES2,EX
209F   JC     11
20A1   ADD    RES2,GX
20A4   ADDC   RES3,#0
20A8   SJMP   14
20AA   ADDC   RES2,#0
20AE   JC     8
20B0   SJMP   7E7
```

Ejecutando el programa
>Pasos Continuo

8096/SIM

REGISTROS DE MEMORIA

```
0000 00 00 00 00 00 00 00 00
0008 00 00 2D 00 00 00 00 00
0010 00 00 00 00 00 00 00 00
0018 00 01 00 00 11 11 22 34
0020 67 78 89 45 D7 D5 CD 28
0028 59 2D 29 0E AE E9 84 18
0030 19 B7 A2 04 00 00 00 00
0038 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00
0058 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00
0068 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00
```

```
PSW
I0 =  0
I1 =  0
I2 =  0
I3 =  0
I4 =  0
I5 =  0
I6 =  0
I7 =  0
ST =  0
 I =  0
 - =  0
 C =  0
VT =  1
 V =  0
 N =  0
 Z =  0
```

EJECUCION MUL.HEX

```
2080   LD     SP,#100
2084   MULU   RES,AX,CX
2088   MULU   HX,BX,CX
208C   MULU   FX,AX,DX
2090   MULU   RES2,BX,DX
2094   ADD    RES1,HX
2097   JC     11
2099   ADD    RES1,FX
209C   ADDC   RES2,EX
209F   JC     11
20A1   ADD    RES2,GX
20A4   ADDC   RES3,#0
20A8   SJMP   14
20AA   ADDC   RES2,#0
20AE   JC     8
20B0   SJMP   7E7
```

Ver el contenido de memoria
>Contenido Modificar Encontra Borrar(B) Transferir(B) Llenar(B)

SIMULADOR DEL MICROCONTROLADOR 8096

ARCHIVO SIMULADO: MUL.HEX

```
2080    LD      SP,#100
2084    MULU    RES,AX,CX
2088    MULU    HX,BX,CX
208C    MULU    FX,AX,DX
2090    MULU    RES2,BX,DX
2094    ADD     RES1,HX
2097    JC      11
2099    ADD     RES1,FX
209C    ADDC    RES2,EX
209F    JC      11
20A1    ADD     RES2,GX
20A4    ADDC    RES3,#0
20A8    SJMP    14
20AA    ADDC    RES2,#0
20AE    JC      8
20B0    SJMP    7E7
20B2    ADDC    RES3,#0
20B6    SJMP    7E9
20B8    ADDC    RES3,#0
20BC    SJMP    7DB
```

REGISTROS ESPECIALES DE MEMORIA

```
0000    00 00 00 00 00 00 00 00 00 00 2D 00 00 00 00 00 ..........-.....
0010    00 00 00 00 00 00 00 00 00 01 00 00 11 11 22 34 ..............."4
```

CONTENIDO DE LOCALIDADES DE MEMORIA

```
0000    00 00 00 00 00 00 00 00 00 00 2D 00 00 00 00 00 ..........-.....
0010    00 00 00 00 00 00 00 00 00 01 00 00 11 11 22 34 ..............."4
0020    67 7B 89 45 D7 D5 CD 28 59 2D 29 0E AE E9 84 18 g{ëEΘ╒═(Y-).«Θ.
0030    19 B7 A2 04 00 00 00 00 00 00 00 00 00 00 00 00 .╖ó.............
0040    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00D0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00E0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00F0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

ESTADOS DE TIEMPO: 0

5.1.4. CONVERSOR ANALOGO DIGITAL

Realiza una conversión de análogo a digital usando del canal 0 hasta el 3 almacenando los resultados en palabras de la RAM.

Este ejemplo ha sido tomado del manual de la Intel del microcontrolador 8096 página 5-31.

PROGRAMA FUENTE

```
        CPU     "8096.TBL"      ; CPU TABLE
         HOF     "INT16"         ; HEX FORMAT

; MCS-96 INTERNAL REGISTERS

; CONVERSION DE A/D

; Este programa convierte el A a D en los canales de 0 al 3
; el resultado esta en RES_N

  AD_RES_LO:        equ    02
  AD_RES_HI:        equ    03
  AD_COMMAND:       equ    02
  SP:               equ    18H
```

```
                    ; Code Segment

000000                          ORG     0000H
000000 A1C00018       start:    LD      SP, #0C0H
000004 A00002                   LD      BX, 00H
000007 910802         next:     ORB     BL, #1000B
00000A B00202                   LDB     AD_COMMAND,BL
00000D 710702                   ANDB    BL, #0111B
000010 FD                       NOP
000011 3B02FD         check:    JBS     AD_RES_LO, 3, check
000014 B00200                   LDB     AL, AD_RES_LO
000017 B00301                   LDB     AH, AD_RES_HI
00001A 54020204                 ADDB    DL, BL, BL
00001E AC0404                   LDBZE   DX, DL
000021 C3040000                 ST      AX, RES_TABLE[DX]
000025 1702                     INCB    BL
000027 710302                   ANDB    BL, #03H
00002A E307                     BR      [next]
000000                          END


00000002  AD_COMMAND            00000003  AD_RES_HI
00000002  AD_RES_LO
00000001  AH                    00000000  AL
00000000  AX
00000002  BL                    00000003  BH
```

8096/SIM

REGISTROS DE MEMORIA

```
2080 A1 00 01 1B 01 20 55 08
2088 20 02 FD FD 3B 02 FD B0
2090 02 1C B0 03 1D 54 20 20
2098 1E AC 1E 1E C3 1E 2B 1C
20A0 17 20 71 03 20 27 DF 00
20A8 00 00 00 00 00 00 00 00
20B0 00 00 00 00 00 00 00 00
20B8 00 00 00 00 00 00 00 00
20C0 00 00 00 00 00 00 00 00
20C8 00 00 00 00 00 00 00 00
20D0 00 00 00 00 00 00 00 00
20D8 00 00 00 00 00 00 00 00
20E0 00 00 00 00 00 00 00 00
20E8 00 00 00 00 00 00 00 00
20F0 00 00 00 00 00 00 00 00
20F8 00 00 00 00 00 00 00 00
```

```
PSW
I0 =  0
I1 =  0
I2 =  0
I3 =  0
I4 =  0
I5 =  0
I6 =  0
I7 =  0
ST =  0
 I =  0
 - =  0
 C =  0
VT =  0
 V =  0
 N =  0
 Z =  0
```

EJECUCION EJEM-4.HEX

```
2080   LD     SP,#100
2084   CLR    BX
2086   ADDB   AD-COML,BL,#8
208A   NOP
208B   NOP
208C   JBS    BL,3,FD
208F   LDB    AL,AD-COML
2092   LDB    AH,AD-COMH
2095   ADDB   DL,BL,BL
2099   LDBZE  DL,DL
209C   ST     AX,28[DX]
20A0   INCB   BL
20A2   ANDB   BL,#3
20A5   SJMP   7DF
```

Ejecutando el programa
>Pasos Continuo

8096/SIM

REGISTROS DE MEMORIA

```
0000 00 00 08 00 00 00 00 00
0008 00 00 79 00 00 00 00 00
0010 00 00 00 00 00 00 00 00
0018 00 01 00 00 42 68 04 00
0020 03 00 00 00 00 00 00 00
0028 C0 FF B1 D6 42 68 00 00
0030 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00
0058 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00
0068 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00
CH : 3   RES :  158
```

```
PSW
I0 =  0
I1 =  0
I2 =  0
I3 =  0
I4 =  0
I5 =  0
I6 =  0
I7 =  0
ST =  0
 I =  0
 - =  0
 C =  0
VT =  0
 V =  0
 N =  0
 Z =  0
```

EJECUCION EJEM-4.HEX

```
2080   LD     SP,#100
2084   CLR    BX
2086   ADDB   AD-COML,BL,#8
208A   NOP
208B   NOP
208C   JBS    BL,3,FD
208F   LDB    AL,AD-COML
2092   LDB    AH,AD-COMH
2095   ADDB   DL,BL,BL
2099   LDBZE  DL,DL
209C   ST     AX,28[DX]
20A0   INCB   BL
20A2   ANDB   BL,#3
20A5   SJMP   7DF
```

Cpu, Memoria, Banderas, Tabla
>Reset Archivo Memoria Fin Setear Ejecutar Tabla Dosc Imprimir

```
                    SIMULADOR DEL MICROCONTROLADOR 8096
                    ===================================
ARCHIVO SIMULADO: EJEM-4.HEX

 2080    LD     SP,#100
 2084    CLR    BX
 2086    ADDB   AD-COML,BL,#8
 208A    NOP
 208B    NOP
 208C    JBS    BL,3,FD
 208F    LDB    AL,AD-COML
 2092    LDB    AH,AD-COMH
 2095    ADDB   DL,BL,BL
 2099    LDBZE  DL,DL
 209C    ST     AX,28[DX]
 20A0    INCB   BL
 20A2    ANDB   BL,#3
 20A5    SJMP   7DF


                    REGISTROS ESPECIALES DE MEMORIA
                    ===============================

0000    00 00 B3 27 00 00 00 00 00 00 82 00 00 00 00 00 ..'............
0010    00 00 00 00 00 00 00 00 00 01 00 00 42 68 04 00 ............Bh..

                    CONTENIDO DE LOCALIDADES DE MEMORIA
                    ===================================
0000    00 00 83 27 00 00 00 00 00 00 82 00 00 00 00 00 ..'............
0010    00 00 00 00 00 00 00 00 00 01 00 00 42 68 04 00 ............Bh..
0020    03 00 00 00 00 00 00 00 C0 FF 81 D6 42 68 00 00 ........└ ╥Bh..
0030    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00D0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00E0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00F0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

ESTADOS DE TIEMPO:  0
```

5.1.5. PORTICO SERIAL

Este ejemplo muestra una simple rutina en la cual recibe un caracter y transmite el mismo caracter.

Este ejemplo ha sido tomado del manual de la Intel del microcontrolador 8096 página 5-29, 5-30.

PROGRAMA FUENTE

```
; PROGRAMA  DEL PORTICO SERIAL
        CPU     "8096.TBL"      ; CPU TABLE
        HOF     "INT16"         ; HEX FORMAT
; MCS-96 INTERNAL REGISTERS
R0:             EQU     00H     ;ZERO REGISTER
AD_RESULT:      EQU     02H     ;A/D RESULT
AD_COMMAND:     EQU     02H     ;A/D COMMAND REGISTER
HSI_MODE:       EQU     03H     ;HIGH SPEED INPUT MODE
HSI_TIME:       EQU     04H     ;HIGH SPEED TRIGGER TIME
HSO_COMMAND:    EQU     06H     ;HSO COMMAND REGISTER
HSI_STATUS:     EQU     06H     ;HSI STATUS REGISTERS
SBUF:           EQU     07H     ;SERIAL BUFFER
INT_MASK:       EQU     08H     ;INTERRUPT MASK
INT_PENDING:    EQU     09H     ;INTERRUPT PENDING REGISTER
```

```
WATCHDOG:     EQU     0AH       ;WATCHDOG TIMER REGISTER
TIMER1:       EQU     0AH       ;TIMER 1
TIMER2:       EQU     0CH       ;TIMER2
IOPORT0:      EQU     0EH       ;PORT 0 REGISTER
BAUD_RATE:    EQU     0EH       ;BAUD RATE REGISTER
IOPORT1:      EQU     0FH       ;PORT 1 REGISTER
IOPORT2:      EQU     10H       ;PORT 2 REGISTER
SP_STAT:      EQU     11H       ;SERIAL PORT STATUS
SP_CON:       EQU     11H       ;SERIAL PORT CONTROL
IOS0:         EQU     15H       ;I/O STATUS REGISTER 0
IOS1:         EQU     16H       ;I/O STATUS REGISTER 1
IOC0:         EQU     15H       ;I/O CONTROL REGISTER 0
IOC1:         EQU     16H       ;I/O CONTROL REGISTER 1
PWM_CONTROL:  EQU     17H       ;PWM CONTROL REGISTER

; Este programa inicializa el portico serial y envia cualquier
; caracter que en el se encuentra

  BAUD_REG:       equ   0EH
  SPCON:          equ   11H
  SPSTAT:         equ   11H
  IOC1:           equ   16H
  IOC0:           equ   15H
  SBUF:           equ   07H
```

```
  INT_PENDING:     equ    09H
  SP:              equ    18H

; Register Segment
                   ORG      28H
       CHR:        dfS      1
       SPTEMP:     dfs      1
       TEMP0:      dfs      1
       TEMP1:      dfs      1
       RCV_FLAG:   dfs      1
                   ORG      200CH
                   DWL      SER_PORT_INT
; Code Segment
        ORG     2080H
start:  LD      SP, #100H          ; Setear Stack Pointer
        LDB     IOC1,#00100000B    ; Setear P2.0 a TXD
;Relacion de baudios=fin./64*valor en baudios
; Valor en baudios= (fin/64)/relacion de baudios
baud_val:       equ     39      ; 2400 baudios a 6 MHz
BAUD_HIGH:      equ    ((baud_val-1)/256) OR 80H  ;setear el MSB
BAUD_LOW:       equ     (baud_val-1) MOD 256
        LDB    BAUD_REG, #BAUD_LOW
        LDB    BAUD_REG, #BAUD_HIGH
        LDB    SPCON, #01001001B         ;habilitado el modo 1
```

```
                                    ;el portico  serial esta
                                   ;inicializado
        STB   SBUF, CHR             ;borrar el portico serial
        LDB   TEMPO, #00100000B     ;setear el TI-temp
        LDB   INT_MASK,#01000000B   ;habilitar interrupcion
        EI
LOOP:
        BR    [LOOP]                ;espera por interrupcion
SER_PORT_INT:
        PUSHF

rd_again:
        LDB   SPTEMP,SPSTAT
        ORB   TEMPO,SPTEMP
        ANDB  SPTEMP,#01100000B
        JNE   rd_again              ;repite hasta que TI y RI
                                    ;sean borradas
get_byte:
        JBC    TEMPO, 6, put_byte    ;si RI-temp no esta
                                   ;seteado
        STB    SBUF, CHR            ;almacenar el byte
        ANDB   TEMPO, #10111111B    ;borrar el RI-temp
        LDB    RCV_FLAG, #0FFH      ;setear  la bandera  de
                                   ;bit de recibo
```

PROGRAMA ENSAMBLADO

```
00000000    =    R0:              EQU    00H     ;ZERO
00000002    =    AD_RESULT:       EQU    02H     ;A/D
00000002    =    AD_COMMAND:      EQU    02H     ;A/D
00000003    =    HSI_MODE:        EQU    03H     ;HIGH
00000004    =    HSI_TIME:        EQU    04H     ;HIGH
00000006    =    HSO_COMMAND:     EQU    06H     ;HSO
00000006    =    HSI_STATUS:      EQU    06H     ;HSI
00000007    =    SBUF:            EQU    07H     ;SERIAL
00000008    =    INT_MASK:        EQU    08H     ;INTERRUPT
00000009    =    INT_PENDING:     EQU    09H     ;INTERRUPT
0000000A    =    WATCHDOG:        EQU    0AH     ;WATCHDOG
0000000A    =    TIMER1:          EQU    0AH     ;TIMER 1
0000000C    =    TIMER2:          EQU    0CH     ;TIMER2
0000000E    =    IOPORT0:         EQU    0EH     ;PORT 0
0000000E    =    BAUD_RATE:       EQU    0EH     ;BAUD RATE
0000000F    =    IOPORT1:         EQU    0FH     ;PORT 1
00000010    =    IOPORT2:         EQU    10H     ;PORT 2
00000011    =    SP_STAT:         EQU    11H     ;SERIAL
00000011    =    SP_CON:          EQU    11H     ;SERIAL
00000015    =    IOS0:            EQU    15H     ;I/O
00000016    =    IOS1:            EQU    16H     ;I/O
00000015    =    IOC0:            EQU    15H     ;I/O
00000016    =    IOC1:            EQU    16H     ;I/O
```

```
00000017     =          PWM_CONTROL:  EQU     17H       ;PWM
0000000E     =            BAUD_REG:       equ   0EH
00000011     =            SPCON:          equ   11H
00000011     =            SPSTAT:         equ   11H
00000016     =            IOC1:           equ   16H
00000015     =            IOC0:           equ   15H
00000007     =            SBUF:           equ   07H
00000009     =            INT_PENDING:    equ   09H
0000001B     =            SP:             equ   18H

000028                                     ORG       28H
000028                       CHR:          dfS       1
000029                       SPTEMP:       dfs       1
00002A                       TEMP0:        dfs       1
00002B                       TEMP1:        dfs       1
00002C                       RCV_FLAG:     dfs       1
00200C                                     ORG       200CH
00200C 9A20                                DWL          002080
                     ORG     2080H
002080 A1000118      start:  LD    SP, #100H
002084 B12016                LDB   IOC1,#00100000B
00000027     =         baud_val:       equ     39       ; 2400
00000000     =         BAUD_HIGH:      equ    ((baud_val-1)/256)
00000000     =         BAUD_LOW:       equ     (baud_val-1) MOD
```

```
002087 B1000E                   LDB    BAUD_REG, #BAUD_LOW
00208A B1000E                   LDB    BAUD_REG, #BAUD_HIGH
00208D B14911                   LDB    SPCON, #01001001B
002090 C42807                   STB    SBUF, CHR
002093 B1202A                   LDB    TEMP0, #00100000B
002096 B14008                   LDB    INT_MASK,#01000000B
002099 FB                       EI
00209A                  LOOP:   BR     [LOOP]
00209A                  SER_PORT_INT:
00209A F2                       PUSHF
00209B                  rd_again:
00209B B01129                   LDB    SPTEMP,SPSTAT
00209E 90292A                   ORB    TEMP0,SPTEMP
0020A1 716029                   ANDB   SPTEMP,#01100000B
0020A4 D7F5                     JNE    rd_again
0020A6                  get_byte:
0020A6 362A09                   JBC    TEMP0, 6, put_byte
0020A9 C42B07                   STB    SBUF, CHR
0020AC 71BF2A                   ANDB   TEMP0, #10111111B
0020AF B1FF2C                   LDB    RCV_FLAG, #0FFH
0020B2                  put_byte:
0020B2 302C19                   JBC    RCV_FLAG, 0, continue
0020B5 352A16                   JBC    TEMP0, 5, continue
0020B8 B02807                   LDB    SBUF, CHR
```

```
00000002  AD_COMMAND          00000002  AD_RESULT
00000000  BAUD_HIGH
00000000  BAUD_LOW            0000000E  BAUD_RATE
0000000E  BAUD_REG
00000027  BAUD_VAL            00000028  CHR
000020CC  CLR_RCV
000020CE  CONTINUE            000020A6  GET_BYTE
00000003  HSI_MODE
00000006  HSI_STATUS          00000004  HSI_TIME
00000006  HSO_COMMAND
00000008  INT_MASK            00000009  INT_PENDING
00000015  IOC0
00000016  IOC1                0000000E  IOPORT0
0000000F  IOPORT1
00000010  IOPORT2             00000015  IOS0
00000016  IOS1
0000209A  LOOP                000020B2  PUT_BYTE
00000017  PWM_CONTROL
00000000  R0                  0000002C  RCV_FLAG
0000209B  RD_AGAIN
00000007  SBUF                0000209A  SER_PORT_INT
00000018  SP
00000011  SPCON               00000011  SPSTAT
00000029  SPTEMP
```

8096/SIM

REGISTROS DE MEMORIA

```
2080 A1 00 01 18 B1 20 16 B1
2088 26 0E B1 80 0E B1 49 11
2090 C4 28 07 B1 20 2A B1 40
2098 08 FB 27 FE F2 B0 11 29
20A0 90 29 2A 71 60 29 D7 F5
20A8 36 2A 09 C4 28 07 71 BF
20B0 2A B1 FF 2C 30 2C 18 35
20B8 2A 15 B0 28 07 71 DF 2A
20C0 71 7F 28 99 0D 28 D7 05
20C8 B1 0A 28 20 02 11 2C F3
20D0 F0 00 00 00 00 00 00 00
20D8 00 00 00 00 00 00 00 00
20E0 00 00 00 00 00 00 00 00
20E8 00 00 00 00 00 00 00 00
20F0 00 00 00 00 00 00 00 00
20F8 00 00 00 00 00 00 00 00
```

```
PSW
I0 = 0
I1 = 0
I2 = 0
I3 = 0
I4 = 0
I5 = 0
I6 = 0
I7 = 0
ST = 0
 I = 0
 - = 0
 C = 0
VT = 0
 V = 0
 N = 0
 Z = 0
```

EJECUCION EJEM-1.HEX

```
2080   LD     SP,#100
2084   LDB    IOC1L,#20
2087   LDB    BA-REGL,#26
208A   LDB    BA-REGL,#80
208D   LDB    SPCOML,#49
2090   STB    SBUFL,BA-VH
2093   LDB    TEMPOL,#20
2096   LDB    INK-MASKL,#40
2099   EI
209A   SJMP   7FE
209C   PUSHF
209D   LDB    SPTEMPL,SPCOML
20A0   ORB    TEMPOL,SPTEMPL
20A3   ANDB   SPTEMPL,#60
20A6   JNE    F5
20A8   JBC    TEMPOL,6,9
```

Ejecutando el programa
>Pasos Continuo

8096/SIM

REGISTROS DE MEMORIA

```
0000 00 00 00 00 00 00 00 00
0008 40 00 0A 00 00 00 26 00
0010 00 49 00 00 00 00 20 00
0018 00 01 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00
0028 00 00 20 00 00 00 00 00
0030 00 00 00 00 00 00 00 00
0038 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00
0048 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00
0058 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00
0068 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00
0078 00 00 00 00 00 00 00 00
ESTADOS: 832
```

```
PSW
I0 = 0
I1 = 0
I2 = 0
I3 = 0
I4 = 0
I5 = 0
I6 = 0
I7 = 0
ST = 0
 I = 1
 - = 0
 C = 0
VT = 0
 V = 0
 N = 0
 Z = 0
```

EJECUCION EJEM-1.HEX

```
2080   LD     SP,#100
2084   LDB    IOC1L,#20
2087   LDB    BA-REGL,#26
208A   LDB    BA-REGL,#80
208D   LDB    SPCOML,#49
2090   STB    SBUFL,BA-VH
2093   LDB    TEMPOL,#20
2096   LDB    INK-MASKL,#40
2099   EI
209A   SJMP   7FE
209C   PUSHF
209D   LDB    SPTEMPL,SPCOML
20A0   ORB    TEMPOL,SPTEMPL
20A3   ANDB   SPTEMPL,#60
20A6   JNE    F5
20A8   JBC    TEMPOL,6,9
```

Ejecutando el programa
>Pasos Continuo

SIMULADOR DEL MICROCONTROLADOR 8096

ARCHIVO SIMULADO: EJEM-1.HEX

```
 2080   LD     SP,#100
 2084   LDB    IOC1L,#20
 2087   LDB    BA-REGL,#26
 208A   LDB    BA-REGL,#80
 208D   LDB    SPCOML,#49
 2090   STB    SBUFL,BA-VH
 2093   LDB    TEMPOL,#20
 2096   LDB    INK-MASKL,#40
 2099   EI
 209A   SJMP   7FE
 209C   PUSHF
 209D   LDB    SPTEMPL,SPCOML
 20A0   ORB    TEMPOL,SPTEMPL
 20A3   ANDB   SPTEMPL,#60
 20A6   JNE    F5
 20A8   JBC    TEMPOL,6,9
 20AB   STB    SBUFL,BA-VH
 20AE   ANDB   TEMPOL,#BF
 20B1   LDB    RC-FLAGL,#FF
 20B4   JBC    RC-FLAGL,0,1B
 20B7   JBC    TEMPOL,5,15
 20BA   LDB    SBUFL,BA-VH
 20BD   ANDB   TEMPOL,#DF
 20C0   ANDB   BA-VH,#7F
 20C3   CMPB   BA-VH,#D
 20C6   JNE    5
 20C8   LDB    BA-VH,#A
 20CB   SJMP   2
 20CD   CLRB   RC-FLAGL
 20CF   POPF
 20D0   RET
```

REGISTROS ESPECIALES DE MEMORIA

```
0000   00 00 00 00 00 00 00 00 40 00 07 00 00 00 26 00 ........@.....&.
0010   00 49 00 00 00 00 20 00 00 01 00 00 00 00 00 00 .I.... ........
```

CONTENIDO DE LOCALIDADES DE MEMORIA

```
0000   00 00 00 00 00 00 00 00 40 00 07 00 00 00 26 00 ........@.....&.
0010   00 49 00 00 00 00 20 00 00 01 00 00 00 00 00 00 .I.... ........
0020   00 00 00 00 00 00 00 00 00 00 20 00 00 00 00 00 .......... .....
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

```
00B0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00D0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00E0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00F0    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

ESTADOS DE TIEMPO:  28
```

5.2 LIMITACIONES

Debido a las características del computador y del lenguaje con los cuales se simuló el microcontrolador 8096 este presenta las siguientes limitaciones.

- La ejecución de un programa no se puede simular a tiempo real, es decir en el mismo tiempo en el que ejecutaría el microcontrolador, por lo cual se utiliza los estados de tiempo que dan una referencia al usuario del tiempo que tomaría la ejecución de dicho programa.

- No se simulan entradas de alta velocidad HSI ni las salidas de alta velocidad HSO.

- El conversor A/D del 8096 realiza una conversión en forma paralela es decir que el programa sigue ejecutándose mientras se produce la conversión. Esto no puede reproducirse en el simulador. Además la entrada análoga debe ser simulada en un archivo.

- El conversor digital análogo tampoco es simulado.

- El pórtico serial del 8096 es full duplex es decir puede transmitir y recibir al mismo tiempo lo cual no puede ser

simulado.Los datos recibidos son ingresados a través de un archivo.

- Los pórticos de entrada y salida no son simulados pues estos son utilizados para ingresar señales al microcontrolador.

- Las señales eléctricas externas en general no se han simulado, tales como las frecuencias de entrada de los osciladores.

- Las interrupciones no son simuladas puesto que para ello sería necesario disponer de las fuentes de interrupción.

- El tiempo en cual el simulador ejecuta un programa no es el mismo que el que le tomaría al microcontrolador 8096 ejecutarlo.

5.3. RECOMENDACIONES

El simulador del microcontrolador 8096 fue realizado con el propósito de fácilitar al usuario la comprobación de un programa sin necesidad de implementar el hardward correspondiente, es decir se aisla los problemas físicos ocacionados del algoritmo utilizado.

Este simulador permite además depurar un programa sin necesidad de cambiar el archivo fuente, es decir, modificando el contenido de las memorias en las que se encuentra el programa, con lo cual una vez logrado el resultado deseado se puede modificar el archivo fuente e implementarlo.

El programa puede ser ejecutado paso a paso chequeándose así el resultado de cada instrucción.

Permite implementar puntos de parada con el fin de chequear instrucciones determinadas.

Debe utilizarse el manual del usuario que le permitirá aprovechar todas la facilidades del simulador.

.3 Packaging

MCS-96 products are available in 48-pin and 68-pin packages, with and without A/D, and with and without on-ROM or EPROM. The MCS-96 numbering system is shown below. Section 14.4 shows the pinouts for the 48- 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin versions come in a stic Leaded Chip Carrier (PLCC), a Pin Grid Array (PGA) or a Type "B" Leadless Chip Carrier.

The MCS®-96 Family Nomenclature

		Without A/D	With A/D
ROMless 809XBH	48 Pin		C8095BH - Ceramic DIP P8095BH - Plastic DIP
	68 Pin	A8096BH - Ceramic PGA N8096BH - PLCC	A8097BH - Ceramic PGA N8097BH - PLCC
ROM 839XBH	48 Pin		C8395BH - Ceramic DIP P8395BH - Plastic DIP
	68 Pin	A8396BH - Ceramic PGA N8396BH - PLCC	A8397BH - Ceramic PGA N8397BH - PLCC
EPROM 879XBH	48 Pin		C8795BH - Ceramic DIP
	68 Pin	A8796BH - Ceramic PGA R8796BH - Ceramic LCC	A8797BH - Ceramic PGA R8797BH - Ceramic LCC

Transistor Count

Device Type	# MOS Gates
839XBH/879XBH	120,000
809XBH	50,000

MTBF Calculations*

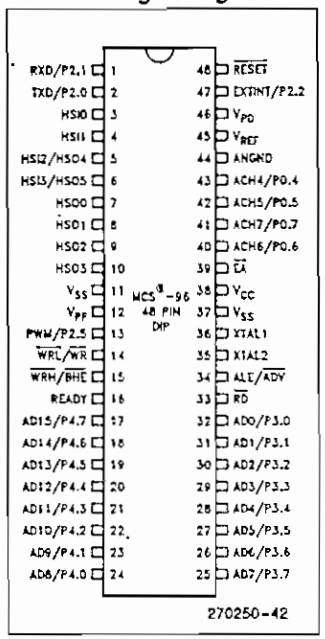
3.8×10^7 Device Hours @ 55°C
1.7×10^7 Device Hours @ 70°C

*MTBF data was obtained through calculations based upon the actual average junction temperatures under stress at 55°C and 70°C ambient.

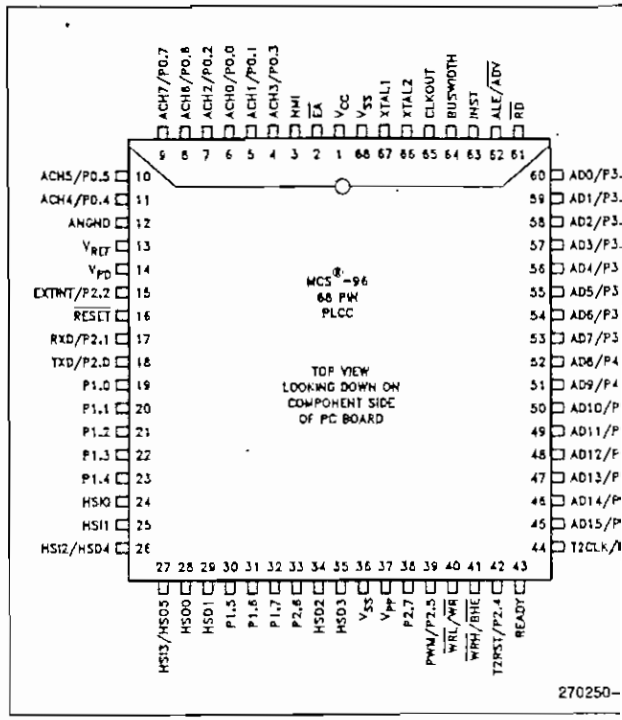
Thermal Characteristics

T_{CASE}		Package Type	θJa	θJc
COMM'L	EXPRESS			
85°C	100°C	PGA	35°C/W	10°C/W
85°C	100°C	PLCC	37°C/W	10°C/W
		LCC	28°C/W	—
		Plastic DIP	38°C/W	—
79.75°C	94.75°C	Ceramic DIP	26°C/W	6.5°C/W

14.4 Package Diagrams

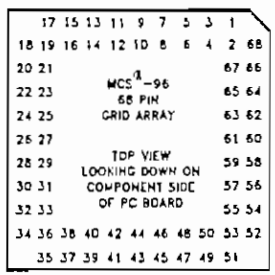


48-Pin Package

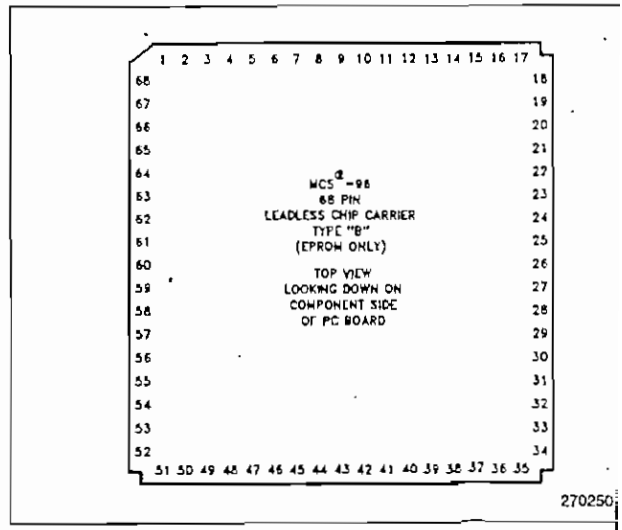


68-Pin Package (PLCC - Top View)

Pins Facing Down



68-Pin Package
(Pin Grid Array - Top View)

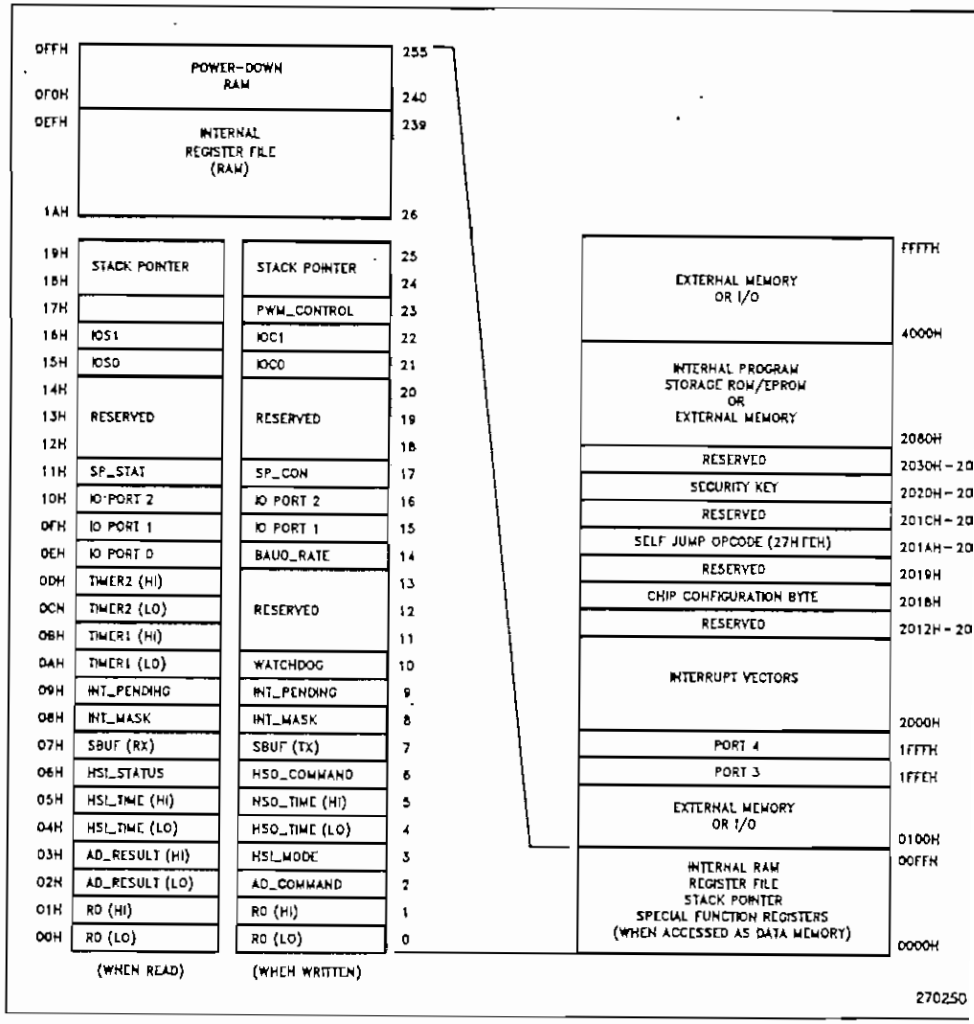


68-Pin Package (LCC - Top View)

5 PGA, PLCC and LCC Function Pinouts

A/ CC	PLCC	Description	PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description
1	9	ACH7/P0.7/PMOD.3	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6/PMOD.2	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{EA}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{BHE}$/$\overline{WRH}$	60	18	TXD/P2.0/PVER/SALE
15	63	INST	38	40	$\overline{WR}$/$\overline{WRL}$	61	17	RXD/P2.1/PALE
16	62	ALE/$\overline{ADV}$	39	39	PWM/P2.5/$\overline{PDO}$/$\overline{SPROG}$	62	16	$\overline{RESET}$
17	61	$\overline{RD}$	40	38	P2.7	63	15	EXTINT/P2.2/$\overline{PROG}$
18	60	AD0/P3.0	41	37	VPP	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4/PMOD.0
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5/PMOD.1
23	55	AD5/P3.5	46	32	P1.7			

14.6 Memory Map



.7 Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
DD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	—	
DD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	—	
DDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	—	
UB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	—	
UB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	—	
UBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	—	
MP/CMPB	2	D − A	✓	✓	✓	✓	↑	—	
IUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
IUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
IULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
IULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
IVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	✓	↑	—	2
IVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	✓	↑	—	3
IV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
IVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	✓	✓	0	0	—	—	
AND/ANDB	3	D ← B and A	✓	✓	0	0	—	—	
OR/ORB	2	D ← D or A	✓	✓	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	✓	✓	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Oper-ands	Operation (Note 1)	Flags						N
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	
JGE	1	Jump if N = 0	—	—	—	—	—	—	
JLT	1	Jump if N = 1	—	—	—	—	—	—	
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	
JV	1	Jump if V = 1	—	—	—	—	—	—	
JNV	1	Jump if V = 0	—	—	—	—	—	—	
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	
JST	1	Jump if ST = 1	—	—	—	—	—	—	
JNST	1	Jump if ST = 0	—	—	—	—	—	—	
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	
EXTB	1	D ← D; D + 1 ← Sign(D)	✓	✓	0	0	—	—	
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb ——— lsb ← 0	✓	?	✓	✓	↑	—	
SHR/SHRB/SHRL	2	0 → msb ——— lsb → C	✓	?	✓	0	—	✓	
SHRA/SHRAB/SHRAL	2	msb → msb ——— lsb → C	✓	✓	✓	0	—	✓	
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	✓	?	0	—	—	—	
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B a must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A ca located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code startin 2080H.
9. The assembler will not accept this mnemonic.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.[8]

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
C	DB	JE	DF	JGE	D6	JGT	D2
NC	D3	JNE	D7	JLT	DE	JLE	DA
H	D9	JV	DD	JVT	DC	JST	D8
NH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.[8]

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

MNEMONIC	OPCODE	BYTES	STATE TIMES
DJNZ	E0	3	5/9 STATE TIME (NOT TAKEN/TAKEN)[8]

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES[8]	MNEMONIC	OPCODE	BYTES	STATES[8]
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES[8]
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT[7]
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT[7]
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT[7]

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES[8]	MNEMONIC	OPCODE	BYTES	STATES[8]
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST[6]	FF	1	166	SKIP	00	2	4

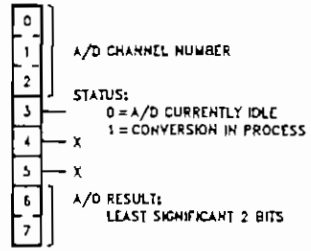
NORMALIZE

MNEMONIC	OPCODE	BYTES	STATE TIMES
NORML	0F	3	11 + 1 PER SHIFT

NOTES:
This instruction takes 2 states to pull RESET low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H. If a capacitor is tied to RESET, the pin may take longer to go low and may never reach the V_{OL} specification.

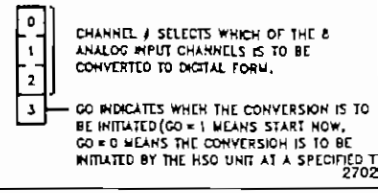
14.9 SFR Summary

A/D Result LO (02H)



270250-48

A/D Command (02H)



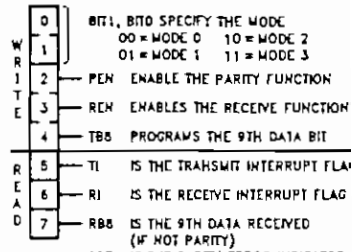
HSI_Mode (03H)

WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

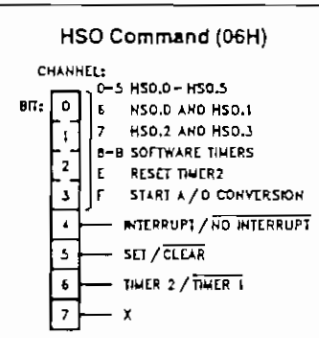
00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

270250-49

SPCON/SPSTAT (11H)



HSO Command (06H)



270250-50

Baud Rate Calculations

Using XTAL1:

$$\text{Mode 0: } \text{Baud Rate} = \frac{\text{XTAL1 frequency}}{4*(B+1)};\ B \neq 0$$

$$\text{Others: } \text{Baud Rate} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$$

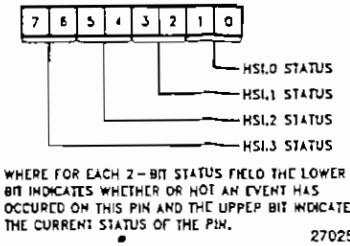
Using T2CLK:

$$\text{Mode 0: } \text{Baud Rate} = \frac{\text{T2CLK frequency}}{B};\ B \neq 0$$

$$\text{Others: } \text{Baud Rate} = \frac{\text{T2CLK frequency}}{16*B};\ B \neq 0$$

Note that B cannot equal 0, except when XTAL1 in other than Mode 0.

HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATE THE CURRENT STATUS OF THE PIN.

2702

.8 Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊙ NORMAL			INDIRECT⊙ AUTO-INC.		INDEXED⊙ SHORT			INDEXED⊙ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES①	BYTES	STATE① TIMES①
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270250-46

NOTES:
Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any Indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.
① Number of state times shown for internal/external operands.
② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.
③ State times shown for 16-bit bus.

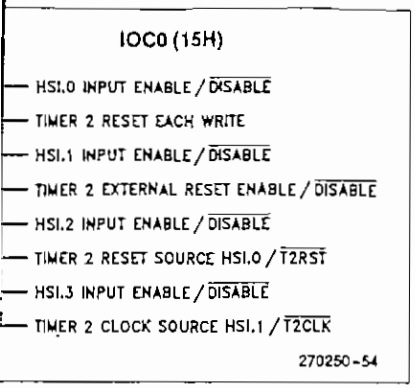
MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊙ NORMAL			INDIRECT⊙ AUTO-INC.		INDEXED⊙ SHORT			INDEXED⊙ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES③	BYTES	STATE①
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	1
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	1
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	1
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	1
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STAT
LJMP	E7	3	8	LCALL	EF	3	13/16
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16
BR[ ]	E3	2	8	RET	F0	1	12/16
				TRAP③	F7	1	21/24

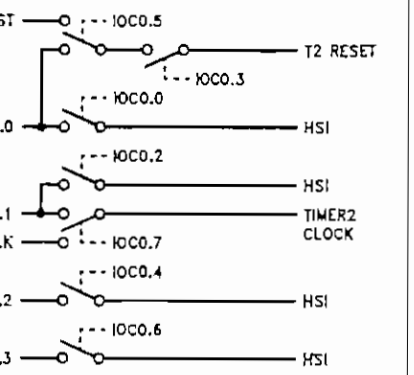
270250

NOTES:
① Number of state times shown for internal/external operands.
④ The assembler does not accept this mnemonic.
④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's comp offset for the relative call or jump.
③ State times for stack located internal/external.

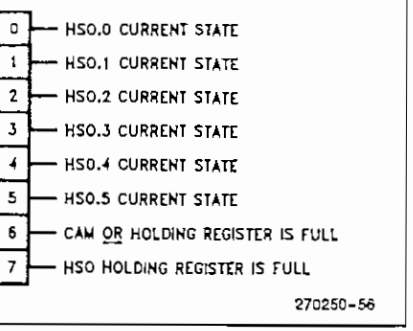
IOC0 (15H)



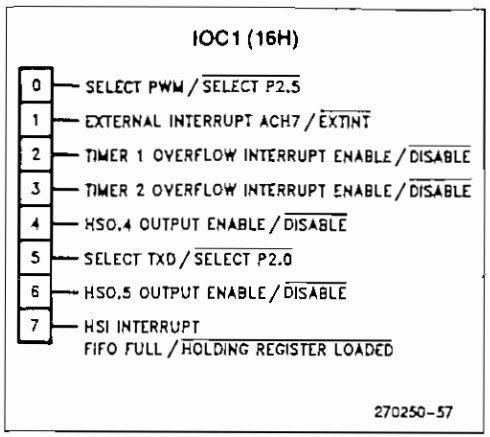
IOC0 (15H)



IOS0 (15H)

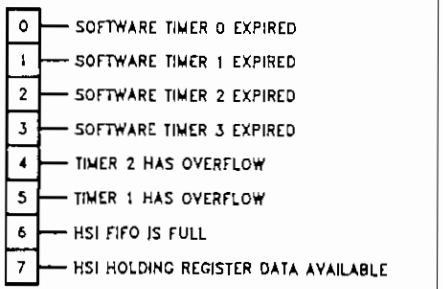


IOC1 (16H)

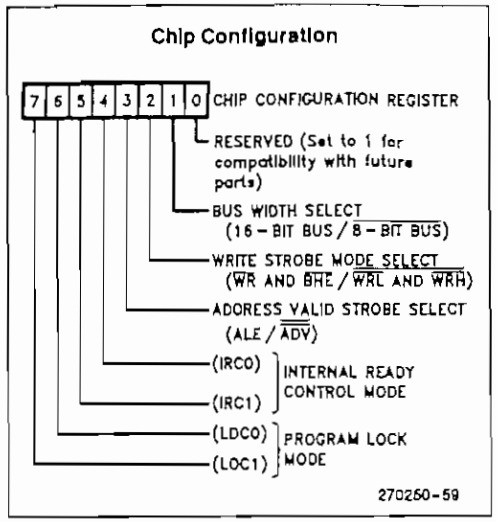


Vector	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

IOS1 (16H)



Chip Configuration



Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Programming Function PMODE Values

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

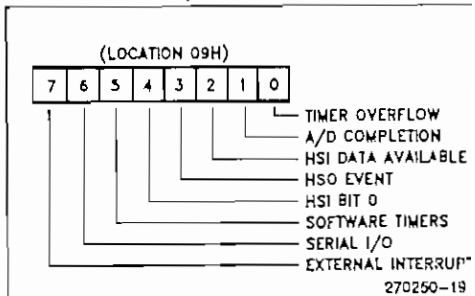
8X9XBH Signature Word

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

Port 2 Pin Functions

Port	Function	Alternate Function
P2.0	Output	TXD (Serial Port Transmit)
P2.1	Input	RXD (Serial Port Receive)
P2.2	Input	EXTINT (External Interrupt)
P2.3	Input	T2CLK (Timer 2 Clock)
P2.4	Input	T2RST (Timer 2 Reset)
P2.5	Output	PWM (Pulse Width Modulation

Interrupt Pending Register



PSW Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

MANUAL DEL USUARIO

El simulador del microcontrolador 8096 nos permite correr un determinado programa para depurarlo o comprobar sus resultados.

Es necesario que el usuario verifique que exista el paquete completo del simulador el mismo que está conformado por los siguientes archivos:

- 8096.PRN
- TI.EXE
- TM.EXE
- TE.EXE
- TD.EXE
- TA.EXE
- TS.EXE

Puede ejecutarse con cualquiera de los archivos .EXE, el ejemplo que se desea probar debe estar ensamblado y el archivo que se utiliza es el . HEX , por lo cual el usuario necesita el ensamblador C16 de la Intel .

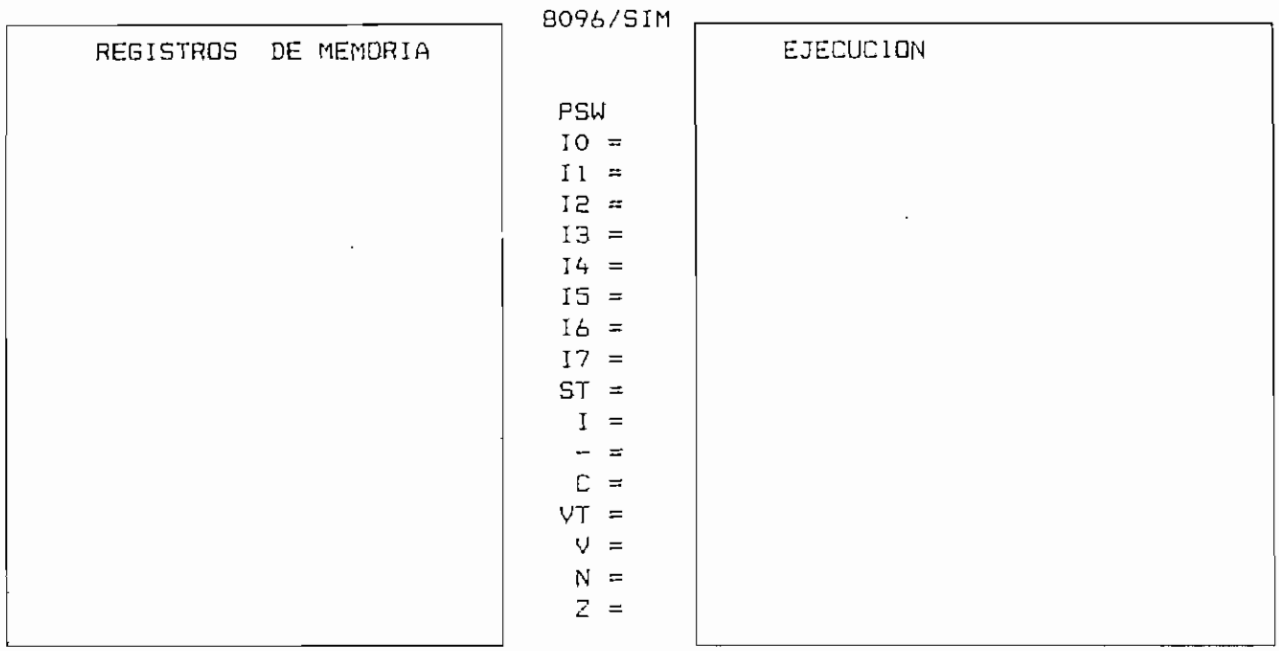
El ejemplo se escribe en cualquier procesador de palabras grabándolo en formato ASCII. Con este archivo que para este caso

se llamará Ejemplo, se utiliza el ensamblador de la siguiente manera:

C> C16 EJEMPLO -H EJEMPLO.HEX

Con lo cual se genera el archivo EJEMPLO.HEX el cual puede ser simulado.

La ejecución se realiza en la siguiente pantalla:



Cpu, Memoria, Banderas, Tabla
>Reset Archivo Memoria Fin Setear Ejecutar Tabla Dosc Imprimir

Las opciones de los menús se escogen con la primera letra,

con las flechas o con el espaciador.

En la opción de Reset nos permite resetear la CPU de manera que realiza un reset del microcontrolador, resetear la memoria que borra todo el contenido de memoria del micro , resetear las banderas y resetear la tabla para volver a cargar otra o simplemente trabajar sin tabla.

En la opción de archivo podemos escoger el drive, el path, y el archivo. En este último en la opción de hexadecimal se pone el archivo .HEX dado por el ensamblador, en análogo el archivo que simula la entrada análoga y en serial el archivo que contiene los datos que van ha ser transmitidos o recibidos en el pórtico serial.

La opción de memoria permite revisar el contenido de la memoria para lo cual le pregunta la dirección inicial de las memorias que desea observar, puede escrolarse las memorias con las flechas arriba o abajo.

Con la opción de modificar se varia el contenido de una determinada memoria dando su dirección, de este modo el cursor se poseciona en dicha memoria con lo cual se varia el valor de esta y se puede variar otras memoria mediante las flechas,

posecionandose en la memoria deseada y cambiando el valor.

En la opción de buscar un valor determinado se debe dar el valor de la memoria a buscarse y la dirección desde la cual debe empezar a buscar , el simulador le dará la dirección de la memoria en la cual esta dicho valor y con cualquier tecla continua buscando. Con ESC termina la busqueda.

En la opción de borrar permite borrar un bloque determinado dando la dirección inicial y la final, de la misma manera en transferir y en llenar en las cuales se debe dar la dirección a la cual se desea transferir o el valor con el que se desea llenar.

La opción de setear nos permite escoger los break points y borrar los mismos total o parcialmente.

La opción de ejecución desencadena la operación del programa, en la modalidad paso a paso o hasta que encuentre un determinado punto de parada, para ejecutar paso a paso se presiona la tecla F1 si se desea seguir en forma continua se presiona C , con la letra M podemos observar las memoria de la 0000H hasta 000fH y si presionamos nuevamente M se observará las

tabla.

La opción de recuperar trae a la memoria una tabla ya realizada anteriormente.

La opción de fin termina la simulación.

La opción de Dos permite ejecutar comandos desde el sistema operativo.

Con imprimir se puede tener en papel o en pantalla el programa en nmemónico , un rango de memória que escoge el usuario, los registros especiales, y los estados de tiempo.

El simulador desensambla automáticamente cuando se ha producido un cambio en el contenido de la memoria, sin variar el archivo que contiene el programa en código hexadecimal, esta caracteritica permite variar ciertos parámetros del programa y ejecutarlo sin necesidad de volver a ensamblarlo, sobre todo en lo que hace relación a saltos o lazos de repetición .

Los puntos de parada "break point" permiten ejecutar un programa hasta una línea determinada y se puede revisar

banderas, valores de determinadas localidades de memoria, registros especiales etc, y luego continuar la ejecución evaluando así el programa simulado.

Si desea salir al menú principal en cualquier momento de la ejecución puede hacerlo con ESC si no realiza ningún cambio en los registro de memoria, break points, tabla , etc,puede seguir la ejecución desde el punto en que se quedó, si se realiza algún cambio vuelve a desensamblarse el programa e iniciará desde el principio.

Con Ctrl flecha izquierda o derecha podemos cambiar de ventana para escrolar las memorias o los nmemónicos de las instrucciones según sea el caso.

PROGRAMA DE INICIALIZACION DEL SIMULADOR

DEL MICROPROCESADOR 8096 (TI.BAS)

```
COMMON M() AS INTEGER
COMMON M1() AS INTEGER
COMMON COD(), NEM$(), BYT(), U(), EST()
COMMON LOC$(), V$(), DASM$(), y$(), A(), AUX(), PSW(), IT$(),
PB(), LSAR(), BP%, RE, TABLA, ARCH$, NLC, NREG
COMMON A, PC%, BPC, IM, B1, L, BP$, NL, BA, ESTADOS, FM14, TIM,
BPS, IPS, BZ, EST, RST, shel
COMMON SP AS DOUBLE
COMMON CLAVE, pi(), menu$(), arcs$, BP(), pasos, F()
CLAVE = 1
IF shel = 1 THEN GOTO fin
REM $DYNAMIC
DIM M(0 TO 32766) AS INTEGER
DIM M1(0 TO 32766) AS INTEGER
DIM LOC$(113), V$(113), DASM$(256), y$(113), F(5)
DIM A(32), AUX(32), PSW(16), IT$(8), PB(16), LSAR(16)
DIM COD(256), NEM$(256), BYT(256), U(256), EST(256), pi(45),
menu$(10), BP(30)
   DATA 3843,3857,3875,3893,3903,3919,3939,3953,3965,3983
   DATA 3843,3853,3871,3891,3907
   DATA 3843,3865,3887,3907,3929,3959,3979
   DATA 3843,3857,3869,3887
   DATA 3843,3865,3885,3899,3921,3937
   DATA 3843,3869,3887,3903
   DATA 3843,3863,3879,3899
   DATA 3843,3857,3879
   ON ERROR GOTO erro
   FOR K = 1 TO 43: READ pi(K): NEXT

   'SE CARGA EL ARCHIVO DEL NEMONICO DE CADA INSTRUCCION
   CLS : LOCATE 3, 25: PRINT "SIMULADOR DEL MICROCONTROLADOR
8096": LOCATE 12, 25: PRINT "DESARROLLADO POR: PATRICIA PROAÑO":
LOCATE 20, 25: PRINT "DIRIGIDO POR ING. FERNANDO FLORES"
90   OPEN "8096.PRN" FOR INPUT AS #1
   i = 1
inicio:
    IF EOF(1) THEN CLOSE : GOTO fin
    LINE INPUT #1, A$
    IF A$ = "" THEN GOTO inicio
    NEM$(i) = MID$(A$, 3, 7): COD(i) = VAL(MID$(A$, 11, 4)):
BYT(i) = VAL(MID$(A$, 14, 2)): U(i) = VAL(MID$(A$, 17, 2))
    EST(i) = VAL(MID$(A$, 19, 3))
    i = i + 1: GOTO inicio
fin:
```

```
    M = 8
      '***************************** IMPRESION DE VENTANAS
      CLS :  COLOR 15, 0: LOCATE 1, 35: PRINT "8096/SIM ": LOCATE
1, 1, 0, 12, 13:  KEY  OFF:  PRINT  CHR$(218);  STRING$(30, 196);
CHR$(191);
      LOCATE 1,  44: KEY  OFF: PRINT CHR$(218); STRING$(35, 196);
CHR$(191)
      FOR J = 1 TO 20: LOCATE 1 + J, 1: PRINT CHR$(179): LOCATE 1
+  J,  32:  PRINT  CHR$(179):  LOCATE 1 + J, 44: PRINT CHR$(179):
LOCATE 1 + J, 80: PRINT CHR$(179): NEXT
      LOCATE 21, 1: PRINT CHR$(192); STRING$(30, 196); CHR$(217);
STRING$(11, 255);  CHR$(192); STRING$(35,  196); CHR$(217): COLOR
7, 0
      'LOCATE 22,2:COLOR 15,0:PRINT  CHR$(25);:COLOR  7,0:PRINT "
continúa   ";:COLOR   15,0:PRINT   CHR$(24);:COLOR   7,0:PRINT  "
regresa"
      LOCATE 22, 56: PRINT "Con "; : COLOR 15, 0:  PRINT "ESC"; :
COLOR 7, 0: PRINT " menú";
      '******************************** IMPRESION DE I1- I9
      FOR I1 = 0 TO 7: LOCATE 5 + I1, 36: PRINT I1; "=": LOCATE 5
+ I1, 36: PRINT "I": NEXT:  LOCATE 13,  36: PRINT  "ST =": LOCATE
14, 36: PRINT " I =": LOCATE 15, 36: PRINT " - =": LOCATE 16, 36:
PRINT " C =": LOCATE 17, 36: PRINT "VT =": LOCATE 18,  _
36: PRINT " V ="
      LOCATE 19, 36: PRINT " N =": LOCATE 20, 36:  PRINT "  Z =":
LOCATE 4, 36: PRINT "PSW"
      ' ***********************  IMPRESION DE R1 - R25 (REGISTROS
ESPECIALES DEL CPU)
      COLOR 7, 0: LOCATE 2, 7: PRINT "REGISTROS   DE  MEMORIA  ":
LOCATE 2, 50: PRINT "EJECUCION " + ARCH$
      CHAIN "TM"
erro:

     '***** SUBRUTINA DE ERROR
      IF ERR  = 53  AND ERL  = 90 THEN LOCATE 24, 32: COLOR 7, 0:
PRINT "NO EXISTE 8096.PRN  "; : RESUME fin1
fin1:
       END
```

```
    FOR ii = 0 TO BP%
    IF PC% = BP(ii) THEN GOSUB impredasm: GOTO PAUSA
    NEXT

codigo:
    GOSUB IMPREG: GOSUB impredasm
codig:
    G = U(M(PC%) + 1)'              Código de instrucción
    IA = 0: IAX = 0
    IF G > 20 THEN G = G - 20: GOTO subrutina
    ON G GOSUB SKIP,  CLR, NOTW,  NEG, DEC,  EXT, INC,  SHR, SHL,
SHRA,  SHRL,  SHLL,  SHRAL,  NORML, CLRB, NOTB, NEGB, DECB, EXTB,
INCB
    GOTO opsigno
subrutina:
    IF G > 20 THEN G = G - 20: GOTO codigo1
    ON G GOSUB SHRB,  SHLB, SHRAB,  SJMP, SCALL,  JBC, JBS, AND3,
ADD3,  SUB3,  MULU3,  ANDB3,  ADDB3,  SUBB3,  MULUB3, AND2, ADD2,
SUBW2, MULU, ANDB2
    GOTO opsigno
codigo1:
    IF G > 20 THEN G = G - 20: GOTO codigo2
    ON G  GOSUB ADDB2,  SUBB, MULUB,  ORW, XORW,  CMP, DIVU, ORB,
XORB,  CMPB,  DIVUB,  LD,  ADDC,  SUBC, LDBZE, LDB, ADDCB, SUBCB,
LDBSE, ST
    GOTO opsigno
codigo2:
    IF G > 20 THEN G = G - 20: GOTO codigo3
    ON G GOSUB STB, PUSH, POP,  JNST, JNH,  JGT, JNC,  JNVT, JNV,
JGE, JNE, JST, JH, JLE, JC, JVT, JV, JLT, JE, BR
    GOTO opsigno
codigo3:
    ON G  GOSUB LJMP,  LCALL, RET,  PUSHF, POPF, NOP, CLRC, SETC,
DI, EI, CLRVT, NOP, opersigno, RST, DJNZ
opsigno:
    IF OS = 1 THEN GOTO break
    TIM = TIM + EST(M(PC%) + 1): TIM1 = INT(TIM / B)
    IF TIM1 >= 65535 AND (M(22) AND 8) = 1 THEN M(9) = M(9) OR 1
    IF TIM1 >= 65535 THEN TIM1 = 0
    DE = INT(TIM1 / 256):  M(11)  =  DE:  M(10)  =  TIM1  -  DE *
256'*******timer 1
    BA1 = M(2) AND 8
    IF BA1 = 8 THEN NA = 1
    IF IPS = 1 THEN GOTO serial2
    IF NA = 1 THEN GOTO intanaloga
    IF BPS = 0 AND M(17) <> 0 THEN GOSUB intserial
    IF PSW(10) = 1 THEN GOTO checkpr
    GOTO checkpc
checkpc:
    IF PC%  <= NREG - BYT(M(PC%) + 1) THEN GOTO entrada ELSE GOTO
regresotm
checkpr:
    M = 8: X = M(9): GOSUB BINARIO      '*****chequeo  del pending
register******
```

```
    I5 = M
checkint:
     IF A(I5)  = 1 THEN GOTO interupcion ELSE IF I5 = 1 THEN GOTO
checkpc ELSE I5 = I5 - 1: GOTO checkint
interupcion:
     ' SUBRUTINA DE INTERRUPCION
     FOR i = 1 TO 8
     PB(i) = A(i): NEXT
     M = 8: X = M(8): GOSUB BINARIO
     I2 = 8
int1:
     IF PB(I2) = 1 OR A(I2) = 1 THEN GOTO int2
     IF I2 = 1 THEN GOTO checkpc ELSE I2 = I2 - 1: GOTO int1
     GOTO checkpc
int2:
     ON I2  GOTO  intanaloga,  intanal1,  intanaloga, intanaloga,
intanaloga, serial5, serial5, intanaloga
intanaloga:
     EST = EST(M(PC%) + 1) + EST
     IF EST < 168 THEN LOCATE 20, 3: PRINT "ESTADOS :"; EST: GOTO
checkpc
     CHAIN "ta"
intanal1:
     IF IAN = 1 THEN M(SP + 1) = INT(PC%  / 256):  M(SP) =  PC% -
M(SP +  1) * 256: PC% = M(8194) + 256 * M(8195): GOTO codigo ELSE
GOTO checkpc
intserial:
     'interrupcion serial
     IF ESTADOS = 0 THEN CHAIN "ts"
     IF ESTADOS <> 1 THEN GOTO serial
     IF i$ = "F" THEN GOTO serial1 ELSE RETURN
serial:
     EST = EST(M(PC%) + 1) + EST
     IF EST < ESTADOS THEN RETURN
serial1:
     M(9) = M(9) OR 64:  RETURN
serial2:
     IF ESTADOS <> 1 THEN GOTO serial3
     IF i$ = "F" OR i$ = "f" THEN GOTO serial4 ELSE GOTO checkpc
serial3:
     EST = EST(M(PC%) + 1) + EST
     LOCATE 20, 17: PRINT "EST "; EST
     IF EST < ESTADOS THEN GOTO checkpc
serial4:
     M = 8: X = M(17): GOSUB BINARIO
     IF A(7) = 1 OR A(6) = 1 THEN CHAIN "ts"
     GOTO checkpc
serial5:
     'PORTICO SERIAL
     M(SP + 1) = INT(PC% / 256)
     M(SP) = PC% - M(SP + 1) * 256
     PC% = M(8204) + 256 * M(8205)
     IPS = 1: GOTO checkpc
opersigno:          'OPERACIONES CON SIGNO
```

```
      PC% = PC% + 1
      OS = 1
      RETURN

DJNZ:  ' decrementa uno y salta si no es cero
      M(PC% + 1) = M(PC% + 1) - 1
      DISP = M(PC% + 2)
      IF DISP > 127 THEN DISP = DISP - 256
      IF M(PC% + 1) <> 0 THEN PC% = PC% + DISP + 2 ELSE PC%  = PC%
+ BYT(M(PC%) + 1) + IAX
      RETURN

SUBW2:     ' DOS OPERANDOS RESTA DE PALABRAS
      M = 16
      IF M(PC%) = 104 THEN GOSUB DIRECTO ELSE IF M(PC%) = 105 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 106 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 107 THEN GOSUB INDEXADO ELSE RETURN
      DEST = DEST - SRC
      IF DEST >= 0 THEN GOSUB banderas: GOTO SBW2
      GOSUB BAND1
SBW2:
      GOTO SBC

ADDB2: ' (DOS OPERANDOS), BYTES
      M = 8
      IF M(PC%) = 116 THEN GOSUB DIRECTO ELSE IF M(PC%) = 117 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 118 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 119 THEN GOSUB INDEXADO ELSE RETURN
      DEST = DEST + SRC
      GOSUB banderas
      M(IND) = DEST
      GOTO CLR1

SKIP: ' Salto no operación
      PC% = PC% + 2
      RETURN

CLR:  ' borrar una palabra
      M(M(PC% +  1)) =  0: M(M(PC%  + 1)  + 1)  = 0:  PSW(16) = 1:
PSW(15) = 0: PSW(12) = 0: PSW(14) = 0
CLR1:
      OS = 0: DW = 0: BI = 0
      PC% = PC% + BYT(M(PC%) + 1) + IAX
      RETURN

NOTW: ' complementa una palabra
      M = 16: GOSUB VERM1
      DEST = DE + 256 * DE1
      DEST = 256 + NOT (DEST)
      DE1 = INT(DEST / 256): DE = DEST - DE1 * 256
      GOSUB PONM1
      GOSUB banderas
      PSW(12) = 0: PSW(14) = 0
      GOTO CLR1
```

```
NEG: ' (Cambia signo) a un integer
     M = 16: GOSUB VERM1
     DEST = DE + DE1 * 256
     IF DEST > 32767 THEN DEST = -1 * (DEST - 65536!) ELSE DEST =
-DEST
     GOSUB bande1
     IF DEST > 0 THEN GOTO NEG1 ELSE DEST = DEST + 65536!
NEG1:
     DE1 = INT(DEST / 256): DE = DEST - DE1 * 256
     GOSUB PONM1
NEG2:
     PC% = PC% + BYT(M(PC%) + 1) + IAX
     RETURN

DEC: '  (resta uno) a una palabra
     M = 16: GOSUB VERM1
     DE = DE - 1
     DEST = DE + 256 * DE1
     IF DEST >= 0 THEN GOSUB banderas: GOTO DEC1
     GOSUB BAND1
DEC1:
     DE1 = INT(DEST / 256): DE = DEST - DE1 * 256
     GOSUB PONM1
     GOTO CLR1

EXT: ' Extiende integer en long-integer 'Revisar
     DE = M(T + 1): DE1 = M(T + 1)
     M = 16: GOSUB VERM1
     DEST = DE + DE1 * 256
     IF DEST < 32768! THEN DEST1 = 0 ELSE DEST1 = 65535!
     DE3 = INT(DEST1 / 256): M(T + 3) = DE3
     DE2 = DEST1 - DE3 * 256: M(T + 2) = DE2
     DEST = DE3 * (256) ^ 3 + DE2 * (256) ^ 2 + DEST
     IF DEST = 0 THEN PSW(16) = 1 ELSE PSW(16) = 0
     IF DEST > 2147483647 THEN PSW(15) = 1 ELSE PSW(15) = 0
     PSW(12) = 0: PSW(14) = 0
     GOTO CLR1

INC: ' Incrementa uno a una palabra
     M = 16: GOSUB VERM1
     DE = DE + 1: DEST = DE + 256 * DE1
     GOSUB banderas
     GOSUB PONM1
     GOTO CLR1

SHR: ' desplazamiento logico a la derecha una palabra
     M = 16: PC% = PC% + 1: GOSUB VERM1: PC% = PC% - 1
     DEST = DE + 256 * DE1
SHR1:
     COUNT = M(PC% + 1)
     IF COUNT  <= 15  AND COUNT  > 0  THEN GOTO SHR2 ELSE COUNT =
M(M(PC% + 1))
SHR2:
```

```
      X = DEST: GOSUB BINARIO
      FOR K = 1 TO COUNT
      C = A(1)
      PSW(12) = C
      IF C = 1 THEN PSW(9) = 1 ELSE PSW(9) = 0
      FOR i = 2 TO M
      A(i - 1) = A(i)
      NEXT: A(M) = 0
      NEXT
      GOSUB DECIMAL
      GOSUB banderas
      PSW(12) = C
      PSW(15) = 0: PSW(14) = 0
      IF M = 32 THEN GOSUB ALDP: GOTO SHR3
      DE1 = INT(DEST / 256): DE = DEST - DE1 * 256
      PC% = PC% + 1: GOSUB PONM1: PC% = PC% - 1
SHR3:
      PC% = PC% + BYT(M(PC%) + 1) + IAX
      EST(M(PC%) + 1) = EST(M(PC%) + 1) + COUNT
      PSW(14) = 0
      RETURN

SHL: 'Desplazamiento logico a la izquierda
      M = 16: PC% = PC% + 1: GOSUB VERM1: PC% = PC% - 1
      DEST = DE + 256 * DE1
      COUNT = M(PC% + 1)
      IF COUNT <= 15 AND COUNT > 0 THEN GOTO SHL1
      COUNT = M(M(PC% + 1))
SHL1:
      FOR K = 1 TO COUNT
      X = DEST: GOSUB BINARIO
      C = A(M)
      PSW(12) = C
      FOR i = 1 TO M - 1
      A(i + 1) = A(i)
      NEXT: A(1) = 0
      NEXT
      GOSUB banderas
      PSW(12) = C
      IF M = 32 THEN GOSUB ALDP: GOTO SHL2
      DE1 = INT(DEST / 256): DE = DEST - DE1 * 256
      PC% = PC% + 1: GOSUB PONM1: PC% = PC% - 1
SHL2:
      PC% = PC% + BYT(M(PC%) + 1) + IAX
      EST(M(PC%) + 1) = EST(M(PC%) + 1) + COUNT
      RETURN

SHRA: ' Desplazamiento arit.a la derecha
      M = 16: PC% = PC% + 1: GOSUB VERM1: PC% = PC% - 1
      DEST = DE + 256 * DE1

SHRA1:
      COUNT = M(PC% + 1): PSW(9) = 0
      IF COUNT <= 15 AND COUNT > 0 THEN GOTO SA1 ELSE COUNT =
```

```
M(M(PC% + 1))
SA1:
      X# = DEST#: GOSUB BINARIO
      C1 = A(M)
      FOR K = 1 TO COUNT
      C = A(1)
      PSW(12) = C
      IF C = 1 THEN PSW(9) = PSW(12) ELSE PSW(9) = 0
      FOR i = 2 TO M
      A(i - 1) = A(i)
      NEXT i
      A(M) = C1
      NEXT K: GOSUB DECIMAL
      GOSUB banderas
      PSW(12) = C
      IF M = 16 OR M = 32 THEN GOSUB ALDP: GOTO SHRA2
       DE = DEST - INT(DEST / 256) * 256: DE1 = INT(DEST / 256)
      PC% = PC% + 1: GOSUB PONM1: PC% = PC% - 1
SHRA2:
      PC% = PC% + BYT(M(PC%) + 1) + IAX
      EST(M(PC%) + 1) = EST(M(PC%) + 1) + COUNT
      PSW(14) = 0
      RETURN

SHRL: 'Desplazamiento logico a la derecha doble palabra
      COUNT = M(PC% + 1)
      M = 16: PC% = PC% + 1: GOSUB VERM1: PC% = PC% - 1
      DE2 = M(M(PC% + 2) + 2): DE3 = M(M(PC% + 2) + 3):
      DEST# = DE + 256 * DE1 + 256 ^ 2 * DE2 + 256 ^ 3 * DE3
      M = 32: GOSUB SHR1
      RETURN

SHLL: ' Desplazamiento logico a la izquierda
      COUNT = M(PC% + 1)
      DE =  M(M(PC% +  2) +  1): DE1  = M(M(PC%  + 2)  + 2): DE2 =
M(M(PC% + 2) + 3):
      DEST# = M(M(PC% + 2)) + 256 * DE + 256 ^ 2 * DE1 + 256 ^ 3 *
DE2
      IF COUNT  <= 15  AND COUNT  > 0  THEN GOTO SHL1 ELSE COUNT =
M(M(PC% + 1))
      M = 32: GOSUB SHL1
      GOSUB ALDP: RETURN
ALDP:
      IND = M(PC% + 2)
ALDPM:

      DE3# = INT(DEST# / 256 ^ 3)
      M(IND + 3) = DE3#
      D# = DEST# - 256 ^ 3 * DE3#
      DE2# = INT(D / 256 ^ 2)
      M(IND + 2) = DE2#
      D# = D# - 256 ^ 2 * DE2#
      DE1# = INT(D / 256)
      M(IND + 1) = DE1#
```

```
      M(IND) = D# - 256 * DE1#
      RETURN
ALDP1:
      PC% = PC% + BYT(M(PC%) + 1) + IAX
      EST(M(PC%) + 1) = EST(M(PC%) + 1) + COUNT
      RETURN

SHRAL: ' Desplazamiento  doble palabra
      DE = M(M(PC% +  2) +  1): DE1  = M(M(PC%  + 2)  + 2):  DE2 =
M(M(PC% + 2) + 3)
      DEST# = M(M(PC% + 2)) + 256 * DE + 256 ^ 2 * DE1 + 256 ^ 3 *
DE2
      M = 32: GOSUB SHRA1
      RETURN

NORML: ' NORMALIZA LONG-INTEGER
      M = 16: GOSUB VERM1
      DE2 = M(M(PC% + 1) + 2): DE3 = M(M(PC% + 1) + 3)
      DEST# = DE + 256 * DE1 + 256 ^ 2 * DE2 + 256 ^ 3 * DE3
      IF DEST# > 2147483647 THEN DEST# = DEST# - 4294967295#
      FOR i = 1 TO 32
      M = 32: X = DEST#: GOSUB BINARIO
      IF A(32) = 1 THEN GOTO NORML1
      DEST# = DEST# * 2
      NEXT i
      IF DEST# = 0 THEN PSW(16) = 1 ELSE PSW(16) = 0
      PSW(12) = 0
      GOTO CLR1
NORML1:
      COUNT = i
      M(PC% + 1) = COUNT
      GOSUB ALDP
      PSW(15) = 1: PSW(12) = 0
      PC% = PC% + BYT(M(PC%) + 1) + IAX
      EST(M(PC%) + 1) = EST(M(PC%) + 1) + 8 + COUNT
      RETURN

CLRB: ' Borrar el byte
      PSW(16) = 1: PSW(15) = 0: PSW(12) = 0: PSW(14) = 0
      M(M(PC% + 1)) = 0
      GOTO CLR1

NOTB: ' Cada bit del byte es complementado
      M = 8
      DEST = M(M(PC% + 1))
      DEST = NOT (DEST)
      GOSUB banderas
      PSW(16) = 0: PSW(14) = 0
      M(M(PC% + 1)) = DEST
      GOTO CLR1

NEGB: ' EL VALOR DEL SHORT INTEGER ES NEGADO
      DEST = M(M(PC% + 1))
      IF DEST > 127 THEN DEST = DEST - 256
```

```
      DEST = -1 * DEST: GOSUB banderas
      IF DEST > 0 THEN GOTO NEGB1 ELSE DEST = DEST + 256
NEGB1:
      M(M(PC% + 1)) = DEST
      GOTO CLR1

DECB: ' DECREMENTO DEL BYTE
      M = B
      DEST = M(M(PC% + 1)) - 1
      IF DEST >= 0 THEN GOSUB banderas: GOTO DECB1
      GOSUB BAND1
DECB1:
      M(M(PC% + 1)) = ABS(DEST)
      GOTO CLR1

EXTB: ' EXTIENDE SHORT INTEGER EN INTEGER
      DE = M(M(PC% + 1))
      IF DE < 128 THEN M(M(PC% + 1) + 1) = 0  ELSE M(M(PC%  + 1) +
1) = 255
      DE1 = M(M(PC% + 1) + 1)
      DEST = DE1 * 256 + DE
      IF DEST = 0 THEN PSW(16) = 0
      IF DEST > 32767 THEN PSW(15) = 1 ELSE PSW(15) = 0
      PSW(12) = 0: PSW(14) = 0
      GOTO CLR1

INCB: ' BYTE INCREMENTADO
      DEST = M(M(PC% + 1)) + 1:
      M(M(PC% + 1)) = DEST: GOSUB banderas
      GOTO CLR1

SHRB: ' DESPLAZAMIENTO A LA DERECHA EL BYTE
      M = 8: DEST = M(M(PC% + 2)): GOTO SHR1

SHLB: ' DESPLAZAMIENTO BYTE A LA IZQUIERDA
      DEST = M(M(PC% + 2))
      COUNT = M(PC% + 1)
      IF COUNT  <= 15  AND COUNT  > 0  THEN GOTO SHL1 ELSE COUNT =
M(M(PC% + 1))
      M = 8: GOSUB SHL1
      RETURN

SHRAB: ' DEZPLAZAMIENTO ARITMETICO DE UN BYTE
      M = 8: DEST = M(M(PC% + 2)): GOTO SHRA1

SJMP: ' SALTO CORTO
      X = M(PC%): M = 8: GOSUB BINARIO: A(9) = A(1): A(10) = A(2):
A(11)  =  A(3):  X  =  M(PC%  +  1): GOSUB BINARIO: M = 11: GOSUB
DECIMAL: DISP = DEST
      IF (DISP > 1023) THEN DISP = DISP - 2048
      IF (DISP < 1023)  AND (DISP  > -1024)  THEN GOTO  SJMP1 ELSE
PRINT "FUERA DE RANGO"
SJMP1:
      PC% = PC% + DISP + 2
```

```
     RETURN

SCALL: ' LLAMADO CORTO
     X = M(PC%): M = 8: GOSUB BINARIO: A(9) = A(1): A(10) = A(2):
A(11) = A(3): X = M(PC% + 1): GOSUB BINARIO: M = 11: GOSUB
DECIMAL: DISP = DEST
     IF (DISP > 1023) THEN DISP = DISP - 2048
     SP = SP - 2: M(SP) = PC% - INT(PC% / 256) * 256: M(SP + 1) =
INT(PC% / 256)
     PC% = PC% + DISP + 2
     IF M(SP) > 255 THEN EST(M(PC%) + 1) = EST(M(PC%) + 1) + 3
     RETURN

JBC: ' SALTO SI EL NB ES CERO
     X = M(M(PC% + 1)): DISP = M(PC% + 2)
     IF DISP > 127 THEN DISP = DISP - 256
     NB = M(PC%) - 47: M = 8: GOSUB BINARIO
     IF A(NB) = 0 THEN PC% = PC% + DISP + 3: EST(M(PC%) + 1) =
EST(M(PC%) + 1) + 4 ELSE PC% = PC% + 3
     RETURN

JBS: ' SALTO SI EL BIT ES UNO
     X = M(M(PC% + 1)): DISP = M(PC% + 2)
     IF (DISP > 127) THEN DISP = DISP - 256
     NB = M(PC%) - 55: M = 8: GOSUB BINARIO
     IF A(NB) = 1 THEN PC% = PC% + DISP + 3: EST(M(PC%) + 1) =
EST(M(PC%) + 1) + 4 ELSE PC% = PC% + 3
     RETURN

AND2: ' LOGICO DE PALABRAS dos operandos
     M = 16
     IF M(PC%) = 96 THEN GOSUB DIRECTO ELSE IF M(PC%) = 97 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 98 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 99 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST AND SRC
     GOSUB banderas
     PSW(12) = 0: PSW(14) = 0
     GOTO SBC

ADD2: ' AÑADIR PALABRAS DE 16 BITS dos operandos
     M = 16
     IF M(PC%) = 100 THEN GOSUB DIRECTO ELSE IF M(PC%) = 101 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 102 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 103 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST + SRC
     GOSUB banderas
     GOTO SBC

MULU: '(palabras) Y MUL(Integers) MULTIPLICACION dos operandos
     M = 16
     IF M(PC%) = 108 THEN GOSUB DIRECTO ELSE IF M(PC%) = 109 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 110 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 111 THEN GOSUB INDEXADO ELSE RETURN
     IF OS = 1 THEN IF DEST > 32767 THEN DEST = DEST - 65536!
```

```
ELSE DEST = DEST
     IF OS  = 1  THEN IF SRC > 32767 THEN SRC = SRC - 65536! ELSE
SRC = SRC
     DEST = DEST * SRC
     IF DEST < 0 THEN DEST = DEST + 4294967296#
     IF BI = 1 THEN FOR i = 0 TO 2: MI(IND + i) = DEST - INT(DEST
/ 256)  * 256:  DEST = INT(DEST / 256): NEXT: MI(IND + 3) = DEST:
GOTO CLR1
     FOR i = 0 TO 2
     M(IND + i) = DEST - INT(DEST / 256) * 256: DEST = INT(DEST /
256)
     NEXT: M(IND + 3) = DEST
     OS = 0: GOTO CLR1

ANDB2: ' (dos operandos) bytes
     M = 8
     IF M(PC%) = 112 THEN GOSUB DIRECTO ELSE IF M(PC%) = 113 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 114 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 115 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST AND SRC
     GOSUB banderas
     PSW(12) = 0: PSW(14) = 0
     M(IND) = DEST
     GOTO CLR1

ANDB3: ' (tres operandos)bytes
     M = 8
     IF M(PC%)  = 80  THEN GOSUB DIRECTO ELSE IF M(PC%) = 81 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 82 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 83 THEN GOSUB INDEXADO ELSE RETURN
     DEST = SRC1 AND SRC2
     GOSUB banderas
     PSW(12) = 0: PSW(14) = 0
     GOTO SBC

SUBB: ' DOS OPERANDOS RESTA DE BYTES
     M = 8
     IF M(PC%) = 120 THEN GOSUB DIRECTO ELSE IF M(PC%) = 121 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 122 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 123 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST - SRC
     IF DEST >= 0 THEN GOSUB banderas: GOTO SB2
     GOSUB BAND1
SB2:
     M(IND) = ABS(DEST)
     GOTO CLR1

MULUB: ' Y MULB DOS OPERANDOS MULTIPLICACION DE BYTES
     M = 8
     IF M(PC%) = 124 THEN GOSUB DIRECTO ELSE IF M(PC%) = 125 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 126 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 127 THEN GOSUB INDEXADO ELSE RETURN
     IF OS  = 1  THEN IF  DEST >  127 THEN DEST = DEST - 256 ELSE
DEST = DEST
```

```
     IF OS = 1 THEN IF SRC > 127 THEN SRC = SRC - 256 ELSE  SRC =
SRC
     DEST = DEST * SRC
     IF DEST < 0 THEN DEST = DEST + 65536!
     GOTO SBC

AND3: ' TRES OPERANDOS AND LOGICO DE PALABRAS
     M = 16
     IF M(PC%)  = 64  THEN GOSUB DIRECTO ELSE IF M(PC%) = 65 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 66 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 67 THEN GOSUB INDEXADO ELSE RETURN
     DEST = SRC1 AND SRC2
     GOSUB banderas
     PSW(12) = 0: PSW(14) = 0
     GOTO SBC

ORW: ' LOGICO DE PALABRAS
     M = 16
     IF M(PC%) = 128 THEN GOSUB DIRECTO ELSE IF M(PC%) = 129 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 130 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 131 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST OR SRC
     GOSUB banderas
     PSW(16) = 0: PSW(14) = 0
     GOTO SBC

XORW: ' EXCLUSIVO DE PALABRAS
     M = 16
     IF M(PC%) = 132 THEN GOSUB DIRECTO ELSE IF M(PC%) = 133 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 134 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 135 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST XOR SRC
     GOSUB banderas
     PSW(12) = 0: PSW(14) = 0
     GOTO SBC

CMP: ' COMPARE PALABRAS
     M = 16
     IF M(PC%) = 136 THEN GOSUB DIRECTO ELSE IF M(PC%) = 137 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 138 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 139 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST - SRC
     IF DEST >= 0 THEN GOSUB banderas: GOTO CLR1
     GOSUB BAND1
     GOTO CLR1

DIVU: ' palabras (dos operandos) Y DIV(integers)
     M = 16: DW = 1
     IF M(PC%) = 140 THEN GOSUB DIRECTO ELSE IF M(PC%) = 141 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 142 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 143 THEN GOSUB INDEXADO ELSE RETURN
     IF OS  = 1  THEN IF DEST# > 2147483647# THEN DEST# = DEST# -
4294967296# ELSE DEST# = DEST#
     IF SRC = 0 THEN BEEP: LOCATE 20, 47: PRINT "DIV / 0  ": S$ =
```

```
INPUT$(1): LOCATE 20, 47: PRINT "          "; : GOTO CLR1
     IF OS  = 1  THEN IF SRC > 32767 THEN SRC = SRC - 65536! ELSE
SRC = SRC
     DEST1# = INT(ABS(DEST# / SRC))
     IF DEST1# > 65535! THEN PSW(14) = 1:  PSW(13) =  1: DEST1# =
65535 ELSE PSW(14) = 0
     IF DEST# / SRC < 0 THEN DEST1# = -DEST1# + 65536!
     DEST2# =  ABS(DEST#) -  INT(ABS(DEST# / SRC)) * ABS(SRC): IF
DEST# / SRC < 0 THEN DEST2# = -DEST2# + 65536!
     IF BI = 1 THEN M1(IND) = DEST1# - INT(DEST1#  / 256)  * 256:
IN# = INT(DEST1# / 256): M1(IND + 1) = IN# - INT(IN# / 256) * 256
ELSE M(IND) = DEST1# - INT(DEST1# / 256) * 256:  IN# = INT(DEST1#
/ 256): M(IND + 1) = IN# - INT(IN# / 256) * 256
     IF BI  = 1  THEN M1(IND  + 2) = DEST2# - INT(DEST2# / 256) *
256: IN# = INT(DEST2# / 256): M1(IND + 3) = IN# -  INT(IN# / 256)
* 256  ELSE M(IND  + 2) = DEST2# - INT(DEST2# / 256) * 256: IN# =
INT(DEST2# / 256): M(IND + 3) = IN# - INT(IN# / 256)
     GOTO CLR1

ORB: ' OR LOGICO DE BYTES
     M = 8
     IF M(PC%) = 144 THEN GOSUB DIRECTO ELSE IF M(PC%) = 145 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 146 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 147 THEN GOSUB INDEXADO ELSE RETURN
     GOSUB banderas
     DEST = DEST OR SRC: M(IND) = DEST
     PSW(12) = 0: PSW(14) = 0
     GOTO CLR1

XORB: ' Or exclusivo de bytes
     M = 8
     IF M(PC%) = 148 THEN GOSUB DIRECTO ELSE IF M(PC%) = 149 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 150 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 151 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST XOR SRC
     GOSUB banderas
     M(IND) = DEST
     PSW(12) = 0: PSW(14) = 0
     GOTO CLR1

CMPB: ' COMPARAR BYTS
     M = 8
     IF M(PC%) = 152 THEN GOSUB DIRECTO ELSE IF M(PC%) = 153 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 154 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 155 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST - SRC
     IF DEST >= 0 THEN GOSUB banderas: GOTO CLR1
     GOSUB BAND1
     GOTO CLR1

DIVUB: ' bytes (dos operandos) Y DIVU(short integer)
     M = 8: DB = 1
     IF M(PC%) = 156 THEN GOSUB DIRECTO ELSE IF M(PC%) = 157 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 158 THEN GOSUB INDIRECTO ELSE IF
```

```
M(PC%) = 159 THEN GOSUB INDEXADO ELSE RETURN
     IF OS = 1 THEN IF  DEST >  32767 THEN  DEST =  DEST - 65536!
ELSE DEST = DEST
     IF OS  = 1 THEN IF SRC > 127 THEN SRC = SRC - 256 ELSE SRC =
SRC
     IF SRC = 0 THEN BEEP: LOCATE 20, 47: PRINT "DIV /  0 "; : S$
= INPUT$(1): LOCATE 20, 47: PRINT "         "; : GOTO CLR1
     DEST1 = INT(ABS(DEST / SRC))
     IF DEST1  > 255  THEN PSW(14)  = 1: PSW(13) = 1: DEST1 = 255
ELSE PSW(14) = 0
     IF DEST / SRC < 0 THEN DEST1 = -DEST1 + 256
     DEST2 = ABS(DEST) - INT(ABS(DEST / SRC)) * ABS(SRC): IF DEST
/ SRC < 0 THEN DEST2 = -DEST2 + 256
     IF BI  = 1  THEN M1(IND)  = DEST1  - INT(DEST1 / 256) * 256:
M1(IND + 1) = DEST2 - INT(DEST2 / 256) * 256: OS = 0: DB = 0 ELSE
M(IND) =  DEST1 -  INT(DEST1 /  256) *  256: M(IND + 1) = DEST2 -
INT(DEST2 / 256) * 256: OS = 0: DB = 0
     GOTO CLR1

LD: 'cargar una palabra
     M = 16
     IF M(PC%) = 160 THEN GOSUB DIRECTO ELSE IF M(PC%) = 161 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 162 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 163 THEN GOSUB INDEXADO ELSE RETURN
     M(IND) = SRC - INT(SRC / 256) * 256: M(IND + 1)  = INT(SRC /
256)
     IF IND = 24 THEN DE = M(25): SP = M(24) + 256 * DE
     GOTO CLR1

ADDC: ' Anadir palabras con carry
     M = 16
     IF M(PC%) = 164 THEN GOSUB DIRECTO ELSE IF M(PC%) = 165 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 166 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 167 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST + SRC + PSW(12): PSW16 = PSW(16): GOSUB banderas
     IF PSW16 = 0 THEN PSW(16) = 0
     GOTO SBC

SUBC: ' Resta de palabras con borrow
     M = 16
     IF M(PC%) = 168 THEN GOSUB DIRECTO ELSE IF M(PC%) = 169 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 170 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 171 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST - SRC - (1 - PSW(12))
     IF DEST >= 0 THEN GOSUB banderas: GOTO SBC
     GOSUB BAND1
SBC:
     DE1 = INT(DEST / 256): M(IND + 1) = DE1: M(IND) = DEST - DE1
* 256
     GOTO CLR1

LDBZE: ' Cargar una palabra con un byte
     M = 8
     IF M(PC%) = 172 THEN GOSUB DIRECTO ELSE IF M(PC%) = 173 THEN
```

```
GOSUB INMEDIATO ELSE IF M(PC%) = 174 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 175 THEN GOSUB INDEXADO ELSE RETURN
     M(IND) = SRC
     M(IND + 1) = 0
     GOTO CLR1

LDB: ' Cargar un byte
      M = 8
     IF M(PC%) = 176 THEN GOSUB DIRECTO ELSE IF M(PC%) = 177 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 178 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 179 THEN GOSUB INDEXADO ELSE RETURN
     IF IND <> 14 THEN GOTO LDB1
     IF BZ = 1  THEN FM14  = M(14)  + 256  * SRC:  GOTO LDB2 ELSE
M(IND) = SRC: BZ = 1: GOTO LDB2
LDB1:
     M(IND) = SRC
LDB2:
     IF IND = 7 THEN M(17) = M(17) OR 32
     GOTO CLR1

ADDCB: ' Añadir bytes con carry
     IF M(PC%) = 180 THEN GOSUB DIRECTO ELSE IF M(PC%) = 181 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 182 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 183 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST + SRC + PSW(12): PSW16 = PSW(16): GOSUB banderas
     IF PSW16 = 0 THEN PSW(16) = 0
     GOTO SBC

SUBCB: ' Restar bytes con el borrow
     M = 8
     IF M(PC%) = 184 THEN GOSUB DIRECTO ELSE IF M(PC%) = 185 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 186 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 187 THEN GOSUB INDEXADO ELSE RETURN
     DEST = DEST - SRC - (1 - PSW(12))
     IF DEST >= 0 THEN GOSUB banderas: GOTO SBCB
     GOSUB BAND1
SBCB:
     M(IND) = DEST
     GOTO CLR1

LDBSE: ' Cargar un integer con short-integer
     IF M(PC%) = 188 THEN GOSUB DIRECTO ELSE IF M(PC%) = 189 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 190 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 191 THEN GOSUB INDEXADO ELSE RETURN
     M(IND) = SRC
     IF SRC < 128 THEN SRCA = 0 ELSE SRCA = 255
     M(IND + 1) = SRCA
     GOTO CLR1

ST: 'Store una palabra (dos operandos)
     BI = 0
     M =  16: IF M(PC%) = 192 THEN GOSUB DIRECTO ELSE IF M(PC%) =
194 THEN GOSUB INDIRECTO ELSE IF M(PC%) = 195 THEN GOSUB INDEXADO
ELSE RETURN
```

```
     IF INSTO > 32766 THEN INSTO = INSTO - 32767: BI = 1
     IF BI  = 0 THEN DE = INT(DEST / 256): M(INSTO + 1) = DE: DE1
= DEST - 256 * DE: M(INSTO) = DE1: GOTO CLR1
     IF INSTO > 32766 THEN INSTO = INSTO - 32767: DE = INT(DEST /
256): F(INSTO  + 1)  = DE: DE1 = DEST - 256 * DE: F(INSTO) = DE1:
GOTO CLR1
     DE = INT(DEST / 256): M1(INSTO + 1) = DE: DE1 = DEST - 256 *
DE: M1(INSTO) = DE1
     GOTO CLR1

STB: ' Almacenar un byte
     BI = 0
     M =  8: IF  M(PC%) = 196 THEN GOSUB DIRECTO ELSE IF M(PC%) =
197  THEN  GOSUB  INMEDIATO  ELSE  IF  M(PC%)  =  198  THEN GOSUB
INDIRECTO ELSE IF M(PC%) = 199 THEN GOSUB INDEXADO ELSE RETURN
     IF INSTO > 32766 THEN INSTO = INSTO - 32767: BI = 1
     IF BI = 0 THEN M(INSTO) = DEST: GOTO CLR1
     IF  INSTO  >  32766  THEN  INSTO = INSTO - 32767: F(INSTO) =
DEST: GOTO CLR1
     M1(INSTO) = DEST
     GOTO CLR1

PUSHF: ' LAS BANDERAS
      M = 16
     SP = SP - 2
     FOR i = 1 TO 8: A(i) = PSW(i + 8): NEXT
     M = 8: GOSUB DECIMAL:
     IF SP > 32766 THEN M1(SP)  = M(8):  M1(SP +  1) =  DEST ELSE
M(SP) = M(8): M(SP + 1) = DEST
     FOR i = 1 TO 16: PSW(i) = 0: NEXT
     PSW = 0: M(8) = 0
     GOTO CLR1

POP: ' Pop Ia palabra
     M = 16: M(M(PC% + 1)) = M(SP): M(M(PC% + 1) + 1) = M(SP + 1)
     SP = SP + 2
     GOTO CLR1

JNST: ' SALTAR SI STICKY BIT ES CERO
     DISP = M(PC% + 1)
     IF (DISP > 127) THEN DISP = DISP - 256
     IF PSW(9)  = 0  THEN PC%  = PC%  + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1) + IAX
     RETURN

JNH: ' SALTO SI NO ES MAYOR
     DISP = M(PC% + 1)
     IF DISP > 127 THEN DISP = DISP - 256
     IF PSW(12) = 0 OR PSW(16) = 1 THEN PC% = PC% + DISP + 2 ELSE
PC% = PC% + BYT(M(PC%) + 1) + IAX
     RETURN

JGT: ' Salto si eI signed es mayor
     DISP = M(PC% + 1)
```

```
      IF DISP > 127 THEN DISP = DISP - 256
      IF PSW(15)  = 0  AND PSW(16)  = 0  THEN PC% = PC% + DISP + 2
ELSE PC% = PC% + BYT(M(PC%) + 1) + IAX
      RETURN

JNC: ' Salto si el carry es cero
      DISP = M(PC% + 1)
      IF DISP > 127 THEN DISP = DISP - 256
      IF PSW(12) = 0 THEN PC% = PC% + DISP  + 2  ELSE PC%  = PC% +
BYT(M(PC%) + 1) + IAX
      RETURN

JNVT: ' Salto si el Overflow es cero
      DISP = M(PC% + 1)
      IF DISP > 127 THEN DISP = DISP - 256
      IF PSW(13)  = 0  THEN PC%  = PC% + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1)
      PSW(13) = 0: RETURN

JNV: ' salto si el signo es mayor o igual
      DISP = M(PC% + 1)
      IF DISP > 127 THEN DISP = DISP - 256
      IF PSW(14) = 0 THEN PC% = PC% + DISP  + 2  ELSE PC%  = PC% +
BYT(M(PC%) + 1)
      RETURN

JGE: ' SALTO SI EL SIGNO ES MAYOR O IGUAL
      DISP = M(PC% + 1)
      IF DISP > 127 THEN DISP = DISP - 256
      IF PSW(15)  = 0  THEN PC%  = PC% + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1)
      RETURN

JNE: ' Salto si no es igual a cero
      DISP = M(PC% + 1)
      IF DISP > 127 THEN DISP = DISP - 256
      IF PSW(16) = 0 THEN PC% = PC% + DISP  + 2  ELSE PC%  = PC% +
BYT(M(PC%) + 1)
      RETURN

JST: ' Salto si el sticky bit es uno
      DISP = M(PC% + 1)
      IF DISP > 127 THEN DISP = DISP - 256
      IF PSW(9)  = 1  THEN PC%  = PC%  + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1)
      RETURN

JH: ' Salto si es mayor (sin signo)
      DISP = M(PC% + 1)
      IF DISP > 127 THEN DISP = DISP - 256
      IF PSW(12) = 1 AND PSW(16) = 0  THEN PC%  = PC%  + DISP  + 2
ELSE PC% = PC% + BYT(M(PC%) + 1)
      RETURN
```

```
JLE: ' Salto si el signo es menor o igual
     DISP = M(PC% + 1)
     IF DISP > 127 THEN DISP = DISP - 256
     IF PSW(15) = 1 AND PSW(16) = 1 THEN PC% = PC% + DISP + 2
ELSE PC% = PC% + BYT(M(PC%) + 1)
     RETURN

JC: ' Salto si el carry es uno
     DISP = M(PC% + 1)
     IF DISP > 128 THEN DISP = DISP - 256
     IF PSW(12) = 1 THEN PC% = PC% + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1)
     RETURN

JVT: ' Salto si el overflow trap es uno
     DISP = M(PC% + 1)
     IF DISP > 128 THEN DISP = DISP - 256
     IF PSW(13) = 1 THEN PC% = PC% + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1)
     PSW(13) = 0: RETURN

JV: ' Salto si el overflow bandera es uno
     DISP = M(PC% + 1)
     IF DISP > 128 THEN DISP = DISP - 256
     IF PSW(14) = 1 THEN PC% = PC% + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1)
     RETURN

JLT: ' Salto si el signo es menor que
     DISP = M(PC% + 1)
     IF DISP > 128 THEN DISP = DISP - 256
     IF PSW(15) = 1 THEN PC% = PC% + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1)
     RETURN

JE: 'Salto si es igual
     DISP = M(PC% + 1)
     IF DISP > 128 THEN DISP = DISP - 256
     IF PSW(16) = 1 THEN PC% = PC% + DISP + 2 ELSE PC% = PC% +
BYT(M(PC%) + 1)
     RETURN

BR: '(indirecto) Salto indirecto

     PC% = M(M(PC% + 1)) + 256 * M(M(PC% + 1) + 1)
     RETURN

LJMP: ' Salto largo
     M = 16: GOSUB VERM1
     DISP = DE + 256 * DE1
     PC% = PC% + DISP + 2
     RETURN

LCALL: ' Llamada larga
```

```
      DE = M(PC% + 2)
      DISP = M(PC% + 1) + 256 * DE
      SP =  SP - 2: DE1 = INT(PC% / 256): M(SP + 1) = DE1: M(SP) =
PC% - DE1 * 256
      PC% = PC% + DISP + 2
      RETURN

RET: ' Return de una subrutina
      DE = M(SP + 1)
      PC% = M(SP) + 256 * DE
      BPS = 0: IPS = 0
      SP = SP + 2
      RETURN checkpc

PUSH: 'PUSH PALABRAS
      IF M(PC%) = 96 THEN GOSUB DIRECTO ELSE  IF M(PC%)  = 97 THEN
GOSUB INMEDIATO  ELSE IF M(PC%) = 98 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 99 THEN GOSUB INDEXADO ELSE RETURN
      SP = SP - 2
      DE1 = INT(SRC / 256): M(SP + 1) = DE1: M(SP) =  SRC -  DE1 *
256
      RETURN

POPF: 'BANDERAS
      E = M(SP): E1 = M(SP + 1)
      M(8) = E
      PSW = E + 256 * E1: M = 16: X = PSW: GOSUB BINARIO
      FOR I1 = 1 TO 16: PSW(I1) = A(I1): NEXT
      SP = SP + 2: PC% = PC% + 1
      RETURN

CLRC: 'Borrar el carry
      PSW(12) = 0: PC% = PC% + 1
      RETURN

SETC: 'Setear el carry
      PSW(12) = 1: PC% = PC% + 1
      RETURN

DI: 'DESABILITA INTERRUPCIONES
      PSW(10) = 0
      PC% = PC% + BYT(M(PC%) + 1)
      RETURN

EI: 'HABILITA INTERRUPCIONES
      PSW(10) = 1
      PC% = PC% + BYT(M(PC%) + 1)
      RETURN

CLRVT: 'Borra el overflow trap
      PSW(13) = 0
      PC% = PC% + BYT(M(PC%) + 1)
      RETURN
```

```
NOP: 'No operacion
     PC% = PC% + BYT(M(PC%) + 1)
     RETURN

RST: 'RESETEA EL SISTEMA
     RST = 1
     FOR i = 9 TO 16
     PSW(i) = 0
     NEXT
     CHAIN "TM"

ADD3: '(tres operandos) anadir palabras
     M = 16
     IF M(PC%)  = 68  THEN GOSUB DIRECTO ELSE IF M(PC%) = 69 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 70 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 71 THEN GOSUB INDEXADO ELSE RETURN
     DEST = SRC1 + SRC2
     GOSUB banderas
     GOTO SBC

SUB3: '(Tres operandos) resta de palabras
     M = 16
     IF M(PC%)  = 72  THEN GOSUB DIRECTO ELSE IF M(PC%) = 73 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 74 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 75 THEN GOSUB INDEXADO ELSE RETURN
     DEST = SRC1 - SRC2
     IF DEST >= 0 THEN GOSUB banderas: GOTO SBW3
     GOSUB BAND1
SBW3:
     GOTO SBC

MULU3: '(Integers) Y MULU(Palabras) multiplicacion tres operandos
     M = 16
     IF M(PC%)  = 76  THEN GOSUB DIRECTO ELSE IF M(PC%) = 77 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 78 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 79 THEN GOSUB INDEXADO ELSE RETURN
     IF OS  = 1  THEN IF  SRC1 >  32767 THEN SRC1 = SRC1 - 65536!
ELSE SRC1 = SRC1
     IF OS = 1 THEN IF  SRC2 >  32767 THEN  SRC2 =  SRC2 - 65536!
ELSE SRC2 = SRC2
     DEST# = SRC1 * SRC2
     IF DEST# < 0 THEN DEST# = DEST# + 4294967296#
     IF BI = 1 THEN GOSUB ALDPM1 ELSE GOSUB ALDPM
     GOTO CLR1

ADDB3: '(tres operandos) anade bytes
     M = 8
     IF M(PC%)  = 84  THEN GOSUB DIRECTO ELSE IF M(PC%) = 85 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 86 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 87 THEN GOSUB INDEXADO ELSE RETURN
     DEST = SRC1 + SRC2: GOSUB banderas
     GOTO SBC

SUBB3: '(tres operandos) resta bytes
```

```
     M = 8
     IF M(PC%)  = 88  THEN GOSUB DIRECTO ELSE IF M(PC%) = 89 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 90 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 91 THEN GOSUB INDEXADO ELSE RETURN
     DEST = SRC1 - SRC2
     IF DEST >= 0 THEN GOSUB banderas: GOTO SBB3
     GOSUB BAND1
SBB3:
    GOTO SBC
MULU83: '(tres operandos) multiplica bytes
     M = 8
     IF M(PC%)  = 92  THEN GOSUB DIRECTO ELSE IF M(PC%) = 93 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 94 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 95 THEN GOSUB INDEXADO ELSE RETURN
     IF OS  = 1  THEN IF  SRC1 >  127 THEN SRC1 = SRC1 - 256 ELSE
SRC1 = SRC1
     IF OS = 1 THEN IF SRC2 >  127 THEN  SRC2 =  SRC2 -  256 ELSE
SRC2 = SRC2
     DEST = SRC1 * SRC2: IF DEST < 0 THEN DEST = 65536! + DEST
     GOTO SBC

BINARIO: 'RUTINA DE CONVERCION A BINARIO
     D = 0: FOR i = 1 TO M
     Y = ABS(INT(X / 2))
     A(i) = X - Y * 2
     X = Y
     NEXT i
     RETURN

DECIMAL:   'RUTINA DE BINARIO A DECIMAL
     SUM = 0
     FOR i = 1 TO M
     SUM = SUM + A(i) * (2 ^ (i - 1))
      NEXT i: DEST = SUM
      RETURN

banderas:   'rutina para chequear banderas
      IF DEST = 0 THEN PSW(16) = 1 ELSE PSW(16) = 0 'Z
      IF M = 16 THEN GOTO BANDERAS1
      IF M = 32 THEN GOTO BANDERAS2
      PSW(15) = DEST AND 128
      IF PSW(15) = 128 THEN PSW(15) = 1 ELSE PSW(15) = 0
      IF DEST  > 255 THEN PSW(14) = 1: PSW(13) = 1 ELSE PSW(14) =
0
      IF DEST <= 511 AND DEST  >= 256  THEN PSW(12)  = 1:  DEST =
DEST AND 127 ELSE PSW(12) = 0
      RETURN


BANDERAS1:
'     PSW(15) = DEST AND 32768
      IF PSW(15) = 32768 THEN PSW(15) = 1 ELSE PSW(15) = 0
      IF DEST  > 65535 THEN PSW(14) = 1: PSW(13) = 1 ELSE PSW(14)
= 0
```

```
      IF DEST <= 131071 AND DEST >= 65536 THEN PSW(12)  = 1: DEST
= DEST AND 32767 ELSE PSW(12) = 0
      RETURN
BANDERAS2:
      X = DEST: GOSUB BINARIO
      PSW(15) = A(32)
      IF DEST  > 4294967295#  THEN PSW(14)  = 1: PSW(13) = 1 ELSE
PSW(14) = 0
      IF DEST >= 4294967296# THEN PSW(12)  = 1:  DEST =  DEST AND
2147483647# ELSE PSW(12) = 0
      RETURN


DIRECTO:
      X = BYT(M(PC%) + 1)
      IF M = 16 THEN GOTO WORDS2
      IF X = 4 THEN GOTO BYTS3
      SRC = M(M(PC% + 1))
      IF DB = 1 THEN SR = M(M(PC% + 2) + 1): DEST = M(M(PC% + 2))
+ 256 * SR ELSE DEST = M(M(PC% + 2))
      IND = M(PC% + 2): INSTO = M(PC% + 1)
      RETURN
BYTS3:
      SRC2 = M(M(PC% + 1))
      SRC1 = M(M(PC% + 2))
      IND = M(PC% + 3)
      RETURN
WORDS2:
      IF X = 4 THEN GOTO WORDS3
      SR = M(M(PC% + 1) + 1)
      SRC = M(M(PC% + 1)) + 256 * SR
      DE = M(M(PC% + 2) +  1): DE1  = M(M(PC%  + 2)  + 2):  DE2 =
M(M(PC% + 2) + 3)
      IF DW  = 1  THEN da#  = 256 * (DE + 256 * DE1 + (256 ^ 2) *
DE2): DEST# = M(M(PC% + 2)) + da# ELSE DEST = M(M(PC% + 2)) + 256
* DE
      IND = M(PC% + 2): INSTO = M(PC% + 1)
      RETURN
WORDS3:
      SR1 = M(M(PC% + 1) + 1)
      SRC2# = M(M(PC% + 1)) + 256 * SR1
      SR2 = M(M(PC% + 2) + 1)
      SRC1# = M(M(PC% + 2)) + 256 * SR2
      IND = M(PC% + 3)
      RETURN

INDIRECTO:
      H = M(PC% + 1) - INT(M(PC% + 1) / 2) * 2
      IF H = 1 THEN IA = 1 ELSE IA = 0
      X = BYT(M(PC%) + 1)
      IF M = 16 THEN GOTO IWORDS2
      IF X = 4 THEN GOTO IBYTS3
      SR = (M(M(PC% + 1) + 1 - IA))
      S1 = M(M(PC% + 1) - IA) + 256 * SR
```

```
      IF S1 > 65533 THEN S1 = S1 - 65534: SRC = F(S1)
      IF S1 > 32766 THEN S1 = S1 - 32767: SRC = M1(S1) ELSE SRC =
M(S1)
      IF DB = 1 THEN SR1 = M(M(PC% + 2)  + 1):  DEST =  M(M(PC% +
2)) + 256 * SR1 ELSE DEST = M(M(PC% + 2))
      IND = M(PC% + 2): INSTO = M(M(PC% + 1) - IA) + 256 * SR
      GOTO IW1
IBYTS3:
      SR1 = M(M(PC% + 1) + 1 - IA)
      SR2 = M(M(PC% + 1) - IA) + 256 * SR1
      IF SR2 > 65533 THEN SR2 = SR2 - 65534: SRC2 = F(SR2)
      IF SR2  > 32766 THEN SR2 = SR2 - 32767: SRC2 = M1(SR2) ELSE
SRC2 = M(SR2)
      SRC1 = M(M(PC% + 2))
      IND = M(PC% + 3)
      GOTO IW1
IWORDS2:
      IF X = 4 THEN GOTO IWORDS3
      DE4 = M(M(PC% + 1) + 1 - IA): SC = M(M(PC% + 1) - IA) + 256
* (DE4)
      DE =  M(M(PC% +  2) +  1): DE1  = M(M(PC%  + 2) + 2): DE2 =
M(M(PC% + 2) + 3)
      IF SC > 65533 THEN SC = SC - 65534: DE5 = F(SC +  1): SRC =
F(SC): SRC = SRC + 256 * DE5
      IF SC > 32766 THEN SC = SC - 32767: DE5 = M1(SC + 1): SRC =
M1(SC): SRC = SRC + 256 * DE5 ELSE DE5 = M(SC + 1):  SRC = M(SC):
SRC = SRC + 256 * DE5
      IF DW  = 1  THEN da  = 256  * (DE + 256 * DE1 + (256 ^ 2) *
DE2): DEST = M(M(PC% + 2)) + da ELSE DEST = M(M(PC% + 2)) + 256 *
DE
      IND = M(PC% + 2): INSTO = M(M(PC% + 1) - IA) + 256 * DE4
      GOTO IW1
IWORDS3:

      SR1 = (M(M(PC% + 1) + 1) - IA)
      SC = M(M(PC% + 1) - IA) + 256 * SR1
      IF SC > 65533 THEN SC = SC - 65534: SR2 = F(SC + 1): SRC2 =
F(SC) + 256 * SR2
      IF SC > 32766 THEN SC = SC - 32767: SR2 = M1(SC +  1): SRC2
= M1(SC)  + 256  * SR2 ELSE SR2 = M(SC + 1): SRC2 = M(SC) + 256 *
SR2
      SR3 = M(M(PC% + 2) + 1)
      SRC1 = M(M(PC% + 2)) + 256 * SR3
      IND = M(PC% + 3)
      GOTO IW1
IW1:
      IF IA = 1 THEN GOTO IW2 ELSE RETURN
IW2:
      IF M = 8 THEN M(M(PC% +  1))  =  M(M(PC%  +  1))  +  1 ELSE
M(M(PC% + 1) - IA) = M(M(PC% + 1) - IA) + 2
      RETURN

INMEDIATO:
      X = BYT(M(PC%) + 1)
```

```
        IF M = 16 THEN GOTO NWORDS2
        IF X = 4 THEN GOTO NBYTS3
        SRC = M(PC% + 1)
        IF DB = 1 THEN SR = M(M(PC% + 2) + 1): DEST = M(M(PC% + 2))
    + 256 * SR ELSE DEST = M(M(PC% + 2))
        IND = M(PC% + 2): INSTO = M(PC% + 1)
        RETURN
    NBYTS3:
        SRC2 = M(PC% + 1)
        SRC1 = M(M(PC% + 2))
        IND = M(PC% + 3)
        RETURN
    NWORDS2:
        IF X = 5 THEN GOTO NWORDS3
        DE4 = M(PC% + 2): SRC = M(PC% + 1) + 256 * DE4:  DE = M(PC%
    + 3): DE1 = M(DE + 1): DE2 = M(DE + 2): DE3 = M(DE + 3)
        IF DW  = 1  THEN da  = 256 * (DE1 + 256 * DE2 + (256 ^ 2) *
    DE3): DEST = M(DE) + da ELSE DEST = M(DE) + 256 * DE1
        IND = M(PC% + 3): INSTO = SRC
        RETURN
    NWORDS3:
        SR1 = M(PC% + 2)
        SRC2 = M(PC% + 1) + 256 * SR1
        SR2 = M(M(PC% + 3) + 1)
        SRC1 = M(M(PC% + 3)) + 256 * SR2
        IND = M(PC% + 4)
        RETURN

    INDEXADO:
        X = BYT(M(PC%) + 1): IAX = 0
        H = M(PC% + 1) - INT(M(PC% + 1) / 2) * 2
        IF H = 0 THEN IAX = 0: GOTO SHORTINX
        SC1 = M(M(PC% + 1))
        SC = M(M(PC% + 1) - 1) + 256 * SC1
        IAX = 1                 '  H=0  short indexado
        OF = M(PC% + 2)
        OF1 = M(PC% + 3)
        OF = OF + 256 * OF1
        IF OF > 32768! THEN OF = OF - 65536!
        GOTO XBYTS2
    SHORTINX:
        SC1 = M(M(PC% + 1) + 1)
        SC = M(M(PC% + 1)) + 256 * SC1
        OF = M(PC% + 2)
        IF OF > 128 THEN OF = OF - 256
    XBYTS2:
        IF M = 16 THEN GOTO XWORDS2
        IF X = 5 THEN GOTO XBYTS3
      ' IF SC + OF > 32766 THEN  DE  = SC  +  OF  -  32767:  SRC =
    M1(ABS(DE)) ELSE SRC = M(ABS(SC + OF))
        IF DB = 1 THEN DE = M(M(PC% + 3 + IAX) + 1): DEST = M(M(PC%
    + 3 + IAX)) + 256 * DE ELSE DEST = M(M(PC% + 3 + IAX))
        IND = M(PC% + 3 + IAX): INSTO = SC + OF
        RETURN
```

```
XBYTS3:
      IF SC + OF > 32766 THEN DE = SC + OF - 32767: SRC2 =
M1(ABS(DE)) ELSE SRC2 = M(ABS(SC + OF))
      SRC1 = M(M(PC% + 3 + IAX))
      IND = M(PC% + 4 + IAX)
      RETURN
XWORDS2:
      IF X = 5 THEN GOTO XWORDS3
      IF SC + OF > 32766 THEN SC1 = SC + OF - 32767: DE =
M1(ABS(SC1 + 1)): SRC = M1(ABS(SC1)) + 256 * DE ELSE DE =
M(ABS(SC + OF + 1)): SRC = M(ABS(SC + OF)) + 256 * DE
      DE = M(M(PC% + 3 + IAX) + 1): DE1 = M(M(PC% + 3 + IAX) +
2): DE2 = M(M(PC% + 3 + IAX) + 3)
      IF DW = 1 THEN da = 256 * (DE + 256 * DE1 + (256 ^ 2) *
DE2): DEST = M(M(PC% + 3 + IAX)) + da ELSE DEST = M(M(PC% + 3 +
IAX)) + 256 * DE
      IND = M(PC% + 3 + IAX): INSTO = SC + OF'var para stor
      RETURN
XWORDS3:
      IF SC + OF > 32766 THEN SC1 = SC + OF - 32767: DE =
M1(ABS(SC1 + 1)): SRC2 = M1(ABS(SC1)) + 256 * DE ELSE DE =
M(ABS(SC + OF + 1)): SRC2 = M(ABS(SC + OF)) + 256 * DE
      SR2 = M(M(PC% + 3 + IAX) + 1)
      SRC1 = M(M(PC% + 3 + IAX)) + 256 * SR2
      IND = M(PC% + 4 + IAX)
      RETURN

regresotm:
      BPC = 1: CHAIN "TM"

impredasm:
      IF M(PC%) = 254 THEN RETURN
      FOR W1 = 1 TO NL - 1
      X$ = MID$(dasm$(W1), 2, 4): GOSUB HEXDEC: PA = SUM
      IF PC% = PA THEN GOTO IMPRE1
      NEXT
      IF im > 15 THEN GOTO IMPRE2
      atr = 7: GOSUB atributo

IMPRE1:
      IF W1 <= 15 THEN im = W1
      IF NL <= 15 THEN GOSUB BVD1: NR = NL ELSE NR = 15
      FOR j = 0 TO NR
      LOCATE 4 + j, 47: PRINT USING "\
\"; dasm$(W1 + j + 1 - im): NEXT
      im1 = 160 * (im - 1): atr = 112: GOSUB atributo
      RETURN
IMPRE2:
      FOR j = 0 TO 15
      LOCATE 4 + j, 47: PRINT USING "\
\"; dasm$(W1 - 12 + j): NEXT
      im = 15: atr = 112: GOSUB atributo
      RETURN
```

```
PAUSA:
      BP$ = INPUT$(1): IF BP$ = CHR$(27) THEN GOTO regresotm ELSE
GOTO codigo' GOSUB 11940:GOSUB IMPREDASM:GOTO codigo

BVD1:
      IF NL < 15 THEN J2 = 15 - NL ELSE J2 = 0
BVD:      FOR j1 = 0 TO 16 - J2
      LOCATE 4 + j1 + J2, 47: PRINT STRING$(30, " "): NEXT
      RETURN
IMPREG:
      COLOR 7,  0: LOCATE  2, 6, 0: PRINT " REGISTROS  DE MEMORIA
"
      LOCATE 2, 60, 0: PRINT STRING$(11, 255);
      LOCATE 2, 50, 0: PRINT "EJECUCION " + arch$;
      COLOR 15, 0
      FOR I22 = 0 TO 15
      IF A > 65430 THEN A = 65430
         B = 4 - LEN(HEX$(A + I22 * 8))
          LOCATE 4 + I22, 3
        PRINT STRING$(B, "0"); HEX$(A + I22 * 8);
      NEXT
      FOR I12 = 0 TO 120 STEP 8
      I11 = B
      FOR I1 = 0 TO 7
      IF A + I1 + I12 > 32766 THEN GOTO CM31
      IF LEN(HEX$(M(A + I1 + I12))) < 2 THEN P1$ = "0" + HEX$(M(A
+ I1 + I12)) ELSE P1$ = HEX$(M(A + I1 + I12))
      GOTO CM32
CM31:
      A = A - 32767
      IF  LEN(HEX$(M1(A  +  I1  +  I12)))  <  2  THEN P1$ = "0" +
HEX$(M1(A + I1 + I12)) ELSE P1$ = HEX$(M1(A + I1 + I12)): STOP
      A = A + 32767

CM32:
      LOCATE 4 + I12 / 8, I11, 0
      COLOR 7, 0: PRINT P1$;
      I11 = I11 + 3
      NEXT: NEXT
      COLOR 15, 0
      FOR I1 = 1 TO 16: LOCATE 4 + 11, 41
      PRINT PSW(I1): NEXT
      COLOR 7, 0
      RETURN
HEXDEC:
      SUM = 0
      FOR IAA = 1 TO LEN(X$)
      Y$ = MID$(X$, LEN(X$) - IAA + 1, 1)
      IF ASC(Y$) >= 65 THEN V1 = 55 ELSE V1 = 48
      SUM = ((ASC(Y$) - V1) * (16 ^ (IAA - 1))) + SUM
      NEXT IAA
RETURN

bandel:
```

```
      IF DEST = 0 THEN PSW(16) = 1 ELSE PSW(16) = 0 'Z
      IF M = 16 THEN GOTO bande2
      IF DEST < 0 THEN PSW(15) = 1 ELSE PSW(15) = 0
      IF DEST > 127 OR DEST < -128 THEN PSW(14) = 1: PSW(13) = 1:
DEST = 255 ELSE PSW(14) = 0
      IF DEST  < 255 AND DEST > 127 THEN PSW(12) = 1: DEST = DEST
AND 127 ELSE PSW(12) = 0
      RETURN
bande2:
      IF DEST < 0 THEN PSW(15) = 1 ELSE PSW(15) = 0
      IF DEST > 32767 OR DEST < -32768 THEN PSW(14)  = 1: PSW(13)
= 1: DEST = 65535 ELSE PSW(14) = 0
      IF DEST  < 65535  AND DEST > 32767 THEN PSW(12) = 1: DEST =
DEST AND 32767 ELSE PSW(12) = 0
      RETURN

BAND1:
     PSW(16) = 0: PSW(15) = 1
     IF M = 16 THEN GOTO BAND2
     IF DEST < -128 THEN PSW(12) = 0 ELSE PSW(12) = 1
     IF DEST < -128 THEN PSW(14) = 1: PSW(13) = 1 ELSE  PSW(14) =
0
     DEST = 256 + DEST
     IF DEST < 0 THEN DEST = 255
     RETURN
BAND2:
     IF DEST < -32768 THEN PSW(12) = 0 ELSE PSW(12) = 1
     IF DEST  < -32768 THEN PSW(14) = 1: PSW(13) = 1 ELSE PSW(14)
= 0
     DEST = 65536 + DEST
     IF DEST < 0 THEN DEST = 65535
     RETURN

VERM1:
     DE = M(M(PC% + 1))
     IF M = 8 THEN RETURN
     DE1 = M(M(PC% + 1) + 1)
     RETURN

PONM1:
     M(M(PC% + 1)) = DE
     IF M = 8 THEN RETURN
     M(M(PC% + 1) + 1) = DE1
     RETURN
ALDPM1:
     DE3 = INT(DEST# / 256 ^ 3)
     IF IND + 3 > 32766 THEN F(IND + 3 - 32767) = DE3
     M1(IND + 3) = DE3
     D = DEST# - 256 ^ 3 * DE3
     DE2 = INT(D / 256 ^ 2)
     IF IND + 2 > 32766 THEN F(IND + 2 - 32767) = DE2
     M1(IND + 2) = DE2
     D = D - 256 ^ 2 * DE2
     DE1 = INT(D / 256)
```

```
      M1(IND + 1) = DE1
      M1(IND) = D - 256 * DE1
      RETURN


atributo:
   FOR K = 575 + im1 TO 629 + im1 STEP 2
   POKE (K), atr
   NEXT K
    RETURN
atrip:
    LOCATE 25,  1: PRINT  ">"; :  COLOR 0,  7: PRINT  "Pasos "; :
COLOR 7, 0: PRINT " Continuo    ";
    RETURN
atric:
    LOCATE 25,  1: PRINT  ">"; :  COLOR 7,  0: PRINT "Pasos  "; :
COLOR 0, 7: PRINT "Continuo ";
    COLOR 7, 0
    RETURN
```

PROGRAMA DE MENUS DEL SIMULADOR DEL

MICROCONTROLADOR 8096 (TM.BAS)

```
COMMON M() AS INTEGER
COMMON M1() AS INTEGER
COMMON COD(), NEM$(), BYT(), U(), EST()
COMMON  LOC$(),  V$(),  DASM$(),  Y$(), A(), AUX(), PSW(), IT$(),
PB(), LSAR(), BP%, RE, tabla, ARCH$, NLC, NREG
COMMON A, PC%, BPC, IM, B1, L, BP$,  NL, BA,  ESTADOS, FM14, TIM,
BPS, IPS, BZ, EST, RST, shel
COMMON SP AS DOUBLE
COMMON CLAVE, pi(), menu$(), arcs$, BP(), pasos, F()
DIM LOP$(16), P2$(16), BPT$(30)
IF CLAVE = 0 THEN RUN "TI"
   ON ERROR GOTO manerror
   DEF SEG = 0
   IF (PEEK(&H410)  AND &H30)  = &H30 THEN DEF SEG = &HB000: GOTO
colr
   DEF SEG = &HB800
colr:

   GOTO menu1
colrint:
vent:
   LOCATE 25, 1, 0
   PRINT menu$;
   FOR K = 1 TO ne - 1
   POKE pi(K + ie), 15
   NEXT K
   RETURN

colrnorm:

   LOCATE 24, 1, 0: PRINT STRING$(70, 255);
   LOCATE 24, 1, 0: PRINT menu$(inxm);
   GOSUB clrmay
   FOR K = pi(inxm + ie) TO pi(inxm + ie + 1) - 4 STEP 2
   POKE (K), 112
   NEXT K
   RETURN
clrmay:

   IF pii = 0 THEN pii = 1
   FOR j = pi(pii + ie) TO pi(pii + ie + 1) - 4 STEP 2
   POKE pi(pii + ie), 15
   POKE j + 2, 7
   NEXT
   RETURN

nombarch:
```

```
    LX = 13: L = 22: CO = 25: PAT = 0
    LOCATE 22, 1: PRINT STRING$(40, 255)
    LOCATE 22, 1, 0: COLOR 7, 0: PRINT "NOMBRE DEL  ARCHIVO HEX :
                                    "; : GOSUB endatos: ARCH$ = YO$
    IF LEN(ARCH$) <= 3 THEN ARCH$ = ARCH$ + ".HEX"
    IF  MID$(ARCH$,  LEN(ARCH$)  -  3,  4) <> ".HEX" THEN ARCH$ =
ARCH$ + ".HEX"

    '***********************************************LECTURA      DE
ARCHIVO A SIMULAR
    path$ = UNIDAD$ + PATT$
310 OPEN path$ + ARCH$ FOR INPUT AS #1
    i = 1: NREG = 0

entra1:
    IF EOF(1) THEN CLOSE #1: LOCATE 22, 1, 0: PRINT STRING$(70, "
") ELSE GOTO elemen1
    IF DAT = 1 THEN NREG = NREG2: DAT = 0 ELSE NREG = NREG
    A = 8320: GOSUB cm3
    RETURN

elemen1:
    INPUT #1, AUX$
    IF i = 1 THEN i = i + 1: GOTO elemen1
    X$ = MID$(AUX$, 2, 2): GOSUB hexadecimal: NEG = SUM
    IF NEG = 0 THEN NREG = NREG + K: GOTO entra1
    NREG1 = NREG
    X$ = MID$(AUX$, 4, 4): GOSUB hexadecimal: NREG = SUM
    'IF NREG - NREG1 > 16 THEN DAT = 1: NREG2 = NREG1 + K
    NREG = NREG: K = 0
    FOR j = 1 TO 2 * NEG STEP 2
    X$ = MID$(AUX$, 9 + j, 2): GOSUB hexadecimal
    M(NREG + K) = SUM
    K = K + 1: NEXT
    i = i + 1
    GOTO entra1
tb1:
tabla = 1
'cargar tabla de default por defecto
RETURN

fintabla:
    GOSUB bvi: GOSUB cm2
    tabla = 1: LOCATE 22, 1, 0: PRINT STRING$(50, " ")
    LOCATE 22, 1, 0: PRINT STRING$(80, " ")
    CLOSE #2: texit = 0
    GOTO menu1

tb2:
    GOSUB PRINTABLA
    LOCATE 22, 1, 0:  PRINT STRING$(45,  " "):  LOCATE 22,  1, 0:
PRINT STRING$(80, " ")
    i = 1
```

```
PRINTABLA:
    COLOR 7, 0: LN = 0
    GOSUB bvi
    LOCATE 2, 7, 0: PRINT "CONTENIDO DE LA TABLA";
    LOCATE 5, 6, 0: PRINT "VARIABLE"; "       "; "LOCALIDAD";
    IF NLC = 0 THEN RETURN
    FOR j = 0 TO NLC
    B = 4 - LEN(LOC$(j))
    LOC$(j) = STRING$(B, "0") + LOC$(j)
    LOCATE 6  + LN,  6, 0:  PRINT USING  "\    \"; Y$(j); TAB(21);
LOC$(j)
    IF LN > 11 THEN LOCATE  20,  6,  0:  PRINT  "continua":  P$ =
INPUT$(1): K1 = 3: GOSUB bvi: LN = 0 ELSE LN = LN + 1
    NEXT:
    RETURN


tb8:
    LX = 13: L = 22: CO = 22: HX = 0
    LOCATE 22,  1, 0:  COLOR 7, 0: PRINT "NOMBRE DE LA TABLA:"; :
GOSUB endatos: ARC$ = Y0$
    LOCATE 22, 1, 0: PRINT STRING$(50, " ")
    IF X$ = CHR$(27) THEN LOCATE  22, 1,  0: PRINT  STRING$(55, "
"): LOCATE 25, 2, 0: GOTO entrada5
    RETURN

hexadecimal:
      SUM = 0
      FOR IA = 1 TO LEN(X$)
      Y$ = MID$(X$, LEN(X$) - IA + 1, 1)
      IF ASC(Y$) >= 65 THEN V1 = 55 ELSE V1 = 48
      SUM = ((ASC(Y$) - V1) * (16 ^ (IA - 1))) + SUM
      NEXT IA
      RETURN
      '***************************** INGRESO AL MENU principal
menu1:

      COLOR 7, 0: LOCATE 25, 1, 0: PRINT STRING$(70, 255);
      menu$ =  ">Reset   Archivo   Memoria  Fin  Setear  Ejecutar
Tabla  Dosc  Imprimir"
      menu$(1) = "Cpu, Memoria, Banderas, Tabla"
      menu$(2) = "Drive, Path, Archivo "
      menu$(3)  =  "Contenido,  Modificar,  Encontrar, Borrar(B),
Transferir(B), Llenar(B)"
      menu$(4) = "Salir al DOS"
      menu$(5) = "Ingresar, Borrar, Corregir"
      menu$(6) = "Ejecutar el programa ( Pasos, Continuo )"
      menu$(7) = "Recuperar, Archivar, Crear, Modificar, Borrar"
      menu$(8) = " Ejecutar comando del DOS"
      menu$(9) = "Imprimir resultados"
      ne = 10: inxm = 1: ie = 0
      LOCATE 22, 1, 0: PRINT STRING$(70, 255);
      LOCATE 25, 1, 0
      GOSUB vent: GOSUB colrnorm
```

```
ingreso1:

      Q$ = INKEY$: IF Q$ = "" THEN GOTO ingreso1
      IF MID$(Q$,  2, 1)  = "M" OR Q$ = CHR$(32) THEN pii = inxm:
IF inxm < ne - 1 THEN inxm = inxm + 1: GOSUB colrnorm ELSE inxm =
1: GOSUB colrnorm
      IF Q$ = CHR$(27) THEN GOTO menu1
      IF Q$ = CHR$(13) THEN GOTO ingresoflech
      IF MID$(Q$,  2, 1)  = "K" THEN pii = inxm: IF inxm > 1 THEN
inxm = inxm - 1:  GOSUB  colrnorm  ELSE  inxm  =  ne  -  1: GOSUB
colrnorm
      IF MID$(Q$, 2, 1) = "t" THEN GOTO mover
      IF  MID$(Q$,  2,  1)  =  "s"  THEN  LOCATE  20, 2, 0: PRINT
CHR$(18): GOTO cm4
      pii = inxm
      IF Q$ = "R" OR Q$ = "r" THEN GOTO resett
      IF Q$ = "A" OR Q$ = "a" THEN inxm = 2: GOTO ingresof
      IF Q$ = "M" OR Q$ = "m" THEN GOTO memoria
      IF Q$ = "F" OR Q$ = "f" THEN GOTO fin
      IF Q$ = "S" OR Q$ = "s" THEN inxm = 5: GOTO ingresof
      IF Q$ = "E" OR Q$ = "e" THEN inxm = 6: GOTO ingresof
      IF Q$ = "T" OR Q$ = "t" THEN inxm = 7: GOTO ingresof
      IF Q$ = "D" OR Q$ = "d" THEN inxm = 8: GOTO ingresof
      IF Q$ = "I" OR Q$ = "i" THEN inxm = 9: GOTO ingresof
      GOTO ingreso1
ingresof:

       GOSUB colrnorm

ingresoflech:
      ON  inxm  GOTO  resett,  archivo,  memoria,  fin,  setear,
ejecutar, tabla, dosc, imprimir

archivo:
       BPC = -1: COLOR 7, 0
   LOCATE 22, 1, 0: PRINT STRING$(70, 255)
   LOCATE 25, 1, 0: PRINT STRING$(72, 255);
   menu$ = ">Drive  Path  Archivo "
   menu$(1) = "Elegir drive de trabajo "
   menu$(2) = "Determinar el directorio de trabajo "
   menu$(3) = "Hexadecimal, Análogo, Pserial "
   ne = 4: inxm = 1: ie = 22
   GOSUB vent: GOSUB colrnorm
entrada4:
      Q$ = INKEY$: IF Q$ = "" THEN GOTO entrada4
      IF Q$ = CHR$(27) THEN GOSUB refresco: GOTO menu1
      IF MID$(Q$,  2, 1)  = "M" OR Q$ = CHR$(32) THEN pii = inxm:
IF inxm < ne - 1 THEN inxm = inxm + 1: GOSUB colrnorm ELSE inxm =
1: GOSUB colrnorm
      IF Q$ = CHR$(13) THEN GOTO entradaflech4
      IF MID$(Q$,  2, 1)  = "K" THEN pii = inxm: IF inxm > 1 THEN
inxm = inxm - 1:  GOSUB  colrnorm  ELSE  inxm  =  ne  -  1: GOSUB
colrnorm
      IF MID$(Q$, 2, 1) = "t" THEN GOTO mover
```

```
      IF MID$(Q$, 2, 1) = "s" THEN LOCATE 20, 2, 0: PRINT
CHR$(18): GOTO cm4
      pii = inxm
      IF Q$ = "D" OR Q$ = "d" THEN inxm = 1: GOTO entradaf4
      IF Q$ = "P" OR Q$ = "p" THEN inxm = 2: GOTO entradaf4
      IF Q$ = "A" OR Q$ = "a" THEN inxm = 3: GOTO entradaf4
      GOTO entrada4

entradaf4:
       GOSUB colrnorm

entradaflech4:
      ON inxm GOTO drive, pahtt, archivo1
drive:
     CO1 = 15: CO2 = 0: LX = 2: L = 22: CO = 27: PAT = 2
     LOCATE 22, 1, 0: COLOR 7, 0: PRINT "INGRESE EL DRIVE DESEADO
:"; : GOSUB endatos: UNIDAD$ = YO$
     UNIDAD$ = MID$(UNIDAD$, 1, 1): UNIDAD$ = UNIDAD$ + ":"
     PAT = 0
195 OPEN UNIDAD$ + "x" FOR RANDOM AS #4
    CLOSE #4
    KILL UNIDAD$ + "x"
    LOCATE 22, 1: PRINT STRING$(40, 255): COLOR 7, 0
    GOTO entrada4
pahtt:
     CO1 = 15: CO2 = 0: LX = 20: L = 22: CO = 27: PAT = 2
     LOCATE 22,  1, 0: COLOR 7, 0: PRINT "INGRESE EL PATH DESEADO
:\"; : GOSUB endatos: PATT$ = YO$
     PATT$ = "\" + PATT$
    PAT = 0
200 OPEN PATT$ + "\X" FOR RANDOM AS #4
    CLOSE #4
    KILL PATT$ + "\X"
    PATT$ = PATT$ + "\"
    LOCATE 22, 1: PRINT STRING$(40, 255): COLOR 7, 0
    GOTO entrada4

archivo1:
       BPC = -1: COLOR 7, 0
   LOCATE 22, 1, 0: PRINT STRING$(70, 255)
   LOCATE 25, 1, 0: PRINT STRING$(72, 255);
   menu$ = ">Hexadecimal  Análogo  Pserial"
   menu$(1) = "Archivo a simularse "
   menu$(2) = "Archivo del conversor análogo/digital"
   menu$(3) = "Archivo del portico serial "
   ne = 4: inxm = 1: ie = 32
   GOSUB vent: GOSUB colrnorm
entrada6: :
      Q$ = INKEY$: IF Q$ = "" THEN GOTO entrada6
      IF Q$ = CHR$(27) THEN GOTO archivo
      IF MID$(Q$, 2, 1) = "M" OR Q$ = CHR$(32)  THEN pii  = inxm:
IF inxm < ne - 1 THEN inxm = inxm + 1: GOSUB colrnorm ELSE inxm =
1: GOSUB colrnorm
      IF Q$ = CHR$(13) THEN GOTO entradaflech6
```

```
      IF MID$(O$, 2, 1) = "K" THEN pii = inxm:  IF inxm  > 1 THEN
inxm  =  inxm  -  1:  GOSUB  colrnorm  ELSE  inxm = ne - 1: GOSUB
colrnorm
      IF MID$(O$, 2, 1) = "t" THEN GOTO mover
      IF MID$(O$, 2,  1)  =  "s"  THEN  LOCATE  20,  2,  0: PRINT
CHR$(18): GOTO cm4
      pii = inxm
      IF O$ = "H" OR O$ = "h" THEN inxm = 1: GOTO entradaf6
      IF O$ = "A" OR O$ = "a" THEN inxm = 2: GOTO entradaf6
      IF O$ = "P" OR O$ = "p" THEN inxm = 3: GOTO entradaf6
      GOTO entrada6

entradaf6:
       GOSUB colrnorm

entradaflech6:
      ON inxm GOTO archhex, analogo, pserial
archhex:
    GOSUB nombarch: LOCATE 25, 2, 0: BPC = -1: GOTO entrada6
analogo:
    LX = 13: L = 22: CO = 30
    LOCATE 22,  1: COLOR 7, 0: PRINT "NOMBRE DEL ARCHIVO ANALOGO:
"; : GOSUB endatos: ARC$ = YO$
    IF LEN(ARC$) <= 3 THEN ARC$ = ARC$ + ".prn"
    IF MID$(ARC$, LEN(ARC$) - 3, 4) <> ".prn" THEN ARC$  = ARC$ +
".prn"

     BA = 1
1000 OPEN path$ + ARC$ FOR INPUT AS #3
    LOCATE 22, 1: PRINT STRING$(70, 255); : COLOR 7, 0
     GOTO entrada6

pserial:
    LX = 13: L = 22: CO = 30
    LOCATE 22,  1: COLOR 7, 0: PRINT "NOMBRE DEL ARCHIVO PSERIAL:
"; : GOSUB endatos: arcs$ = YO$
    IF LEN(arcs$) <= 3 THEN arcs$ = arcs$ + ".prn"
    IF MID$(arcs$, LEN(arcs$) - 3,  4)  <>  ".prn"  THEN  arcs$ =
arcs$ + ".prn"
    arcs$ = path$ + arcs$
     PS = 1
1001   OPEN path$ + arcs$ FOR INPUT AS #4
       LOCATE 22, 1: PRINT STRING$(70, 255); : COLOR 7, 0
       GOTO entrada6

fin:
        CLOSE : COLOR 7, 0: CLS : LOCATE , , , 12, 13: END
ejecutar:
   COLOR 7, 0
   LOCATE 22, 1, 0: PRINT STRING$(70, 255)
   LOCATE 25, 1, 0: PRINT STRING$(72, 255);
   menu$ = ">Pasos  Continuo"
   menu$(1) = "Ejecutar el programa por pasos"
   menu$(2) = "Ejecutar en forma continua"
```

```
  ne = 3: inxm = 1: ie = 40
   GOSUB vent: GOSUB colrnorm
entrada8:
      Q$ = INKEY$: IF Q$ = "" THEN GOTO entrada8
      IF Q$ = CHR$(27) THEN GOSUB bvi: GOSUB cm2: GOTO menu1
      IF MID$(Q$, 2, 1) = "M" OR Q$ = CHR$(32) THEN pii = inxm:
IF inxm < ne - 1 THEN inxm = inxm + 1: GOSUB colrnorm ELSE inxm =
1: GOSUB colrnorm
      IF Q$ = CHR$(13) THEN GOTO entradaflech8
      IF MID$(Q$, 2, 1) = "K" THEN pii = inxm: IF inxm > 1 THEN
inxm = inxm - 1: GOSUB colrnorm ELSE inxm = ne - 1: GOSUB
colrnorm
      IF MID$(Q$, 2, 1) = "t" THEN GOTO mover
      IF MID$(Q$, 2, 1) = "s" THEN LOCATE 20, 2, 0: PRINT
CHR$(18): GOTO cm4
      pii = inxm
      IF Q$ = "P" OR Q$ = "p" THEN inxm = 1: GOTO entradaf8
      IF Q$ = "C" OR Q$ = "c" THEN inxm = 2: pasos = 0: GOTO
entradaf8
      GOTO entrada8

entradaf8:
       GOSUB colrnorm

entradaflech8:
      ON inxm GOTO porpasos, desen

porpasos:
      pasos = 1
      GOTO desen
setear:
   BPC = -1: COLOR 7, 0
   LOCATE 22, 1, 0: PRINT STRING$(70, 255)
   LOCATE 25, 1, 0: PRINT STRING$(72, 255);
   menu$ = ">Ingresar  Borrar  Corregir"
   menu$(1) = "Elegir puntos de parada"
   menu$(2) = "Eliminar todos los puntos de parada"
   menu$(3) = "Borrar un punto de parada"
   ne = 4: inxm = 1: ie = 36
   GOSUB vent: GOSUB colrnorm
entrada7:
      Q$ = INKEY$: IF Q$ = "" THEN GOTO entrada7
      IF Q$ = CHR$(27) THEN GOSUB bvi: GOSUB cm2: GOTO menu1
      IF MID$(Q$, 2, 1) = "M" OR Q$ = CHR$(32) THEN pii = inxm:
IF inxm < ne - 1 THEN inxm = inxm + 1: GOSUB colrnorm ELSE inxm =
1: GOSUB colrnorm
      IF Q$ = CHR$(13) THEN GOTO entradaflech7
      IF MID$(Q$, 2, 1) = "K" THEN pii = inxm: IF inxm > 1 THEN
inxm = inxm - 1: GOSUB colrnorm ELSE inxm = ne - 1: GOSUB
colrnorm
      IF MID$(Q$, 2, 1) = "t" THEN GOTO mover
      IF MID$(Q$, 2, 1) = "s" THEN LOCATE 20, 2, 0: PRINT
CHR$(18): GOTO cm4
      pii = inxm
```

```
      IF Q$ = "I" OR Q$ = "i" THEN inxm = 1: GOTO entradaf7
      IF Q$ = "B" OR Q$ = "b" THEN inxm = 2: GOTO entradaf7
      IF Q$ = "C" OR Q$ = "c" THEN inxm = 3: GOTO entradaf7
      GOTO entrada7

entradaf7:
       GOSUB colrnorm

entradaflech7:
      ON inxm GOTO breakp, brel, breakp1


tabla:
       BPC = -1: COLOR 7, 0
   LOCATE 22, 1, 0: PRINT STRING$(70, 255)
   LOCATE 25, 1, 0: PRINT STRING$(72, 255);
   menu$ = ">Recuperar  Archivar  Crear  Modificar  Borrar"
   menu$(1) = "Recuperar una tabla archivada"
   menu$(2) = "Grabar la tabla creada"
   menu$(3) = "Generar una tabla o incrementarla"
   menu$(4) = "Modificar la tabla en memoria"
   menu$(5) = "Borrar la tabla en memoria"
   ne = 6: inxm = 1: ie = 26
   GOSUB vent: GOSUB colrnorm
entrada5:
      Q$ = INKEY$: IF Q$ = "" THEN GOTO entrada5
      IF Q$ = CHR$(27) THEN GOSUB refresco: GOTO menu1
      IF MID$(Q$, 2, 1) = "M" OR Q$ = CHR$(32)  THEN pii  = inxm:
IF inxm < ne - 1 THEN inxm = inxm + 1: GOSUB colrnorm ELSE inxm =
1: GOSUB colrnorm
      IF Q$ = CHR$(13) THEN GOTO entradaflech5
      IF MID$(Q$, 2, 1) = "K" THEN pii = inxm:  IF inxm  > 1 THEN
inxm  =  inxm  -  1:  GOSUB  colrnorm  ELSE  inxm = ne - 1: GOSUB
colrnorm
      IF MID$(Q$, 2, 1) = "t" THEN GOTO mover
      IF MID$(Q$, 2,  1)  =  "s"  THEN  LOCATE  20,  2,  0: PRINT
CHR$(18): GOTO cm4
      pii = inxm
      IF Q$ = "R" OR Q$ = "r" THEN inxm = 1: GOTO entradaf5
      IF Q$ = "A" OR Q$ = "a" THEN inxm = 2: GOTO entradaf5
      IF Q$ = "C" OR Q$ = "c" THEN inxm = 3: GOTO entradaf5
      IF Q$ = "M" OR Q$ = "m" THEN inxm = 4: GOTO entradaf5
      IF Q$ = "B" OR Q$ = "b" THEN inxm = 5: GOTO entradaf5
      GOTO entrada5

entradaf5:
       GOSUB colrnorm

entradaflech5:
      ON  inxm  GOTO  recuperart,  archivart, creart, modificart,
borrart
recuperart:
    texit = 2
    GOSUB tb8
```

```
441 OPEN path$ + ARC$ FOR INPUT AS #2
    FOR NLC = 0 TO 255
    IF EOF(2) THEN CLOSE #2: NLC = NLC - 1: GOSUB PRINTABLA:  i =
NLC + 1: texit = 0: GOTO entrada5
    INPUT #2, V$(NLC), Y$(NLC), LOC$(NLC)
    NEXT

archivart:
    texit = 2
    GOSUB tb8
482 OPEN path$ + ARC$ FOR OUTPUT AS #2
    Q$ = CHR$(34)
    FOR i = 0 TO NLC
    PRINT #2, Q$; V$(i); Q$; Q$; Y$(i); Q$; Q$; LOC$(i); Q$
    NEXT: CLOSE #2
    NLC = NLC - 1
    texit = 0
    GOTO entrada5

creart:
    GOSUB PRINTABLA
TB3:
    IF i - 1 > NLC THEN NLC = i - 1
    LX = 10: L = 22: CO = 10: HX = 0: texit = 2
    COLOR 7, 0: LOCATE 22, 1, 0: PRINT STRING$(80, " ")
    LOCATE 22, 1, 0: PRINT "VARIABLE: "
    GOSUB endatos: V$(i) = Y0$
    IF MID$(V$(i), 2, 1) = "X" THEN F = 1: Y$(i) = MID$(V$(i), 1,
1) + "L": Y$(i + 1) = MID$(V$(i), 1, 1) + "H" ELSE Y$(i) = V$(i)
    LX = 4: L = 22: CO = 65: HX = 2: texit = 2
tlocal:
    COLOR 7, 0: LOCATE 22, 45, 0: PRINT "LOCALIDAD EN HEX : ";
    GOSUB endatos: LOC$(i) = Y0$
    X$ = Y0$: GOSUB hexadecimal
    IF SUM > 255 THEN BEEP: GOTO tlocal
    IF F <> 1 THEN GOTO tb31
    X$ = LOC$(i): GOSUB hexadecimal: LOC$(i + 1) = HEX$(SUM + 1)
    'LOCATE 6 + LN, 3, 0: PRINT USING "##"; I;
tb31:
    B = 4 - LEN(LOC$(i))
    LOC$(i) = STRING$(B, "0") + LOC$(i)
    LOCATE 6  + LN,  6, 0:  PRINT USING  "\    \"; Y$(i); TAB(21);
LOC$(i)
    IF LN > 12 THEN LN = 0 ELSE LN = LN + 1
    IF F = 1 THEN i = i + 2: F = 0: GOTO TB3
    i = i + 1
    F = 0
    GOTO TB3

modificart:
    GOSUB PRINTABLA
    LX = 2: L = 22: CO = 26: texit = 2: ' HX = 0
    LOCATE 22, 1, 0: PRINT STRING$(80, " ")
    LOCATE  22,  1,  0:  PRINT "LOCALIDAD A MODIFICARSE: ": GOSUB
```

```
endatos: X$ = YO$
    B = 4 - LEN(X$)
    LO$ = STRING$(B, "0") + X$
    FOR J1 = NLC + 1 TO 0 STEP -1
      IF LO$ = LOC$(J1) THEN GOTO MODT
      NEXT J1
      GOTO modificart
MODT:
    LX = 10: L = 22: CO = 10: HX = 0: texit = 2
    COLOR 7, 0: LOCATE 22, 1, 0: PRINT STRING$(80, " ")
    LOCATE 22, 1, 0: PRINT "VARIABLE: "
    GOSUB endatos: V$(J1) = YO$
    IF MID$(V$(J1), 2, 1) = "X" THEN Y$(J1) = MID$(V$(J1), 1, 1)
+ "L": Y$(J1 + 1) = MID$(V$(J1), 1, 1) + "H" ELSE Y$(J1) = V$(J1)
    GOTO modificart

borrart:
    GOSUB PRINTABLA
    LX = 2: L = 22: CO = 26: texit = 2: ' HX = 0
    LOCATE 22, 1, 0: PRINT STRING$(80, " ")
    LOCATE 22, 1, 0: PRINT "TODA LA TABLA S/N : ": GOSUB
endatos: X$ = YO$
    IF X$ = "S" THEN GOTO borr
bor:
    LOCATE 22, 1, 0: PRINT "LOCALIDAD A BORRARSE: ": GOSUB
endatos: X$ = YO$
    B = 4 - LEN(X$)
    LO$ = STRING$(B, "0") + X$
    FOR J1 = NLC + 1 TO 0 STEP -1
      IF LO$ = LOC$(J1) THEN GOTO BORR1
      NEXT JI
     GOTO bor
BORR1:
     LOC$(J1) = "": V$(J1) = "": Y$(JI) = ""
     FOR J2 = J1 TO NLC - 1
     LOC$(J2) = LOC$(J2 + 1): V$(J2) = V$(J2 + 1): Y$(J2) = Y$(J2
+ 1)
     NEXT
     NLC = NLC - 1
     i = i - 1
     GOSUB PRINTABLA
     GOTO bor
borr:
           FOR J2 = 0 TO NLC
           LOC$(J2) = "": V$(J2) = "": Y$(J2) = ""
           NEXT
           i = 0: NLC = 0: j = 0
           LOCATE 22, I, Q: PRINT STRING$(78, 255): COLOR 7, 0
           GOSUB PRINTABLA
           texit = 0
           GOTO entrada5


dosc:
```

```
        shel = 1
        CLS
        SHELL
        GOSUB refresco
        GOTO menu1
imprimir:
      CO = 26: L = 20: LX = 4: CO1 = 15: CO2 = 0: HX = 2: ME = 2
      C = 0
      LOCATE 20, 2, 0:  PRINT "MEMORIA  INICIAL (HEX)=";  : GOSUB
endatos: X$  = YO$:  GOSUB hexadecimal: A = SUM: LOCATE 20, 2, 0:
PRINT STRING$(28, " ")
      COLOR 7, 0
impe:
      LOCATE 20, 2, 0: PRINT "MEMORIA   FINAL   (HEX)=";  : GOSUB
endatos: X$  = YO$:  GOSUB hexadecimal: C = SUM: LOCATE 20, 2, 0:
PRINT STRING$(28, " ")
      COLOR 7, 0
      IF C < A THEN BEEP: GOTO impe
impr:
      LOCATE 22, 1: INPUT "DESEA EN PANTALLA (P)  O IMPRESORA (I)
:", R$
      IF R$ = "P" OR R$ = "p" THEN salid$ = "scrn:"
      IF R$ = "I" OR R$ = "i" THEN salid$ = "lpt1:"
      LOCATE 22,  1: PRINT "PRESIONE ENTER PARA CONTINUAR :
                    "
imprs: R$ = INKEY$: IF R$ = "" THEN GOTO imprs
      IF R$ = CHR$(13) THEN GOTO imprimir1
      IF R$ = CHR$(27) THEN GOTO ingreso1 ELSE GOTO imprs
imprimir1:
      IF salid$ = "scrn:" THEN CLS
      OPEN salid$ FOR OUTPUT AS #5

      PRINT #5, TAB(20); "SIMULADOR DEL MICROCONTROLADOR 8096"
      PRINT #5, TAB(20); "==================================="
      PRINT #5, "ARCHIVO SIMULADO: "; ARCH$
      FOR i = 0 TO NL
      PRINT #5, TAB(0); DASM$(i)
      NEXT
      PRINT #5, CHR$(13)
      PRINT #5, TAB(20); "REGISTROS ESPECIALES DE MEMORIA"
      PRINT #5, TAB(20); "================================="
      PRINT #5, CHR$(13)
      FOR I22 = 0 TO 1
      B = 4 - LEN(HEX$(I22 * 16))
      LOP$(I22) = STRING$(B, "0") + HEX$(I22 * 16)
      NEXT
      FOR I12 = 0 TO 16 STEP 16
      I11 = 8
      PRINT #5, TAB(0); LOP$(I12 / 16);
      FOR I1 = 0 TO 15
      IF LEN(HEX$(M(I1 + I12))) < 2 THEN P1$ = "0"  + HEX$(M(I1 +
I12)) ELSE P1$ = HEX$(M(I1 + I12))
      X$ = P1$: GOSUB hexadecimal:
      IF SUM < 32 THEN P2$(I1) = ".": GOTO cmp1321
```

```
      P2$(I1) = CHR$(SUM)
cmp1321:
      PRINT #5, TAB(I11); P1$;
      I11 = I11 + 3
      NEXT
      PRINT #5,  TAB(56); :  FOR i = 0 TO 15: PRINT #5, P2$(i); :
NEXT
      NEXT
      PRINT #5, CHR$(13)
      PRINT #5, TAB(20); "CONTENIDO DE LOCALIDADES DE MEMORIA"
      PRINT #5, TAB(20); "================================="
impfi:
      FOR I22 = 0 TO 15
      IF A > 65430 THEN A = 65430
      B = 4 - LEN(HEX$(A + I22 * 16))
      LOP$(I22) = STRING$(B, "0") + HEX$(A + I22 * 16)
      NEXT
      FOR I12 = 0 TO 240 STEP 16
      I11 = 8
      PRINT #5, TAB(0); LOP$(I12 / 16);
      FOR I1 = 0 TO 15
      IF A + I1 + I12 > 32766 THEN GOTO CMP31
      IF LEN(HEX$(M(A + I1 + I12))) < 2 THEN P1$ = "0" + HEX$(M(A
+ I1 + I12)) ELSE P1$ = HEX$(M(A + I1 + I12))
      GOTO CMP32
CMP31:
      A = A - 32767
      IF  LEN(HEX$(M1(A  +  I1  +  I12)))  <  2  THEN P1$ = "0" +
HEX$(M1(A + I1 + I12)) ELSE P1$ = HEX$(M1(A + I1 + I12))
      A = A + 32767
CMP32:
      X$ = P1$: GOSUB hexadecimal:
      IF SUM < 32 THEN P2$(I1) = ".": GOTO cmp321
      P2$(I1) = CHR$(SUM)
cmp321:
      PRINT #5, TAB(I11); P1$;
      I11 = I11 + 3
      NEXT
      PRINT #5, TAB(56); : FOR i = 0 TO 15:  PRINT #5,  P2$(i); :
NEXT
      NEXT
      IF (C - A) > 256 THEN A = A + 256: GOTO impfi
      PRINT #5, CHR$(13)
      PRINT #5, "ESTADOS DE TIEMPO: "; EST
      shel = I
      CLOSE #5
      IF salid$  = "lpt1:"  THEN LOCATE  22, 1: PRINT STRING$(70,
255); : GOTO ingreso1
      LOCATE  25,  1:  PRINT   "presione  cualquier   tecla  para
continuar.....";
      D$ = INPUT$(1)
      GOSUB refresco
      GOTO menu1
resett:
```

```
   BPC = -1: COLOR 7, 0
   LOCATE 22, 1, 0: PRINT STRING$(70, 255)
   LOCATE 25, 1, 0: PRINT STRING$(72, 255);
   menu$ = ">Cpu  Memoria  Banderas  Tabla "
   menu$(1) = "Reset del cpu"
   menu$(2) = "Borrar la memoria del microcontrolador"
   menu$(3) = "Borrar las banderas"
   menu$(4) = "Borrar la tabla"
   ne = 5: inxm = 1: ie = 10
   GOSUB vent: GOSUB colrnorm
entrada2:
      Q$ = INKEY$: IF Q$ = "" THEN GOTO entrada2:
      IF Q$ = CHR$(27) THEN GOTO menu1
      IF MID$(Q$,  2, 1)  = "M" OR Q$ = CHR$(32) THEN pii = inxm:
IF inxm < ne - 1 THEN inxm = inxm + 1: GOSUB colrnorm ELSE inxm =
1: GOSUB colrnorm
      IF Q$ = CHR$(13) THEN GOTO entradaflech2
      IF MID$(Q$,  2, 1)  = "K" THEN pii = inxm: IF inxm > 1 THEN
inxm = inxm - 1:  GOSUB  colrnorm  ELSE  inxm  =  ne  -  1: GOSUB
colrnorm
      IF MID$(Q$, 2, 1) = "t" THEN GOTO mover
      IF  MID$(Q$,  2,  1)  =  "s"  THEN  LOCATE  20, 2, 0: PRINT
CHR$(18): GOTO cm4
      pii = inxm
      IF Q$ = "C" OR Q$ = "c" THEN inxm = 1: GOTO entradaf2
      IF Q$ = "M" OR Q$ = "m" THEN inxm = 2: GOTO entradaf2
      IF Q$ = "B" OR Q$ = "b" THEN inxm = 3: GOTO entradaf2
      IF Q$ = "T" OR Q$ = "t" THEN inxm = 4: GOTO entradaf2
      GOTO entrada2

entradaf2:
       GOSUB colrnorm

entradaflech2:
      ON inxm GOTO resetcpu, resetmemoria, banderas, resetabla
banderas:
         FOR I3 = 9 TO 16: PSW(I3) = 0: NEXT:
      COLOR 15, 0
      FOR I1 = 1 TO 16: LOCATE 4 + I1, 41
      PRINT PSW(I1): NEXT
      COLOR 7, 0

         GOTO entrada2:

memoria:
   COLOR 7, 0
   LOCATE 22, 1, 0: PRINT STRING$(70, 255)
   LOCATE 25, 1, 0: PRINT STRING$(70, 255);
   menu$  =  ">Contenido     Modificar     Encontra     Borrar(B)
Transferir(B)  Llenar(B)"
   menu$(1) = "Ver el contenido de memoria"
   menu$(2) = "Modificar un valor de una localidad de memoria "
   menu$(3) = "Encontrar un valor determinado"
   menu$(4) = "Borrar un bloque de la memoria"
```

```
   menu$(5) =  "Transferir valores  de un bloque a otro bloque de
memoria"
   menu$(6) = "Llenar un bloque de memoria con un valor"
   ne = 7: inxm = 1: ie = 15
   GOSUB vent: GOSUB colrnorm

entrada3:
   LOCATE 19, 2, 0: PRINT CHR$(255)
   O$ = INKEY$: IF O$ = "" THEN GOTO entrada3
   IF O$ = CHR$(27) THEN GOTO menu1
   IF MID$(O$, 2, 1) = "M" OR O$ = CHR$(32)  THEN pii  = inxm: IF
inxm < ne - 1 THEN inxm = inxm + 1: GOSUB colrnorm ELSE inxm = 1:
GOSUB colrnorm
   IF O$ = CHR$(13) THEN GOTO entradaflech3
   IF MID$(O$, 2, 1) = "K" THEN pii = inxm: IF inxm > 1 THEN inxm
= inxm - 1: GOSUB colrnorm ELSE inxm = ne - 1: GOSUB colrnorm
   IF MID$(O$, 2, 1) = "t" THEN GOTO mover
   IF MID$(O$,  2, 1) = "s" THEN LOCATE 20, 2, 0: PRINT CHR$(18):
GOTO cm4

      pii = inxm
      IF O$ = "C" OR O$ = "c" THEN inxm = 1: GOTO entradaf3
      IF O$ = "M" OR O$ = "m" THEN inxm = 2: GOTO entradaf3
      IF O$ = "E" OR O$ = "e" THEN inxm = 3: GOTO entradaf3
      IF O$ = "B" OR O$ = "b" THEN inxm = 4: GOTO entradaf3
      IF O$ = "T" OR O$ = "t" THEN inxm = 5: GOTO entradaf3
      IF O$ = "L" OR O$ = "l" THEN inxm = 6: GOTO entradaf3
      IF O$ = CHR$(27)  THEN LOCATE  22, 1,  0: PRINT STRING$(70,
255):  GOSUB  cm2:  LOCATE  25,  2,  0:  GOTO  ingreso1 ELSE GOTO
entrada3

entradaf3:

      GOSUB colrnorm

entradaflech3:
       ON   inxm   GOTO    contenido,    modmemoria,    encontrar,
borrarbloques, transbloques, llenar
contenido:
        IF BPC = -1 THEN BPC = -1: GOTO conmemoria ELSE BPC = 1:
GOTO conmemoria

encontrar:
         IF BPC = -1 THEN BPC = -1: GOTO buscmem  ELSE BPC  = 1:
GOTO buscmem

     '*****************************  BORRA VENTANA IZQUIERDA
bvi:
     FOR  K  =  2  TO  20  -  K1:  LOCATE  K  +  K1, 2, 0: PRINT
STRING$(30, " "): NEXT: K1 = 0: RETURN

manerror:
     '*****************************************************MANEJO
DE ERROR
```

```
      IF ERR = 53 AND ERL = 310 THEN LOCATE 24, 60,  0: COLOR 16,
6: PRINT  "NO EXISTE "; : s$ = INPUT$(1): LOCATE 24, 60, 0: COLOR
7, 0: PRINT "           "; : ARCH$ = "": YO$ = "": RESUME nombarch
      IF ERR = 53 AND ERL = 441 THEN LOCATE 24, 60,  0: COLOR 16,
6: PRINT  "NO EXISTE     "; :  s$ =  INPUT$(1): LOCATE 24, 60, 0:
COLOR 7, 0: PRINT "           "; : RESUME tb8
      IF ERR = 53 AND ERL = 482 THEN LOCATE 24, 60,  0: COLOR 16,
6: PRINT  "NO EXISTE     "; :  s$ =  INPUT$(1): LOCATE 24, 60, 0:
COLOR 7, 0: PRINT "           "; : RESUME tb8
      IF ERR = 53 AND ERL = 1000 THEN LOCATE 24, 60, 0: COLOR 16,
6: PRINT  "NO EXISTE "; : s$ = INPUT$(1): LOCATE 24, 60, 0: COLOR
7, 0: PRINT "           "; : RESUME analogo
      IF ERR = 53 AND ERL = 1001 THEN LOCATE 24, 60, 0: COLOR 16,
6: PRINT  "NO EXISTE "; : s$ = INPUT$(1): LOCATE 24, 60, 0: COLOR
7, 0: PRINT "           "; : RESUME pserial
      IF ERR = 76 AND ERL = 200 THEN LOCATE 24, 60,  0: COLOR 16,
6: PRINT  "NO EXISTE  "; : PATT$ = "": s$ = INPUT$(1): LOCATE 24,
60, 0: COLOR 7, 0: PRINT "           "; : RESUME pahtt
      IF ERR = 68 OR ERR = 71 AND ERL = 195  THEN LOCATE  24, 60,
0:  COLOR  16,  6:  PRINT  "NO  EXISTE  ";  :  UNIDAD$ = "": s$ =
INPUT$(1): LOCATE 24, 60, 0: COLOR 7, 0: PRINT "              "; :
RESUME drive
      'IF ERR = 71 AND ERL = 195 THEN LOCATE 24, 60, 0: COLOR 16,
6: PRINT "NO LISTO. "; : s$ = INPUT$(1): LOCATE 24, 60,  0: COLOR
7, 0: PRINT "           "; : RESUME drive
      IF ERR = 9 THEN RESUME NEXT
      IF ERR  = 5  THEN CLOSE  #1: LOCATE 24, 60, 0: COLOR 16, 6:
PRINT "NO EJECUTABLE"; : RESUME ingresol
      IF ERR = 64 AND ERL = 310 THEN LOCATE 24, 60,  0: COLOR 16,
6: PRINT  "NO EXISTE     "; :  s$ =  INPUT$(1): LOCATE 24, 60, 0:
COLOR 7, 0: PRINT "           "; : RESUME nombarch
RESUME NEXT
conmemoria:
      '*******************************      rutina  (contenido de
memoria)
      CO = 24: L = 20: LX = 4: CO1 = 15: CO2 = 0: HX = 2: ME = 1
      COLOR 7,  0: LOCATE  2, 6, 0: PRINT "LOCALIDADES DE MEMORIA
"
      LOCATE 20, 2, 0: PRINT "  DIRECCION EN  HEXADC= ";  : GOSUB
endatos: X$  = YO$:  GOSUB hexadecimal: A = SUM: LOCATE 20, 2, 0:
PRINT STRING$(28, " ")
      HX = 0:
cm1:
      LOCATE 20, 2, 0: PRINT CHR$(18)
      GOSUB cm2
      GOTO cm4
cm2:
      COLOR 7, 0: LOCATE 2, 6, 0: PRINT  " REGISTROS   DE MEMORIA
": LOCATE 2, 50, 0: PRINT "EJECUCION " + ARCH$
cm3:
      COLOR 15, 0
      FOR I22 = 0 TO 15
      IF A > 65408 THEN A = 65408
      B = 4 - LEN(HEX$(A + I22 * 8))
```

```
      LOCATE 4 + I22, 3
      PRINT STRING$(8, "0"); HEX$(A + I22 * 8);
      NEXT
      FOR I12 = 0 TO 120 STEP 8
      I11 = 8
      FOR I1 = 0 TO 7
      IF A + I1 + I12 > 32766 THEN GOTO CM31
      IF LEN(HEX$(M(A + I1 + I12))) < 2 THEN P1$ = "0" + HEX$(M(A
+ I1 + I12)) ELSE P1$ = HEX$(M(A + I1 + I12))
      GOTO CM32
CM31:
      IF A + I1 + I12 > 65533 THEN GOTO CM311
      A = A - 32767
      IF LEN(HEX$(M1(A + I1 + I12))) < 2 THEN P1$ = "0" +
HEX$(M1(A + I1 + I12)) ELSE P1$ = HEX$(M1(A + I1 + I12))
      A = A + 32767
      GOTO CM32
CM311:
      A = A - 65534
      IF LEN(HEX$(F(A + I1 + I12))) < 2 THEN P1$ = "0" + HEX$(F(A
+ I1 + I12)) ELSE P1$ = HEX$(F(A + I1 + I12))
      A = A + 65534

CM32:
      LOCATE 4 + I12 / 8, I11, 0
      COLOR 7, 0: PRINT P1$;
      I11 = I11 + 3
      NEXT: NEXT
     RETURN
cm4:
      O1$ = INKEY$: IF O1$ = "" THEN GOTO cm4
      IF MID$(O$, 2, 1) = "K" THEN GOTO entrada3
      IF MID$(O$, 2, 1) = "M" THEN GOTO entrada3
      IF MID$(O1$, 2, 1) = "t" THEN LOCATE 20, 2, 0: PRINT
CHR$(255): GOTO mover
      IF MID$(O1$, 2, 1) = "P" THEN A = A + 104: GOSUB cm3
      IF MID$(O1$, 2, 1) = "H" THEN IF A <= 17 THEN A1 = 65430:
GOSUB cm3 ELSE A = A - 104: GOSUB cm3
      IF O1$ = CHR$(27) THEN
      LOCATE 20, 2, 0: PRINT CHR$(255)
      IF ie = 0 THEN HX = 0: ME = 0: LOCATE 20, 2, 0: PRINT
STRING$(25, " "): GOTO ingreso1
      IF ie = 8 THEN HX = 0: ME = 0: LOCATE 20, 2, 0: PRINT
STRING$(25, " "): GOTO entrada2:
      LOCATE 20, 2, 0: PRINT STRING$(25, " ")
      HX = 0: ME = 0
      GOTO entrada3
      ELSE
      GOTO cm4
      END IF

:
endatos:
      '**************************************SUBRUTINA DE ENTRADA
```

```
DE DATOS
      CO1 = 15: CO2 = 0
enblanqueo:
      XO$ = STRING$(LX, " ")
      P = 0: LX1 = LX
      LOCATE L, CO: COLOR CO1, CO2: PRINT XO$: LOCATE L, CO, 1
eningreso:
      X$ = INKEY$: IF X$ = "" THEN GOTO eningreso
      X$ = UCASE$(X$)
      IF X$ = CHR$(27) AND texit = 2 THEN
      texit = 0
      LOCATE 22, 1, 0: PRINT STRING$(78, 255): COLOR 7, 0
      RETURN entrada5
      ELSE
      END IF
      IF X$  = CHR$(27)  AND ME = 1 THEN PAT = 0: HX = 0: ME = 0:
COLOR 7, 0: LOCATE 20,  2,  0:  PRINT  STRING$(26,  "  "): RETURN
entrada3
      IF X$  = CHR$(27) AND ie = 0 THEN PAT = 0: HX = 0: COLOR 7,
0: LOCATE  20, 2,  0: PRINT  STRING$(26, "  "): LOCATE  22, 1, 0:
PRINT STRING$(30,  " "): LOCATE 22, 1, 0: PRINT STRING$(70, " "):
RETURN ingresol
      IF X$ = CHR$(27) AND ie = 8 THEN PAT = 0: HX =  0: COLOR 7,
0: LOCATE 22, 1, 0: PRINT STRING$(70, " "): RETURN entrada2:
      IF X$ = CHR$(27) AND ie = 32 THEN ARCH$ = "": PAT = 0: HX =
0: COLOR 7, 0: LOCATE 22,  1, 0:  PRINT STRING$(70,  " "): RETURN
entrada6:
      IF X$  = CHR$(27)  AND BPEXIT  = 1 THEN HX = 0: BPEXIT = 0:
COLOR 7, 0: LOCATE 20, 2, 0: PRINT  STRING$(29, "  "): LOCATE 22,
1, 0: PRINT STRING$(70, " "): RETURN entrada7
      IF X$ = CHR$(27) AND ie = 22 THEN PAT = 0: HX = 0: COLOR 7,
0: LOCATE 22, I, 0: PRINT STRING$(70, " "): RETURN entrada4:
      IF LEN(X$) = 2  THEN  X  =  -ASC(RIGHT$(X$,  1))  ELSE  X =
ASC(X$)
      IF LX <> 1 THEN GOTO salidaI
      IF X  = 78  OR X = 83 OR X = 13 THEN GOTO salida1 ELSE GOTO
eninqreso
salida1:
      IF P = 0 AND X = 13 AND LX = 1  THEN YO$  = LEFT$(XO$,  P +
1): LOCATE , , 0: RETURN
      IF P = 0 AND X = I3 THEN GOTO eningreso
      IF X = 46 OR X = 45 AND HX = 0 THEN GOTO en4
      IF X = -75 OR X = 8 THEN GOTO E1 ELSE GOTO E2
E1:
      IF P  <= 0  THEN P  = 0:  LOCATE L, CO + P, 1: PRINT " "; :
GOTO enblanqueo
      X = 255: P = P - 1
      LOCATE L, CO + P, 1:  PRINT CHR$(X); CHR$(X);
      MID$(XO$, P + 1, 1) = CHR$(X)
      LOCATE L, CO + P, 1
      GOTO eningreso
E2:
      IF HX = 2 THEN GOTO ennumeros ELSE GOTO en1     'HEXADECIMAL
ennumeros:
```

```
      IF X > 47 AND X < 58 THEN GOTO en4
      IF X > 64 AND X < 71 THEN GOTO en4 ELSE GOTO salida2
en1:
      IF PAT = 2 THEN GOTO en2 ELSE GOTO en3
en2:
      IF P = 1 AND X = 58 THEN GOTO en4
      IF X = 92 THEN GOTO en4
en3:
      IF X >= 65 AND X < 91 THEN GOTO en4
      IF X > 47 AND X < 58 THEN GOTO en4
salida2:
      IF P <= LX AND X = 13 THEN YO$ = LEFT$(XO$, P): LOCATE  , ,
0: RETURN
      GOTO eningreso
en4:
      IF P >= LX THEN GOTO eningreso
      LOCATE L, CO + P, 1: PRINT CHR$(X);
      MID$(XO$, P + 1, 1) = CHR$(X)
      IF LX <> I THEN P = P + 1
      GOTO eningreso


resetcpu:
      '*************************** RESET DE LA CPU
      PC% = 8320
      FOR i = 1 TO 16: PSW(i) = 0: NEXT
      M(15) = 255                              'portico 1
      M(16) = 193                              'portico 2
      M(17) =  0                                    'portico serial
control
      M(8) = 0                                        'mascara de
interrupcion
      M(10) = 0: M(11) = 0                     'timer 1
      M(13) = 0: M(12) = 0                     'timer 2
      M(21) = 0: M(22) = 0                     'Ios0 Ios1
      ESTADOS =  0                                'BANDERA PORTICO
SERIAL
      BPS = 0
      BA = 0                                       'bandera analogo
conversion
      IM = 0
      LOCATE 20, 2, 0: PRINT STRING$(28, 255): LOCATE 25, 1, 0
      LOCATE 22, 1, 0: PRINT STRING$(30, 255): LOCATE 25, 1, 0
      BPC = 0
      CLOSE #1
      GOTO desen

resetabla:
      '*************************** RESET DE TABLA
      ERASE LOC$, V$, Y$
      DIM LOC$(113), V$(113), Y$(113)
      RE = 0: NLC = 0:  ESTADOS = 0: tabla = 0
       LOCATE 22, 1, 0: PRINT "Reset tabla....."
         FOR i = 1 TO 100: NEXT
```

```
          LOCATE 22, 1, 0: PRINT "                    "

        IF ie = 26 THEN GOTO entrada5
        GOTO entrada2:

resetmemoria:
      '*************************RESET DE MEMORIA
      'QUE ES RESET DE MEMORIA
      ERASE M, M1, DASM$
      REDIM M(32766) AS INTEGER
      REDIM M1(32766) AS INTEGER
      REDIM DASM$(255)
      GOSUB cm2
      ARCH$ = ""

      GOTO entrada2:

desen:
      '************************** DESENSAMBLADOR
      IF ARCH$  = ""  THEN LOCATE  24, 1: PRINT "Elija primero el
nombre del archivo"; : GOTO ingreso1                          'GOSUB
nombrearch
      LOCATE 24,  1: PRINT "Ejecutando el programa
 ";
desen1:
      IF BPC = -1 OR BPC = 0 THEN GOTO desen2 ELSE CHAIN "TE"
desen2:
      A = 8320
      LOCATE 20, 48, 0: PRINT "Desensamblando.....": CHAIN "TD"

transbloques:
      '************************ TRANFERENCIA DE BLOQUES
      BPC = -1
      L = 20: CO = 23: LX = 4: HX = 2: ME = 1
      LOCATE 20, 2, 0:  PRINT "DIRECCION  INICIAL(H)= ";  : GOSUB
endatos: X$  = YO$: GOSUB hexadecimal: AA = SUM: LOCATE 20, 2, 0:
PRINT STRING$(29, " ")
      IF A > AA OR AA > A + 127 THEN A = AA
      GOSUB cm2
tra:
      LOCATE 20,  2, 0:  PRINT "DIRECCION  FINAL (H)=  "; : GOSUB
endatos: X$  = YO$:  GOSUB hexadecimal: B = SUM: LOCATE 20, 2, 0:
PRINT STRING$(29, " ")
      IF B < AA THEN BEEP: GOTO tra
      LOCATE 20, 2, 0:  PRINT "DIRECCION  DESEADA(H)= ";  : GOSUB
endatos: X$  = YO$:  GOSUB hexadecimal: C = SUM: LOCATE 20, 2, 0:
PRINT STRING$(29, " ")
      HX = 0
      IF C < AA THEN GOTO TRA1
      FOR T = (B - AA) TO 0 STEP -1
      IF (C + T) > 65533 THEN GOTO TRAF
      IF (AA + T) > 65533 THEN GOTO TRAF1
      IF (C + T) >= 32767 THEN IF (AA + T) >= 32767 THEN M1(C + T
- 32767)  = M1(AA  + T  - 32767):  M1(AA +  T -  32767) = 0: GOTO
```

```
TRANS1 ELSE M1(C + T - 32767) = M(AA +  T): M(AA  + T)  = 0: GOTO
TRANS1
      IF (AA  + T)  >= 32767  THEN M(C + T) = M1(AA + T - 32767):
M1(AA + T - 32767) = 0: GOTO TRANS1 ELSE  M(C +  T) =  M(AA + T):
M(AA + T) = 0: GOTO TRANS1
TRAF:
      IF (AA  + T)  > 65533  THEN F(C  + T  - 65534) = F(AA + T -
65534): F(AA + T - 65534) = 0: GOTO TRANS1
      IF (AA + T) > 32766 THEN F(C  + T  - 65534)  = M1(AA  + T -
32767): M1(AA + T - 32767) = 0: GOTO TRANS1
      F(C + T - 65534) = M(AA + T): M(AA + T) = 0: GOTO TRANS1
TRAF1:
     IF (C  + T)  >= 32767  THEN M1(C  + T  - 32767) = F(AA + T -
65534): F(AA + T - 65534) = 0: GOTO TRANS1
     M(C + T) = F(AA + T - 65534): F(AA  + T  - 65534)  = 0: GOTO
TRANS1
TRANS1:
      NEXT

trans2:
      IF A > C OR C > A + 127 THEN A = C
      GOSUB cm2
      GOTO transbloques
TRA1:
      FOR T = 0 TO (B - AA)
      IF (C + T) > 65533 THEN GOTO TRAFF
      IF (AA + T) > 65533 THEN GOTO TRAFF1
      IF (C + T) >= 32767 THEN IF (AA + T) >= 32767 THEN M1(C + T
- 32767) = M1(AA  + T  - 32767):  M1(AA +  T -  32767) =  0: GOTO
TRANS11 ELSE  M1(C +  T - 32767) = M(AA + T): M(AA + T) = 0: GOTO
TRANS11
      IF (AA + T) >= 32767 THEN M(C + T)  = M1(AA  + T  - 32767):
M1(AA +  T -  32767) = 0: GOTO TRANS11 ELSE M(C + T) = M(AA + T):
M(AA + T) = 0: GOTO TRANS11
TRAFF:
      IF (AA + T) > 65533  THEN F(C  + T  - 65534)  = F(AA  + T -
65534): F(AA + T - 65534) = 0: GOTO TRANS11
      IF (AA  + T)  > 32766  THEN F(C  + T - 65534) = M1(AA + T -
32767): M1(AA + T - 32767) = 0: GOTO TRANS11
      F(C + T - 65534) = M(AA + T): M(AA + T) = 0: GOTO TRANS11
TRAFF1:
     IF (C + T) >= 32767  THEN M1(C  + T  - 32767)  = F(AA  + T -
65534): F(AA + T - 65534) = 0: GOTO TRANS11
     M(C +  T) =  F(AA +  T - 65534): F(AA + T - 65534) = 0: GOTO
TRANS11

TRANS11:
      NEXT
      GOTO trans2

borrarbloques:
      '************************** BORRADO DE BLOQUES
      BPC = -1
      L = 20: CO = 23: LX = 4: HX = 2: ME = 1
```

```
      LOCATE 20, 2, 0: PRINT  "DIRECCION  INCIAL(H)=  ";  : GOSUB
endatos: X$  = YO$: GOSUB hexadecimal: AA = SUM: LOCATE 20, 2, 0:
PRINT STRING$(29, " ")
      IF A > AA OR AA > A + 127 THEN A = AA
      GOSUB cm2:
bor1:
      LOCATE 20,  2, 0:  PRINT "DIRECCION  FINAL (H)=  "; : GOSUB
endatos: X$  = YO$:  GOSUB hexadecimal: B = SUM: LOCATE 20, 2, 0:
PRINT STRING$(29, " ")
      IF B < AA THEN BEEP: GOTO bor1
      HX = 0
      FOR TA = 0 TO (B - AA)
      IF 65534 > (AA + TA) AND (AA + TA) >= 32767 THEN M1(AA + TA
- 32767) = 0: GOTO borrar1
      IF  AA  +  TA  >  65533  THEN  F(AA + TA - 65534) = 0: GOTO
borrar1 ELSE M(AA + TA) = 0: GOTO borrar1


borrar1:
      NEXT
      GOSUB cm2
      GOTO borrarbloques

breakp:
      '************SETEADO (BP BANDERA DE BREAK POINT)
      L = 20: CO = 23: LX = 4: HX = 2: BPEXIT = 1
      COLOR 7, 0
      GOSUB PRINTBREAK
bree:
      HX = 2: BPEXIT = 1
      LOCATE 20, 2, 0: PRINT "DIRECCION BREAK (H)= ";
      GOSUB endatos: X$ = YO$
      GOSUB hexadecimal:
      FOR J1 = 0 TO BP% - 1
      IF SUM = BP(J1) THEN BEEP: GOTO bree
      NEXT J1
      BP(BP%) = SUM
      B = 4 - LEN(HEX$(BP(BP%)))
     BPT$(BP%) = STRING$(B, "0") + HEX$(BP(BP%))
      LOCATE 6 + LN, 15, 0: PRINT USING "\    \"; BPT$(BP%)
      IF LN > 11 THEN P$ = INPUT$(1): K1 = 3: GOSUB  bvi: LN  = 0
ELSE LN = LN + 1
      BP% = BP% + 1
      HX = 0: COLOR 7, 0
      GOTO bree
breakp1:
    LX = 4: L = 20: CO = 26: BPEXIT = 1: HX = 2
    GOSUB PRINTBREAK
    LOCATE  20,  2,  0:  PRINT  "BREAKPOINT  A BORRARSE: ": GOSUB
endatos: X$ = YO$
    GOSUB hexadecimal: PB = SUM
    FOR J1 = BP% TO 0 STEP -1
      IF PB = BP(J1) THEN GOTO bre
      NEXT Ji
```

```
      GOTO breakp1
bre:
      BP(J1) = 0
      FOR J2 = J1 TO BP% - 1
      BP(J2) = BP(J2 + 1)
      NEXT
      BP% = BP% - 1
      GOTO breakp1
bre1:
     GOSUB PRINTBREAK
     '       FOR J2 = 0 TO BP%
             BP(J2) = 0
             NEXT
             BP% = 0
             LOCATE 22, 1, 0: PRINT STRING$(78, 255): COLOR 7, 0
             GOTO entrada7
PRINTBREAK:
     COLOR 7, 0: LN = 0
     GOSUB bvi
     LOCATE 2, 7, 0: PRINT "CONTENIDO DE BREAKPOINT";
     LOCATE 4, 12, 0: PRINT "BREAKPONINT"
     IF BP% = 0 THEN RETURN
     FOR j = 0 TO BP% - 1
     B = 4 - LEN(HEX$(BP(j)))
     BPT$(j) = STRING$(B, "0") + HEX$(BP(j))
     LOCATE 6 + LN, 15, 0: PRINT USING "\    \"; BPT$(j)
     IF LN > 11 THEN LOCATE 20, 10,  0: PRINT  "continua...": P$ =
INPUT$(1): K1 = 3: GOSUB bvi: LN = 0 ELSE LN = LN + 1
     NEXT:
     RETURN

buscmem:
       '************ ENCONTRAR VALOR DE MEMORIA
       L = 20: CO = 23: LX = 2: HX = 2: ME = 1
       LOCATE 20,  2, 0:  PRINT "  VALOR DE  MEM (HEX)= "; : GOSUB
endatos: X$ = YO$: GOSUB hexadecimal: A =  SUM: LOCATE  20, 2, 0:
PRINT STRING$(29, " ")
       LOCATE 20,  2, 0:  PRINT "DIRECCION  INICIAL= ";  : LX = 4:
GOSUB endatos: X$ = YO$: GOSUB hexadecimal: B  = SUM:  LOCATE 20,
2, 0: PRINT STRING$(29, " ")
       HX = 0
       LOCATE 20, 2, 0: PRINT "LOCALIDAD: "
       IF B > 32766 THEN GOTO buscar1
       FOR I4 = B TO 32766
       s$ = INKEY$
       IF M(I4)  = A  THEN LOCATE  20, 14, 0: PRINT HEX$(I4): s$ =
INPUT$(1)
       LOCATE 20, 14, 0: PRINT HEX$(I4)
       IF s$ = CHR$(27) THEN LOCATE 20, 2, 0: PRINT  STRING$(28, "
"): COLOR 7, 0: GOTO entrada3
       NEXT
       B = 32767

buscar1:
```

```
      IF B > 65533 THEN GOTO buscar2
      FOR I4 = B - 32767 TO 32766
      s$ = INKEY$
      IF M1(I4) = A THEN LOCATE 20, 14, 0: PRINT HEX$(I4 +
32767): s$ = INPUT$(1)
      LOCATE 20, 14, 0: PRINT HEX$(I4 + 32767)
      IF s$ = CHR$(27) THEN LOCATE 20, 2, 0: PRINT STRING$(28, "
"): GOTO entrada3
      NEXT
      B = 65534
buscar2:
      FOR I4 = B - 65534 TO 1
      s$ = INKEY$
      IF F(I4) = A THEN LOCATE 20, 14, 0: PRINT HEX$(I4 + 65534):
s$ = INPUT$(1)
      LOCATE 20, 14, 0: PRINT HEX$(I4 + 65534)
      IF s$ = CHR$(27) THEN LOCATE 20, 2, 0: PRINT STRING$(28, "
"): GOTO entrada3
      NEXT
      LOCATE 20, 2, 0: PRINT "NO HAY OTRA ": COLOR 7, 0: GOTO
entrada3


llenar:
      '**********************************LLENADO DE BLOQUE CON
UNA CONSTANTE
      BPC = -1
      L = 20: CO = 23: LX = 4: HX = 2: ME = 1
      LOCATE 20, 2, 0: PRINT "DIRECCION INICIAL(H)= "; : LX = 4:
GOSUB endatos: X$ = YO$: GOSUB hexadecimal: AA = SUM: LOCATE 20,
2, 0: PRINT STRING$(29, " ")
      IF A > AA OR AA > A + 127 THEN A = AA
      GOSUB cm2
lle:
      LOCATE 20, 2, 0: PRINT "DIRECCION FINAL (H)= "; : GOSUB
endatos: X$ = YO$: GOSUB hexadecimal: BB = SUM: LOCATE 20, 2, 0:
PRINT STRING$(29, " ")
      IF BB < AA THEN BEEP: GOTO lle
      LOCATE 20, 2, 0: PRINT "VALOR DESEADO (HEX)= "; : LX = 2:
GOSUB endatos: X$ = YO$: GOSUB hexadecimal: C = SUM: LOCATE 20,
2, 0: PRINT STRING$(29, " ")
      HX = 0
      FOR TA = 0 TO (BB - AA)
      IF AA + TA > 65533 THEN F(AA + TA - 65534) = C: GOTO llen
      IF AA + TA >= 32767 THEN M1(AA + TA - 32767) = C ELSE M(AA
+ TA) = C
llen:
      NEXT
      GOSUB cm2
      GOTO llenar

mover:
      '**********************************RUTINA DE MOVIMIENTO DE
```

```
DASM$
      LOCATE 20, 47, 0: PRINT CHR$(18)
mover1:
      L = 0
mover2:
      FOR i = 0 TO 14
      LOCATE 3  + i, 47, 0: PRINT USING "\
  \"; DASM$(i): NEXT

mover3:
      Q$ = INKEY$: IF Q$ = "" THEN GOTO mover3
      IF Q$ = CHR$(27) THEN
      IF ie = 0 THEN HX  = 0:  ME =  0: LOCATE  20, 47,  0: PRINT
STRING$(25, " "): GOTO ingreso1
      IF ie  = 8  THEN HX  = 0:  ME =  0: LOCATE 20, 47, 0: PRINT
STRING$(25, " "): GOTO entrada2:
      IF ie = 13 THEN HX = 0:  ME =  0: LOCATE  20, 47,  0: PRINT
STRING$(25, " "): GOTO entrada3
      END IF
      IF  MID$(Q$,  2,  1)  =  "s"  THEN  LOCATE  20, 2, 0: PRINT
CHR$(18): LOCATE 20, 47, 0: PRINT CHR$(255): GOTO cm4
      IF MID$(Q$, 2, 1) = "P" THEN L = L + 1:  GOTO mover4
      IF MID$(Q$, 2, 1) = "H" THEN L = L - 1:  GOTO mover5

mover4:
      IF L + 15 > NL THEN L = 1: GOTO mover2
      FOR i = L TO L + 15
      LOCATE 4 + i - L, 47, 0: PRINT USING "\
      \"; DASM$(i): NEXT
      GOTO mover3
mover5:
      IF L <= 0 THEN GOTO mover1
      FOR i = L TO L + 15
      LOCATE 4  + i - L, 47, 0: PRINT USING "\
      \"; DASM$(i): NEXT
      GOTO mover3
refresco:
      '**************************** IMPRESION DE VENTANAS
      CLS : COLOR 15, 0: LOCATE 1, 35: PRINT "8096/SIM  ": LOCATE
1,  1,  0,  12,  13:  KEY OFF: PRINT CHR$(218); STRING$(30, 196);
CHR$(191);
      LOCATE 1, 44: KEY  OFF: PRINT  CHR$(218); STRING$(35, 196);
CHR$(191)
      FOR j = 1 TO 20: LOCATE 1 + j, 1: PRINT CHR$(179): LOCATE 1
+ j, 32: PRINT CHR$(179): LOCATE  1  +  j,  44:  PRINT CHR$(179):
LOCATE 1 + j, 80: PRINT CHR$(179): NEXT
      LOCATE 21, 1: PRINT CHR$(192); STRING$(30, 196); CHR$(217);
STRING$(11, 255);  CHR$(192); STRING$(35,  196); CHR$(217): COLOR
7, 0
      'LOCATE  22,2:COLOR  15,0:PRINT CHR$(25);:COLOR 7,0:PRINT "
continúa   ";:COLOR   15,0:PRINT   CHR$(24);:COLOR   7,0:PRINT  "
regresa"
      LOCATE 22,  56: PRINT "Con "; : COLOR 15, 0: PRINT "ESC"; :
COLOR 7, 0: PRINT " menú";
```

```
      '******************************** IMPRESION DE I1- I9
      FOR I1 = 0 TO 7: LOCATE 5 + I1, 36: PRINT I1; "=": LOCATE 5
+ I1,  36: PRINT  "I": NEXT:  LOCATE 13, 36: PRINT "ST =": LOCATE
14, 36: PRINT " I =": LOCATE 15, 36: PRINT " - =": LOCATE 16, 36:
PRINT " C =": LOCATE 17, 36: PRINT "VT =": LOCATE 18,  _
36: PRINT " V ="
      LOCATE 19,  36: PRINT  " N =": LOCATE 20, 36: PRINT " Z =":
LOCATE 4, 36: PRINT "PSW"
      ' ********************** IMPRESION DE R1 -  R25 (REGISTROS
ESPECIALES DEL CPU)
      COLOR 7,  0: LOCATE  2, 7: PRINT "REGISTROS  DE MEMORIA  ":
LOCATE 2, 50: PRINT "EJECUCION " + ARCH$
      GOSUB cm2
      RETURN

      'Mover a través de la ventana izquierda

modmemoria:
      '************************ MODIFICAR MEMORIA
      col = 9: lin = 4
      BPC = -1: INM = 0
      CO = 24: L = 20: LX = 4: CO1 = 15: CO2 = 0: HX = 2: ME = 1
      COLOR 7, 0: LOCATE 2, 6,  0: PRINT  "LOCALIDADES DE MEMORIA
"
modm1:
      LOCATE 20,  2, 0:  PRINT " DIRECCION EN HEXAD= "; : LX = 4:
GOSUB endatos: X$ = YO$: GOSUB hexadecimal: AA = SUM:  LOCATE 20,
2, 0: PRINT STRING$(28, " ")
      IF A > AA OR AA > A + 127 THEN A = AA
modm2:
      GOSUB cm2:
  atr = 15: GOSUB atrib
menum1:
   Q$ = INKEY$: IF Q$ = "" THEN GOTO menum1
    IF Q$ = CHR$(27) THEN COLOR 7, 0: GOSUB cm2: GOTO entrada3
    IF MID$(Q$, 2, 1) = "M" THEN GOTO derecha
     IF MID$(Q$, 2, 1) = "K" THEN GOTO izquierda
     IF MID$(Q$, 2, 1) = "H" THEN GOTO arriba
     IF MID$(Q$, 2, 1) = "P" THEN GOTO abajo
     IF MID$(Q$, 2, 1) = "Q" THEN GOTO pagdn
     IF MID$(Q$, 2, 1) = "I" THEN GOTO pagup
     IF 47 < ASC(Q$) AND ASC(Q$) < 58 THEN GOTO ingreso
     IF 64 < ASC(Q$) AND ASC(Q$) < 71 THEN GOTO ingreso
   GOTO menum1

pagdn:
   IF AA >= 65407 THEN AA = 65407: A = AA: GOTO modm2
   AA = AA + 128: A = AA
   GOTO modm2
pagup:
   IF AA <= 128 THEN AA = 0: A = 0: GOTO modm2
   AA = AA - 128: A = AA
   GOTO modm2
```

```
derecha:
   IF col >= 30 THEN atr = 7: GOSUB atrib: col = 9: GOTO d1
   posi = posi + 2: atr = 7: GOSUB atrib: col = col + 3
d1:
   atr = 15: GOSUB atrib
 GOTO menum1

izquierda:
   IF col <= 10 THEN atr = 7: GOSUB atrib: col = 30: GOTO iz1
   posi = posi - 4: atr = 7: GOSUB atrib: col = col - 3
iz1:
   atr = 15: GOSUB atrib
 GOTO menum1

abajo:
   IF lin >= 19 THEN atr = 7: GOSUB atrib: lin = 4: GOTO ab1
   atr = 7: GOSUB atrib: lin = lin + 1
ab1:
   atr = 15: GOSUB atrib
 GOTO menum1
arriba:
   IF lin <= 4 THEN atr = 7: GOSUB atrib: lin = 19:  GOTO arr1
   atr = 7: GOSUB atrib: lin = lin - 1
arr1:
   atr = 15: GOSUB atrib
 GOTO menum1


atrib:
   posi = ((lin - 1) * 160) + 1 + (2 * (col - 1) - 2)
   POKE posi, atr
   POKE posi + 2, atr
    RETURN
ingreso:
  CO =  col - 1: L = lin: LX = 2: CO1 = 15: CO2 = 0: HX = 2: ME =
1
      XO$ = O$ + "O"
      P = 1: LX1 = LX
      LOCATE L, CO: COLOR CO1, CO2: PRINT XO$: LOCATE L,  CO + 1,
1

  GOSUB eningreso: X$ = YO$: GOSUB hexadecimal: B = SUM:
  INM = AA + B * (lin - 4) + CO / 3 - 3
      IF INM > 65533 THEN F(INM - 65534) = B:  GOTO derecha
      IF INM <= 32766 THEN M(INM) = B ELSE M1(INM - 32767) = B

  GOTO derecha
```

PROGRAMA DE DESENSAMBLADO DEL SIMULADOR

DEL MICROCONTROLADOR 8096 (TD.BAS)

```
COMMON M() AS INTEGER
COMMON M1() AS INTEGER
COMMON COD(), NEM$(), BYT(), U(), EST()
COMMON  LOC$(),  V$(),  dasm$(),  Y$(), A(), AUX(), PSW(), IT$(),
PB(), LSAR(), BP%, RE, TABLA, arch$, NLC, NREG
COMMON A, PC%, BPC, im, B1, L, BP$,  NL, BA,  ESTADOS, FM14, TIM,
BPS, IPS, BZ, EST, RST, shel
COMMON SP AS DOUBLE
COMMON CLAVE, pi(), menu$(), arcs$, BP(), pasos, F()
   IF CLAVE = 0 THEN RUN "TI"
   DEFINT M: J2 = 0
   DEFDBL D, O, S
   DS = 0: M = 8: BM = 1: NA = 0: BZ = 0: EST = 0: EJM = 1: im1 =
0
   DEF SEG = 0
   IF (PEEK(&H410) AND &H30) = &H30 THEN DEF  SEG =  &HB000: GOTO
inicio
   DEF SEG = &HB800


    'PROGRAMA DE EJECUCION
inicio:
    GOSUB BVD: PC% = 8320: O = 1: B1 = 0

break:

    FOR ii = 0 TO BP%
    IF PC% = BP(ii) THEN da$ = CHR$(175): GOTO codig
    NEXT
    da$ = " ": GOTO codig

codig:
    'Q$ = INKEY$
    'IF Q$ = CHR$(27) THEN GOTO REGRESOTM
    G = U(M(PC%) + 1)'            Código de instrucción
    IA = 0: IAX = 0
    IF G > 20 THEN G = G - 20: GOTO subrutina
    ON G  GOSUB SKIP,  CLR, NOTW,  NEG, DEC,  EXT, INC, SHR, SHL,
SHRA, SHRL, SHLL, SHRAL, NORML,  CLRB,  NOTB,  NEGB,  DECB, EXTB,
INCB
    GOTO opsigno
subrutina:
    IF G > 20 THEN G = G - 20: GOTO codigo1
```

```
    ON G  GOSUB SHRB,  SHLB, SHRAB,  SJMP, SCALL, JBC, JBS, AND3,
ADD3, SUB3,  MULU3,  ANDB3,  ADDB3,  SUBB3,  MULUB3,  AND2, ADD2,
SUBW2, MULU, ANDB2
    GOTO opsigno
codigo1:
    IF G > 20 THEN G = G - 20: GOTO codigo2
    ON G  GOSUB ADDB2,  SUBB, MULUB,  ORW, XORW,  CMP, DIVU, ORB,
XORB, CMPB, DIVUB, LD,  ADDC,  SUBC,  LDBZE,  LDB,  ADDCB, SUBCB,
LDBSE, ST
    GOTO opsigno
codigo2:
    IF G > 20 THEN G = G - 20: GOTO codigo3
    ON G  GOSUB STB,  PUSH, POP,  JNST, JNH, JGT, JNC, JNVT, JNV,
JGE, JNE, JST, JH, JLE, JC, JVT, JV, JLT, JE, BR
    GOTO opsigno
codigo3:
    ON G GOSUB LJMP,  LCALL, RET,  PUSHF, POPF,  NOP, CLRC, SETC,
DI, EI, CLRVT, NOP, opersigno, RST, DJNZ
opsigno:
    IF OS = 1 THEN GOTO break
    LOCATE 4,  48: PRINT  USING "\                              \";
dasm$(0): GOTO fin1
    GOTO checkpc
fin1:
    dasm$(0) = da$ + dasm$(0)
    NL = 0: O = O + I
    IF O > 255 THEN LOCATE 25, 2, 1:  GOTO REGRESOTM
checkpc:
    IF PC% <= NREG - BYT(M(PC%) + 1) THEN GOTO  break ELSE  BPC =
0: im1 = 0: CHAIN "TE"

opersigno:          'OPERACIONES CON SIGNO

     PC% = PC% + 1
     OS = 1
     RETURN

DJNZ:  ' decrementa uno y salta si no es cero
     M(PC% + 1) = M(PC% + 1) - 1
     DISP = M(PC% + 2)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP): GOTO CLR1

SUBW2:    ' DOS OPERANDOS RESTA DE PALABRAS
     M = 16
     IF M(PC%) = 104 THEN GOSUB DIRECTO ELSE IF M(PC%) = 105 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 106 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 107 THEN GOSUB INDEXADO ELSE RETURN
     GOTO CLR1
```

```
ADDB2: ' (DOS OPERANDOS), BYTES
     M = 8
     IF M(PC%) = 116 THEN GOSUB DIRECTO ELSE IF M(PC%) = 117 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 118 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 119 THEN GOSUB INDEXADO ELSE RETURN
     GOTO CLR1

SKIP: ' Salto no operación
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     PC% = PC% + 2
     RETURN

CLR:  ' borrar una palabra
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J): GOTO CLR1
CLR1:
     OS = 0: DW = 0
     PC% = PC% + BYT(M(PC%) + 1) + IAX
     RETURN

NOTW: ' complementa una palabra
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J)
     GOTO CLR1

NEG: ' (Cambia signo) a un integer
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J): GOTO NEG2

NEG2:
     PC% = PC% + BYT(M(PC%) + 1) + IAX
     RETURN

DEC: '  (resta uno) a una palabra
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J)
     GOTO CLR1

EXT: ' Extiende integer en long-integer 'Revisar
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J)
     GOTO CLR1

INC: ' Incrementa uno a una palabra
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J)
     GOTO CLR1

SHR: ' desplazamiento logico a la derecha una palabra
     X = M(PC% + 2): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
```

```
NEM$(M(PC%) + 1) + " " + V$(J) + ",#" + HEX$(M(PC% + 1))
     GOTO CLR1

SHL: 'Desplazamiento logico a la izquierda
     X = M(PC% + 2): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J) + ",#" + HEX$(M(PC% + 1))
     GOTO CLR1

SHRA: ' Desplazamiento arit.a la derecha
     M = 16
     X = M(PC% + 2): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J) + ",#" + HEX$(M(PC% + 1))
     GOTO CLR1

SHRL: 'Desplazamiento logico a la derecha doble palabra
     COUNT = M(PC% + 1)
     X = M(PC% + 2): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J) + ",#" + HEX$(M(PC% + 1))
     GOTO CLR1

SHLL: ' Desplazamiento logico a la izquierda
     COUNT = M(PC% + 1)
     X = M(PC% + 2): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J) + ",#" + HEX$(M(PC% + 1))
     GOTO CLR1
SHRAL: ' Desplazamiento  doble palabra
     X = M(PC% + 2): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J) + ",#" + HEX$(M(PC% + 1))
     GOTO CLR1

NORML: ' NORMALIZA LONG-INTEGER
     X = M(PC% + 2): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) +  1) +  " "  + V$(J)  + ",":  X =  M(PC% + 1): GOSUB
EQULOCAL: dasm$(0) = HEX$(PC%) + " " + dasm$(0) + V$(J)
     GOTO CLR1

CLRB: ' Borrar el byte
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J)
     GOTO CLR1

NOTB: ' Cada bit del byte es complementado
     M = 8
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J)
     GOTO CLR1

NEGB: ' EL VALOR DEL SHORT INTEGER ES NEGADO
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J)
```

```
      GOTO CLR1

DECB: ' DECREMENTO DEL BYTE
      M = 8
      X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J)
      GOTO CLR1

EXTB: ' EXTIENDE SHORT INTEGER EN INTEGER
      X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J)
      GOTO CLR1

INCB: ' BYTE INCREMENTADO
      X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J)
      GOTO CLR1

SHRB: ' DESPLAZAMIENTO A LA DERECHA EL BYTE
      X = M(PC% + 2): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J) + ",#" + HEX$(M(PC% + 1))
      GOTO CLR1

SHLB: ' DESPLAZAMIENTO BYTE A LA IZQUIERDA
      X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J) + ",#" + HEX$(M(PC% + 1))
      GOTO CLR1

SHRAB: ' DEZPLAZAMIENTO ARITMETICO DE UN BYTE
      X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + Y$(J) + ",#" + HEX$(M(PC% + 1))
      GOTO CLR1

SJMP: ' SALTO CORTO
      X = M(PC%): M = 8: GOSUB BINARIO: A(9) = A(1): A(10) = A(2):
A(11) = A(3): X = M(PC% + 1): GOSUB BINARIO: M = 11: GOSUB
DECIMAL: DISP = DEST
      dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
      GOTO CLR1

SCALL: ' LLAMADO CORTO
      dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1) + " " +
HEX$(M(PC% + 1))
      GOTO CLR1

JBC: ' SALTO SI EL NB ES CERO
      X = M(PC% + 1): DISP = M(PC% + 2): GOSUB EQULOCAL
      IF DISP > 127 THEN DISP = DISP - 256
      NB = M(PC%) - 47: M = 8: GOSUB BINARIO
```

```
     dasm$(0) =  HEX$(PC%) + " " + NEM$(M(PC%) + 1) + " " + Y$(J)
+ "," + HEX$(NB - 1) + ","  + HEX$(M(PC%  + 2)):  PC% =  PC% + 3:
RETURN

JBS: ' SALTO SI EL BIT ES UNO
     X = M(PC% + 1): DISP = M(PC% + 2)
     IF (DISP > 127) THEN DISP = DISP - 256
     NB = M(PC%) - 55: M = 8: GOSUB BINARIO
     dasm$(0) =  HEX$(PC%) + " " + NEM$(M(PC%) + 1) + " " + Y$(J)
+ "," + HEX$(NB - 1) + ","  + HEX$(M(PC%  + 2)):  PC% =  PC% + 3:
RETURN

AND2: ' LOGICO DE PALABRAS dos operandos
     M = 16
     IF M(PC%)  = 96  THEN GOSUB DIRECTO ELSE IF M(PC%) = 97 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 98 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 99 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

ADD2: ' AÑADIR PALABRAS DE 16 BITS dos operandos
     M = 16
     IF M(PC%) = 100 THEN GOSUB DIRECTO ELSE IF M(PC%) = 101 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 102 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 103 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

MULU: '(palabras) Y MUL(Integers)  MULTIPLICACION dos operandos
     M = 16
     IF M(PC%) = 108 THEN GOSUB DIRECTO ELSE IF M(PC%) = 109 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 110 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 111 THEN GOSUB INDEXADO ELSE RETURN
     IF OS = 1 THEN MID$(dasm$(0), 11, 2) = "  "
     GOTO CLRI

AND82: ' (dos operandos) bytes
     M = 8
     IF M(PC%) = 112 THEN GOSUB DIRECTO ELSE IF M(PC%) = 113 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 114 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 115 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

AND83: ' (tres operandos)bytes
     M = 8
     IF M(PC%)  = 80  THEN GOSUB DIRECTO ELSE IF M(PC%) = 81 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 82 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 83 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

SUBB: ' DOS OPERANDOS RESTA DE BYTES
     M = 8
```

```
     IF M(PC%) = 120 THEN GOSUB DIRECTO ELSE IF M(PC%) = 121 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 122 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 123 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

MULUB: ' Y MULB DOS OPERANDOS MULTIPLICACION DE BYTES
     M = 8
     IF M(PC%) = 124 THEN GOSUB DIRECTO ELSE IF M(PC%) = 125 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 126 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 127 THEN GOSUB INDEXADO ELSE RETURN
     IF OS = 1 THEN MID$(dasm$(0), 11, 2) = "B "
     GOTO CLR1

AND3: ' TRES OPERANDOS AND LOGICO DE PALABRAS
     M = 16
     IF M(PC%)  = 64  THEN GOSUB DIRECTO ELSE IF M(PC%) = 65 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 66 THEN GOSUB INDIRECTO  ELSE IF
M(PC%) = 67 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

ORW: ' LOGICO DE PALABRAS
     M = 16
     IF M(PC%) = 128 THEN GOSUB DIRECTO ELSE IF M(PC%) = 129 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 130 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 131 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

XORW: ' EXCLUSIVO DE PALABRAS
     M = 16
     IF M(PC%) = 132 THEN GOSUB DIRECTO ELSE IF M(PC%) = 133 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 134 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 135 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

CMP: ' COMPARE PALABRAS
     M = 16
     IF M(PC%) = 136 THEN GOSUB DIRECTO ELSE IF M(PC%) = 137 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 138 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 139 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

DIVU: ' palabras (dos operandos) Y DIV(integers)
     M = 16: DW = 1
     IF M(PC%) = 140 THEN GOSUB DIRECTO ELSE IF M(PC%) = 141 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 142 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 143 THEN GOSUB INDEXADO ELSE RETURN
     IF OS = 1 THEN MID$(dasm$(0), 11, 2) = "  "
     GOTO CLR1

ORB: ' OR LOGICO DE BYTES
```

```
     M = 8
     IF M(PC%) = 144 THEN GOSUB DIRECTO ELSE IF M(PC%) = 145 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 146 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 147 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC%,+ BYT(M(PC%) + 1) + IAX: RETURN

XORB: ' Or exclusivo de bytes
     M = 8
     IF M(PC%) = 148 THEN GOSUB DIRECTO ELSE IF M(PC%) = 149 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 150 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 151 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

CMPB: ' COMPARAR BYTS
     M = 8
     IF M(PC%) = 152 THEN GOSUB DIRECTO ELSE IF M(PC%) = 153 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 154 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 155 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

DIVUB: ' bytes (dos operandos) Y DIVU(short integer)
     M = 8: DB = 1
     IF M(PC%) = 156 THEN GOSUB DIRECTO ELSE IF M(PC%) = 157 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 158 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 159 THEN GOSUB INDEXADO ELSE RETURN
     IF OS = 1 THEN MID$(dasm$(0), 11, 2) = "B "
     GOTO CLR1

LD: 'cargar una palabra
     M = 16
     IF M(PC%) = 160 THEN GOSUB DIRECTO ELSE IF M(PC%) = 161 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 162 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 163 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

ADDC: ' Anadir palabras con carry
     M = 16
     IF M(PC%) = 164 THEN GOSUB DIRECTO ELSE IF M(PC%) = 165 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 166 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 167 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

SUBC: ' Resta de palabras con borrow
     M = 16
     IF M(PC%) = 168 THEN GOSUB DIRECTO ELSE IF M(PC%) = 169 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 170 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 171 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

LDBZE: ' Cargar una palabra con un byte
```

```
      M = 8
      IF M(PC%) = 172 THEN GOSUB DIRECTO ELSE IF M(PC%) = 173 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 174 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 175 THEN GOSUB INDEXADO ELSE RETURN
      PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

LDB: ' Cargar un byte
       M = 8
      IF M(PC%) = 176 THEN GOSUB DIRECTO ELSE IF M(PC%) = 177 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 178 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 179 THEN GOSUB INDEXADO ELSE RETURN
      PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

ADDCB: ' Añadir bytes con carry
      IF M(PC%) = 180 THEN GOSUB DIRECTO ELSE IF M(PC%) = 181 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 182 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 183 THEN GOSUB INDEXADO ELSE RETURN
      PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

SUBCB: ' Restar bytes con el borrow
      M = 8
      IF M(PC%) = 184 THEN GOSUB DIRECTO ELSE IF M(PC%) = 185 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 186 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 187 THEN GOSUB INDEXADO ELSE RETURN
      PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

LDBSE: ' Cargar un integer con short-integer
      IF M(PC%) = 188 THEN GOSUB DIRECTO ELSE IF M(PC%) = 189 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 190 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 191 THEN GOSUB INDEXADO ELSE RETURN
      PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

ST: 'Store una palabra (dos operandos)
      M =  16: IF M(PC%) = 192 THEN GOSUB DIRECTO ELSE IF M(PC%) =
194 THEN GOSUB INDIRECTO ELSE IF M(PC%) = 195 THEN GOSUB INDEXADO
ELSE RETURN
      PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

STB: ' Almacenar un byte
      M =  8: IF  M(PC%) = 196 THEN GOSUB DIRECTO ELSE IF M(PC%) =
197  THEN  GOSUB  INMEDIATO  ELSE  IF  M(PC%)  =  198  THEN GOSUB
INDIRECTO ELSE IF M(PC%) = 199 THEN GOSUB INDEXADO ELSE RETURN
      PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

PUSHF: ' LAS BANDERAS
       M = 16
      dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
      GOTO CLR1

POP: ' Pop la palabra
```

```
     X = M(PC% + 1): GOSUB EDULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J)
     GOTO CLR1

JNST: ' SALTAR SI STICKY BIT ES CERO
     DISP = M(PC% + 1)
     dasm$(0) = HEX$(PC%) + " "  +  NEM$(M(PC%)  +  1)  +  "  " +
HEX$(DISP)
     GOTO CLR1

JNH: ' SALTO SI NO ES MAYOR
     DISP = M(PC% + 1)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

JGT: ' Salto si el signed es mayor
     DISP = M(PC% + 1)
     dasm$(0) = HEX$(PC%) + " "  +  NEM$(M(PC%)  +  1)  +  "  " +
HEX$(DISP)
     GOTO CLR1

JNC: ' Salto si el carry es cero
     DISP = M(PC% + 1)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

JNVT: ' Salto si el Overflow es cero
     DISP = M(PC% + 1)
     dasm$(0) = HEX$(PC%) + " "  +  NEM$(M(PC%)  +  1)  +  "  " +
HEX$(DISP)
     GOTO CLR1

JNV: ' salto si el signo es mayor o igual
     DISP = M(PC% + 1)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

JGE: ' SALTO SI EL SIGNO ES MAYOR O IGUAL
     DISP = M(PC% + 1)
     dasm$(0) = HEX$(PC%) + " "  +  NEM$(M(PC%)  +  1)  +  "  " +
HEX$(DISP)
     GOTO CLR1

JNE: ' Salto si no es igual a cero
     DISP = M(PC% + 1)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP): PC% = PC% + 2: RETURN
```

```
JST: ' Salto si el sticky bit es uno
     DISP = M(PC% + 1)
     dasm$(0) = HEX$(PC%) + " "  +  NEM$(M(PC%)  +  1)  +  "  " +
HEX$(DISP)
     GOTO CLR1

JH: ' Salto si es mayor (sin signo)
     DISP = M(PC% + 1)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

JLE: ' Salto si el signo es menor o igual
     DISP = M(PC% + 1)
     dasm$(0) = HEX$(PC%) + " "  +  NEM$(M(PC%)  +  1)  +  "  " +
HEX$(DISP)
     GOTO CLR1

JC: ' Salto si el carry es uno
     DISP = M(PC% + 1)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

JVT: ' Salto si el overflow trap es uno
     DISP = M(PC% + 1)
     dasm$(0) = HEX$(PC%) + " "  +  NEM$(M(PC%)  +  1)  +  "  " +
HEX$(DISP)
     GOTO CLR1

JV: ' Salto si el overflow bandera es uno
     DISP = M(PC% + 1)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

JLT: ' Salto si el signo es menor que
     DISP = M(PC% + 1)
     dasm$(0) = HEX$(PC%) + " "  +  NEM$(M(PC%)  +  1)  +  "  " +
HEX$(DISP)
     GOTO CLR1

JE: 'Salto si es igual
     DISP = M(PC% + 1)
     dasm$(0)  =  HEX$(PC%)  +  "  "  +  NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

BR: '(indirecto) Salto indirecto
```

```
     X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
NEM$(M(PC%) + 1) + " " + V$(J)
     GOTO CLR1

LJMP: ' Salto largo
     DISP = M(PC% + 1) + 256 * M(PC% + 2)
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

LCALL: ' Llamada larga
     DISP = M(PC% + 1) + 256 * M(PC% + 2)
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1) + " " +
HEX$(DISP)
     GOTO CLR1

RET: ' Return de una subrutina
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1): PC% = PC% +
1: RETURN

PUSH: 'PUSH PALABRAS
     IF M(PC%) = 96 THEN GOSUB DIRECTO ELSE IF M(PC%) = 97 THEN
GOSUB INMEDIATO ELSE IF M(PC%) = 98 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 99 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

POPF: 'BANDERAS
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     GOTO CLR1

CLRC: 'Borrar el carry
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     GOTO CLR1

SETC: 'Setear el carry
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     GOTO CLR1

DI: 'DESABILITA INTERRUPCIONES
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     GOTO CLR1

EI: 'HABILITA INTERRUPCIONES
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     GOTO CLR1

CLRVT: 'Borra el overflow trap
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     GOTO CLR1
```

```
NOP: 'No operacion
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     GOTO CLR1

RST: 'RESETEA EL SISTEMA
     dasm$(0) = HEX$(PC%) + " " + NEM$(M(PC%) + 1)
     GOTO CLR1

ADD3: '(tres operandos) anadir palabras
     M = 16
     IF M(PC%) = 68 THEN GOSUB DIRECTO ELSE  IF M(PC%)  = 69 THEN
GOSUB INMEDIATO  ELSE IF M(PC%) = 70 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 71 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

SUB3: '(Tres operandos) resta de palabras
     M = 16
     IF M(PC%) = 72 THEN GOSUB DIRECTO ELSE  IF M(PC%)  = 73 THEN
GOSUB INMEDIATO  ELSE IF M(PC%) = 74 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 75 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

MULU3: '(Integers) Y MULU(Palabras) multiplicacion tres operandos
     M = 16
     IF M(PC%) = 76 THEN GOSUB DIRECTO ELSE  IF M(PC%)  = 77 THEN
GOSUB INMEDIATO  ELSE IF M(PC%) = 78 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 79 THEN GOSUB INDEXADO ELSE RETURN
     IF OS = 1 THEN MID$(dasm$(0), 11, 2) = "  "
     GOTO CLR1

ADDB3: '(tres operandos) anade bytes
     M = 8
     IF M(PC%) = 84 THEN GOSUB DIRECTO ELSE  IF M(PC%)  = 85 THEN
GOSUB INMEDIATO  ELSE IF M(PC%) = 86 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 87 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

SUBB3: '(tres operandos) resta bytes
     M = 8
     IF M(PC%) = 88 THEN GOSUB DIRECTO ELSE  IF M(PC%)  = 89 THEN
GOSUB INMEDIATO  ELSE IF M(PC%) = 90 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 91 THEN GOSUB INDEXADO ELSE RETURN
     PC% = PC% + BYT(M(PC%) + 1) + IAX: RETURN

MULUB3: '(tres operandos) multiplica bytes
     M = 8
     IF M(PC%) = 92 THEN GOSUB DIRECTO ELSE  IF M(PC%)  = 93 THEN
GOSUB INMEDIATO  ELSE IF M(PC%) = 94 THEN GOSUB INDIRECTO ELSE IF
M(PC%) = 95 THEN GOSUB INDEXADO ELSE RETURN
     IF OS = 1 THEN MID$(dasm$(0), 11, 2) = "B "
```

```
      GOTO CLR1


DIRECTO:

       X = BYT(M(PC%) + 1)
       IF M = 16 THEN GOTO WORDS2
       IF X = 4 THEN GOTO BYTS3
       SRC = M(M(PC% + 1))
       IF DB = 1 THEN SR = M(M(PC% + 2) + 1): DEST = M(M(PC% + 2))
+ 256 * SR ELSE DEST = M(M(PC% + 2))
       IND = M(PC% + 2): INSTO = M(PC% + 1)
       IF IND > 32766 THEN IND = IND - 32767: BI = 1
       X = M(PC% + 2): GOSUB EQULOCAL
       dasm$(0) =  NEM$(M(PC%) + 1) + " " + Y$(J) + ",": X = M(PC%
+ 1): GOSUB EQULOCAL
       dasm$(0) = HEX$(PC%) + " " + dasm$(0) + Y$(J)
       RETURN
BYTS3:
       SRC2 = M(M(PC% + 1))
       SRC1 = M(M(PC% + 2))
       IND = M(PC% + 3)
       IF IND > 32766 THEN IND = IND - 32767: BI = 1
       X = M(PC% + 3): GOSUB EQULOCAL
       dasm$(0) = NEM$(M(PC%) + 1) + " " + Y$(J) + ",":  X = M(PC%
+ 2): GOSUB EQULOCAL
       dasm$(0) =  dasm$(0) +  Y$(J) +  ",": X = M(PC% + 1): GOSUB
EQULOCAL
       dasm$(0) = HEX$(PC%) + " " + dasm$(0) + Y$(J)
       RETURN
WORDS2:
       IF X = 4 THEN GOTO WORDS3
       SR = M(M(PC% + 1) + 1)
       SRC = M(M(PC% + 1)) + 256 * SR: DE = M(M(PC% + 2) + 1): DE1
= M(M(PC% + 2) + 2): DE2 = M(M(PC% + 2) + 3)
       IF DW  = 1  THEN da#  = 256 * (DE + 256 * DE1 + (256 ^ 2) *
DE2): DEST = M(M(PC% + 2)) + da ELSE DEST = M(M(PC% + 2)) + 256 *
DE
       IND = M(PC% + 2): INSTO = M(PC% + 1)
       IF IND > 32766 THEN IND = IND - 32767: BI = 1
       X = M(PC% + 2): GOSUB EQULOCAL
       dasm$(0) =  NEM$(M(PC%) + 1) + " " + V$(J): X = M(PC% + 1):
GOSUB EQULOCAL: dasm$(0) =  HEX$(PC%) +  " "  + dasm$(0)  + "," +
V$(J): RETURN
WORDS3:
       SR1 = M(M(PC% + 1) + 1)
       SRC2# = M(M(PC% + 1)) + 256 * SR1
       SR2 = M(M(PC% + 2) + 1)
       SRC1# = M(M(PC% + 2)) + 256 * SR2
       IND = M(PC% + 3)
```

```
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
       X = M(PC% + 3): GOSUB EQULOCAL
      dasm$(0) =  NEM$(M(PC%) + 1) + " " + V$(J): X = M(PC% + 2):
GOSUB EQULOCAL: dasm$(0) = dasm$(0) +  "," +  V$(J): X  = M(PC% +
1): GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " + dasm$(0) + "," +
V$(J): RETURN

INDIRECTO:
      H = M(PC% + 1) - INT(M(PC% + 1) / 2) * 2: IF H =  1 THEN IA
= 1 ELSE IA = 0
      X = BYT(M(PC%) + 1)
      IF M = 16 THEN GOTO IWORDS2
      IF X = 4 THEN GOTO IBYTS3
      SR = (M(M(PC% + 1) + 1))
      SRC = M(M(M(PC% + 1)) + 256 * SR)
      IF DB  = 1  THEN SR1  = M(M(PC% + 2) + 1): DEST = M(M(PC% +
2)) + 256 * SR1 ELSE DEST = M(M(PC% + 2))
      IND = M(PC% + 2): INSTO = M(M(PC% + 1)) + 256 * SR
      IF IND > 32766 OR INSTO > 32766 THEN INSTO = INSTO - 32767:
IND = IND - 32767: BI = 1
       X = M(PC% + 2): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + Y$(J): X = M(PC% + 1) -
IA: GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +  dasm$(0) + ",["
+ V$(J) + "]": RETURN
IBYTS3:

      SR1 = (M(M(PC% + 1) + 1))
      SRC2 = M(M(M(PC% + 1)) + 256 * SR1)
      SRC1 = M(M(PC% + 2))
      IND = M(PC% + 3)
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
      X = M(PC% + 3): GOSUB EQULOCAL
      dasm$(0) =  NEM$(M(PC%) + 1) + " " + Y$(J): X = M(PC% + 2):
GOSUB EQULOCAL: dasm$(0) = dasm$(0) + "," + Y$(J): X = M(PC% + 1)
- IA:  GOSUB EQULOCAL:  dasm$(0) =  HEX$(PC%) +  " " + dasm$(0) +
",[" + V$(J) + "]": RETURN
IWORDS2:
      IF X = 4 THEN GOTO IWORDS3
      DE4 = M(M(PC% + 1) + 1): SC = M(M(PC% + 1)) + 256  * (DE4):
DE =  M(M(PC% + 2) + 1): DE1 = M(M(PC% + 2) + 2): DE2 = M(M(PC% +
2) + 3)
      IF SC > 32766 THEN SC = SC - 32767: DE5 = M1(SC + 1): SRC =
M1(SC): SRC  = SRC + 256 * DE5 ELSE DE5 = M(SC + 1): SRC = M(SC):
SRC = SRC + 256 * DE5
      IF DW = 1 THEN da = 256 * (DE  + 256  * DE1  + (256  ^ 2) *
DE2): DEST = M(M(PC% + 2)) + da ELSE DEST = M(M(PC% + 2)) + 256 *
DE
      IND = M(PC% + 2): INSTO = M(M(PC% + 1)) + 256 * DE4
      IF IND > 32766 OR INSTO > 32766 THEN INSTO = INSTO - 32767:
IND = IND - 32767: BI = 1
```

```
      X = M(PC% + 2): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + V$(J): X = M(PC% + 1) -
IA: GOSUB EQULOCAL: X = M(PC% + 1): GOSUB EQULOCAL: dasm$(0) =
HEX$(PC%) + " " + dasm$(0) + ",[" + V$(J) + "]": RETURN
IWORDS3:
      SR1 = (M(M(PC% + 1) + 1))
      SC = M(M(PC% + 1)) + 256 * SR1
      SR2 = M(SC + 1)
      SRC2 = M(SC) + 256 * SR2
      SR3 = M(M(PC% + 2) + 1)
      SRC1 = M(M(PC% + 2)) + 256 * SR3
      IND = M(PC% + 3)
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
      X = M(PC% + 3): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + V$(J): X = M(PC% + 2):
GOSUB EQULOCAL: dasm$(0) = dasm$(0) + "," + V$(J): X = M(PC% + 1)
- IA: GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " + dasm$(0) +
",[" + V$(J) + "]": RETURN
IW1:
      IF IA = 1 THEN GOTO IW2 ELSE RETURN
IW2:
      IF M = 8 THEN M(M(PC% + 1)) = M(M(PC% + 1)) + 1 ELSE
M(M(PC% + 1)) = M(M(PC% + 1)) + 2
      RETURN

INMEDIATO:
      X = BYT(M(PC%) + 1)
      IF M = 16 THEN GOTO NWORDS2
      IF X = 4 THEN GOTO NBYTS3
      SRC = M(PC% + 1)
      IF DB = 1 THEN SR = M(M(PC% + 2) + 1): DEST = M(M(PC% + 2))
+ 256 * SR ELSE DEST = M(M(PC% + 2))
      IND = M(PC% + 2)
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
      X = M(PC% + 2): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + Y$(J): dasm$(0) =
HEX$(PC%) + " " + dasm$(0) + ",#" + HEX$(M(PC% + 1)): RETURN
NBYTS3:

      SRC2 = M(PC% + 1)
      SRC1 = M(M(PC% + 2))
      IND = M(PC% + 3)
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
      X = M(PC% + 3): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + Y$(J): X = M(PC% + 2):
GOSUB EQULOCAL: dasm$(0) = dasm$(0) + "," + Y$(J): dasm$(0) =
HEX$(PC%) + " " + dasm$(0) + ",#" + HEX$(M(PC% + 1)): RETURN
NWORDS2:
      IF X = 5 THEN GOTO NWORDS3
      DE4 = M(PC% + 2): SRC = M(PC% + 1) + 256 * DE4: DE = M(PC%
```

```
+ 3): DE1 = M(DE + 1): DE2 = M(DE + 2): DE3 = M(DE + 3)
      IF DW = 1 THEN da = 256 * (DE1 + 256  * DE2  + (256  ^ 2) *
DE3): DEST = M(DE) + da ELSE DEST = M(DE) + 256 * DE1
      IND = M(PC% + 3)
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
      X = M(PC% + 3): GOSUB EQULOCAL
      dasm$(0)  =  NEM$(M(PC%)  +  I)  +  " " + V$(J): dasm$(0) =
HEX$(PC%) + " " + dasm$(0) + ",#" + HEX$(SRC): RETURN
NWORDS3:
      SR1 = M(PC% + 2)
      SRC2 = M(PC% + 1) + 256 * SR1
      SR2 = M(M(PC% + 3) + 1)
      SRC1 = M(M(PC% + 3)) + 256 * SR2
      IND = M(PC% + 4)
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
      X = M(PC% + 4): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + V$(J): X =  M(PC% + 3):
GOSUB EQULOCAL:  dasm$(0) =  dasm$(0) +  "," +  V$(J): dasm$(0) =
HEX$(PC%) + " " + dasm$(0) + ",#" + HEX$(SRC1): RETURN

INDEXADO:
      X = BYT(M(PC%) + 1): IAX = 0
      SC1 = M(M(PC% + 1) + 1)
      SC = M(M(PC% + 1)) + 256 * SC1
      H = M(PC% + 1) - INT(M(PC% + 1) / 2) * 2
      IF H = 0 THEN IAX = 0: GOTO SHORTINX
      IAX = 1                '  H=0  short indexado
      OF = M(PC% + 2)
      OF1 = M(PC% + 3)
      OF = OF + 256 * OF1
      IF OF > 32768! THEN OF = OF - 65536!
      GOTO XBYTS2

SHORTINX:
      SC1 = M(M(PC% + 1) + 1)
      SC = M(M(PC% + 1)) + 256 * SC1
      OF = M(PC% + 2)
      IF OF > 128 THEN OF = OF - 256
XBYTS2:
      IF M = 16 THEN GOTO XWORDS2
      IF X = 5 THEN GOTO XBYTS3
      SRC = M(ABS(SC + OF))
      IF DB = 1 THEN DE = M(M(PC% + 3 + IAX) + 1): DEST = M(M(PC%
+ 3 + IAX)) + 256 * DE ELSE DEST = M(M(PC% + 3 + IAX))
      IND = M(PC% + 3 + IAX): INSTO = SC + OF
      IF IND > 32766 OR INSTO > 32766 THEN INSTO = INSTO - 32767:
IND = IND - 32767: BI = 1
      X = M(PC% + 3 + IAX): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + Y$(J) + ",":  X = M(PC%
+ 1) - IAX:  GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " + dasm$(0)
```

```
+ HEX$(OF) + ",[" + V$ + "]": RETURN
XBYTS3:
      SRC1 = M(ABS(SC + OF))
      SRC2 = M(M(PC% + 3 + IAX))
      IND = M(PC% + 4 + IAX): INSTO = SC + OF
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
      X = M(PC% + 4 + IAX): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + Y$(J): X = M(PC% + 3 +
IAX): GOSUB EQULOCAL: dasm$(0) = dasm$(0) + "," + Y$ + ",": X =
M(PC% + 1) - IAX: GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + "  " +
dasm$(0) + HEX$(OF) + ",[" + V$ + "]": RETURN
XWORDS2:
      IF X = 5 THEN GOTO XWORDS3
      DE = M(ABS(SC + OF + 1)): SRC = M(ABS(SC + OF)) + 256 * DE:
DE = M(M(PC% + 3 + IAX) + 1): DE1 = M(M(PC% + 3 + IAX) + 2): DE2
= M(M(PC% + 3 + IAX) + 3)
      IF DW = 1 THEN da = 256 * (DE + 256 * DE1 + (256 ^ 2) *
DE2): DEST = M(M(PC% + 3 + IAX)) + da ELSE DEST = M(M(PC% + 3 +
IAX)) + 256 * DE
      IND = M(PC% + 3 + IAX): INSTO = SC + OF'var para stor
      IF IND > 32766 OR INSTO > 32766 THEN INSTO = INSTO - 32767:
IND = IND - 32767: BI = 1
      X = M(PC% + 3 + IAX): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + V$(J) + ",": X = M(PC%
+ 1) - IAX: GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " + dasm$(0)
+ HEX$(OF) + "[" + V$(J) + "]": RETURN
XWORDS3:
      SR1 = M(ABS(SC + OF + 1))
      SRC1 = M(ABS(SC + OF)) + 256 * SR1
      SR2 = M(M(PC% + 3 + IAX) + 1)
      SRC2 = M(M(PC% + 3 + IAX)) + 256 * SR2
      IND = M(PC% + 4 + IAX): INSTO = SC + OF
      IF IND > 32766 THEN IND = IND - 32767: BI = 1
      X = M(PC% + 4 + IAX): GOSUB EQULOCAL
      dasm$(0) = NEM$(M(PC%) + 1) + " " + V$(J): X = M(PC% + 3 +
IAX): GOSUB EQULOCAL: dasm$(0) = dasm$(0) + "," + V$(J) + ",": X
= M(PC% + 1) - IAX: GOSUB EQULOCAL: dasm$(0) = HEX$(PC%) + " " +
dasm$(0) + HEX$(OF) + ",[" + V$ + "]": RETURN

REGRESOTM:

      BPC = 1: CHAIN "TM"


EQULOCAL:
      IF X > 255 THEN LOCATE 20, 50: PRINT "Ilegal asignacion ":
q$ = INPUT$(1): CHAIN "TM"
      IF NLC = 0 THEN V$(J) = HEX$(X): Y$(J) = HEX$(X): RETURN
      b = 4 - LEN(HEX$(X))
      X$ = STRING$(b, "0") + HEX$(X)
```

```
      FOR J = NLC + 1 TO 1 STEP -1
      IF X$ = LOC$(J) THEN RETURN
      NEXT
      RETURN
BVD1:
      IF NL < 15 THEN J2 = 15 - NL ELSE J2 = 0

BVD:
      FOR j1 = 0 TO 16 - J2
      LOCATE 4 + j1 + J2, 47: PRINT STRING$(30, " "): NEXT
      RETURN
BINARIO: 'RUTINA DE CONVERCION A BINARIO
     D = 0: FOR I = 1 TO M
     Y = ABS(INT(X / 2))
     A(I) = X - Y * 2
     X = Y
     NEXT I
     RETURN
DECIMAL:   'RUTINA DE BINARIO A DECIMAL
     SUM = 0
     FOR I = 1 TO M
     SUM = SUM + A(I) * (2 ^ (I - 1))
      NEXT I: DEST = SUM
      RETURN
```

PROGRAMA DEL CONVERSOR ANALOGO DIGITAL DEL

SIMULADOR DEL MICROCONTROLADOR 8096 (TA.BAS)

```
COMMON M() AS INTEGER
COMMON MI() AS INTEGER
COMMON COD(), NEM$(), BYT(), U(), EST()
COMMON  LOC$(),  V$(),  DASM$(),  y$(), A(), AUX(), psw(), IT$(),
PB(), LSAR(), BP%, RE, TABLA, ARCH$, NLC, NREG
COMMON A, PC%, BPC, IM, B1, L, BP$,  NL, BA,  ESTADOS, FM14, TIM,
BPS, IPS, BZ, EST, RST, shel
COMMON SP AS DOUBLE
COMMON CLAVE, pi(), menu$(), ARC$, BP(), pasos, F()
IF CLAVE = 0 THEN RUN "TI"
ON ERROR GOTO serror
BA = BA + 1
IF BA = 1 THEN GOTO nombre ELSE GOTO inicio
nombre:
LOCATE 24, 1: PRINT "No ha elegido el nombre del arch. analogo ";
CHAIN "TM"

inicio:


     IF EOF(3) THEN BA = 0: CLOSE : GOTO fin
     LINE INPUT #3, A$
     IF A$ = "" THEN GOTO inicio
     VANALOGO = ABS(VAL(A$))
     M = 8: CH = M(2) AND 7: X = M(2): GOSUB conbinaria: FOR I4 =
1 TO 8: LSAR(I4) = A(I4): NEXT
     VALMAX = 1023    'NIVEL TTL
     FESCALA = 5
     FACTOR = VALMAX / FESCALA
     VALDIGI = INT(FACTOR * VANALOGO + .5)
     LOCATE 20, 10: PRINT "RES : "; VALDIGI;
     LOCATE 20, 3: PRINT "CH :"; CH; "  RES : "; VALDIGI;
     M = 10: X = VALDIGI: GOSUB conbinaria
     LSAR(7) = A(1): LSAR(8) = A(2)
     FOR I4 = 1 TO 8: A(I4) = A(I4 + 2): NEXT
     M = 8: GOSUB condecimal: M(3) = SUM
     FOR I4 = 1 TO 8: A(I4) = LSAR(I4): NEXT: A(4) = 0
     M = 8: GOSUB condecimal: M(2) = SUM
     PB(12) = 0

fin:
    BPC = 1: M(2) = M(2) AND 247: AMK =  M(8) AND  2 'AMK BANDERA
DE MASCARA DE INT. ANALOGA
     IF psw(2)  = 1  AND AMK  = 1 THEN IAN = 1 ELSE IAN = 0  'IAN
```

```
bandera de int. analg
     CHAIN "TE"

conbinaria:

     D = 0: FOR I = 1 TO M
     y = ABS(INT(X / 2))
     A(I) = X - y * 2
     X = y
     NEXT I
     RETURN

condecimal:

     SUM = 0
     FOR I = 1 TO M
     SUM = SUM + A(I) * (2 ^ (I - 1))
      NEXT I: DEST = SUM
      RETURN
serror:
      IF ERR = 53 AND ERL = 1081 THEN LOCATE 24, 60: COLOR 16, 6:
PRINT "NO EXISTE "; : RESUME nombre
      RESUME NEXT
```

PROGRAMA DEL PORTICO SERIAL DEL SIMULADOR

DEL MICROCONTROLADOR 8096 (TS.BAS)

```
COMMON M() AS INTEGER
COMMON M1() AS INTEGER
COMMON COD(), NEM$(), BYT(), U(), EST()
COMMON  LOC$(),  V$(),  DASM$(),  y$(), A(), AUX(), PSW(), IT$(),
PB(), LSAR(), BP%, RE, TABLA, ARCH$, NLC, NREG
COMMON A, PC%, BPC, IM, B1, L, BP$,  NL, BA,  ESTADOS, FM14, TIM,
BPS, IPS, BZ, EST, RST, shel
COMMON SP AS DOUBLE
COMMON CLAVE, pi(), menu$(), ARCS$, BP(), pasos, F()
IF CLAVE = 0 THEN RUN "TI"
     ON ERROR GOTO erro
     IF ESTADOS <> 0 THEN GOTO PORT
     FREC = FM14 AND 32768
     IF FREC = 32768 THEN F = 1.2E+07: GOTO demod
     LOCATE 20, 3: PRINT "Timer2 M o A: ":
e:
     G$ = INKEY$
     LOCATE 20, 18: PRINT G$
     IF G$ = CHR$(27) THEN CHAIN "TM"
     BPC = 1
     IF G$  = "M"  THEN ESTADOS = 1: BPC = 1: LOCATE 20, 3: PRINT
STRING$(20, 255): CHAIN "TE"
     IF G$ = "A" THEN GOTO e1 ELSE GOTO e
e1:
     LOCATE 20, 3: INPUT "FRECUENCIA EN Hz : ", F:  LOCATE 20, 3:
PRINT STRING$(20, 255)
     IF F >= 48 AND F <= 58 THEN GOTO demod ELSE GOTO e1
demod:
     B = FM14 AND 32767
     modo = M(17) AND 3
     IF modo = 0 THEN GOTO frecuencia ELSE GOTO frecuencia1

frecuencia:
     IF B  = 0 THEN LOCATE 20, 3: PRINT " ERROR EN PORTICO SERIAL
": LOCATE 20, 3: PRINT STRING$(30, 255): CHAIN "TE"
     IF FREC = 32768 THEN VELO = F / (4 * (B + 1)) ELSE VELO  = F
/ B
     ESTADOS = INT(1 / (VELO * (2.5E-07)) + .5): GOTO BPC

frecuencia1:
     IF FREC = 32768 THEN VELO = F / (64 * (B + 1)) ELSE VELO = F
/ (16 * B)
     ESTADOS = INT(1 / (VELO * (2.5E-07)) + .5)
     LOCATE 20, 3: PRINT "ESTADOS:"; ESTADOS
BPC:
     BPC = 1: CHAIN "TE"
PORT:
```

```
      BPS = 1 + BPS
      LOCATE 20, 47: PRINT "BPS "; BPS
      modo = M(17) AND 3
      modo = modo + 1
      M = 8: X = M(17): GOSUB binario
      PAR = A(3)
      REN = A(4)
      TBB = A(5)
      TI = A(6)
      RI = A(7)
      RBB = A(8)
      FREC = M(14) AND 32768
      ON modo GOSUB MODO0, MODO1, MODO2, MODO3
      BPC = 1: CHAIN "TE"

MODO0:
      IF REN = 0 THEN GOTO MODOOTX ELSE GOTO MODOORX
      'TRANSMISION
MODOOTX:

      M = 8: X = M(7): GOSUB binario
      IOC1.5 = M(22) AND 32
      IF IOC1.5 = 0 THEN LOCATE 20, 3:  PRINT "ERROR TRANSMISION":
LOCATE 20, 3: PRINT STRING$(30, 255): CHAIN "TE"
      LOCATE 20, 3: PRINT "TXD= "
      TXD = A(BPS): LOCATE 20, 7: PRINT TXD
      IF BPS = 8 THEN M(17) = M(17) AND 223: BPC = 1: BPS = 0
      RETURN

      'RECEPCION
MODOORX:
      FOR I = 1 TO 8: A(I) = 0: NEXT

lectura:
      IF EOF(4) THEN CLOSE : GOTO finlectura
      LINE INPUT #4, A$
      IF A$ = "" THEN GOTO lectura
      A(BPS) = ABS(VAL(A$))
      M = 8: GOSUB decimal

finlectura:
      M(7) = M(7) OR SUM
      IF BPS = 8 THEN M(17) = M(17) AND 191: BPC = 1: BPS = 0
      RETURN


MODO1:
      IF RI = 1 THEN GOTO modo1rx     'RI bandera de recepcion
      IF TI = 1 THEN GOTO modo1tx ELSE RETURN
'TRANSMISION
```

```
modo1tx:
     M = 8: X = M(7): GOSUB binario
     IOC1.5 = M(22) AND 32
     IF IOC1.5  = 0 THEN LOCATE 20, 3: PRINT "ERROR TRANSMISION":
LOCATE 20, 3: PRINT STRING$(30, 255): CHAIN "TE"
     FOR I = 8 TO 1 STEP -1
     A(1 + I) = A(I)
     NEXT
     IF PAR = 0 THEN GOTO parid
     PARIDAD = A(2) XOR A(3) XOR A(4) XOR A(5) XOR  A(6) XOR A(7)
XOR A(8) XOR A(9)
     A(9) = PARIDAD
parid:
     A(1) = 0
     A(10) = 1
     TXD = A(BPS)
     LOCATE 20, 3: PRINT "TXD=    "; TXD
     IF BPS = 10 THEN M(17) = M(17) AND 223: BPC = 1: BPS = 0
     RETURN
'RECEPCION

modo1rx:
     FOR I = 1 TO 10: A(I) = 0: NEXT

     IF EOF(4) THEN CLOSE : GOTO finlectura1
     LINE INPUT #4, A$
     IF BPS = 1 THEN RETURN ELSE A(BPS - 1) = ABS(VAL(A$))

finlectura1:
     IF BPS  = 10  THEN M(17)  = M(17) AND 191: BPC = 1: BPS = 0:
RETURN
     M = 8: GOSUB decimal
     M(7) = M(7) OR SUM
     RETURN


MODO2:
     IF RI = 1 THEN GOTO modo2rx
     IF TI = 1 THEN GOTO modo2tx ELSE RETURN
'TRANSMISION

modo2tx:
     M = 8: X = M(7): GOSUB binario
     IOC1.5 = M(22) AND 32
     IF IOC1.5 = 0 THEN LOCATE 20, 3:  PRINT "ERROR TRANSMISION":
LOCATE 20, 3: PRINT STRING$(30, 255): CHAIN "TE"
     FOR I = 8 TO 1 STEP -1
     A(1 + I) = A(1)
     NEXT
```

```
     A(1) = 0
     A(10) = TBB
     A(11) = 1
     TXD = A(BPS)
     LOCATE 20, 7: PRINT "TXD=  "; TXD
     IF BPS  = 11  THEN M(17)  = M(17) AND 223: BPC = 1: IPS = 0:
RETURN
'RECEPCION

modo2rx:
     FOR I = 1 TO 11: A(I) = 0: NEXT
     IF BPS = 11 THEN M(17) = M(17) AND 191: BPS = 0: RETURN

lectura2:
     IF EOF(4) THEN CLOSE : GOTO finmodo2
     LINE INPUT #4, A$
     IF A$ = "" THEN GOTO lectura2
     IF BPS = 1 THEN RETURN ELSE A(BPS + 1) = ABS(VAL(A$))
     IF BPS <> 10 THEN GOTO finmodo2
     IF A(10) = 0  THEN M(9)  = M(9)  AND 191:  BPC =  1: M(17) =
M(17) + 2E+07 * A(10): M(17) = M(17) AND 191: BPS = 0: RETURN

finmodo2:
     M = 8: GOSUB decimal
     M(7) = M(7) OR SUM
     RETURN


MODO3:
     IF RI = 1 THEN GOTO modo3rx
     IF TI = 1 THEN GOTO modo3tx ELSE RETURN
'TRANMISION
modo3tx:

     M = 8: X = M(7): GOSUB binario
     IOC1.5 = M(22) AND 32
     IF IOC1.5  = 0 THEN LOCATE 20, 3: PRINT "ERROR TRANSMISION":
LOCATE 20, 3: PRINT STRING$(30, 255): CHAIN "TE"
     FOR I = B TO 1 STEP -1
     A(I + I) = A(I)
     NEXT
     A(I) = 0
     A(10) = TBB
     A(11) = I
     TXD = A(BPS)
     LOCATE 20, 7: PRINT "TXD= "; TXD
     IF BPS = 11 THEN M(17) = M(17) AND 223: BPC = 1: BPS = 0
     RETURN
'RECEPCION
```

```
modo3rx:
     FOR I = 1 TO 11: A(I) = 0: NEXT
     IF BPS = 11 THEN M(17) = M(17) AND 191:  BPC =  1: BPS  = 0:
RETURN

lectura3:
     IF EOF(4) THEN CLOSE : GOTO finmodo3
     LINE INPUT #4, A$
     IF A$ = "" THEN GOTO lectura3
     IF BPS = 1 THEN RETURN ELSE A(BPS + 1) = ABS(VAL(A$))

finmodo3:
     M = 8: GOSUB decimal
     M(7) = M(7) OR SUM
     RETURN


binario:
     'RUTINA DE CONVERCION A BINARIO
     '
     D = 0: FOR I = 1 TO M
     y = ABS(INT(X / 2))
     A(I) = X - y * 2
     X = y
     NEXT I
     RETURN

decimal:
     'RUTINA DE BINARIO A DECIMAL
     '
      SUM = 0
      FOR I = 1 TO M
      SUM = SUM + A(I) * (2 ^ (I - 1))
      NEXT I: DEST = SUM
      RETURN

erro:
      '****** SUBRUTINA DE ERROR
      IF ERR  = 53  AND ERL = 31 THEN LOCATE 24, 60: COLOR 16, 6:
PRINT "NO EXISTE "; : END
      IF ERR = 53 AND ERL = 61 THEN LOCATE  24, 60:  COLOR 16, 6:
PRINT "NO EXISTE   ": : END
      IF ERR  = 53 AND ERL = 101 THEN LOCATE 24, 60: COLOR 16, 6:
PRINT "NO EXISTE   "; : END
      IF ERR = 53 AND ERL = 141 THEN LOCATE 24, 60:  COLOR 16, 6:
PRINT "NO EXISTE   "; : END
      END
```

B I B L I O G R A F I A

- INTEL, Microcontroller Handbook, Santa Clara California, U.S.A. 1984.
- INTEL, Microcontroller Handbook, Santa Clara California, U.S.A. 1986.
- INTEL, 16-Bit Embedded Controller Handbook, Santa Clara California, U.S.A. 1989.
- GORDON GEOFFREY, Simulación de Sistemas,Diana, México,1981
- NAYLOR THOMAS, Técnicas de simulación de Computadores, Paraninfo, España, 1974.
- INTEL, Cross Assembler C16, California, U.S.A.