ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

DEPURADOR Y KIT DE DESARROLLO PARA EL MICROCONTROLADOR DS5000T

TESIS PREVIA A LA OBTENCION DEL TITULO DE INGENIEROS EN LA ESPECIALIZACION DE ELECTRONICA Y TELECOMUNICACIONES

FABIO GONZALEZ GONZALEZ

CESAR MORENO FLORES

QUITO, NOVIEMBRE 1991

Certifico que el presente trabajo ha sido realizado en su totalidad por los señores Fabio González González y César Moreno Flores, bajo mi dirección.

Ing. Luis Montalvo Ramirez

DIRECTOR DE TESIS

DEDICATORIA

A mi Padre,

a mi Madre,

a mis Hermanos

Fabio

A mis Padres,

a mis Hermanos,

a mis Amigos

César

AGRADECIMIENTO

Al Ing. Luis Montalvo, por su acertada dirección en la realización de la presente Tesis.

Al Sr. Santiago Yépez, por su decidida y desinteresada colaboración y sobre todo por el consejo oportuno que fue determinante para la culminación de esta Tesis.

A todos aquellos profesores y compañeros que de una u otra forma colaboraron con este trabajo.

CAPITULO I

I. CONSIDERACIONES GENERALES

1.1 JUSTIFICACION

El DS5000T es un microcontrolador compatible con el estándar industrial i51 tanto en su set de instrucciones como en su configuración de pines. La familia de microcontroladores i51 es muy conocida y muy utilizada. El microcontrolador DS5000T tiene todas las caracteristicas del i8051 y adicionalmente el siguiente conjunto de prestaciones propias que lo hacen adecuado para constituirse en una herramienta didáctica y profesional útil:

a) Bateria de litio interna que preserva la memoria y las funciones de reloj en ausencia de energia.
b) 32 Kbytes de memoria incluidos en el chip que pueden ser utilizados como ROM y/o RAM.
c) Partición interna de memoria de programa y/o datos dinámica, realizable por el usuario.
d) Reloj/calendario en tiempo real incluido en el chip, lo cual permite el registro de eventos relacionados con fecha y hora.
e) Descarga del programa del usuario por medio de un puerto serial full-dúplex del propio chip.
f) Encriptor residente que protege el programa de un acceso no autorizado.

Estas características del microcontrolador permiten que se pueda realizar sistemas con poca circuiteria adicional, lo

que redunda en una simplicidad del hardware requerido para una aplicación determinada.

Por lo expuesto, se considera necesario desarrollar herramientas que faciliten la manipulación de este microcontrolador, lo cual se reflejará en una mayor utilización del mismo en la realización de proyectos didácticos y profesionales.

Adicionalmente, el sistema serviría también para desarrollar proyectos en base a microcontroladores de la familia i51.

1.2 OBJETIVOS

Considerando que la Facultad de Ingenieria Eléctrica de la Escuela Politécnica Nacional no cuenta con las herramientas que permitan desarrollar programas en los cuales se aprovechen todas las caracteristicas del microcontrolador DS5000T, nuestro objetivo es ofrecer un sistema útil que llene los requerimientos necesarios para personas que quieran desarrollar aplicaciones para el DS5000T.

Para cumplir con el objetivo planteado las herramientas a crear son un Depurador de Programas y un Kit de Desarrollo, las cuales le faciliten al programador la creación de sistemas basados en el microcontrolador DS5000T.

Tomando en cuenta la existencia de depuradores de programas para microcontroladores y microprocesadores, bastante conocidos y utilizados tales como AVSIM51, AVSIMZ80, AVSIM48 y PRO, el Depurador de Programas para el DS5000T debe ser similar a estos. Las caracteristicas principales que tienen son:

- Ejecutables en computadores personalales.
- Utilización intensiva de la pantalla del computador para visualizar los elementos más importantes para el programador.
- Manejo de opciones en base a menúes.

Por tanto, se debe desarrollar un paquete de software que permita visualizar en la pantalla de un computador IBM o compatible el efecto que produce la ejecución de un programa del usuario en el microcontrolador.

Un depurador es un medio para facilitar el desarrollo de programas, pero el objetivo final es llegar a obtener un sistema que tenga una utilización práctica. Por esto es necesario disponer de un kit de desarrollo, en el cual se pueda probar la ejecución del programa del usuario en el propio microcontrolador.

Para aprovechar la capacidad que tiene el DS5000T de recibir un programa mediante su puerto serial, el kit debe permitir la descarga de programas desde el computador personal en forma serial hacia el microcontrolador.

El kit de desarrollo estará conformado por una tarjeta donde se encuentre el microcontrolador DS5000T con las conexiones que permitan la comunicación serial con el computador utilizando el protocolo EIA RS-232C.

Como se manifestó anteriormente el depurador de programas y el kit de desarrollo tratan de asemejarse a paquetes existentes; sin embargo, las características propias del microcontrolador hacen que sea factible realizar una opción nueva y novedosa: "controlar la ejecución de un programa en el kit de desarrollo".

El software estará constituido por un conjunto de módulos que permitirá realizar las siguientes tareas:

a) Simulación, en un computador personal (IBM o compatible), de programas para el DS5000T con la posibilidad de observar en la pantalla del computador el efecto que produce el programa en el estatus del microcontrolador; pudiendo además establecer puntos de ruptura, contar los ciclos de máquina requeridos para la ejecución y modificar el estatus del microcontrolador mediante edición de la pantalla

b) Ejecución controlada desde un computador personal (IBM o compatible) de un programa del usuario en el kit de desarrollo, con las mismas posibilidades que para el módulo de simulación.

c) Descarga serial de programas para el microcontrolador DS5000T desde un computador personal (IBM o compatible), con la posibilidad de configurar la partición de memoria y encriptar el programa.

Se aspira a que el depurador y el kit de desarrollo para el DS5000T, se constituyan en una herramienta útil para estudiantes y profesionales que quieran desarrollar sistemas basados en este microcontrolador, pudiendo servir también como un instrumento para un mejor aprovechamiento de las materias relacionadas con hardware y software de microprocesadores que

ha optado por que el lenguaje de programación sea un lenguaje assembly. Siendo los computadores personales IBM y compatibles los más conocidos y difundidos, los cuales están basados en el microprocesador i8086, el programa ha sido desarrollado en el lenguaje assembly de este microprocesador.

El lenguaje assembly del microprocesador i8086 permite programación modular, lo que facilita el desarrollo del programa y además posibilita dividir el trabajo total en módulos individuales, los cuales son concebidos para cumplir una tarea especifica, y luego poder ser encadenados dentro del programa total el cual va a satisfacer los objetivos planteados.

Las tareas que deben ser realizadas por el programa general se han distribuido en los siguientes módulos:

INTHMAQ.ASM	Interfaz hombre máquina
DEP.ASM	Depurador de programas
ENS.ASM	Ensamblador de un mnemónico
DESENS.ASM	Desensamblador de un código de máquina
SIMULAR.ASM	Simulador de instrucciones del DS5000T en el computador personal.
COMSER.ASM	Administrador de comunicación serial

En la figura 2.1 se muestra con un diagrama de bloques la forma en que se encadenan estos módulos.

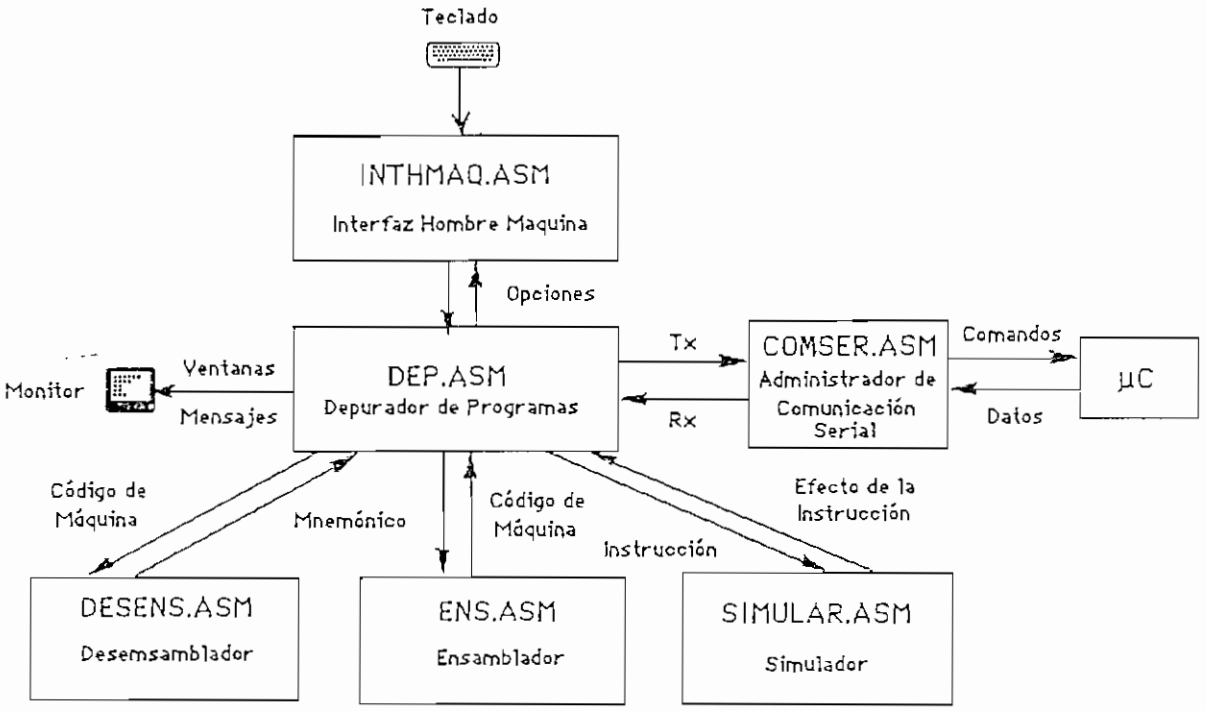


Figura 2.1.- Interrelaciones entre los módulos que conforman el programa

El módulo INTHMAQ.ASM es el que posibilita al usuario administrar el depurador, por medio de menúes le permite elegir el proceso que quiere desarrollar; una vez seleccionado el procedimiento a seguirse este módulo invoca a la rutina correspondiente la cual se encuentra en el módulo DEP.ASM.

El módulo DEP.ASM es el que realmente controla las tareas de depuración, en el se cumplen las rutinas mandadas a ejecutar desde el interfaz hombre máquina. En este módulo es donde se preparan y muestran ventanas y mensajes de información al usuario sobre los cambios que produce la acción por él seleccionada.

Una vez que el módulo DEP.ASM toma el control, en adelante es el que decide que tareas se deben cumplir y dependiendo de ellas transfiere control a los otros módulos.

La pantalla del depurador esta compuesta por ventanas, las cuales presentan el estatus del microcontrolador. Una de estas ventanas es la ventana de código, la misma que presenta el código desensamblado de las instrucciones adyacentes a la instrucción apuntada por el PC del usuario. Las rutinas del módulo DESENS.ASM se encargan de desensamblar el código de máquina correspondiente antes de mostrarlo en pantalla.

Dentro del depurador existe la posibilidad de reemplazar y ejecutar instrucciones ingresadas por el usuario, estas instrucciones son ingresadas con sus mnemónicos los mismos que

2.2 DESARROLLO DEL PROGRAMA PRINCIPAL

El interfaz hombre máquina (INTHMAQ.ASM) y el depurador (DEP.ASM) constituyen los módulos principales, ya que son los encargados de comandar las tareas realizadas por todos los otros módulos del programa.

A continuación se hará una descripción general de los procedimientos más importantes de cada uno de estos módulos, los procedimientos no mencionados y algunos otros detalles se encuentran debidamente explicados en los listados de los programas.

2.2.1 MODULO INTHMAQ.ASM

El módulo INTHMAQ.ASM es el encargado de mantener informado al usuario sobre todas las opciones del programa permitiéndole elígir, mediante menúes, aquella que le permita realizar la tarea por el deseada. Una vez cumplida la tarea seleccionada, este módulo recobra el control sobre las opciones del depurador.

Cuando al usuario se le presente un menú puede tomar alguna de estas opciones.

a) Presionar una de las flechas y moverse a una opción cualquiera.

b) Digitar la tecla ENTER entrando con ésto a cumplir la rutina asociada a la opción sobre la que estaba ubicado y luego a recibir el tratamiento adecuado de acuerdo al "tipo de función" de que se trate.

c) Digitar la letra clave de una opción, con lo que entra directamente a esta opción sin necesidad de ubicarse sobre ella. Realiza la rutina asociada y luego recibe el tratamiento adecuado al "tipo de función".

d) Cualquier otra alternativa no es tomada en cuenta.

Una vez ejecutada la rutina asociada a una opción existen varias alternativas (A esto se refieren las palabras "tipo de función" indicadas anteriormente). Estas alternativas son:

a) Se cumple la rutina asociada y se regresa al mismo menú que contiene la opción. Estas rutinas podrían llamarse "normales".

b) Se cumple la rutina asociada y se emerge un nivel en el árbol de opciones, es decir, se transfiere el control al menú "padre" del que contenía la opción. Estas funciones serán llamadas "inversas".

c) Se ejecuta la rutina y se profundiza un nivel, mostrando un nuevo menú, hijo de la opción que fue digitada. Estas opciones son del tipo "rama".

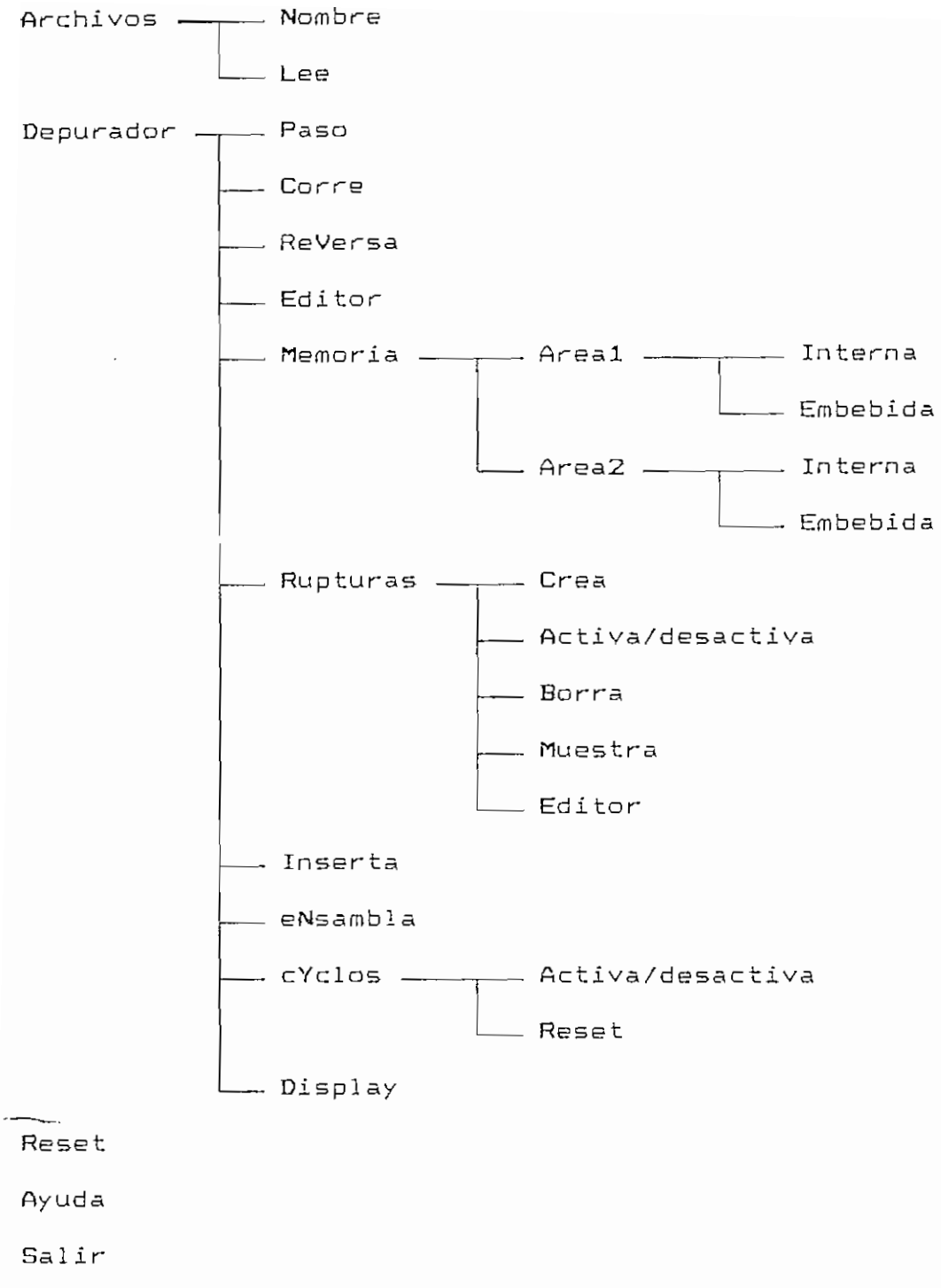


Figura 2.2.- Arbol de opciones del depurador

El interfaz hombre máquina usado para el paquete de depuración del DS5000T es una adaptación del controlador de flujo utilizado en la tesis de los Ingenieros Fernando Ortega

Cuando el usuario se posicione con las flechas sobre la subopción Crea, en la línea 24, aparecerá una pequeña explicación sobre la opción, el mensaje correspondiente en este caso es "Marca un punto de ruptura".

En este caso la opción Crea no tiene subopciones, sin embargo de lo cual se le asigna el menú de opciones principal; aunque el módulo interfaz hombre máquina no va a solicitar esta información, se la debe poner para no alterar el formato de la tabla.

En el momento que el usuario seleccione esta opción, se transfiere el control al sitio etiquetado rdrc. En este sitio se manda a ejecutar la rutina asociada a la opción, la cual se encuentra en el módulo DEP.ASM; cuando esta rutina se cumpla el módulo INTHMAQ.ASM nuevamente recobra el control del programa.

Por tratarse de una opción normal, representada con el número 1 en la tabla tabinf, el módulo INTHMAQ.ASM mantendrá el mismo menú de opciones.

En lo referente a los menués de opciones, éstos se indican en una cadena de caracteres con los nombres de las opciones, separados entre ellos por lo menos con un espacio (un 0 indica el fin de las opciones), a continuación está el número de opciones del menú y por último las letras claves para acceso rápido.

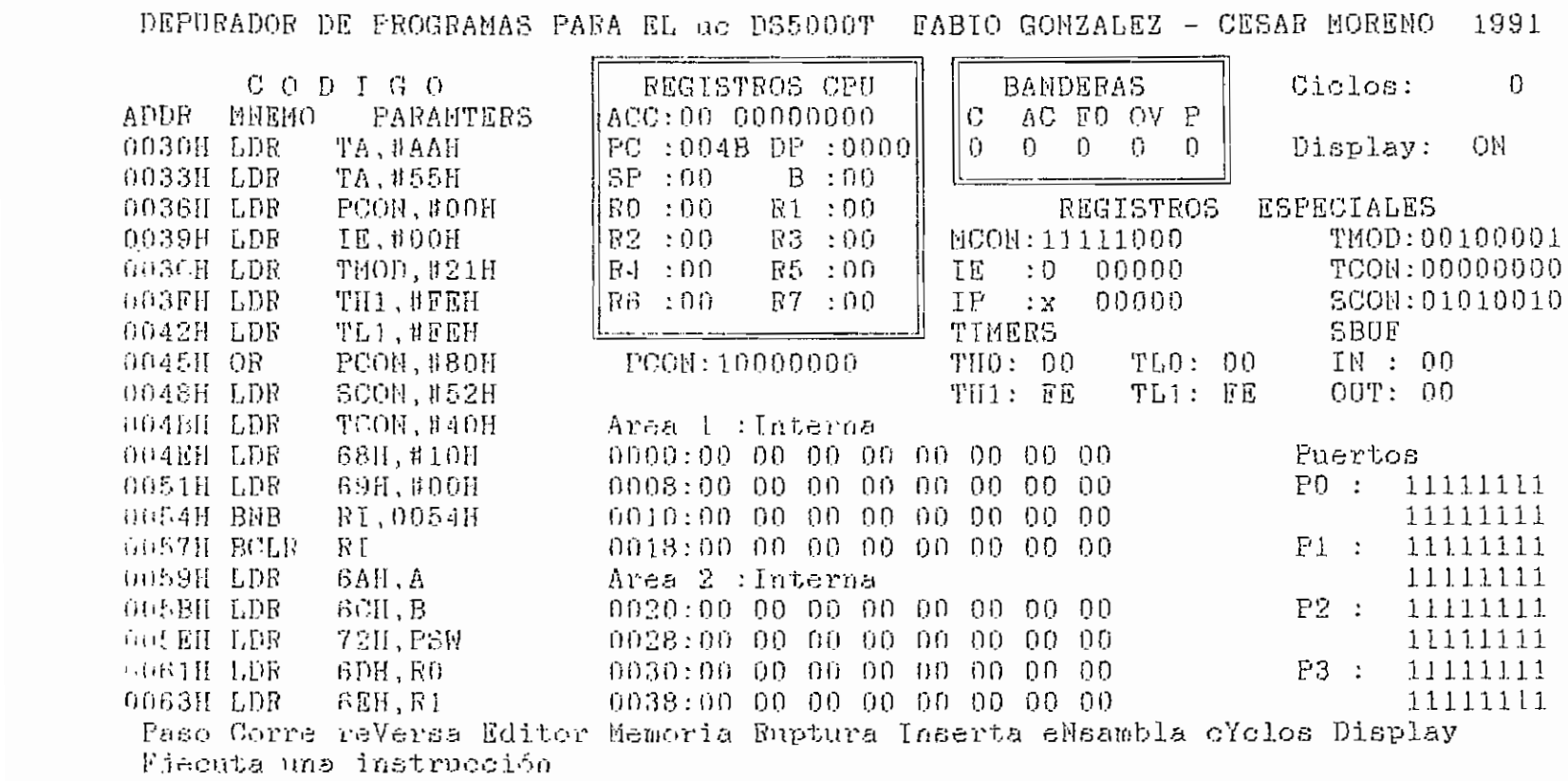


Figura 2.3.- Pantalla del depurador de programas para el DS5000T,

GETLINE: Se utiliza para ingresar cadenas de caracteres desde teclado para lo cual se requiere el nombre de la variable donde se desea guardar la cadena de caracteres.

GETKEY: Se lo utiliza frecuentemente para ingresar un solo caracter desde teclado, el cual puede ser para continuar con algún proceso o para responder a un cuestionamiento.

Adicionalmente al manejo de pantalla, en el módulo DEP.ASM están contenidas todas las rutinas asociadas a las opciones de los menúes de depuración, estas rutinas son controladas y enviadas a ejecutar desde el módulo INTHMAQ.ASM. Además de las rutinas que satisfacen las opciones del programa, en este módulo se encuentra la rutina inicial que es la encargada de inicializar diferentes párametros los cuales determinan que proceso se debe llevar a cabo en un determinado momento. Conviene mencionar que todos los procedimientos a seguirse para los diferentes modos de operación están incluidos en el módulo DEP.ASM, las diferenciaciones se las va haciendo dentro de las distintas rutinas que se cumplen para satisfacer con las tres opciones del depurador (simulación, ejecución controlada y carga serial).

A continuación se explican las rutinas más importantes de este módulo.

RINIC: Es la rutina inicial, la cual es ejecutada una vez que empieza a correr el programa. La primera tarea es

verificar si existe y no está alterado el archivo de ventanas (DALLAS.VNT), si este archivo no existe o está alterado, se presenta un mensaje indicando el error, y el programa termina su ejecución. Si el archivo de ventanas existe y no ha sufrido alteraciones se procede a verificar si se han ingresado párametros al invocar el programa, en caso de que si hayan párametros ingresados, estos son procesados y asignados al nombre del archivo del usuario o archivo objeto, y al modo de operación. Si el usuario no ingresa con el nombre del programa estos párametros, éstos serán indagados de manera que sea posible fijarlos para continuar con la ejecución del depurador. Una vez establecidos el nombre del programa del usuario y modo de operación se procede a leer y cargar en memoria el programa del usuario. Si en la lectura y carga de este programa no se tienen problemas se continúa con el flujo normal del depurador. Si se tienen problemas con el archivo del programa del usuario, el depurador termina su ejecución.

Lo anteriormente expuesto se ilustra en la figura 2.4.

LEE_ARC: Esta rutina puede ser invocada desde la rutina "rinic" o desde el módulo INTHMAQ.ASM. Recibe como entrada el nombre del archivo objeto y el modo de operación, de acuerdo al modo de operación el procedimiento a seguirse es diferente. Como primer paso se lee, verifica el formato INTEL y carga en memoria el archivo con el programa del usuario. Si se trabaja en modo de simulación no se realiza ninguna tarea adicional. En modo de ejecución controlada, este archivo es transmitido serialmente hacia el microcontrolador, luego de lo cual se

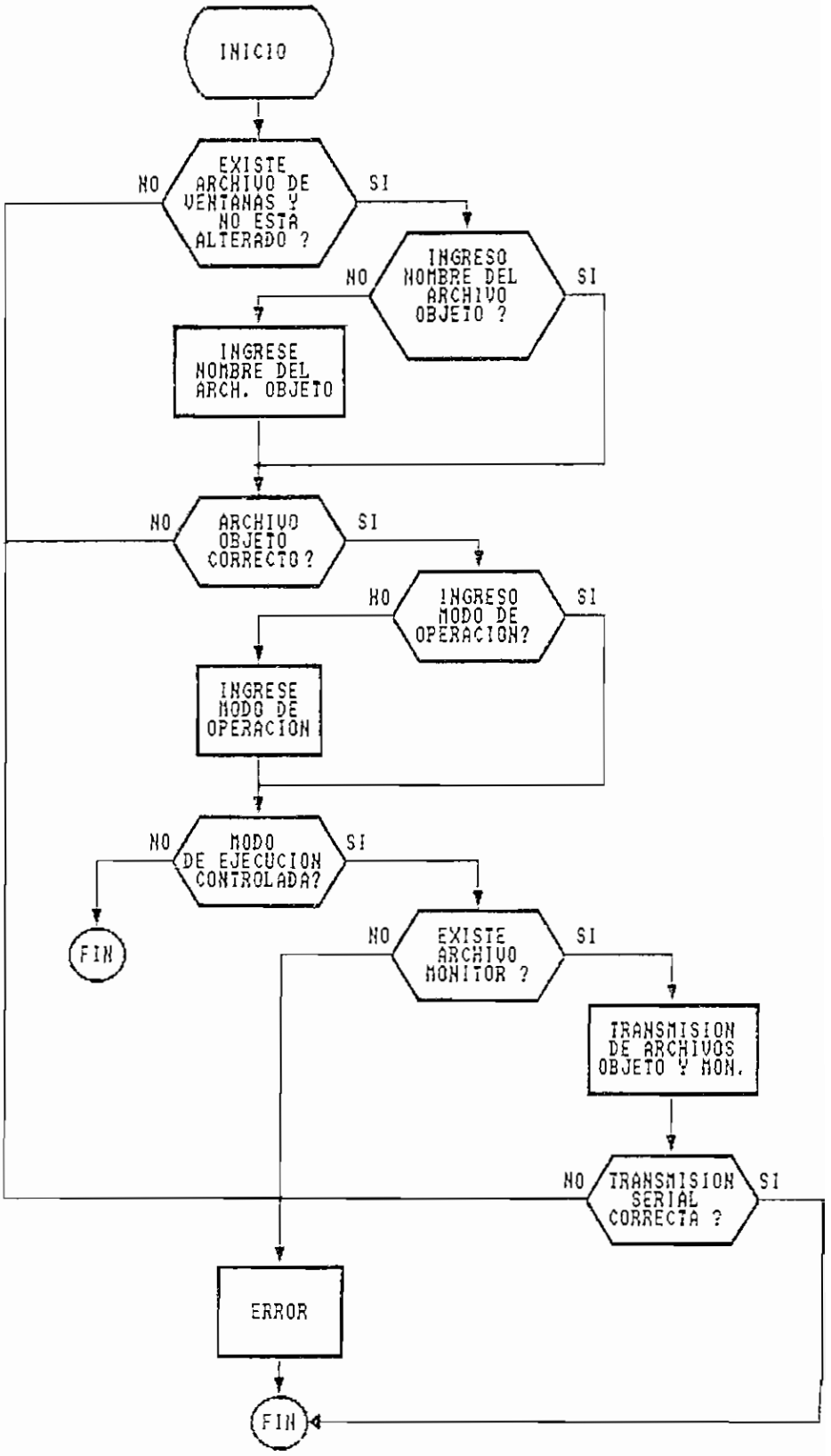


Figura 2.4.- Diagrama de flujo de la rutina RINIC.

procede a leer, verficar el formato INTEL, cargar en memoria y finalmente descargar serialmente el archivo monitor DALLAS.HEX hacia el microcontrolador. Para el modo de operación de carga serial, el archivo objeto es descargado serialmente hacia el microcontrolador, con lo que el depurador concluye la ejecución. Si durante la lectura y verificación de formato INTEL de los archivos objeto y monitor no se tiene error, el depurador continua la ejecución; en el caso contrario se setea el error, el cual será procesado de acuerdo a como fue invocada esta rutina, si ésta fue llamada desde la rutina "rinic", el depurador termina la ejecución, y si fue invocada desde el módulo INTHMAQ.ASM, el archivo previamente cargado permanece vigente como el archivo objeto.

INICIAL: En esta rutina están los procedimientos necesarios para la presentación de las distintas ventanas que constituyen la pantalla del depurador. Las ventanas deben ser preparadas antes de desplegarlas en la pantalla, esto se lo realiza cargando en los buffers de la ventana las variables correspondientes; estas variables antes de guardarse en los buffers deben ser procesadas convenientemente. En modo de simulación las variables son extraídas desde la memoria del computador personal en tanto que en modo de ejecución controlada estos valores son leídos desde el microcontrolador mediante comunicación serial.

DEP_PASO: Es la rutina que satisface la opción Paso del menú de depuración. Si se trabaja en modo de simulación invoca

a la rutina "ejecutor", dentro del módulo SIMULAR.ASM. En modo de ejecución controlada se emite un comando hacia el microcontrolador ordenando la ejecución de la instrucción apuntada por el PC del usuario. Si la cuenta de ciclos de reloj está activada, se actualiza el número de ciclos. Esta rutina puede ser mandada a ejecutar desde la rutina dep_corre, por lo que no siempre se actualiza pantalla.

DEP_CORRE: Va simulando o ejecutando secuencialmente las instrucciones del listado del programa del usuario. Se cumple un lazo en el que la instrucción apuntada por el PC del usuario es mandada a ejecutar a través de la rutina "paso", para salir de este lazo se constata que desde el teclado el usuario haya presionado la tecla ESC, la dirección de la instrucción a ejecutarse esté dentro de la tabla de puntos de ruptura o que, si la actualización de pantalla está desactivada, el programa del usuario entró en un lazo. En la figura 2.5 se ilustra la ejecución secuencial.

INSERTA: Esta rutina permite insertar una nueva instrucción en el listado del programa del usuario, reemplazando al código original del programa. Esta instrucción es ingresada a través de su mnemónico, el cual es procesado en la rutina "assem" del módulo ENS.ASM, si no hay error en el mnemónico éste es ensamblado a su código de máquina, en modo de simulación el código es cargado en la memoria del computador, y en modo de ejecución controlada se transmite el código de máquina de la instrucción ingresada la cual

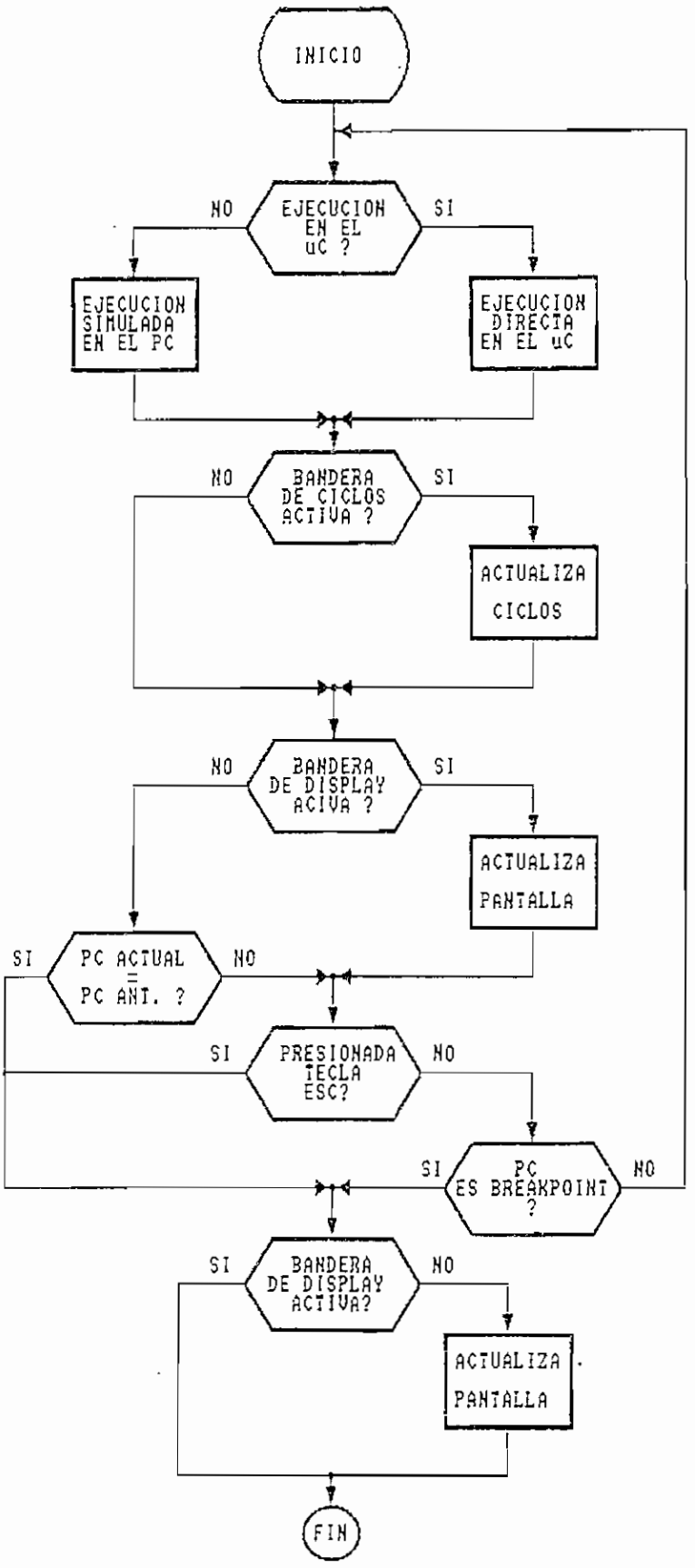


Figura 2.5.- Diagrama de flujo de la rutina DEP_CORRE.

finalmente es cargada en el microcontrolador.

ENSAMBLA: Ensambla y ejecuta fuera del listado del programa una instrucción que es ingresada por el usuario, esta instrucción es ensamblada por la rutina "assem" del módulo ENS.ASM, si no hay error es ejecutada. Si la instrucción es de salto el PC del usuario es alterado, caso contrario el PC del usuario no cambia.

DESCARGA: Es llamada cuando se trabaja en modo de ejecución controlada; mediante comunicación serial, el estatus del microcontrolador, registros, áreas de memoria y puertos, son ingresados al computador personal para ser procesados. Los datos están codificados en formato INTEL para corrección de errores. En caso de existir errores se ordena una nueva transmisión.

CREA_RUPT: Permite ingresar un nuevo punto de ruptura en el programa, estos puntos de ruptura son almacenados en una tabla en la cual se guarda información sobre el punto de ruptura, número, dirección, estado (activo/desactivo) y tipo (dinámico/estático). Máximo se pueden ingresar ocho puntos de ruptura y es necesario que las direcciones de todos estos puntos sean diferentes.

CARGA: Se la utiliza cuando se trabaja en modo de carga serial. Mediante comunicación serial se descarga el archivo con el programa del usuario. Permite además que el usuario

2.3 DESARROLLO DE MODULOS AUXILIARES

2.3.1 MODULO ENSAMBLADOR

El módulo ensamblador ENS.ASM permite convertir cualquier mnemónico válido del DS5000T en su código de máquina correspondiente.

Este módulo es utilizado para realizar correcciones en el programa del usuario durante la simulación o la depuración (opción INSERTA). También es utilizado cuando se quiere ejecutar una instrucción fuera del código del programa que se está depurando o simulando (opción ENSAMBLA).

La rutina que debe ser llamada para realizar este proceso se denomina "assem" la cual hace que se ejecuten dos pasos de ensamblado mediante las rutinas "enspaso1" y "enspaso2", de la manera que se muestra en la figura 2.6.

El primer paso de ensamblado, mediante la rutina "enspaso1", se inicia verificando que los caracteres iniciales de la cadena alfanumérica a ensamblar coincidan con alguno de los nombres de instrucciones del DS5000T listados en la tabla "mnemo". Este proceso lo realiza la rutina "encontrar". Si no existe ningún nombre que coincida con los primeros caracteres, el procedimiento de ensamblado termina, utilizando la bandera de carry para indicar que existió un error. Si se encontró un nombre de instrucción válido, a continuación se analiza

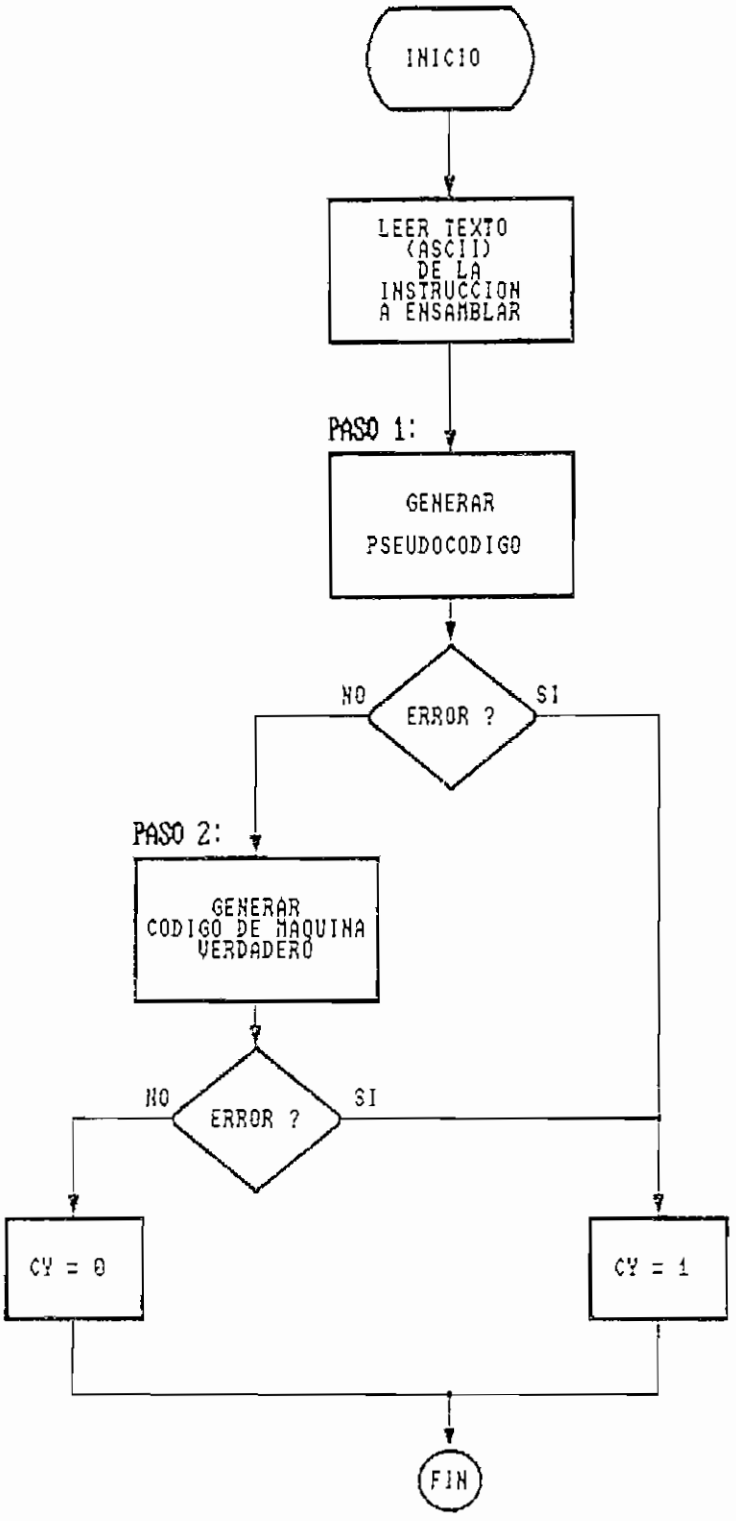


Figura 2.6.- Ensamblado de una instruccion.

cuántos parámetros debe tener esa instrucción y se los busca mediante la rutina "buscar_arg"; si el número de argumentos no coincide con el que debe tener la instrucción, también termina el primer paso de ensamblado indicando el error. Conviene indicar que entre el nombre de la instrucción y los parámetros debe existir al menos un espacio y el separador entre parámetros debe ser una coma.

Cuando se desea ensamblar una instrucción en donde interviene algún byte o bit de los registros de función especial (SFR's), se puede utilizar como parámetro la dirección hexadecimal o el nombre respectivo del bit o byte. Los nombres aceptados por este módulo ensamblador se listan en las tablas 2.1 y 2.2.

BYTE	DIRECCION	BYTE	DIRECCION
P0	80	P2	A0
SP	81	IE	A8
DPL	82	P3	B0
DPH	83	IP	B8
PCON	87	EK0	C1
TCON	88	EK1	C2
TMOD	89	EK2	C3
TL0	8A	EK3	C4
TL1	8B	EK4	C5
TH0	8C	MCON	C6
TH1	8D	TA	C7
P1	90	PSW	D0
SCON	98	ACC	E0
SBUF	99	B	F0

Tabla 2.1.- Bytes directamente direccionables

FF									
	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0	
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	CY	AC	F0	RS1	RS0	OV		P	
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
	RWT			PS	PT1	PX1	PT0	PX0	
B8	BF	--	--	BC	BB	BA	B9	B8	IP
	RD	WR	T1	T0	INT1	INT0	TXD	RXD	
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA			ES	ET1	EX1	ET0	EX0	
A8	AF	AE	AD	AC	AB	AA	A9	A8	IE
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
98	9F	9E	9D	9C	9B	9A	99	98	SCON
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	
90	97	96	95	94	93	92	91	90	P1
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
88	8F	8E	8D	8C	8B	8A	89	88	TCON
	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	
80	87	86	85	84	83	82	81	80	P0

Tabla 2.2.- Bits directamente direccionables

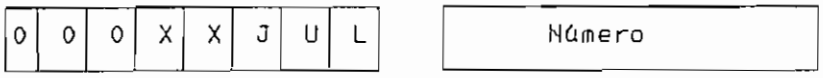
Las rutinas que chequean si los parámetros coinciden con alguno de estos nombres especiales son "encontrar", para los bytes directamente direccionables y "encontrar_bit", para los bits directamente direccionables.

Si no existen errores en el primer paso de ensamblado, la rutina "enspaso1" traduce la cadena de caracteres a ensamblar a un pseudocódigo numérico en el cual se han considerado todos los posibles elementos que se pueden encontrar en un mnemónico del DS5000T. El pseudocódigo correspondiente al nombre de la instrucción se ha escogido de tal manera que sea igual al primer byte de código de la instrucción ensamblada o pueda ser fácilmente convertido a dicho primer byte.

Dentro del mnemónico de una instrucción existe el nombre de la instrucción y los parámetros. Los parámetros válidos para una instrucción del microcontrolador DS5000T pueden ser los siguientes: registro, registro índice, memoria (bytes o bits directamente direccionables), dato de 8 bits, dato de 16 u 11 bits.

El pseudocódigo generado para el nombre de la instrucción y cada uno de los parámetros pueden ser de hasta 3 bytes; en el primer byte, los 3 bits más significativos indican a qué corresponde el pseudocódigo. A continuación se explica como está estructurado el pseudocódigo para cada uno de los posibles elementos de una instrucción del DS5000T:

NOMBRE DE INSTRUCCION:



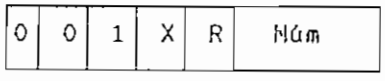
J = 1 si PJMP ó PCALL
U = 1 si la instrucción admite solo un parámetro
0 en otro caso
L = 1 si la instrucción no admite parámetros
0 en otro caso

Número = número de la instrucción, generalmente igual al primer byte del código de máquina de la instrucción

PSEUDOCODIGOS ASIGNADOS A LOS NOMBRES DE LAS INSTRUCCIONES:

INSTR.	Nº	INSTR.	Nº	INSTR.	Nº	INSTR.	Nº
NOP	00	PJMP	01	JMP	02	RR	03
INC	04	BBC	10	PCALL	11	CALL	12
RRC	13	DEC	14	BB	20	RET	22
RL	23	ADD	24	BNB	30	RETI	32
RLC	33	ADC	34	BC	40	OR	42
BNC	50	AND	52	BZ	60	XOR	62
BNZ	70	LDR	74	BRA	80	LDC	83
DIV	84	LDB	92	SBC	94	MUL	A4
BCOM	B2	COM	B3	CBNE	B4	PUSH	C0
BCLR	C2	CLR	C3	SWAP	C4	EXH	C5
PULL	D0	BSET	D2	DAA	D4	DBNZ	D5
EXHD	D6	LDX	E0				

REGISTRO:



R = 0 si se trata de uno de los registros R0 a R7
1 en otro caso

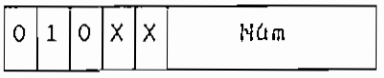
R = 0 -->

Núm	Registro
0 0 0	R0
0 0 1	R1
0 1 0	R2
0 1 1	R3
1 0 0	R4
1 0 1	R5
1 1 0	R6
1 1 1	R7

R = 1 -->

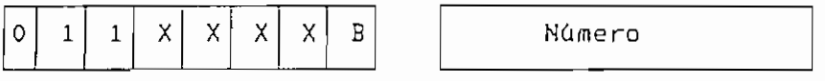
Núm	Registro
0 0 0	A
0 0 1	DPTR
0 1 0	AB
0 1 1	C

REGISTRO INDICE:



Núm	Registro
0 0 0	@R0
0 0 1	@R1
0 1 0	@DPTR
0 1 1	@A+DPTR
1 0 0	@A+PC

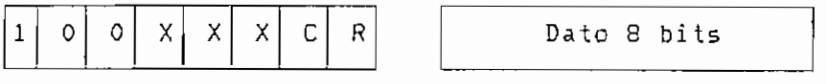
MEMORIA:



B = 1 si se trata de un byte directamente direccionable
0 si se trata de un bit directamente direccionable

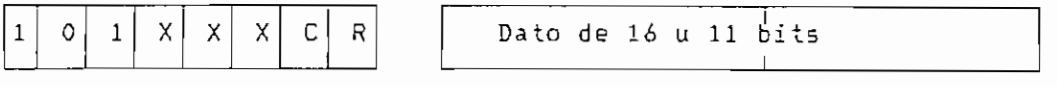
Número = código hexadecimal del byte o bit

DATO DE 8 BITS:



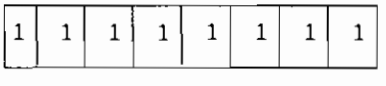
R = 0 si no existe el simbolo #
1 si existe el simbolo #
C = 1 si existe el simbolo /
0 si no existe el simbolo /

DATO DE 11 ó 16 BITS:



D = 1 si es dato de 16 bits
0 si es dato de 11 bits
R = 1 si existe el simbolo #
0 si no existe el simbolo #

FIN DE PSEUDOCODIGO:



primer pseudocódigo para escoger el procedimiento adecuado, dentro de la rutina "enspaso2", que determine si los parámetros que acompañan al nombre de la instrucción son válidos para esa instrucción. El chequeo del pseudocódigo correspondiente a los parámetros se lo realiza por medio de la rutina "tipo". Si alguno de los parámetros no es válido, o falta algún parámetro, la rutina "enspaso2" termina indicando el error; caso contrario, la rutina finaliza con el código de la instrucción almacenado en memoria.

Esta forma de realizar el ensamblado en dos pasos, facilita el análisis de la cadena de caracteres a ensamblar ya que una vez que se ha realizado el primer paso de ensamblado, se dispone de una serie de códigos numéricos en donde es más fácil analizar la estructura y la sintaxis de la instrucción que se desea ensamblar.

La rutina "assem" también entrega, en el registro CX. la longitud en bytes del código de máquina generado para que el procedimiento invocante pueda saber cuántos bytes de memoria fueron ocupados por la instrucción ensamblada.

El diagrama de flujo correspondiente a este módulo se muestra en la figura 2.7.

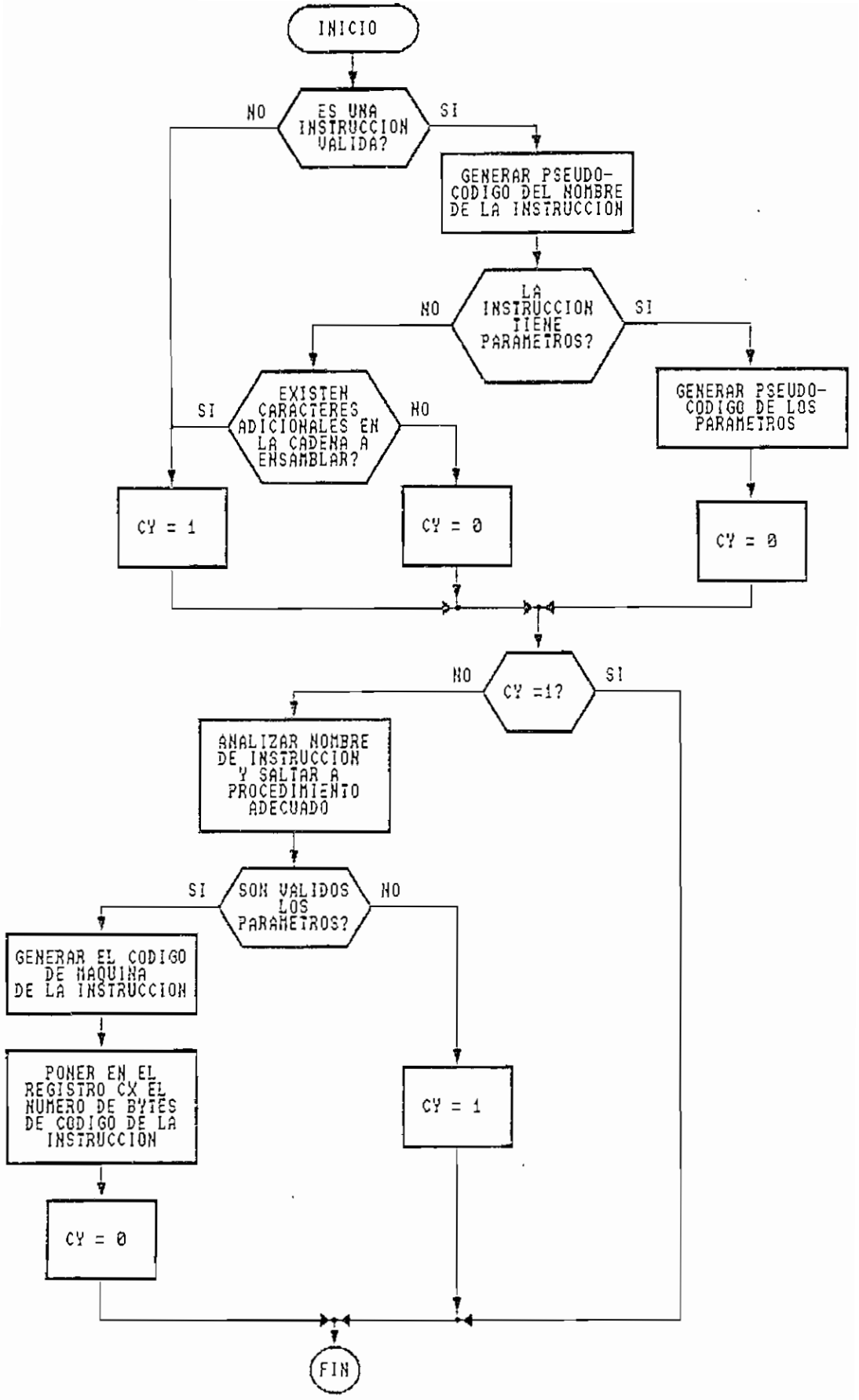


Figura 2.7.- Diagrama de flujo del modulo ensamblador.

2.3.2 MODULO DESENSAMBLADOR

El módulo desensamblador DESENS.ASM permite convertir el código de máquina de cualquier instrucción válida para el DS5000T en una cadena alfanumérica que contenga el mnemónico de la instrucción.

La rutina que se debe invocar para hacer el desensamblado es "desensdet", la cual genera 4 caracteres hexadecimales correspondientes al valor del contador de programa (PC) donde se inicia el código de la instrucción, luego llama a la rutina "desens", para obtener el nombre y parámetros de la instrucción, y finalmente reformatea la cadena de caracteres para presentarla en pantalla.

Cuando se necesita desensamblar una instrucción se lee el primer byte de código, para poder saber con precisión a qué instrucción corresponde el código que se pretende desensamblar y además saber cuántos bytes se deben leer para obtener los parámetros de la instrucción. Con este primer byte de código la rutina "desens" escoge el procedimiento adecuado para realizar el desensamblado. Para poder hacer esto, se ha creado en memoria la tabla "casos" donde está definido, en base al primer byte de código, qué rutina debe ser invocada para desensamblar la instrucción.

Los nombres de las instrucciones están definidos en las tablas "oper00_04" hasta "operf5_ff", las cuales serían

fácilmente cambiadas si se deseara generar otros nombres de instrucciones; por ejemplo, los nombres estándar de instrucciones de la familia de microcontroladores i51. Los nombres de las instrucciones son leídos desde estas tablas por la rutina "readarray".

Cuando uno de los parámetros de la instrucción desensamblada sea un registro de función especial, en la cadena de caracteres se pone el nombre del registro en vez de su dirección en la memoria interna del microcontrolador. Los nombres de los bytes y bits directamente direccionables se encuentran en la tablas 2.1 y 2.2, presentadas en la descripción del módulo ensamblador y son leídos por los macros "trans_bit", en el caso de bits directamente direccionables, y "trans_byte", si es un byte directamente direccionable. Si no se encuentra ningún nombre especial, simplemente se colocará la dirección en hexadecimal del registro o localidad de memoria que interviene como parámetro de la instrucción.

Una vez que se ha generado la cadena de caracteres correspondiente a la instrucción desensamblada, la rutina "desensdet" formatea la cadena de caracteres de forma de presentar las instrucciones en la pantalla de una manera uniforme (4 caracteres del valor del contador del programa donde se encuentra el primer byte de la instrucción, 5 caracteres de nombre de la instrucción, 14 caracteres de parámetros de la instrucción). En el caso de que alguna instrucción tenga menos caracteres, en cualquiera de los

campos especificados, se completa el número de caracteres mediante espacios en blanco.

Adicionalmente, la rutina "desens" proporciona, en el registro CX, la longitud en bytes del código de máquina de la instrucción desensamblada, para que el programa principal pueda ubicar la siguiente instrucción del programa del usuario.

Este módulo contiene también las rutinas que permiten cargar un programa formato INTEL de 8 bits en la memoria del computador. Estas rutinas chequean que se cumpla la sintaxis del formato INTEL de 8 bits y verifican los cheksums.

Cuando se desea cargar un programa del usuario en la memoria del computador, se debe invocar a la rutina "intel" del módulo MACDESEN.ASM. Esta rutina a su vez llama a la rutina "lin_intel" del mismo módulo la cual se encarga de procesar cada una de las líneas del archivo de formato Intel de 8 bits. Se ha asumido que el archivo con el programa del usuario tendrá un máximo de 4000 líneas que equivale a un programa de 64 Kbytes de extensión. Cada una de las líneas del archivo de formato Intel tiene la siguiente estructura:

```
:03000B0002058467
```

donde:

- : inicio de línea
- 03 número de bytes de código presentes en la línea
- 000B dirección de memoria desde donde se carga el código
- 00 byte propio del formato Intel de 8 bits
- 02...84 bytes de código
- 67 Checksum. Este byte es calculado como el complemento de dos de la suma módulo OFFH de los restantes bytes de la línea.

En la rutina "lin_intel" se chequea que existan los dos puntos iniciales de cada línea, a continuación se obtiene el número de bytes de código que existen en la línea y la dirección de memoria donde serán almacenados esos bytes de código; luego se procede a leer y guardar en memoria esos bytes de código. Conforme se van leyendo los bytes de la línea Intel, se va calculando un checksum que posteriormente será comparado con el último byte de la línea que corresponde al checksum generado cuando se ensambló el programa del usuario, si el checksum calculado coincide con este último byte, se pasa a procesar la siguiente línea del archivo, de no ser asi, se setea la bandera de carry para informar al procedimiento que llamó a esta rutina que existió un error. Finalmente, esta rutina chequea que exista un "line feed" y un "carry return" al final de la línea de archivo Intel. De no ser asi, igualmente se setea el carry para indicar que existió un error. Cada vez que se ha finalizado el procesamiento de una línea del archivo, la rutina INTEL verifica si existen los dos

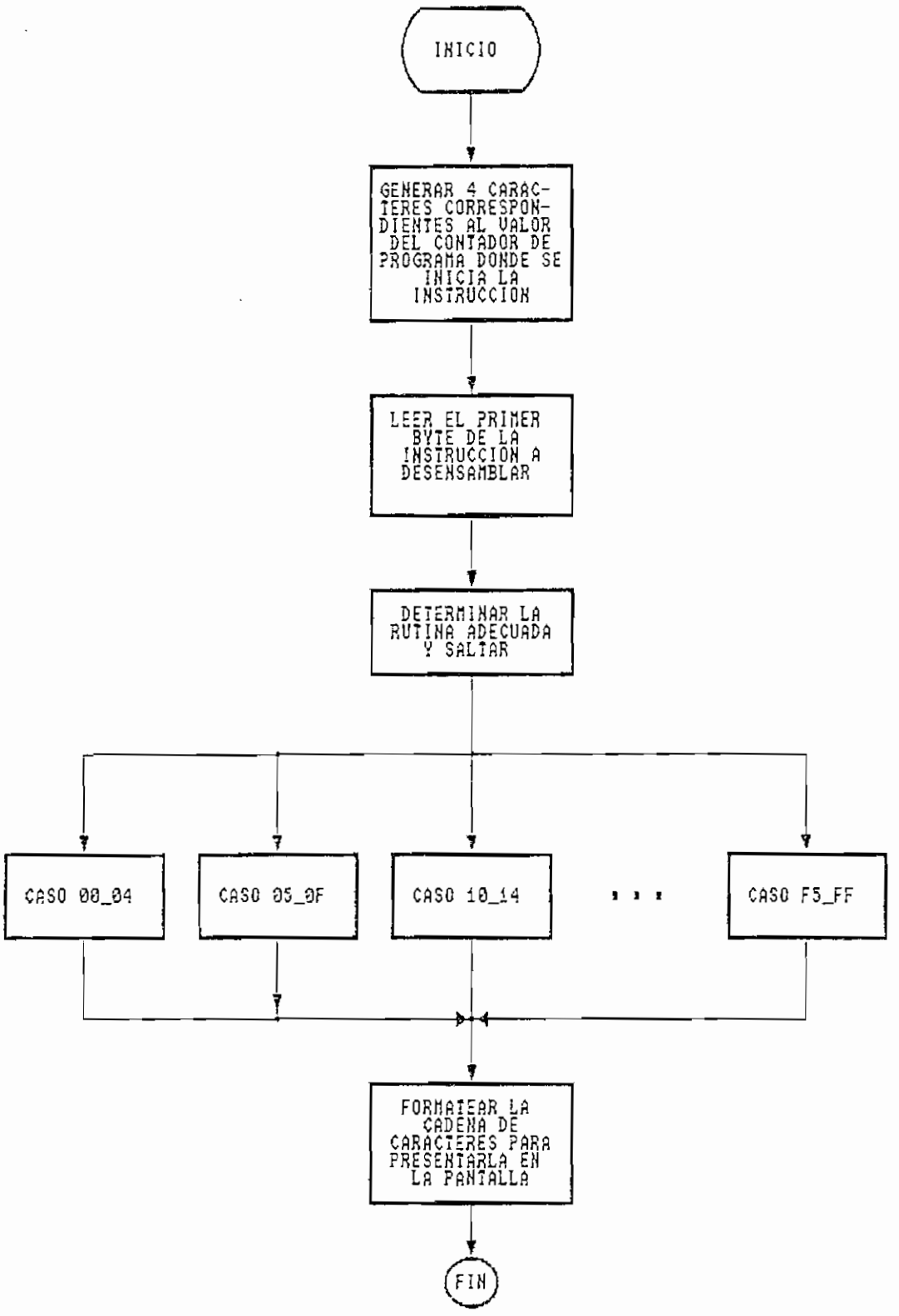


Figura 2.8.- Diagrama de flujo del modulo desensamblador.

2.3.3 MODULO SIMULADOR

El módulo SIMULAR.ASM permite simular la ejecución de cualquier instrucción válida para el DS5000T; incluida la simulación de TIMERS/COUNTERS, interrupciones, puerto serial y reloj en tiempo real interno del DS5000T.

La simulación de una instrucción la realiza la rutina "ejecutor" basándose en la información proporcionada por el fabricante sobre el efecto que tiene una determinada instrucción en el estatus del microcontrolador. Este proceso se inicia leyendo el primer byte del código de la instrucción para poder determinar de qué instrucción se trata y además saber cuántos bytes se deben leer para obtener los parámetros que completan la instrucción.

Analizando este primer byte, se determina cuál es la rutina apropiada para realizar la simulación y se salta a esa rutina. Las rutinas que realizan la simulación de las instrucciones se las ha nombrado "ejec_00" hasta "ejec_ff" y el procedimiento para escoger estas rutinas es similar al descrito en el módulo desensamblador para las rutinas que hacen el desensamblado de las instrucciones del DS5000T.

La simulación se realiza en la memoria del computador personal en donde han sido definidas ciertas localidades de memoria para que actúen como registros, memoria interna del microcontrolador, banderas, etc. Si una determinada

instrucción altera las banderas, la determinación de cuáles banderas deben ser modificadas se realiza mediante la ejecución de una instrucción similar en el microprocesador i8086. Se hace esto porque analizando las instrucciones del DS5000T que alteran banderas, se notó que siempre existe una instrucción equivalente del i8086 que altera las banderas con iguales criterios.

Estas instrucciones se listan en la tabla 2.3.

Instrucción del DS5000T	Instrucción del 8086
RRC A	RCR ACC,1
RLC A	RCL ACC,1

ADD A,#data
ADD A,direct
ADD A,@Ri
ADD A,Rn
	ADD ACC,AL

ADC A,#data ADC A,direct ADC A,@Ri ADC A,Rn	ADC ACC,AL
SBC A,#data SBC A,direct SBC A,@Ri SBC A,Rn	SBB ACC,AL
DIV AB	DIV B
MUL AB	MUL B
DAA	DAA

Tabla 2.3.- Instrucciones que alteran banderas.

Una vez que se ha simulado una instrucción que modifica

el estado de las banderas del DS5000T, el seteo de las mismas se hace en base al estado de las banderas del i8086 mediante la rutina "setflags".

Dentro de las instrucciones del DS5000T existen ciertos casos que requieren un tratamiento especial al realizar la simulación, estos son:

- En las instrucciones con los puertos de entrada/salida, se ha considerado que en los puertos existen latches internos y pines externos. En las instrucciones que escriben en los puertos, se realiza la escritura en el latch y en el pin, mediante los macros "check_pin1" y "check_pin2", si la instrucción solo modifica un pin, o con el macro "check_port" si la instrucción afecta a todos los pines de un puerto. También se ha considerado la existencia de instrucciones que leen los latches y no los pines del puerto; estas instruciones se muestran en la tabla 2.4.

MNEMONICO	DESCRIPCION
AND	AND lógico
OR	OR lógico
XOR	XOR lógico
BBC	Saltar si Bit = 1 y hacer Bit = 0
BCOM	Complementar un bit
INC	Incrementar
DEC	Decrementar
DBNZ	Decrementar y saltar si no es cero
MOV Px.n,C	Mover el carry al bit n del puerto x
CLR Px.n	Hacer cero al bit n del puerto x
SET Px.n	Hacer uno al bit n del puerto x

Tabla 2.4.- Instrucciones especiales con los puertos

- Cuando se simulan instrucciones con los registros MCON y PCON existen ciertos bits que solo pueden ser accedidos a través del registro Timed Access (TA). En ese caso, para permitir el acceso a esos bits se chequea que se haya cumplido la secuencia de instrucciones siguiente:

```
LDR  TA,#0AAH
LDR  TA,#55H
     .
     .
     .
```

Esta verificación se realiza mediante la rutina "check_ta" y si está habilitado el acceso por medio del registro TA, el seteo de los bits especiales de MCON y PCON se efectúa mediante los macros "check_dir1" y "check_dir2".

Una vez que se realiza la simulación de cualquier instrucción, el siguiente paso es simular los TIMERS/COUNTERS, para lo cual primero se chequea el registro TMOD para determinar si estos registros deben operar como temporizadores, si es asi la simulación es realizada mediante la rutina "timers" y en ese caso los registros se van incrementando conforme se simula la ejecución de las instrucciones, de acuerdo al número de ciclos de máquina requeridos por cada una de las instrucciones. Se puede simular todos los modos de operación de los TIMERS/COUNTERS asi como las interrupciones provocadas por el overflow de estos registros.

Luego se verifica si están habilitadas las interrupciones, de ser así, si existe alguna interrupción se procede a simular el salto a la rutina de atención a la interrupción. En la simulación de la atención a una interrupción se toma en cuenta la secuencia de prioridad de interrupciones propia del microcontrolador y además se verifica si ese orden de prioridad no fue alterado por el usuario, por medio del registro de prioridad de interrupciones.

Si se está simulando la rutina de atención a una interrupción, solo se atenderá una nueva interrupción si aquella ha sido programada como de máxima prioridad en el registro de prioridad de interrupciones.

Las interrupciones externas son simuladas mediante la edición en la pantalla de los pines 2 y 3 del puerto 3. La simulación de la interrupción serial se realiza en la rutina "ejecutor", donde lo que se hace es setear las banderas RI o TI del registro SCON cuando se realiza alguna operación con el registro SBUF. Las interrupciones por overflow de los TIMERS/COUNTERS son simuladas mediante las rutinas que realizan el procesamiento de los TIMERS/COUNTERS, las mismas que se mencionaron anteriormente.

La simulación del reloj en tiempo real se realiza en base a la información de fecha y hora del computador personal, transferida a los registros del ECC mediante las rutinas

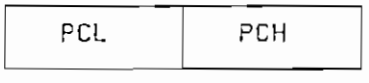
"lee_hora", "lee_fecha" y "write_ecc". El acceso al reloj en tiempo real se realiza mediante las instrucciones de movimiento de datos hacia memoria externa (LDX), cuando el bit ECE2 del registro MCON ha sido seteado a 1. Esto significa que el reloj en tiempo real es visto por el microcontrolador como una parte más de la memoria que puede direccionar. En la simulación del reloj en tiempo real, lo primero que se analiza es que se hayan realizado 64 lecturas para resetear ese reloj, existe una bandera en el simulador que controla este hecho. De no realizarse este reseteo del ECC, las subsiguientes operaciones con el ECC producirán resultados indeterminados tal como ocurre en el microcontrolador. Una vez que se ha ejecutado este reset del ECC, a continuación se espera que el programa del usuario escriba en el ECC un patrón de 64 bits (referirse al Anexo D) que van a posibilitar que se tenga acceso al mismo.

Si alguno de los bits que se van escribiendo no coincide con ese patrón, se bloquea el acceso al ECC y se debe reinicializar el proceso desde el reset del ECC. Si todos los 64 bits corresponden a este patrón, se tendrá acceso al ECC y se podrá leer o escribir bit a bit la información de fecha y hora del ECC. Cuando se escribe en el ECC, la rutina "check_ecc" verifica que ciertos bits que no pueden ser modificados, permanezcan inalterados a pesar de que el usuario intente escribir en ellos. La información escrita en el ECC es transferida al reloj-calendario del computador personal por las rutinas "igual_hora" e "igual_fecha".

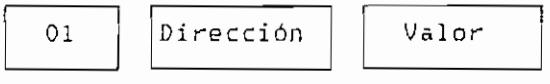
Dentro del depurador se ha incluido la posibilidad de simular ejecución reversa, lo cual se realiza invocando a la rutina "reversa". Para hacer posible esta opción, antes de simular la ejecución normal de una instrucción, se guardan en un buffer de 32 bytes los parámetros que serán modificados por esa instrucción. De esta manera, al realizar la simulación de la ejecución reversa simplemente se recuperan desde ese buffer los valores que tenían los parámetros antes de ejecutarse la instrucción. Se ha previsto que se puedan ejecutar en forma reversa hasta 64 instrucciones, por lo cual se ha reservado en memoria un espacio de (64x32) bytes para preservar los parámetros de las instrucciones. Una vez que se han ejecutado 64 instrucciones del programa del usuario, las siguientes reemplazan a las primeras de tal manera que siempre es posible realizar la ejecución reversa de hasta 64 instrucciones. El buffer para preservar los parámetros de las instrucciones ha sido estructurado como un stack tipo LIFO.

La preservación de parámetros para ejecución reversa se hace utilizando un código en el que se especifica cuál es el parámetro que se ha guardado y cuál era su valor antes de la ejecución de la instrucción. Este código es el siguiente:

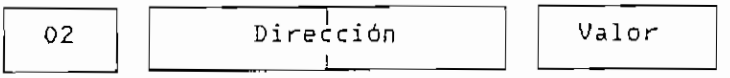
CONTADOR DE PROGRAMA:



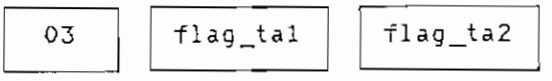
MEMORIA RAM INTERNA:



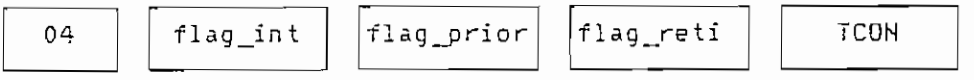
MEMORIA RAM EMBEBIDA:



BANDERAS PARA SIMULAR INSTRUCCIONES CON TA:

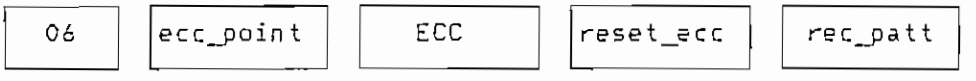


BANDERAS PARA SIMULAR INTERRUPCIONES:

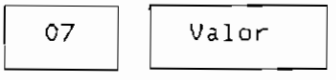


TIMERS:

RELOJ / CALENDARIO (ECC):



CICLOS:



Por ejemplo, antes de ejecutar la instrucción:

LDR A,#34

asumiendo que los Timers están activados, el PC tiene el valor 72BH y en el acumulador hay el valor 56H, se guardaría la siguiente información para ejecución reversa de la instrucción:

07 2B 01 E0 56 05 TL0 TH0 TL1 TH1 TCON 07 01

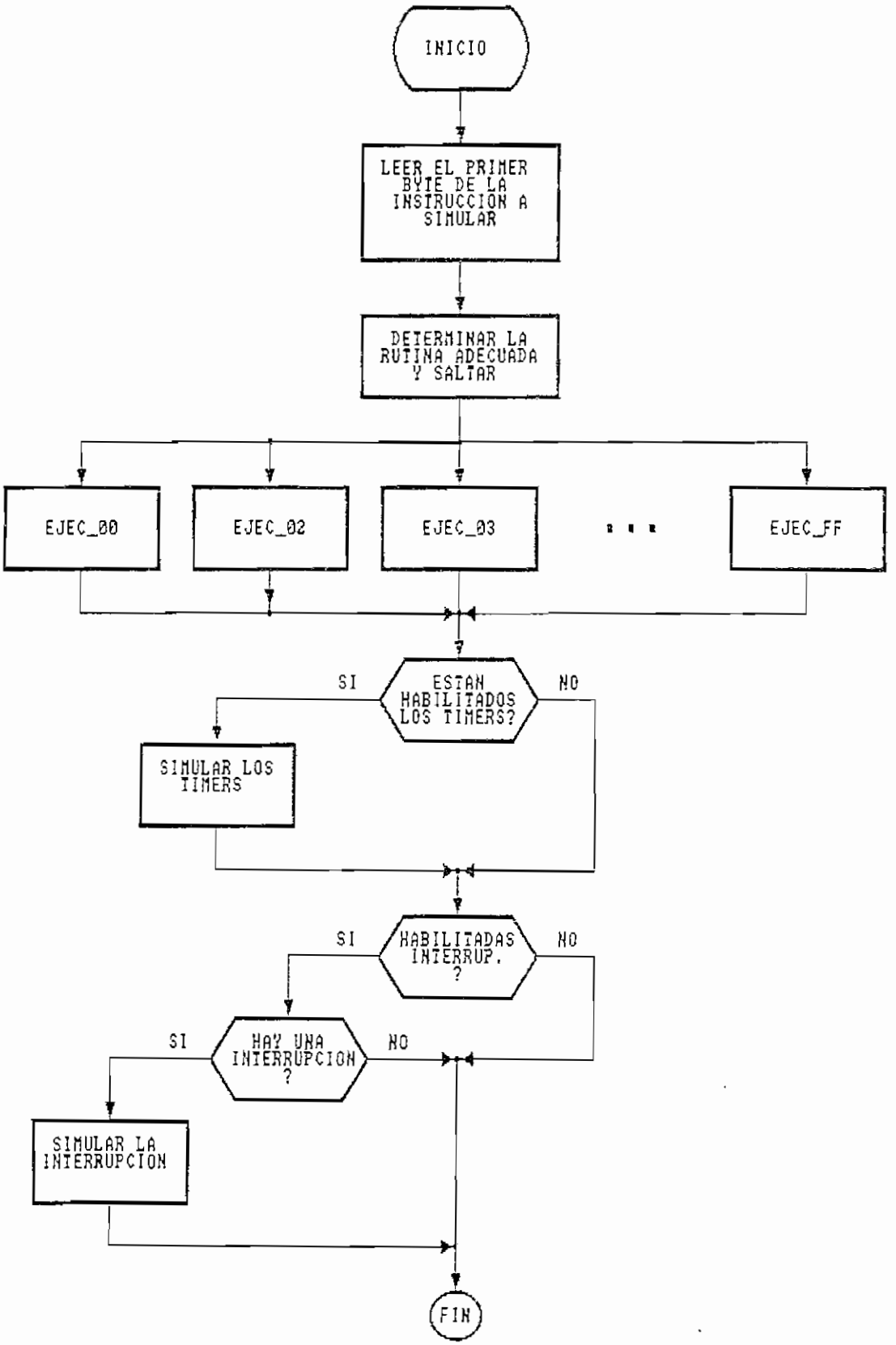


Figura 2.9.- Diagrama de flujo del modulo simulador.

2.3.4 MODULO DE COMUNICACION SERIAL

Para poder llevar a cabo la ejecución controlada se hace necesaria la existencia de un interfaz que sea el encargado de controlar este modo de operación del depurador. Se escogió como interfaz, el serial, por las siguientes razones: necesidad de comunicación de datos en los dos sentidos entre el computador personal y el microcontrolador, por la disponibilidad de puertos para comunicación serial tanto en el computador personal como en el microcontrolador y sobre todo por la característica interna del microcontrolador que es poseer un programa monitor para carga de programas a través del puerto serial.

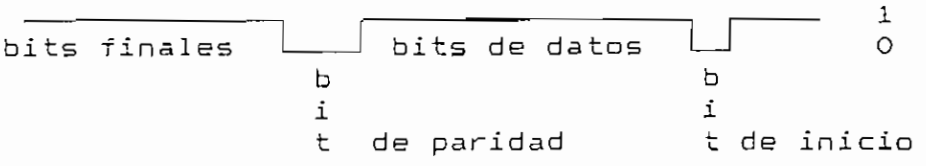
En el módulo COMSER.ASM están incluidas las rutinas que administran la comunicación serial entre el computador personal y el microcontrolador. Estas rutinas son invocadas cuando el depurador opera en modo de carga o ejecución controlada.

Siendo el lenguaje assembly del i8086 el utilizado para desarrollar el programa depurador para el DS5000T, todo lo referente a comunicación serial debe ser hecho al más bajo nivel, lo que implica el manejo directo del UART (Universal Asynchronous Receiver Transmitter).

Antes de usar el UART se debe inicializar la velocidad, número de bits por caracter (5 a 8), tipo de paridad (par,

impar o ninguna) y número de bits finales (1, 1.5 ó 2).

El formato de transmisión de datos es el siguiente.



El primer bit en transmitirse es el bit menos significativo; tanto el transmisor como el receptor deben setearse para comunicarse a igual velocidad, bit de datos y bits de parada.

El protocolo de comunicación serial utilizado es el siguiente: transmisión full-dúplex, 8 bits de datos, 1 bit de parada, ninguna paridad y puede ser seteado a dos velocidades; 4800 baudios para carga de programas en el microcontrolador y 9600 baudios para ejecución controlada. Hay esta diferencia en la velocidad de transmisión porque se utiliza un cristal de 3.58 MHz que de acuerdo a las especificaciones del fabricante del DS5000T, permite descarga serial de programas al microcontrolador hasta una velocidad máxima de 4800 baudios; mientras que para modo de ejecución controlada, el programa monitor DALLAS.ASM permite trabajar a 9600 baudios.

Por la forma de funcionamiento del programa monitor para carga serial de programas en el microcontrolador DS5000T y para poder utilizar las mismas rutinas en modo de ejecución

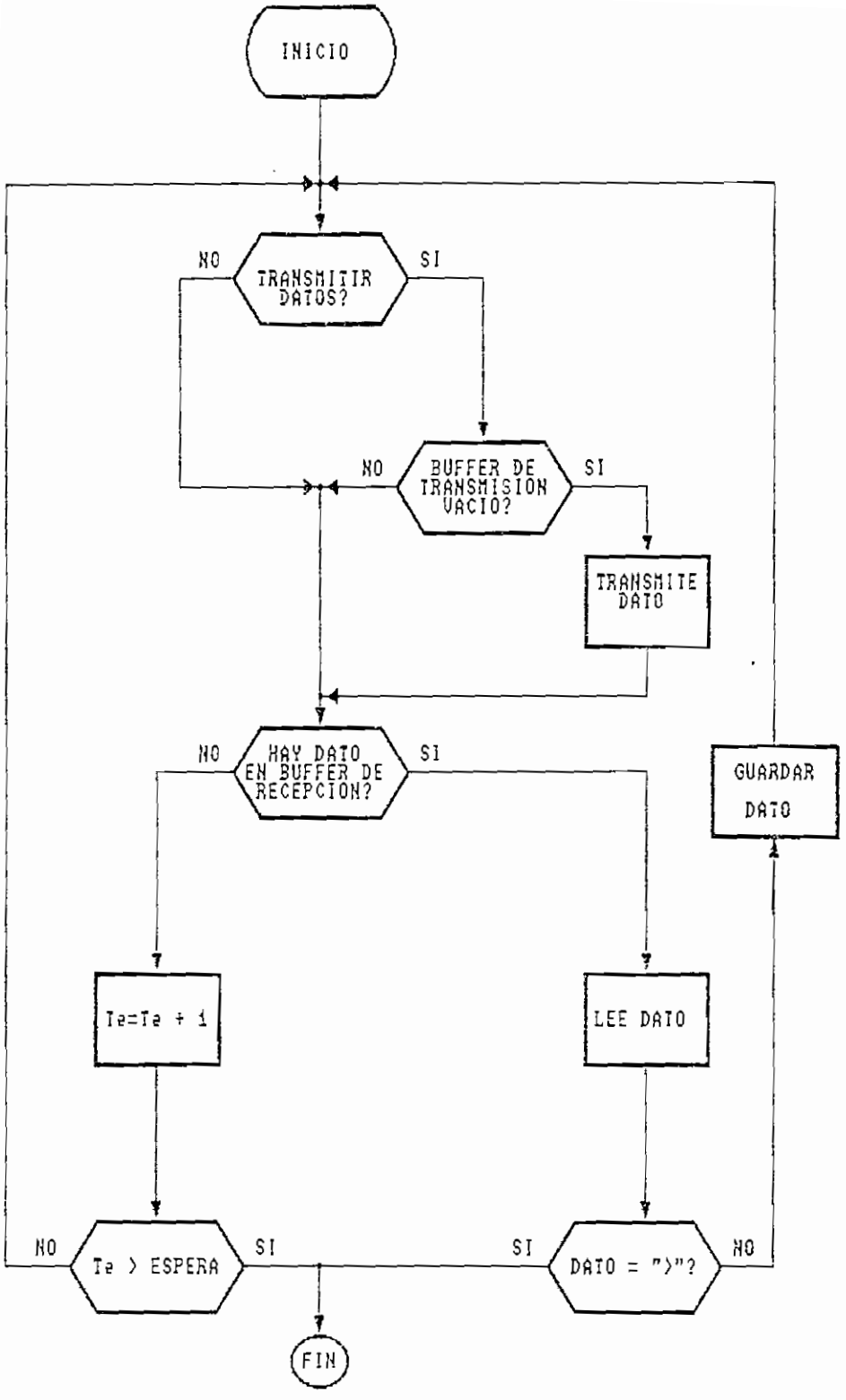


Figura 2.10.- Esquema de la comunicacion serial.

2.4 DESCRIPCION DEL PROGRAMA MONITOR PARA EL DS5000T

El módulo DALLAS.ASM contiene el programa monitor para el microcontrolador DS5000T cuando se trabaja en el modo de ejecución controlada. A diferencia de los otros módulos este es un programa escrito en lenguaje assembly para el microcontrolador DS5000T y no en lenguaje assembly para el microprocesador i8086.

Este programa monitor es el encargado de mantener en comunicación al uC con el computador personal. La comunicación es full-dúplex y está formulado para que el microcontrolador cumpla con alguna tarea especifica, lo que se logra a través de comandos los cuales deben ser procesados en este programa monitor, estos comandos son enviados por el puerto serial del computador personal, al pin de recepción del uC.

El programa monitor del microcontrolador está subordinado al programa depurador del computador personal, esto es necesario para que el programa depurador no pierda el control sobre el microcontrolador.

El programa monitor empieza seteando los parámetros necesarios para poder efectuar la transmisión con las siguientes características 8 bits de datos, 1 bit de parada, ninguna paridad y 9600 baudios de velocidad.

El protocolo de comunicación debe ser el mismo que el

fijado por el programa depurador; una vez hecho el seteo, el programa monitor está listo para recibir vía serial algún comando válido.

De acuerdo al comando recibido se cumple con alguna tarea especifica, estas tareas están relacionadas con las opciones del depurador de programas para el DS5000T.

Los comandos válidos son los siguientes:

Comando "S": Ejecución de una instrucción fuera del programa del usuario, adicionalmente al caracter "S", se requieren el número de bytes de la instrucción y el código de máquina de la instrucción. Si la instrucción es del tipo transferencia de control de programa, se inserta la instrucción recibida en la dirección apuntada por el PC y se ejecuta como parte del programa del usuario. En caso contrario se ejecuta la instrucción fuera de la secuencia del programa del usuario.

Comando "P": Ejecuta la instrucción apuntada por el PC del usuario, luego de lo cual carga en PC la dirección de la próxima instrucción a ejecutarse de acuerdo al flujo del programa del usuario.

Comando "O": Descarga información del microcontrolador hacia el computador, la información enviada hacia el computador son los valores de PC, ACC, DPTR, B, SP, banco de

registros, MCON, y puertos.

Comando "M": Descarga información de la RAM interna del microcontrolador, adicionalmente al comando "M" requiere un byte con la dirección inicial del área de memoria interna visible para el usuario.

Comando "N": Descarga información de la RAM Embebida del microcontrolador, adicionalmente al comando "N" requiere dos bytes con la dirección inicial del área de memoria embebida visible para el usuario.

Tanto el comando Q, M y N transmiten datos en formato ASCII; son enviados el número de bytes de transmisión, los bytes de datos y además un byte para control de error. Se ha adoptado el control de errores mediante checksum, para ser coherente con la codificación INTEL del archivo objeto.

Comando "I": Inserta una instrucción ingresada por el usuario en la dirección apuntada por PC; adicionalmente al comando "I" requiere el número de bytes de la instrucción y el código de máquina de la instrucción. La inserción de la instrucción implica el reemplazo de la información almacenada en la memoria del microcontrolador por la enviada por el computador personal.

Comando "E": Edita registros de la CPU, banderas, área de memoria interna y área de memoria embebida; adicionalmente al

comando "E" requiere un byte para determinar la correspondencia de edición y de acuerdo a este se requieren los respectivos bytes de edición.

Ya que la tarea principal del módulo monitor es permitir la ejecución controlada de una instrucción, se va a describir el procedimiento seguido.

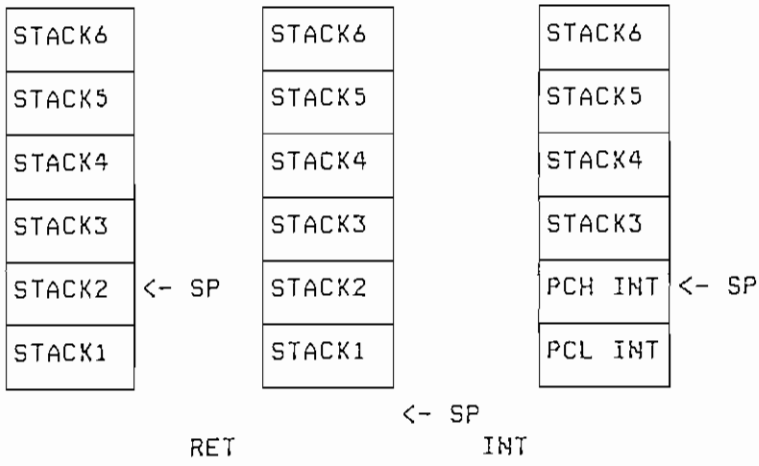
Para ejecutar una instrucción, ya sea dentro del flujo normal del programa o una instrucción fuera del listado del programa del usuario, el procedimiento a seguirse es el siguiente:

- Definir como RAM las localidades superiores a la dirección 0800H. En la dirección 0800H, se encuentra el primer byte de código correspondiente a una instrucción de salto largo.

- Cargar la dirección de la instrucción a ejecutarse, en las localidas correspondientes a las direcciones 0801H y 0802H. En el caso de ejecución dentro del listado del programa esta dirección es la apuntada por el PC del usuario. Y en el caso de ejecución fuera del listado del programa, es la dirección 0803H. Con esto se consigue completar el código de la instrucción de salto.

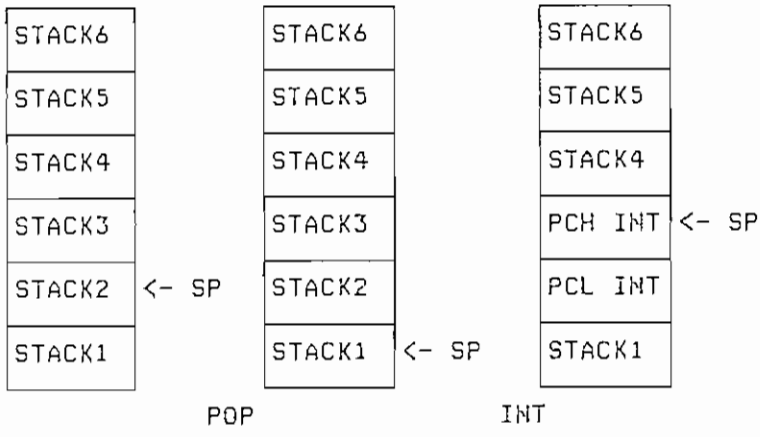
- Redefinir estas localidades como ROM, para que se pueda ejecutar el código almacenado en ellas.

RET:



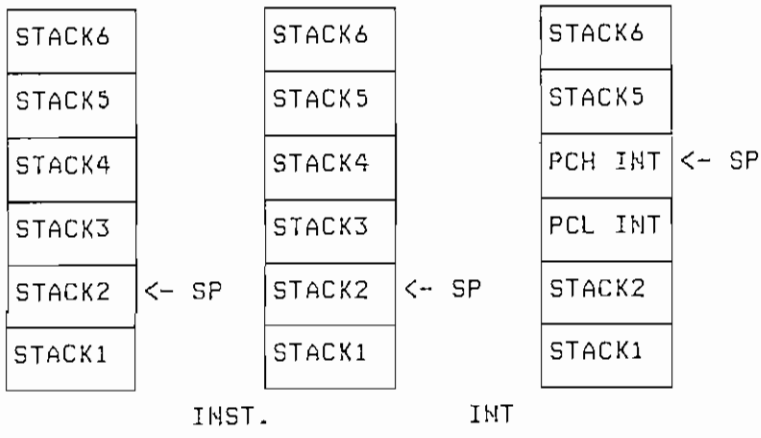
SP = SP , alterados stack1 y stack2

POP:



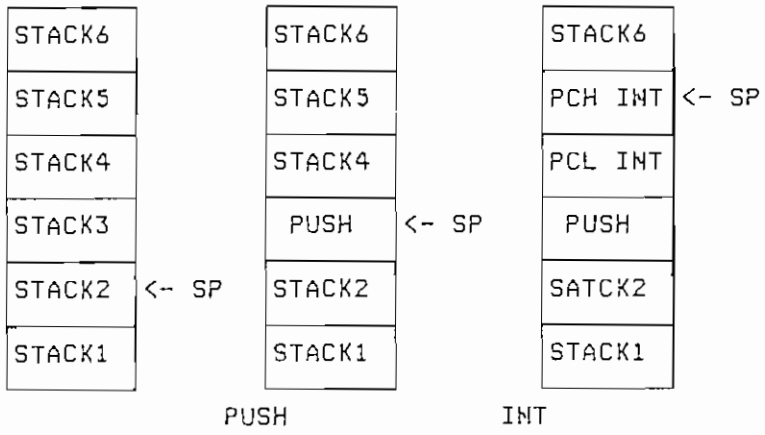
SP = SP + 1, alterados stack2 y stack3

INSTRUCCION QUE NO MODIFICA EL STACK:



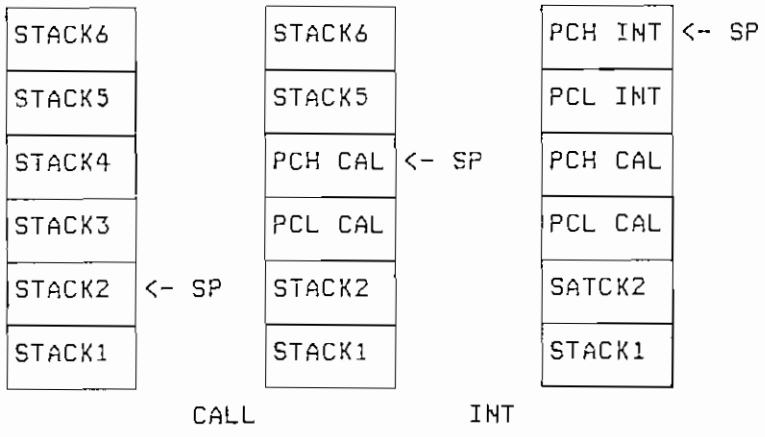
SP = SP + 2, alterados stack3 y stack4

PUSH:



SP = SP + 3, alterados stack4 y stack5

CALL:



SP = SP + 4, alterados stack5 y stack6

III. DESARROLLO DEL HARDWARE

3.1 DESCRIPCION GENERAL DEL KIT DE DESARROLLO

El kit de desarrollo le va a permitir al programador de sistemas basados en el microcontrolador DS5000T, probar sus programas, además de depurarlos en hardware, eligiendo el modo de ejecución controlada.

El kit está constituido por un módulo central que contiene el microcontrolador DS5000T con los dispositivos necesarios para comunicación serial con un computador personal y para el control respectivo de carga y ejecución de programas.

Complementariamente a éste se han diseñado otros módulos, los cuales están concebidos básicamente para ser usados como entrada o salida de datos a través de los puertos del microcontrolador.

3.2 DESCRIPCION DEL MODULO CENTRAL

Este módulo es el que contiene al microcontrolador DS5000T, y por tanto es el alma del kit de desarrollo, por lo que es imprescindible, a diferencia de los módulos de hardware complementarios que son opcionales.

El módulo central tiene que satisfacer los siguientes

requerimientos:

- Permitir descargar serialmente el programa del usuario desde el computador personal al microcontrolador.

- Permitir al depurador del DS5000T trabajar en modo de ejecución controlada.

- Trabajar autónomamente una vez cargado el programa del usuario en el microcontrolador.

La configuración que responde a todos estos planteamientos es la que se muestra en la figura 3.1.

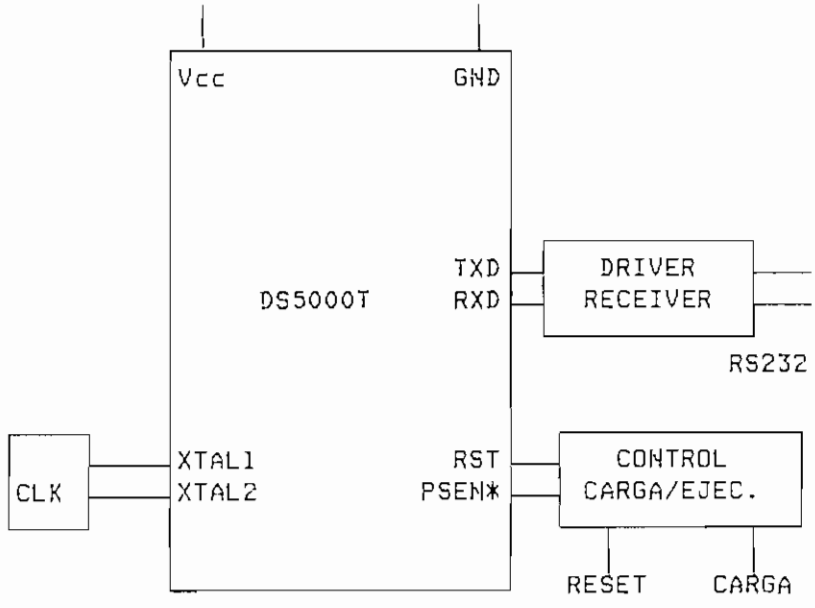


Figura 3.1.- Estructura del módulo central

CARGA	RESET	RST	PSEN	
0	0	1	1	Ejecución
0	1	0	1	Reset en ejecución
1	0	1	0	Carga
1	1	1	0	Carga

Tabla 3.1.- Funcionamiento del control CARGA/EJEC.

De donde:

$$PSEN = \overline{CARGA}$$

$$RST = \overline{CARGA}.\ RESET$$

DRIVER Y RECEIVER:

Es el circuito que nos permite cambiar de niveles TTL a niveles EIA RS-232C y viceversa, esto es necesario para poder manejar la comunicación serial con el computador personal. Se utiliza un MAXIM 232 para tal efecto, este circuito tiene la ventaja de poder utilizar una sola fuente de 5 voltios para polarización.

En la figura 3.2 se encuentra el diagrama del circuito del módulo central.

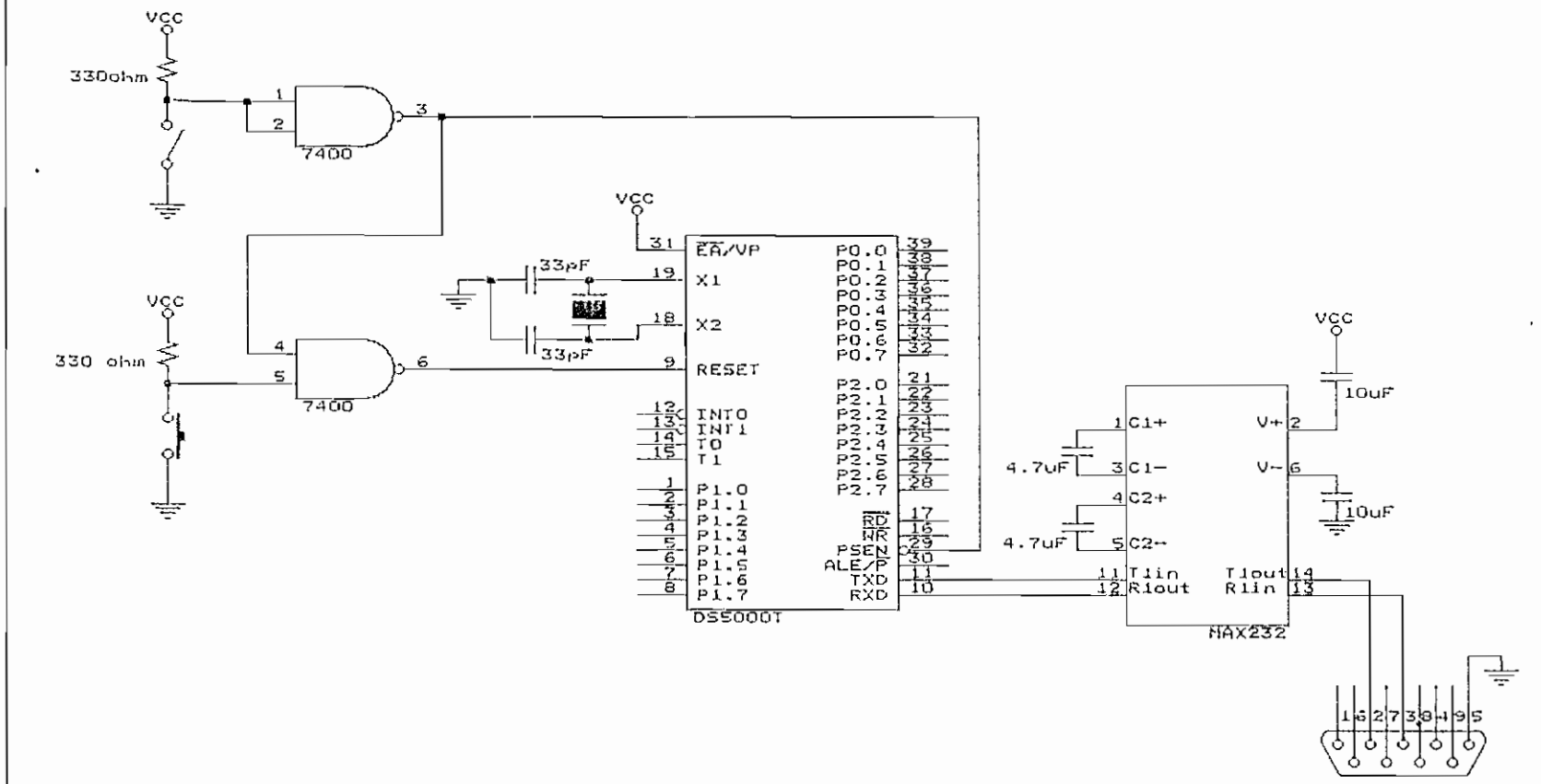


Figura 3.2.- Diagrama circuital del Kit de Desarrollo.

3.3 DESCRIPCION DE LOS MODULOS COMPLEMENTARIOS

3.3.1 MODULOS DE ENTRADA

Se ha considerado necesario disponer de módulos que permitan el ingreso de datos hacia el microcontrolador. Siendo las formas de entrada más frecuentes, a través de dip-switches y teclado, se han diseñado circuitos que posibiliten la entrada de datos mediante estos dos dispositivos.

DIP-SWITCHES

Este módulo está concebido para ser utilizado como entrada de datos por cualquiera de los puertos del microcontrolador. Como cada puerto posee ocho líneas para entrada o salida, el módulo tiene ocho dip-switches para ingreso de datos hacia el puerto.

Tomando en cuenta la posibilidad de una cortocircuito, que podría destruir el microcontrolador, este arreglo de dip-switches se ha diseñado de tal manera que impida su destrucción por mal uso del operador.

El módulo consta de los 8 dip-switches, que permiten ingresar un byte de dato; la entrada hacia el puerto se lo hace a través de optoacopladores. Además de los dip-switches y optoacopladores se requieren resistencias que limiten las corrientes correspondientes.

El circuito diseñado se muestra en la figura 3.3, donde:

$$R1 = \frac{Vcc1 - VD}{ID}$$

$$R1 = \frac{5v - 1.7v}{30\ mA}$$

$$R1 = 150\ ohm.$$

$$R2 = \frac{Vcc2 - Vce\ (sat)}{Ic}$$

$$R2 = \frac{5v - 0.2v}{15\ mA}$$

$$R2 = 330\ ohm.$$

TECLADO

El módulo de teclado permite ingresar información codificada en binario a través de los puertos del microcontrolador.

Este módulo está constituido por una matriz de teclado de 19 teclas, manejada mediante un codificador de teclado MM74C923. Este codificador permite obtener en sus salidas un código binario correspondiente a la tecla presionada, de acuerdo al detalle mostrado en la tabla 3.2.

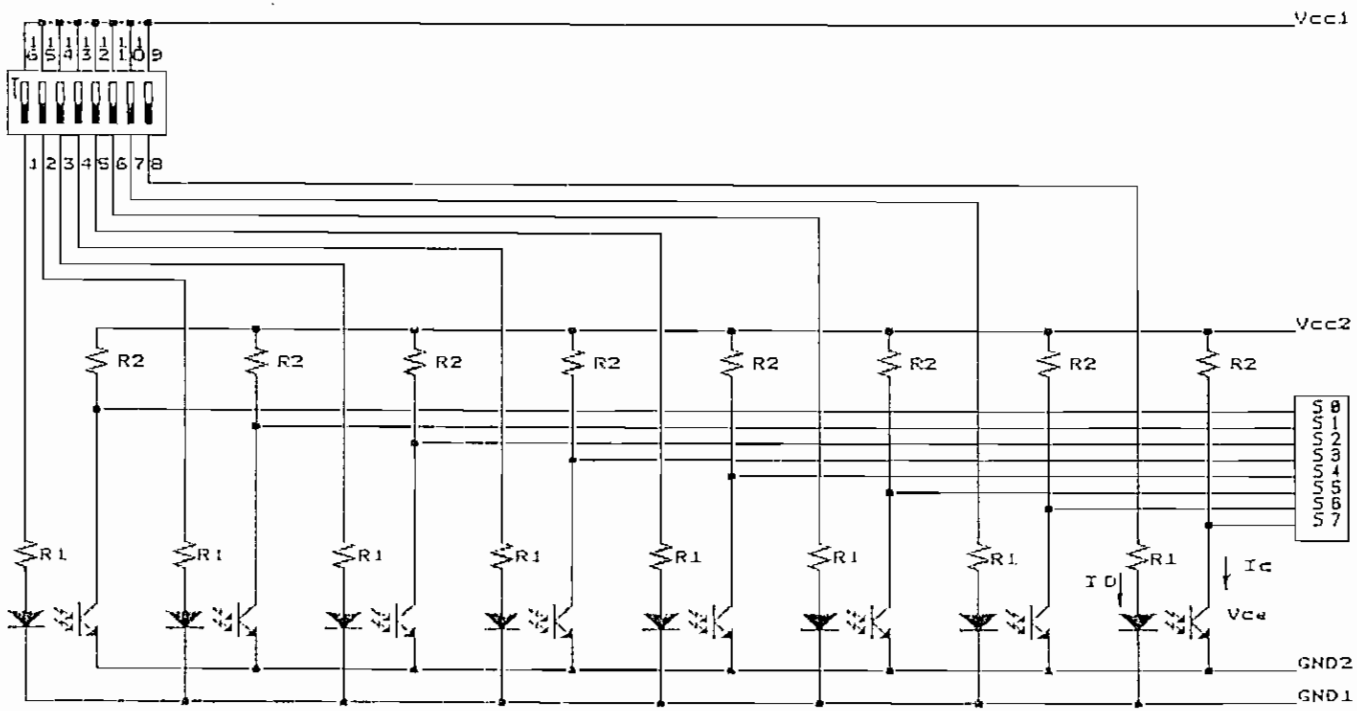


Figura 3.3.- Diagrama circuital de los dip_switches.

Tecla Presionada	Código				
	E	D	C	B	A
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
A	0	1	0	1	0
B	0	1	0	1	1
C	0	1	1	0	0
D	0	1	1	0	1
E	0	1	1	1	0
F	0	1	1	1	1
Shift	1	0	0	0	0
H	1	0	0	0	1
L	1	0	0	1	0

Tabla 3.2.- Códigos generados por la matriz de teclado

Adicionalmente se dispone de una salida de dato válido (DATA AVAILABLE), la cual puede ser usada como una señal de interrupción, que indica al microcontrolador que se ha presionado una tecla.

Las salidas del codificador mantienen el código de la tecla mientras permanece presionada. Cuando se la libera, estas salidas se ponen en estado de alta impedancia.

El circuito diseñado se muestra en la figura 3.4.

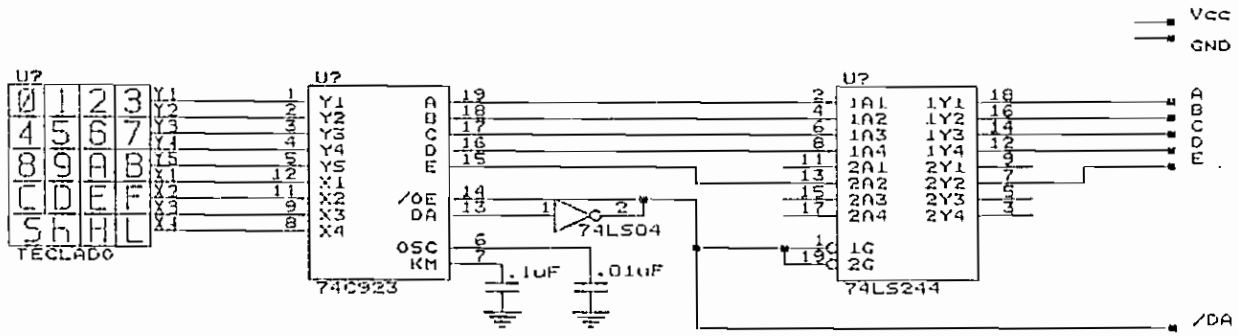


Figura 3.4.- Diagrama circuital del teclado.

3.3.2 MODULOS DE SALIDA

Los módulos de salida, posibilitan al programador observar el estado de los puertos, ya sea directamente en cada pin por medio de leds, o como información codificada mediante displays.

LEDS

Este módulo está constituido por 8 leds que pueden ser conectados a cualquier puerto del microcontrolador.

La corriente de los leds es manejada mediante 8 transistores NPN (2N3904), los cuales a su vez son conectados al microcontrolador a través de un buffer tres estados SN74244 que permite incrementar la capacidad de manejo de corriente propia de los puertos del microcontrolador, además de poder colocar las salidas de este buffer en estado de alta impedancia para aislar este módulo del puerto del microcontrolador.

El circuito diseñado se muestra en la figura 3.5 donde:

$$R = \frac{Vcc - Vce(sat) - VD}{ID}$$

$$R = \frac{5v - 0.2v - 1.7v}{10mA}$$

$$R = 330 \text{ ohm.}$$

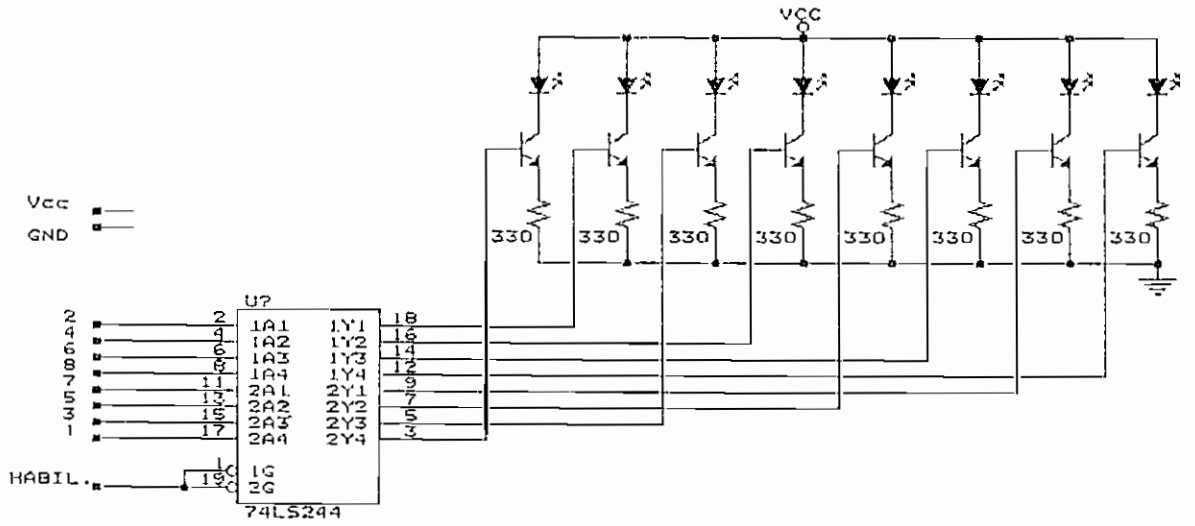


Figura 3.5.- Diagrama circuital de los leds.

ARREGLOS DE DISPLAYS

El arreglo está constituido por 6 displays, D0...D5. Los cuales pueden ser utilizados para mostrar 6 datos, como por ejemplo fechas en formato AA MM DD, horas en formato HH MM SS, etc.

Este arreglo está orientado a ser usado como salida BCD de cualquiera de los puertos, y el usuario no puede manejar directamente los segmentos de los displays.

Por lo tanto se ha adoptado la siguiente configuración:

- Se puede acceder a un display a la vez.
- El manejo de datos es a nivel de byte.
- Se puede acceder al punto decimal de cada display.
- Los datos tienen formato BCD.

Los siete segmentos y punto decimal de todos los displays han sido unidos de modo que se puedan activar con un solo decodificador de segmentos (SN7447), cada display es manejado por un transistor PNP (2N3902), el cual actúa como un interruptor que activa o desactiva el display, esta conmutación la realiza el decodificador (SN74138) el cual selecciona el display que va a operar de acuerdo a los datos presentes en el puerto del microcontrolador.

La figura 3.6 ilustra el circuito diseñado.

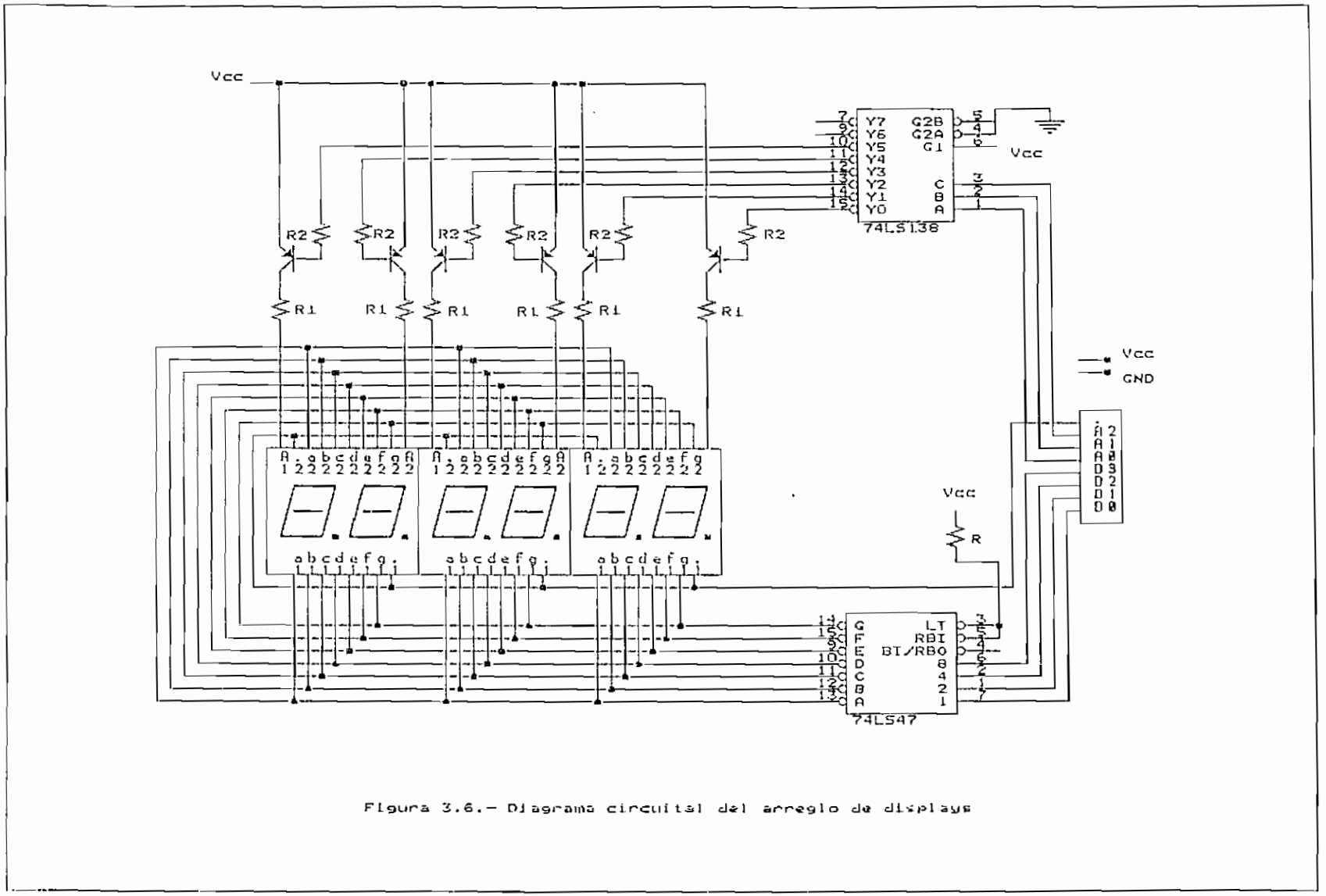


Figura 3.6.- Diagrama circuital del arreglo de displays

IV. RESULTADOS

Al iniciar el presente trabajo de tesis se habia planteado la ejecución de un sistema que permita la depuración de programas desarrollados para el microcontrolador DS5000T, el cual debia ser una herramienta didáctica y profesional útil.

Una vez realizado el trabajo se ha logrado obtener un paquete de software más algunos módulos de hardware que presentan las siguientes características:

- Depuración de programas para el microcontrolador DS5000T, mediante simulación en un computador personal

- Depuración de programas para el microcontrolador DS5000T, en el propio microcontrolador, mediante ejecución controlada desde un computador personal

- Similitud con paquetes desarrollados antes, incluidas sus opciones más importantes

- Descarga de programas hacia el microcontrolador, a través del puerto serial EIA RS-232C del computador personal

- Disponibilidad de módulos de hardware de entrada/salida que permiten probar aplicaciones desarrolladas para el DS5000T

- Facilidad de uso

La consecución de los objetivos planteados implicó que se deba establecer ciertas restricciones, especialmente en el caso de depuración de programas mediante ejecución en el propio microcontrolador, ya que se debió utilizar ciertos recursos de éste para llevar a cabo este tipo de depuración. Estas restricciones se detallan a continuación:

- El modo de simulación no contempla la posibilidad de uso de memoria externa al microcontrolador

- El modo de ejecución controlada no permite al usuario utilizar los siguientes recursos del microcontrolador:

    - Area de memoria embebida comprendida entre 0000H y 0FFFH
    - Area de memoria RAM interna comprendida entre 68H y 7FH
    - puerto serial
    - interrupciones
    - timers
    - partición dinámica de memoria

Considerando que el paquete de depuración de programas para el microcontrolador DS5000T debe cumplir índices altos de confiabilidad, fue puesto a prueba por estudiantes de los

últimos semestres de la Facultad de Ingeniería Eléctrica de la Escuela Politécnica Nacional.

A tal efecto se plantearon dos proyectos en los cuales se utilizó el microcontrolador DS5000T y que fueron planificados de manera que se usen todas las herramientas del depurador.

El desarrollo de estos proyectos involucró la simulación de los programas, la ejecución controlada en el propio microcontrolador, de aquellas rutinas donde era posible este tipo de depuración, y finalmente la carga serial del programa del usuario en el microcontrolador y su ejecución autónoma, haciendo uso de los módulos de hardware necesarios.

En los proyectos planteados se consideró el uso de la mayoría de los recursos del microcontrolador, para que se pudiese trabajar con todos ellos en la detección de posibles errores para su posterior correcccción. Estos proyectos fueron:

PROYECTO # 1:

Reloj - calendario:

Diseño y realización de un reloj-calendario dotado de:

- Teclado para entrada de datos y arreglo de 6 displays para despliegue de su hora y fecha.

Si bien es cierto que el código de máquina es el mismo para los microcontroladores de la familia i51 y el DS5000T, la representación del mismo en mnemónicos es diferente, habiéndose optado por utilizar la especificada por el fabricante del DS5000T. Sin embargo de lo cual, el tipo de mnemónicos utilizados puede ser fácilmente modificado mediante el cambio de una tabla en el programa fuente del módulo desensamblador del depurador.

El registro Timed Access no puede ser accedido para lectura, por lo que su presentación en pantalla sería contradictoria con esta característica del microcontrolador.

El paquete de depuración para el DS5000T ha sido concebido de manera que pueda ser integrado en el paquete SIDES en un etapa posterior a este trabajo.

Los cambios que se hagan en el programa del usuario dentro del depurador no modifican el archivo objeto que se está depurando. La creación de un archivo que incluya estas modificaciones involucra una tarea bastante laboriosa, la cual no se justifica considerando que esos cambios pueden ser fácilmente hechos en el programa fuente del usuario.

La utilización de menúes de opciones hace que el uso del programa sea muy simple, la presentación de opciones viene acompañada de una ayuda que, siendo breve, es lo bastante explicativa sobre las tareas que cumplen estas opciones.

V. CONCLUSIONES Y RECOMENDACIONES

El Depurador de Programas para el microcontrolador DS5000T cumple con los objetivos propuestos, esto es ser una herramienta para desarrollo de programas muy útil y fácil de usar puesto que presenta la información más importante, respecto al estatus del microcontrolador, en la pantalla del computador personal.

El programa desarrollado es muy similar con paquetes existentes como AVSIM51, AVSIMZ80, AVSIM48 y PRO; lo que beneficia a los programadores de microcontroladores y microprocesadores ya que no requieren dedicar mucho tiempo a la familiarización con este nuevo paquete.

El paquete de software y los módulos de hardware fueron probados por los estudiantes de la materia Interfaces para Microprocesadores de la Facultad de Ingeniería Eléctrica de la Escuela Politécnica Nacional habiéndose podido detectar ciertos errores que fueron oportunamente corregidos, al igual que se tomaron en cuenta los comentarios surgidos respecto a la manera de lograr mejoras al programa.

Sin embargo de lo expuesto, se cree que existe la posibilidad de introducir otras mejoras a este paquete lo cual se verá facilitado por la concepción modular con que ha sido desarrollado. Adicionalmente, hay la posibilidad de que existan errores que hayan sido pasados por alto en la

depuración del paquete, los cuales podrían ser superados mediante un trabajo posterior puesto que se espera que este paquete tenga una amplia utilización por los estudiantes de las materias relacionadas con microprocesadores y microcontroladores.

Al igual que en esta Tesis de Grado se han utilizado algunas rutinas desarrolladas en la Tesis de Grado de los señores ingenieros Fernando Ortega e Iván Ordoñez, las rutinas desarrolladas en este trabajo pueden ser utilizadas en proyectos posteriores que tengan las mismas características.

Considerando las características que debe tener un depurador de programas, esto es permitir visualizar rápidamente los cambios producidos en el estatus del microcontrolador, el lenguaje assembly resultó el más adecuado ya que se tiene un acceso directo a todos los recursos del computador personal, principalmente teclado, pantalla y puerto de comunicación serial.

La programación modular, factible en el lenguaje assembly, resultó un modelo de trabajo muy adecuado para desarrollar esta Tesis de Grado entre dos personas ya que posibilitó la realización de tareas separadas que posteriormente fueron integradas para conformar este depurador. Este tipo de programación, además de permitir trabajo conjunto, es muy adecuado para desarrollar paquetes relativamente extensos puesto que el objetivo general puede

ser resuelto mediante módulos que cumplan objetivos particulares encaminados a la consecución del objetivo general.

El desarrollo de este paquete implicó la creación de 6 módulos individuales en lenguaje assembly del microprocesador i8086, los cuales se explican brevemente a continuación.

Los módulos ensamblador y desensamblador fueron desarrollados tomando como base los programas de los ingenieros Fernando Ortega e Iván Ordoñez; sin embargo, fue necesario hacer importantes reformas debido a que la forma de codificación de instrucciones del microprocesador i8086 y del microcontrolador DS5000T difieren notablemente. En el caso del programa ensamblador se ha mantenido la idea de realizar el ensamblado en dos pasos, mediante la generación de un pseudocódigo en el primer paso, ya que ello facilita el análisis de las cadenas de caracteres a ensamblar.

El módulo simulador ha sido concebido para producir en la memoria y reflejar en la pantalla del computador, el efecto que tienen todas y cada una de las instrucciones del microcontrolador en el estatus del mismo. Este módulo fue desarrollado de manera que permite simular la utilización de cualquier recurso del microcontrolador excepto memoria externa, tratando de reproducir lo más fielmente posible el comportamiento del microcontrolador DS5000T.

Aunque el protocolo de comunicación serial utilizada en este trabajo de Tesis de Grado es sencillo, es importante resaltar el uso directo del UART (Universal Asynchronous Receiver Transmitter), ya que es generalizado el uso de lenguajes de alto nivel para comunicación serial, por desconocimiento del manejo directo del UART. Las rutinas utilizadas por el módulo administrador de comunicación serial son una buena base para desarrollar proyectos de comunicacion serial con protocolos más complejos.

Adicionalmente a los módulos de programas en lenguaje assembly del microprocesador i8086, fue necesario desarrollar un programa en lenguaje assembly del microcontrolador DS5000T, el cual le posibilita al depurador trabajar en modo de ejecución controlada. Este programa monitor enlaza el computador personal con el microcontrolador mediante comunicación serial y a través de comandos.

El programa monitor utilizado para depurar programas en modo de ejecución controlada si bien es cierto está subordinado al programa depurador que se ejecuta en el computador personal, tiene cierta libertad de concepción, lo que posibilita modificaciones de este programa monitor sin tener que alterar los módulos del programa depurador; lo cual permitiría cambiar alguna de las rutinas para mejorar la capacidad del depurador en lo que se refiere al modo de ejecución controlada.

La ejecución controlada está realizada mediante la puesta en práctica de dos ideas centrales:

- Partición dinámica de la RAM Embebida
- Generación de la interrupción del Timer 1, justo el instante después de la ejecución de la instrucción

Tomando en cuenta que el microcontrolador DS5000T tiene todas las características del i8051, se podría pensar en utilizar esta forma de realizar la ejecución controlada en el i8051; el único inconveniente sería la no disponibilidad de la capacidad de particionamiento dinámico de memoria; sin embargo, esto podría superarse mediante algún artificio, como almacenar el programa en una memoria RAM, y utilizando algún circuito externo que posibilite este modo de ejecución.

Esta forma de ejecución controlada, no es la única y posiblemente tampoco la mejor por lo que cabría la posibilidad de modificarla o reemplazarla, con el objeto de hacer más práctica esta forma de ejecución y así el depurador brinde una mejor ayuda al diseñador de programas basados en el microcontrolador DS5000T.

Al igual que los otros módulos desarrollados en lenguaje assembly del microprocesador i8086, existen en el programa monitor del DS5000T muchas rutinas que bien pueden ser utilizadas para desarrollo de proyectos posteriores, entre las cuales están las rutinas sobre comunicación serial,

```
;*************************************
;*********     INTHMAQ.ASM    **********
;*************************************

SSEG SEGMENT STACK
DB 512 DUP (?)
SSEG ENDS

PUBLIC  VIDEOSEG,PANBUF

EXTRN   AUX:WORD

DSEG SEGMENT PUBLIC
nmaxbuf   equ 31

tabbuf    db 01h,00h,00h,00h,00h,00h,00h,00h    ;inicial
          db 11h,00h,00h,00h,00h,00h,00h,00h    ;archivos
          db 11h,01h,00h,00h,00h,00h,00h,00h    ;nombre
          db 11h,02h,00h,00h,00h,00h,00h,00h    ;lee
          db 21h,00h,00h,00h,00h,00h,00h,00h    ;depurador
          db 21h,01h,00h,00h,00h,00h,00h,00h    ;paso
          db 21h,02h,00h,00h,00h,00h,00h,00h    ;corre
          db 21h,03h,00h,00h,00h,00h,00h,00h    ;reversa
          db 21h,04h,00h,00h,00h,00h,00h,00h    ;editor
          db 21h,05h,00h,00h,00h,00h,00h,00h    ;memoria
          db 21h,15h,00h,00h,00h,00h,00h,00h    ;area1
          db 21h,15h,01h,00h,00h,00h,00h,00h    ;area1 int
          db 21h,15h,02h,00h,00h,00h,00h,00h    ;area1 ext
          db 21h,25h,00h,00h,00h,00h,00h,00h    ;area2
          db 21h,25h,01h,00h,00h,00h,00h,00h    ;area2 int
          db 21h,25h,02h,00h,00h,00h,00h,00h    ;area2 ext
          db 21h,06h,00h,00h,00h,00h,00h,00h    ;rupturas
          db 21h,16h,00h,00h,00h,00h,00h,00h    ;crea
          db 21h,26h,00h,00h,00h,00h,00h,00h    ;activa/desactiva
          db 21h,36h,00h,00h,00h,00h,00h,00h    ;borra
          db 21h,46h,00h,00h,00h,00h,00h,00h    ;muestra
          db 21h,56h,00h,00h,00h,00h,00h,00h    ;editor
          db 21h,07h,00h,00h,00h,00h,00h,00h    ;inserta
          db 21h,08h,00h,00h,00h,00h,00h,00h    ;ensambla
          db 21h,09h,00h,00h,00h,00h,00h,00h    ;ciclos
          db 21h,19h,00h,00h,00h,00h,00h,00h    ;seteo
          db 21h,29h,00h,00h,00h,00h,00h,00h    ;activa/desactiva
          db 21h,0ah,00h,00h,00h,00h,00h,00h    ;display
          db 31h,00h,00h,00h,00h,00h,00h,00h    ;reset
          db 41h,00h,00h,00h,00h,00h,00h,00h    ;ayuda
          db 51h,00h,00h,00h,00h,00h,00h,00h    ;salir

tabinf    dw offset exp0,offset mem0,offset rut0        ;inicial
          db 1
          dw offset ea,offset ma,offset rut0            ;archivos
          db 3
          dw offset ean,offset mem0,offset ran          ;nombre
          db 1
          dw offset eal,offset mem0,offset ral          ;lee
          db 2
          dw offset ed,offset md,offset rut0            ;depurador
          db 3
          dw offset edp,offset mem0,offset rdp          ;paso
```

```
            db 1
            dw offset edc,offset mem0,offset rdc          ;corre
            db 1
            dw offset erev,offset mem0,offset rrev        ;reversa
            db 1
            dw offset ee,offset mem0,offset re            ;editor
            db 1
            dw offset em,offset mm,offset rut0            ;memoria
            db 3
            dw offset em1,offset min_ex,offset rut0       ;area1
            db 3
            dw offset eint,offset mem0,offset rm1in       ;area1 int
            db 2
            dw offset eext,offset mem0,offset rm1ex       ;area1 ext
            db 2
            dw offset em2,offset min_ex,offset rut0       ;area 2
            db 3
            dw offset eint,offset mem0,offset rm2in       ;area2 int
            db 2
            dw offset eext,offset mem0,offset rm2ex       ;area2 ext
            db 2
            dw offset edr,offset mdr,offset rut0          ;rupturas
            db 3
            dw offset edrc,offset mem0,offset rdrc        ;crea
            db 1
            dw offset edra,offset mem0,offset rdra        ;activa/desactiva
            db 1
            dw offset edrb,offset mem0,offset rdrb        ;borra
            db 1
            dw offset edrm,offset mem0,offset rdrm        ;muestra
            db 1
            dw offset ee,offset mem0,offset re            ;editor
            db 1
            dw offset ei,offset mem0,offset ri            ;inserta
            db 1
            dw offset een,offset mem0,offset ren          ;ensambla
            db 1
            dw offset edci,offset mdci,offset rut0        ;ciclos
            db 3
            dw offset edcis,offset mem0,offset rdcis      ;seteo
            db 2
            dw offset edcia,offset mem0,offset rdcia      ;activa/desactiva
            db 2
            dw offset edd,offset mem0,offset rdd          ;display
            db 1
            dw offset erest,offset mem0,offset rrest      ;reset
            db 1
            dw offset eayu,offset mem0,offset rayu        ;ayuda
            db 1
            dw offset esal,offset mem0,offset rsal        ;salir
            db 1


expO        db " ",0
ea          db "Manejo de Archivos",0
ean         db "Fija el nombre del <Objetivo>",0
```

```
eal       db "Lee el Archivo nombrado",0
ed        db "Depurador de Programas",0
edp       db "Ejecuta una instrucci",162,"n",0
edc       db "Va ejecutando secuencialmente el programa",0
erev      db "Ejecuci",162,"n reversa",0
ee        db "Editor de Registros y Areas de Memoria",0
em        db "Cambio de area de visualizaci",162,"n de Memoria",0
em1       db "Area 1 de vizualizaci",162,"n de memoria",0
em2       db "Area 2 de vizualizaci",162,"n de memoria",0
eint      db "Mapa de memoria interna",0
eext      db "Mapa de memoria RAM Embebida",0
edr       db "Puntos de ruptura",0
edrc      db "Marca un punto de ruptura",0
edra      db "Activa o desactiva puntos de ruptura",0
edrb      db "Elimina un punto de ruptura",0
edrm      db "Muestra puntos de ruptura",0
ei        db "Inserta una instrucci",162,"n",0
een       db "Ensambla una instrucci",162,"n y la ejecuta",0
edci      db "Manejo de ciclos de reloj de uC",0
edcis     db "Resetea el n",163,"mero de ciclos de reloj",0
edcia     db "Activa/desactiva la cuenta de ciclos",0
edd       db "Activa/desactiva actualizaci",162,"n de pantalla durante"
          db " ejecuci",162,"n",0
erest     db "Reset de la CPU",0
eayu      db "Muestra ventana de ayuda",0
esal      db "Permite salir del depurador",0


mem0      db "Archivos  Depurador  Reset  aYuda  Salir",0,5,"ADRYS"
ma        db "Nombre  Lee",0,2,"NL"
md        db "Paso Corre reVersa Editor Memoria Ruptura Inserta eNsambla"
          db " cYclos Display ",0,10,"PCVEMRINYD"
mdr       db "Crea   Activa/desactiva  Borra   Muestra  Editor",0,5,"CABME"
mdci      db "Reset  Activa/desactiva",0,2,"RA"
mm        db "area1   area2",0,2,"12"
min_ex    db "Interna   Embebida",0,2,"IE"

bopc    db      0               ;bandera para mostrar subopciones
nivel   db      0               ;nivel del menu actual
nivel1  db      0               ;nivel de la opcion actual
nivel2  db      0               ;nivel de la subopcion actual
respnivel db    0
buffer  db      8 dup(0)        ;indica camino hacia menu actual
buffer1 db      8 dup(0)        ;camino hacia opcion actual
buffer2 db      8 dup(0)        ;camino hacia subopcion actual
respbuf db      8 dup(0)
nclave  db      0               ;ubicacion de BUFFER's en TABBUF
nclave1 db      0
nclave2 db      0
x       db      0               ;# de opcion actual
y       db      0               ;# de subopcion actual
xprev   db      0               ;X que origino menu actual
acabar? db      'Quiere terminar (S/N)?',0,2,'SN'
miodir  dw      0
mopaux  db      0               ;sitios para paso de dato de
mopaux1 db      0               ;teclado si hay rutina de escape
```

```
mopact  db      0
mbcbyte db      0
mbcmenu db      0
mbcpos  dw      0
mbcbuf  dw      0
videoseg dw     0               ;segmento de video
panbuf  db      3680 dup(0)     ;sitio para salvar pantalla
mmvx1   db      0
mmvlin  db      0
mmvlin1 db      0
mmvcar  db      0
mmvdir  dw      0
mv?aux  db      0
mv?aux1 db      0
mpmprx  dw      0
maylin  db      0
maycol  db      0
dirsal  dw      2 dup(0)        ;sitio para dir. de rutinas de usuario
tiprut  db      0               ;tipo de rutina: 1, 2 o 3
tiprut1 db      0               ;tipo de rutina de opcion actual
mraux1  db      0
mraux2  dw      0               ;Aqui se guardan datos mientras se
mraux3  dw      0               ;corre la rutina del usuario.
dirret  dw      2 dup(0)        ;Direccion de retorno
diract  dw      2 dup(0)
inword  dw      0
getword dw      0               ;salida de GETKEY
mnmos   db      0
mdirop  dw      0
moslin  db      0
moscol  db      0
mpddir  dw      0
mcudir  dw      0
mibdir  dw      0
mibx    db      0
msfdir  dw      0
otroaux db      0               ;bandera que indica paso por "ootr".
blanco  db      78 dup (20h),0
insprog db      "                DEPURADOR DE PROGRAMAS PARA EL DS5000T",0
        db      " ",0
        db      "       Estas son las opciones validas para los menus",0
        db      "  Se puede elegir una opción por su `letra clave' o "
        db      "con las flechas y ENTER.",0
        db      "  F2  presenta/quita las subopciones de una "
        db      "opción en una ventana.",0
        db      "  ESC permite salir al nivel superior de menúes.",0
        db      "  F1  presenta esta pantalla de ayuda.",0
        db      "                    ...Presione cualquier tecla...",0
colven  db      bnnorm          ;atributos: color de ventana
collin  db      bnnorm          ;color de linea
colexp  db      bnnorm          ;color de explicacion
colact  db      bnsubb          ;color de subopcion actual
colIact db      bninv           ;color de opcion actual
primera db      1

DSEG ENDS
```

```
EXTRN RINIC:NEAR,EDITOR:NEAR
EXTRN NOM_ARC:NEAR,LEE_ARC:NEAR
EXTRN CAMM1IN:NEAR,CAMM1EX:NEAR,CAMM2IN:NEAR,CAMM2EX:NEAR
EXTRN DEF_PASO:NEAR,DEF_CORRE:NEAR
EXTRN MOST_RUP:NEAR,ACT_RUP:NEAR,CREAR_RUP:NEAR,BORRAR_RUP:NEAR
EXTRN SET_CIC:NEAR,ACT_CIC:NEAR,ACT_DIS:NEAR
EXTRN INSERTA:NEAR,ENSAMBLA:NEAR
EXTRN RESET:NEAR,HELP:NEAR,ATRAS:NEAR

CSEG SEGMENT PUBLIC

ASSUME CS:CSEG,DS:DSEG,SS:SSEG,ES:NOTHING

INCLUDE FMAC.ASM
SUBRUT

ninfbuf equ     7               ;informaciones asociadas a buffer
longlin equ     160             ;80*2 (incluyendo atributos)
fleder  equ     77              ;teclas pulsadas:
fleizq  equ     75
flearr  equ     72
fleaba  equ     80
cr      equ     13
f1      equ     3bh
f2      equ     3ch
ctrlk   equ     11
esqsi   equ     0dah            ;codigos para dibujo de cuadro
linsup  equ     0c4h
esqsd   equ     0bfh
linlat  equ     0b3h
esqii   equ     0c0h
esqid   equ     0d9h
bnnorm  equ     7               ;video normal (B/N)
bnsubb  equ     09h             ;subrayado intenso
bninv   equ     78h             ;inverso
maxmac  equ     64              ;macros de 64 words maximo

retorno macro
        cld
        mov     ax,dseg
        mov     ds,ax
        mov     ax,videoseg
        mov     es,ax
        mov     ax,mraux3
        mov     bp,offset dirret
        mov     ds:[bp],ax
        mov     ax,mraux2
        mov     ds:[bp+2],ax
        jmp     dword ptr dirret
        endm


START PROC NEAR

inicio: mov     ax,dseg
        mov     ds,ax
```

```
        mov     ax,sseg
        mov     ss,ax
        mov     ax,40h
        mov     es,ax
        mov     ax,es:[63h]
        mov     dx,0b000h
        cmp     ax,3b4h
        je      vidador
        add     dx,800h
vidador:
        mov     videoseg,dx
        mov     es,dx
        mov     ax,0720h
        mov     cx,2000
        cld
        xor     di,di
        rep     stosw
        mov     ah,2            ;fija la posicion del cursor
        mov     dh,24           ;en la fila 24
        mov     dl,1            ;columna 1
        int     10h
        call    rinic
        guarpantalla
        lmppnt
        ayuda   insprog,8,collact
        getkey
        restpantalla
        mov     nivel,0         ;inicializo variables que
        mov     buffer,1        ;manejan flujo del programa.
        mov     buffer1,11h
        mov     bopc,0          ;no subopciones.
        mov     x,1             ;opcion actual
a_nivel:
        mov     y,0             ;subopcion actual
b_subnivel:
        bufcla  nclave,buffer   ;Variables del menu actual
        movbuf  respbuf,buffer
        mov     al,nivel
        mov     respnivel,al
        movbuf  buffer1,buffer
        mov     al,nivel
        mov     nivel1,al       ;Variables de opcion actual
        inc     nivel1
        nunivel buffer1,nivel1,x
        bufcla  nclave1,buffer1
        movbuf  buffer2,buffer1
        mov     al,nivel1
        mov     nivel2,al       ;Variables de subopcion actual
        inc     nivel2
        nunivel buffer2,nivel2,y
        bufcla  nclave2,buffer2
        explica nclave2         ;Explico que hace la subopcion
e_ventana:
        cmp     bopc,1          ;muestro ventana de subopciones?
        jne     b_s5
        guarpantalla            ;si, pero salvo pantalla.
```

```
        menuvent nclave1
b_s5:
        menulinea nclave        ;muestro menu,
        mov     otroaux,0
c_opcion:
        opcion?                 ;recibe datos de teclado.
        mov     al,mopact       ;# de la opcion elegida.
c_op1:  actuar                  ;salta a la rutina correcta.
        dw      offset oizq,offset oder,offset oarr
        dw      offset oaba,offset oret,offset ootr
        dw      offset octo,offset octp,offset octk
        dw      offset octl
oizq:
        menizq  nclave          ;pasar a opcion a la izquierda
        jmp     b_subnivel
oder:
        mender  nclave          ;opcion a la derecha
        jmp     b_subnivel
oarr:
        cmp     bopc,1          ;subopcion superior si
        je      c_oa1           ;la hay y si BOPC=1
        jmp     c_opcion
c_oa1:
        cmp     tipruti,3       ;opcion tiene subopciones?
        jz      c_oa2
        jmp     c_opcion        ;no.
c_oa2:  menarr  nclave1         ;si.
        jmp     b_subnivel
oaba:
        cmp     bopc,1          ;subopcion inferior si
        je      c_ob1           ;la hay y si BOPC=1
        jmp     c_opcion
c_ob1:
        cmp     tipruti,3
        jz      c_ob2
        jmp     c_opcion
c_ob2:  menaba  nclave1
        jmp     b_subnivel
oret:
        inc     nivel           ;se presiono ENTER, correr
        nunivel buffer,nivel,x  ;rutina de opcion actual.
c_or2:  cmp     y,0
        jne     c_or1
        cmp     tipruti,3
        je      c_or3
        jmp     c_oo2
c_or3:  mov     x,1
        jmp     c_oo2
c_or1:  mov     al,y
        mov     x,al
        jmp     c_oo2
ootr:
        valida? nclave          ;se digito una opcion desconocida
        je      c_oo1           ;esta entre las validas del menu?
        jmp     e_ventana       ;no.
c_oo1:
```

```
        mov     y,0             ;si, correr rutina de esa opcion.
        mov     x,al
        inc     nivel
        nunivel buffer,nivel,x
        mov     otroaux,1
c_oo2:
        bufcla  nclave,buffer   ;aqui se transfiere el control al
        rutina  nclave          ;usuario y este lo devuelve.
c_oo3:  actuar                  ;actuar de acuerdo a tiprut.
        dw      offset rnorm,offset rinv,offset rrama
rnorm:
        movbuf  buffer,respbuf  ;normal, quedarse en nivel actual.
        mov     al,respnivel
        mov     nivel,al
        jmp     b_subnivel
rinv:
        movbuf  buffer,respbuf  ;inversa, salir a nivel superior.
        mov     al,respnivel
        mov     nivel,al
        jmp     d_supernivel
rrama:
        cmp     otroaux,1       ;rama, profundizar.
        jne     rrama1
        mov     x,1
rrama1: jmp     a_nivel
octo:
        cmp     bopc,1          ;si BOPC=1 hacer 0 y viceversa.
        je      c_oco1
        mov     bopc,1
        jmp     c_oco2
c_oco1: mov     bopc,0
        mov     y,0
c_oco2: jmp     b_subnivel
octp:
        guarpantalla            ;mostrar instrucciones.
        lmppnt
        ayuda   insprog,8,collact
        getkey
        restpantalla
        jmp     b_subnivel
octk:                           ;Rutinas de macros.
        jmp     b_subnivel

octl:
d_supernivel:
        getxpr                  ;xprev=ultimo nibble del buffer <>0
        nunivel buffer,nivel,0  ;y que ahora pasa a ser 0.
        dec     nivel
        cmp     nivel,0ffh      ;estaba en menu principal?
        je      d_si
        mov     al,xprev        ;no, voy a nivel superior.
        mov     x,al
        jmp     a_nivel
d_si:
        mostrar acabar?,24,colexp
        actuar
```

```
        dw      offset d_s3,offset d_s2
d_s2:
        mov     x,1             ;no acabar.
        mov     nivel,0
        nunivel buffer,nivel,1
        jmp     a_nivel
d_s3:
final:  lmppnt                  ;si acabar
        mov     ax,4c00h
        int     21h

salida: retorno

rut0:   jmp     salida

re:     call    editor
        jmp     salida
rm1in:  call    camm1in
        jmp     salida
rm1ex:  call    camm1ex
        jmp     salida
rm2in:  call    camm2in
        jmp     salida
rm2ex:  call    camm2ex
        jmp     salida
ran:    call    nom_arc
        jmp     salida
ral:    mov     aux,1
        call    lee_arc
        jmp     salida
rdp:    mov     aux,0
        call    dep_paso
        jmp     salida
rdc:    mov     aux,1
        call    dep_corre
        jmp     salida
rdrc:   call    crear_rup
        jmp     salida
rdra:   call    act_rup
        jmp     salida
rdrb:   call    borrar_rup
        jmp     salida
rdrm:   call    most_rup
        jmp     salida
rdcis:  call    set_cic
        jmp     salida
rdcia:  call    act_cic
        jmp     salida
rdd:    call    act_dis
        jmp     salida
ri:     call    inserta
        jmp     salida
ren:    call    ensambla
        jmp     salida
rrev:   call    atras
        jmp     salida
```

```
;*****************************************************
;***************         FMAC.ASM          ***************
;*****************************************************
;*****     Macros usados en el archivo INTHMAQ.ASM    *****
;*****************************************************
;
;-------------------------------------------------------------------
;Macro              Parametros        Funcion
;-------------------------------------------------------------------
;actuar                               salta a una de las dir. en una tabla
;ayuda              dir,nlin,color    presenta ventana con instrucciones
;bufcla             pnclave,pbuf      encuentra la posicion de un buffer
;curs               x                 pone un "cursor" en x,24
;explica            pnclave           muestra la explicacion(PNCLAVE)
;getkey                               recibe dato del teclado (BIOS)
;getxpr                               obtiene ultimo nibble <>0 en BUFFER
;guarpantalla                         guarda una pantalla
;iluminaop                            resalta la opcion actual
;inbuff             dir               recibe string desde teclado (BIOS)
;lmppnt                               limpia la pantalla
;menaba                               cursor abajo en ventana de subopc.
;menarr                               cursor arriba
;mender                               cursor a la derecha
;menizq                               cursor a la izquierda
;menulinea          pnclave           muestra menu(PNCLAVE)
;menuopc                              pone otro atributo a la tecla clave
;menuvent                             muestra menu en forma de ventana
;mostrar            pdir,plin,pcol    muestra pregunta y espera respuesta
;movbuf             pbuf1,pbuf2       PBUF1 <-- PBUF2
;nocurs             x                 pone atributo normal en x,24
;nunivel            pbuf,pniv,pnum    pone PNUM en PBUF(PNIV)
;opcion?                              espera que se elija una opcion
;pchar              car               escribe CAR (DOS)
;pmenu              pnlin,pcolor      muestra menu string apuntado por AX
;posdir             pvarx,pvary,dir   convierte pos. de pantalla en direccion
;restpantalla                         restaura ultima pantalla salvada
;rutina                               pasa control al usuario y lo recibe
;salvareg                             push a todo
;traereg                              pop a todo
;valida?                              ve si una opcion es del menu del usuar.
;subrut                               subrutinas de todos los macros
;-------------------------------------------------------------------

;******  ACTUAR   ***************
;Busca el AL-esimo elemento de la tabla de direcciones
;que ira despues de MTABLA para saltar a esa direccion.
actuar   macro
         local    mtabla
         push     bx
         mov      bx,offset mtabla
         call     mact
         pop      bx
         jmp      dword ptr diract
mtabla:
         endm

;******  AYUDA    ***************
;Escribe NLIN lineas separadas por ceros desde DIR en COLOR.
```

```
;Ocupa columnas 1 a 78 y las filas del centro de la pantalla.
ayuda    macro    dir,nlin,color
         push     ax
         push     cx
         mov      cl,nlin
         mov      ch,color
         mov      ax,offset dir
         call     may
         pop      cx
         pop      ax
         endm

;******  BUFCLA   ***************
;Busca PBUF en TABBUF y pone su ubicacion en PNCLAVE
bufcla   macro    pnclave,pbuf
         push     si
         push     ax
         mov      si,offset pbuf
         call     mbc
         mov      al,mbcbyte
         mov      pnclave,al
         pop      ax
         pop      si
         endm

;******  CURS     ***************
;pone el atributo de (x,24) en inverso.
curs     macro    x
         push     ax
         mov      al,x
         mov      ah,bnsubb
         call     mcu
         pop      ax
         endm

;******  EXPLICA  ***************
;presenta explicacion de opcion cuyo # es pnclave
explica  macro    pnclave
         push     cx
         mov      cl,pnclave
         call     mex
         pop      cx
         endm

;******  GETKEY   ***************
;Espera hasta recibir un dato del teclado y lo manda en GETWORD
getkey   macro
         call     mgk
         mov      ax,getword
         endm


;****** GETXPR    ***************
;Obtiene de BUFFER el ultimo nibble distinto de 0, en XPREV.
;Sirve para regresar (con ^L o rutina inversa) a
;la opcion que dio origen a cierto menu.
```

```
getxpr    macro
          call      mgx
          endm

;******  GUARPANTALLA **********
;guarda la pantalla en PANBUF
guarpantalla      macro
          call      mgp
          endm

;******  ILUMINAOP **************
;Resalta la opcion actual (indicada por la variable X)
iluminaop         macro
          call      mio
          endm

;******  INBUFF   ****************
;Recibe desde el teclado el buffer DIR cuya longitud se indica en
;el primer byte del buffer. En el segundo byte se obtendra el
;numero de bytes recibidos. Solo se aceptan letras o numeros.
;Para edicion, solo Backspace. Pita si se ha llegado al limite.
inbuff    macro     dir
          push      si
          mov       si,offset dir
          call      mib
          pop       si
          endm


;******  LMPPNT   ****************
;Borra la pantalla
lmppnt    macro
          push      di
          push      cx
          push      ax
          mov       cx,2000
          mov       di,0
          mov       ax,0720h
          rep       stosw
          pop       ax
          pop       cx
          pop       di
          endm

;******  MENABA   ****************
;Mueve cursor abajo en la ventana de subopciones.
menaba    macro
          call      mmab
          endm

;******  MENARR   ****************
;Similar a MENABA.
menarr    macro
          call      mma
          endm
```

```
;******  MENDER   ***************
;Mueve cursor de opcion actual a la der
mender   macro
         call     mmd
         endm

;******  MENIZQ   ***************
;Similar a MENDER.
menizq   macro
         call     mmi
         endm

;******  MENULINEA ************
;Muestra menu (PNCLAVE)
menulinea         macro    pnclave
         push     cx
         mov      cl,pnclave
         call     mml
         pop      cx
         endm

;******  MENUOPC  ************
;Pone otro atributo a letra clave de un menu
menuopc  macro
         call     mmo
         endm

;******  MENUVENT ***************
;Muestra menu(NCLAVE1) en forma de ventana.
menuvent macro
         call     mmv
         endm

;******  MOSTRAR  ***************
;Muestra pregunta apuntada por PDIR y espera respuesta. La pregunta
;debe tener esta forma: Mensaje,0,# de respuestas, lista de
;respuestas. En AL sale el numero de la respuesta elegida.
mostrar  macro    pdir,plin,pcol
         push     ax
         mov      ah,plin
         mov      al,pcol
         mov      moslin,ah
         mov      moscol,al
         mov      ax,offset pdir
         call     mmos
         pop      ax
         mov      al,mnmos
         endm

;******  MOVBUF   ***************
;Mueve PBUF1 a PBUF2.
movbuf   macro    pbuf2,pbuf1
         push     si
         push     di
         mov      si,offset pbuf1
         mov      di,offset pbuf2
```

```
          call      mmb
          pop       di
          pop       si
          endm

;******  NOCURS    ***************
;pone el atributo de (x,24) en normal.
nocurs    macro     x
          push      ax
          mov       al,x
          mov       ah,bnnorm
          call      mcu
          pop       ax
          endm

;******  NUNIVEL   ***************
;Pone PNUM en el buffer PBUF en el nivel PNIV.
nunivel   macro     pbuf,pniv,pnum
          push      cx
          push      dx
          mov       cl,pniv
          mov       ch,pnum
          mov       dx,offset pbuf
          call      mnn
          pop       dx
          pop       cx
          endm

;******  OPCION?   ***************
;Espera que se digite una opcion. Saca en MOPACT el numero de
;opcion. Las opciones son: Flechas,ENTER,F2,^K,^L,^F1 y otros.
;Ademas es aqui donde se almacenan los datos al grabar macros
;y se leen del macro en vez del teclado al correr o probarlos.
opcion?   macro
          call      mop
          endm

;******  PCHAR     ***************
;Usa funcion del DOS para escribir CAR en pantalla.
pchar     macro     car
          push      dx
          mov       dl,car
          call      mpch
          pop       dx
          endm

;******  PMENU     ***************
;Muestra menu apuntado por AX. Al salir MPMPRX
;da la direccion del primer byte luego del 0.
pmenu     macro     pnlin,pcolor
          push      cx
          push      dx
          mov       cl,pnlin
          mov       dh,pcolor
          call      mpm
          pop       dx
```

```
          pop       cx
          endm

;******  POSDIR   ***************
;Da la direccion PDIR correspondiente a (PVARX,PVARY)
;para acceso directo a la memoria de video.
posdir    macro     pvarx,pvary,pdir
          push      ax
          push      bx
          push      cx
          mov       cl,pvary
          mov       bl,pvarx
          call      mpd
          mov       bx,mpddir
          mov       pdir,bx
          pop       cx
          pop       bx
          pop       ax
          endm

;******  RESTPANTALLA **********
;Trae de PANBUF la ventana que fue salvada en GUARPANTALLA.
restpantalla      macro
          call      mrp
          endm

;******  RUTINA   ***************
;Pone la direccion de la rutina(NCLAVE) de usuario en DIRSAL y la de
;regreso en MRAUX2 y MRAUX3. Pasa control al usuario, que debe regresar
;poniendo RETORNO al fin de su rutina para regresar a MRUT1.
rutina    macro
          mov       cl,nclave
          mov       al,ninfbuf
          mul       cl
          add       ax,offset tabinf          ;busco dir. de rutina(nclave)
          add       ax,4
          mov       si,ax
          mov       bx,[si]
          mov       al,[si+2]                 ;y tipo de rutina.
          mov       tiprut,al                 ;Preparo el salto.
          mov       mraux1,al                 ;guardo tipo de rutina.
          mov       bp,offset dirsal
          mov       ds:[bp],bx
          mov       ds:[bp+2],cs
          mov       bx,offset mrut1           ;Preparo dir. de regreso.
          mov       mraux2,cs
          mov       mraux3,bx
          jmp       dword ptr dirsal  ;Control al usuario.
mrut1:    mov       al,tiprut                 ;Regreso a FLUJO.
          cmp       al,mraux1                 ;Ha cambiado TIPRUT?
          je        mrut2
          mov       x,1                       ;ademas X,Y previos pueden
          mov       y,0                       ;estar errados.
mrut2:
          endm
```

```
;******  SALVAREG **************
;Push a todo.
salvareg          macro
          push    ax
          push    bx
          push    cx
          push    dx
          push    bp
          push    si
          push    di
          push    ds
          push    es
          push    ss
          endm

;******  TRAEREG  **************
;Pop a todo.
traereg   macro
          pop     ss
          pop     es
          pop     ds
          pop     di
          pop     si
          pop     bp
          pop     dx
          pop     cx
          pop     bx
          pop     ax
          endm

;******  VALIDA?  **************
;Si en OPCION? se digito una opcion distinta de las que maneja
;normalmente, ACTUAR manda el flujo aca para ver si es una opcion
;puesta por el usuario en el menu. Si es asi, AL=numero de opcion.
;Si el primer byte despues de la ultima opcion del menu es FF se
;supone que la ultima es la "opcion de escape" para todas las
;posibilidades que no consten en el menu.
valida?   macro
          call    mv?
          mov     al,mv?aux
          endm

;******  SUBRUT *********
;Este es un macro que contiene todas las subrutinas llamadas por los
;otros macros de este archivo. Se hizo de esta manera para evitar el
;uso de demasiadas banderas, y que su ausencia no produzca "Phase
;errors" al momento del ensamblado.
subrut    macro

mact:
          salvareg
          dec     al                ;pongo el AL-esimo elemento de la
          mov     cl,2              ;tabla en DIRACT y CS en DIRACT+2
          mul     cl                ;para hacer el JMP al regresar.
          add     ax,bx
          mov     bp,offset diract
```

```
        push    si
        mov     si,ax
        mov     bx,cs:[si]
        pop     si
        mov     ds:[bp],bx
        mov     ds:[bp+2],cs
        traereg
        ret

may:
        push    dx
        push    bx
        mov     bx,ax
        mov     dl,23           ;centro el texto en pantalla:
        sub     dl,cl           ;linea inicial= (23-#lineas)/2 -1
        shr     dl,1
        dec     dl
        mov     maylin,dl
        dec     maylin
        mov     maycol,ch
        mov     ax,offset blanco
may1:   pmenu   maylin,maycol   ;primero una linea en blanco.
        inc     maylin
        mov     mpmprx,bx
may2:   mov     ax,mpmprx       ;Ahora el resto de lineas. MPMPRX
        pmenu   maylin,maycol   ;da la direccion del primer byte
        inc     maylin          ;despues del 0.
        dec     cl
        jnz     may2
        mov     ax,offset blanco
        pmenu   maylin,maycol   ;Linea en blanco al final.
        pop     bx
        pop     dx
        ret

mbc:
        salvareg
        mov     al,nmaxbuf      ;Busca un buffer en la tabla TABBUF
        cmp     al,0            ;y entrega el numero clave
        jne     mbc0            ;que servira para obtener toda la
        jmp     mbc4            ;informacion necesaria acerca de la
mbc0:   mov     mbcmenu,0       ;opcion representada por el buffer.
        mov     mbcbuf,si       ;Si el buffer no existe, clave=FF.
        mov     mbcpos,offset tabbuf
mbc1:   mov     mbcbyte,0
        mov     bx,mbcbuf
mbc3:   mov     ah,[bx]
        inc     bx
        mov     bp,mbcpos
        xor     ch,ch
        mov     cl,mbcbyte
        inc     mbcbyte
        mov     di,cx
        cmp     ah,ds:[bp+di]
        jnz     mbc2
        cmp     mbcbyte,8
```

```
        jnz     mbc3
        mov     al,mbcmenu
        jmp     mbc5
mbc2:   inc     mbcmenu
        mov     ah,nmaxbuf
        cmp     ah,mbcmenu
        je      mbc4
        add     mbcpos,8
        jmp     mbc1
mbc4:   mov     al,0ffh
mbc5:   mov     mbcbyte,al
        traereg
        ret

mcu:
        salvareg                ;Pone el atributo dado por AH en
        posdir  0,24,mcudir     ;la posicion (AL,24)
        mov     si,mcudir
        mov     bh,ah
        mov     cl,2
        mul     cl
        add     si,ax
        inc     si
        mov     es:[si],bh
        traereg
        ret

mex:
        salvareg                ;Busca en TABINF la direccion de la
        mov     al,ninfbuf      ;explicacion y la pone en linea 23.
        mul     cl
        add     ax,offset tabinf
        mov     bx,ax
        mov     ax,[bx]
        pmenu   24,colexp
        traereg
        ret

mgk:
        salvareg                ;Uso funcion del BIOS que espera
        mov     ah,0            ;que se pulse una tecla.
        int     16h
        mov     getword,ax
        traereg
        ret

mgx:
        salvareg                ;se divide NIVEL/2, se suma a direccion
        mov     cl,nivel        ;inicial de BUFFER, y se obtiene el msn
        shr     cl,1            ;distinto de 0 en ese byte.
        xor     ch,ch           ;Ese es el ultimo nibble<>0 en BUFFER
        mov     si,offset buffer
        add     si,cx
        mov     al,[si]
        mov     ah,al
        shr     ah,1
```

```
        shr     ah,1
        shr     ah,1
        shr     ah,1
        cmp     ah,0
        jz      mgx1
        mov     al,ah
mgx1:   mov     xprev,al
        traereg
        ret

mgp:
        salvareg                    ;paso pantalla a PANBUF.
        xor     si,si
        mov     di,offset panbuf
        mov     bx,ds
        mov     es,bx
        mov     ds,videoseg
        mov     cx,3680
        rep     movsb
        traereg
        ret

mio:
        salvareg                    ;Resalta la opcion actual, dada por X.
        mov     es,videoseg
        mov     dl,collact
        mov     al,x
mio13:  posdir  1,23,miodir         ;trabajo directo en pantalla.
        mov     si,miodir           ;busco la X-esima palabra en la
        mov     di,si               ;linea 23 y cambio su atributo.
        push    es
        pop     ds
        mov     cl,al
mio131: lodsw
        cmp     al,20h              ;salto espacios iniciales.
        je      mio131
        sub     si,2
        dec     cl
        jz      mio16
mio14:  lodsw
        cmp     al,20h              ;salto palabras.
        jne     mio14
mio15:  lodsw
        cmp     al,20h
        je      mio15
        dec     si
        dec     si
        dec     cl
        jnz     mio14
mio16:  mov     di,si               ;resalto palabra.
        lodsw
        mov     ah,dl
        cmp     al,20h
        je      mio17
        cmp     al,0
        je      mio17
```

```
        stosw
        jmp     mio16
mio17:  traereg
        ret

mib:
        salvareg                        ;obtengo string desde teclado.
        mov     cl,[si]                 ;maximo # de caracteres
        mov     ch,0                    ;contador
        mov     dx,si                   ;puntero auxiliar
        add     si,2                    ;bytes 0 y 1 de uso del macro
        posdir  1,24,mibdir             ;direccion de primer caracter
        mov     bx,mibdir
mib1:   mov     mibx,ch
        inc     mibx
        curs    mibx
mib2:   getkey
        cmp     al,08h                  ;Backspace?
        jne     mib21
        jmp     mib4
mib21:  cmp     al,0dh                  ;Enter?
        jne     mib22
        jmp     mib5
mib22:  cmp     al,30h                  ;<"0"?
        jb      mib2
        cmp     al,7bh                  ;>"z"?
        jnb     mib2
        cmp     al,3ah                  ;es un numero?
        jb      mib3
        cmp     al,41h                  ;"9"<al<"A"?
        jb      mib2
        cmp     al,5bh                  ;es una mayuscula?
        jb      mib3
        cmp     al,61h                  ;"Z"<al<"a"?
        jb      mib2
        sub     al,20h                  ;es minuscula, hacer mayuscula
mib3:   nocurs  mibx
        cmp     ch,cl                   ;contador=maximo?
        jne     mib31
        pchar   07h                     ;si, beep.
        jmp     mib1
mib31:  mov     [si],al                 ;AL al buffer
        inc     si                      ;apunto prox. direccion
        mov     es:[bx],al              ;AL a pantalla
        add     bx,2                    ;apunto prox. direccion
        inc     ch                      ;contador+1
        jmp     mib1
mib4:   nocurs  mibx
        cmp     ch,0                    ;buffer vacio?
        jne     mib41
        pchar   07h                     ;si, beep.
        jmp     mib1
mib41:  dec     si                      ;borro ultimo caracter
        mov     byte ptr[si],0          ;en el buffer
        sub     bx,2                    ;y en pantalla
        mov     byte ptr es:[bx],20h
```

```
        dec     ch              ;contador-1
        jmp     mib1
mib5:   nocurs  mibx
        mov     si,dx           ;fin de rutina: byte 1
        inc     si              ;indica # de bytes recibidos.
        mov     [si],ch
        traereg
        ret

mmab:
        salvareg
        mov     cl,nclave1      ;Busca el # de subopciones que tiene
        mov     al,ninfbuf      ;el menu de la opcion actual dada por
        mul     cl              ;NCLAVE1. Si Y supera ese #, Y=0.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mmab1:  lodsb
        cmp     al,0
        jne     mmab1
        lodsb
        inc     al
        inc     y
        cmp     al,y
        jne     mmab2
        mov     y,0
mmab2:  traereg
        ret

mma:
        salvareg
        mov     cl,nclave1      ;Busca # de subopc. de menu(NCLAVE1)
        mov     al,ninfbuf      ;Si Y=FF, Y igual a ese #.
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mma1:   lodsb
        cmp     al,0
        jne     mma1
        lodsb
        dec     y
        cmp     y,0ffh
        jne     mma2
        mov     y,al
mma2:   traereg
        ret
```

```
mmd:
        salvareg
        mov     cl,nclave           ;Busca # de opciones de menu actual.
        mov     al,ninfbuf          ;Si X supera ese valor, X=1.
        mul     cl                  ;Siempre Y=0.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mmd1:   lodsb
        cmp     al,0
        jne     mmd1
        lodsb
        inc     al
        inc     x                   ;a la derecha.
        cmp     al,x
        jne     mmd2
        mov     x,1
mmd2:   mov     y,0
        traereg
        ret

mmi:
        salvareg
        mov     cl,nclave           ;Busca # de opciones de menu actual.
        mov     al,ninfbuf          ;Si X=0, X igual a ese #. Siempre Y=0.
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]
        pop     bx
mmi1:   lodsb
        cmp     al,0
        jne     mmi1
        lodsb
        dec     x
        cmp     x,0
        jne     mmi2
        mov     x,al
mmi2:   mov     y,0
        traereg
        ret

mml:
        salvareg
        mov     al,ninfbuf          ;Muestra el menu(CL) y resalta la
        mul     cl                  ;opcion actual.
        add     ax,offset tabinf
        inc     ax
        inc     ax
```

```
        mov     bx,ax
        mov     ax,[bx]
        push    ax
        pmenu   23,collin
        iluminaop
        pop     ax
        menuopc
        traereg
        ret

mmo:    salvareg
        mov     si,ax
        push    si
mmo1:   lodsb                   ;encuentra fin del menu
        cmp     al,0
        jne     mmo1
        mov     cl,ds:[si]      ;cx = # de opciones
        xor     ch,ch
        mov     di,si
        inc     di              ;di apunta a letra clave
        pop     si
        mov     bl,0            ;bl = # de columna
mmo2:   mov     ah,ds:[di]      ;ah = letra clave
mmo3:   lodsb                   ;ah = letra del menu
        inc     bl              ;actualizo # de columna
        cmp     al,ah           ;letra del menu = letra clave?
        jne     mmo3
        push    di
        posdir  bl,23,di        ;di = posicion de letra clave
        inc     di              ;cambia atributo
        mov     byte ptr es:[di],0ah
        pop     di
        inc     di
        loop    mmo2
        traereg
        ret
mmv:
        salvareg                ;Muestra ventana se subopc.
        mov     cl,nclave1      ;primero veo si la opcion
        mov     al,ninfbuf      ;tiene algun submenu.
        mul     cl              ;Si es asi, la rutina
        add     ax,offset tabinf        ;debe ser de tipo 3.
        add     ax,6
        mov     bx,ax
        cmp     byte ptr[bx],3
        jz      mmv3
        jmp     mmv7
mmv3:   mov     cl,nclave               ;Ahora busco la direccion
        mov     al,ninfbuf              ;del menu al que pertenece
        mul     cl                      ;esta opcion en TABINF.
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]                 ;SI apunta al inicio del menu.
```

```
        pop     bx                      ;Voy a contar los bytes hasta
        mov     cl,0                    ;la opcion actual (x) para
saber
        mov     dl,x                    ;columna inicial de ventana.
mmv30:  lodsb                           ;Salto espacios iniciales
        inc     cl                      ;pero los cuento.
        cmp     al,20h
        je      mmv30
        cmp     x,1                     ;si x=1 ya tengo la columna
        jnz     mmv31
        dec     cl
        mov     mmvx1,cl
        jmp     mmv4
mmv31:  dec     dl                      ;DL: cuantas opc mas debo
saltar.
mmv32:  lodsb                           ;Cuento bytes hasta encontrar
        inc     cl                      ;un espacio (CL contador).
        cmp     al,20h
        jne     mmv32
mmv33:  lodsb                           ;sumo espacios entre opciones
        inc     cl
        cmp     al,20h
        je      mmv33
        dec     dl                      ;una opc menos que saltar
        jnz     mmv32
        dec     cl                      ;Termino. Ventana empieza en
        mov     mmvx1,cl                ;columna anterior.
mmv4:   mov     mmvcar,0
        mov     cl,nclave1              ;Ahora busco direccion de menu
        mov     al,ninfbuf              ;asociado a la opcion actual
        mul     cl                      ;en TABINF
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax                   ;SI apunta a primer byte
        mov     si,[bx]                 ;de ese menu
        pop     bx
        push    si
mmv40:  lodsb                           ;no tomo en cuenta los
        cmp     al,20h                  ;espacios iniciales.
        je      mmv40
        dec     si
        mov     cx,1                    ;Veo cuantas filas y columnas
mmv41:  lodsb                           ;ocupa la ventana
        cmp     al,20h                  ;Los espacios no cuentan
        jz      mmv42
        cmp     al,0                    ;termina si AL=0
        jne     mmv410
        mov     mmvlin,cl               ;# de subopciones del menu
        jmp     mmv5
mmv410: inc     ch                      ;CH: # de car de subopcion
        cmp     ch,mmvcar               ;MMVCAR: maximo CH logrado
        jb      mmv41
        mov     mmvcar,ch
        jmp     mmv41
```

```
mmv42:  lodsb                           ;saltar espacios
        cmp     al,20h
        je      mmv42
        cmp     al,0                    ;termina si AL=0
        jne     mmv43
        mov     mmvlin,cl               ;# de subopciones del menu
        jmp     mmv5
mmv43:  inc     cl                      ;otra subopcion
        mov     ch,0                    ;# de car = 0
        dec     si
        jmp     mmv41
mmv5:   pop     si
        mov     al,mmvlin               ;Busco la linea inicial
        inc     al                      ;de la ventana incluidas
        inc     al                      ;dos lineas de recuadro.
        mov     ah,23
        sub     ah,al
        mov     mmvlin,ah
        mov     mmvlin1,ah              ;servira despues
        inc     mmvcar
        posdir  mmvx1,mmvlin,mmvdir     ;dir. de esquina superior
        mov     di,mmvdir               ;izquierda en pantalla
        mov     es,videoseg
        mov     al,esqsi                ;dibujar esa esquina
        mov     ah,colven               ;en color de ventana
        stosw
        mov     al,linsup               ;dibuja linea superior
        mov     cl,mmvcar
        xor     ch,ch
        rep     stosw
        mov     al,esqsd                ;dibuja esq. sup. der.
        stosw
mmv50:  inc     mmvlin                  ;proxima linea
        mov     bl,mmvlin
        cmp     bl,22                   ;linea inferior si BL=22
        je      mmv6
        posdir  mmvx1,mmvlin,mmvdir     ;escribir las subopciones
        mov     di,mmvdir
        mov     al,linlat               ;pero antes el recuadro.
        mov     bl,mmvcar               ;long de la linea
        stosw
mmv501: lodsb                           ;no tomo en cuenta los
        cmp     al,20h                  ;espacios iniciales
        je      mmv501
        dec     si
mmv51:  lodsb                           ;leo byte del menu
        stosw                           ;lo escribo en pantalla
        dec     bl
        cmp     al,20h                  ;He terminado la opcion?
        je      mmv52
        cmp     al,0                    ;He terminado el menu?
        jne     mmv51
        dec     di                      ;en ese caso borro el 00H
        dec     di                      ;que escribi
        mov     al,20h                  ; y pongo un " "
        stosw
```

```
mmv52:  cmp     bl,0                    ;lleno el resto de la linea
        je      mmv53                   ;con espacios
        mov     al,20h
mmv521: stosw
        dec     bl
        jnz     mmv521
mmv53:  mov     al,linlat               ;y termino con recuadro.
        stosw
mmv54:  lodsb                           ;salto espacios
        cmp     al,20h
        jz      mmv54
        dec     si
        jmp     mmv50
mmv6:   posdir  mmvxi,mmvlin,mmvdir     ;dibuja linea inferior
        mov     di,mmvdir
        mov     al,esqii
        mov     ah,colven
        stosw
        mov     al,linsup
        mov     cl,mmvcar
        xor     ch,ch
        rep     stosw
        mov     al,esqid
        stosw                           ;fin de dibujo de ventana.
        cmp     y,0                     ;existe              subopcion
seleccionada?
        jz      mmv7                    ;no, termina trabajo.
        mov     al,y                    ;si, resaltarla cambiando
mmv63:  mov     bl,mmvlin1              ;el atributo de la linea.
        add     al,bl
        mov     mmvlin,al
        posdir  mmvxi,mmvlin,mmvdir
        mov     di,mmvdir
        mov     ax,3
        add     di,ax
        mov     bl,mmvcar
        mov     al,colact
mmv64:  stosb
        inc     di
        dec     bl
        jnz     mmv64
mmv7:   traereg
        ret

mmos:
        salvareg
        pmenu   moslin,moscol   ;Presenta pregunta y espera respuesta.
        mov     si,ax
mmos0:  lodsb
        cmp     al,0
        jne     mmos0
        mov     mdirop,si
mmos1:  lodsb                   ;Obtengo el # de opciones.
        mov     cl,al
        mov     dl,al
        getkey                  ;obtengo respuesta.
```

```
          cmp       al,'a'              ;minusculas --> mayusculas.
          jc        mmos11
          cmp       al,'z'+1
          jnc       mmos11
          sub       al,20h
mmos11:   mov       bx,ax
mmos2:    mov       al,[si]             ;comparo y si no es ninguna de las
          inc       si                  ;posibles, vuelve a pedir respuesta.
          sub       cl,1
          jc        mmos4
          cmp       al,0
          je        mmos3
mmos3:    cmp       al,bl
          jne       mmos2
          jmp       mmos5
mmos4:    mov       si,mdirop
          jmp       mmos1
mmos5:    sub       dl,cl
          mov       mnmos,dl            ;# de respuesta en MNMOS.
          traereg
          ret

mmb:
          salvareg                      ;mueve 8 bytes.
          push      ds
          pop       es
          mov       cx,8
          rep       movsb
          traereg
          ret

mnn:
          salvareg
          test      cl,1                ;pongo CH en el CL-esimo nibble
          jz        mnn1                ;del buffer apuntado por DX.
          mov       al,16
          mul       ch
          cmp       al,0
          jnz       mnn4
          mov       ch,0fh
          mov       al,0ffh
          jmp       mnn1
mnn4:     mov       ch,al
mnn1:     shr       cl,1
          mov       bl,cl
          xor       bh,bh
          add       bx,dx
          cmp       ch,0
          jz        mnn5
          cmp       al,0ffh
          jnz       mnn2
mnn5:     and       [bx],ch
          jmp       mnn3
mnn2:     or        [bx],ch
mnn3:     traereg
          ret
```

```
mop:
        salvareg                        ;maneja opciones.
        mov     cl,nclave1              ;Primero veo de que tipo es la
        mov     al,ninfbuf              ;opcion actual. (Esto es usado
        mul     cl                      ;al salir de este macro).
        add     ax,offset tabinf
        add     ax,6
        mov     si,ax
        mov     al,[si]
        mov     tiprut1,al
mop2:   getkey
mop3:   cmp     al,0                    ;es un car.especial?
        je      mop32
mop31:  cmp     al,cr                   ;no, es Enter?
        jne     mop311
        mov     al,5
        jmp     mop4
mop311: cmp     al,ctrlk                ;es ^K?
        jne     mop314
        mov     al,9
        jmp     mop4
mop314: cmp     al,1bh                  ;es ESC?
        jne     mop315
        mov     al,10
        jmp     mop4
mop315: mov     mopaux,al               ;es otro?
        mov     al,6
        jmp     mop4
mop32:  cmp     ah,fleizq               ;es <--?
        jne     mop321
        mov     al,1
        jmp     mop4
mop321: cmp     ah,fleder               ;es -->?
        jne     mop322
        mov     al,2
        jmp     mop4
mop322: cmp     ah,flearr               ;es flecha arriba?
        jne     mop323
        mov     al,3
        jmp     mop4
mop323: cmp     ah,fleaba               ;es flecha abajo?
        jne     mop324
        mov     al,4
        jmp     mop4
mop324: cmp     ah,f1                   ;es F1?
        jne     mop325
        mov     al,8
        jmp     mop4
mop325: cmp     ah,f2                   ;es F2?
        jne     mop326
        mov     al,7
        jmp     mop4
mop326: mov     mopaux1,ah              ;es otro.
        mov     mopaux,0
        mov     al,6
mop4:   mov     mopact,al
```

```
        cmp     bopc,1          ;tengo ventana de opciones?
        jne     mop5
        restpantalla            ;si,recupero pantalla.
mop5:   traereg
        ret

mpch:
        salvareg
        mov     ah,2            ;Uso rutina del DOS para escribir DL.
        int     21h
        traereg
        ret

mpm:
        salvareg
        push    dx              ;pone un string terminado en 0
        xor     ch,ch           ;en linea CL con atributo DH.
        mov     si,ax           ;EL string es apuntado por AX.
        mov     al,longlin
        mul     cl
        mov     es,videoseg
        mov     di,ax
        pop     dx
        mov     ah,dh
        push    di
        mov     al,20h
        mov     cx,80           ;borro la linea.
        rep     stosw
        pop     di
        inc     di
        inc     di
mpm0:   lodsb                   ;pongo el string.
        cmp     al,0
        je      mpm1
        stosw
        jmp     mpm0
mpm1:   mov     mpmprx,si       ;MPMPRX apunta a proximo string.
        traereg
        ret

mpd:
        salvareg
        mov     al,160          ;Direccion = CL*160 + BL.
        mul     cl
        xor     bh,bh
        add     ax,bx
        add     ax,bx
        mov     mpddir,ax
        traereg
        ret

mrp:
        salvareg                ;Paso PANBUF a pantalla.
        xor     di,di
        mov     es,videoseg
        mov     si,offset panbuf
```

```
        mov     cx,3680
        rep     movsb
        traereg
        ret

mv?:
        salvareg                        ;Busca si la eleccion esta
        mov     cl,nclave               ;en el menu del usuario.
        mov     al,ninfbuf
        mul     cl
        add     ax,offset tabinf
        inc     ax
        inc     ax
        push    bx
        mov     bx,ax
        mov     si,[bx]                 ;SI apunta al menu
        pop     bx
mv?1:   lodsb
        cmp     al,0
        jne     mv?1
        lodsb                           ;SI apunta a primera opcion
        mov     cl,al                   ;AL tiene el # de opciones
        mov     dl,al
        push    si                      ;si el primer byte despues
        push    ax                      ;de las opciones validas
        mov     mv?aux1,0
        xor     ah,ah                   ;es FF, entonces evacuar
        add     si,ax                   ;las teclas no validas
        cmp     byte ptr[si],0ffh       ;por la ultima opcion.
        jne     mv?11
        mov     mv?aux1,al              ;dejo MV?AUX1 como bandera
mv?11:  pop     ax
        pop     si
        mov     al,mopaux               ;A las minusculas
        cmp     al,'a'
        jb      mv?2
        cmp     al,'z'
        ja      mv?2
        sub     al,20h                  ;las hago mayusculas.
mv?2:   mov     bl,al
mv?21:  lodsb
        cmp     al,bl                   ;veo si opcion es valida
        je      mv?3
        dec     cl
        jnz     mv?21
        cmp     mv?aux1,0               ;no fue valida, veo si hay
        je      mv?22                   ;rutina de escape
        mov     al,mv?aux1              ;si hay esa rutina
        mov     mv?aux,al               ;salgo con respuesta
        xor     ah,ah                   ;y ZF=1
        jmp     mv?4
mv?22:  or      al,1                    ;no hay esa rutina,ZF=0
        jmp     mv?4
mv?3:   dec     cl                      ;opcion valida,
        sub     dl,cl                   ;guardo respuesta
        mov     mv?aux,dl
```

```
;*************************************
;*********      DEF.ASM         **********
;*************************************

IF1
        INCLUDE MACVNT.ASM              ;macro de ventanas
        INCLUDE MACPRO.ASM              ;macros del PRO
        INCLUDE MACARC.ASM              ;macros de archivos
ENDIF

        EXTRN VIDEOSEG:WORD, PANBUF:BYTE
        EXTRN BUFF_RX:BYTE

        PUBLIC PC,RAM_EMB,AUX
        PUBLIC R0,R1,R2,R3,R4,R5,R6,R7
        PUBLIC P0,REGSP,DPL,DPH,PCON,TCON,TMOD
        PUBLIC TL0,TL1,TH0,TH1,P1,SCON,SBUF,SBUF_IN
        PUBLIC P2,IE,P3,IP,EK0,EK1,EK2,EK3,EK4
        PUBLIC P0_PIN,P1_PIN,P2_PIN,P3_PIN
        PUBLIC MCON,TA,PSW,ACC,B
        PUBLIC SEGDAT2,SEGDAT1
        PUBLIC ROM_RAM_E,RAM_EMB_M

DSEG SEGMENT PUBLIC

segdat   dw 0                           ;segmento DSEG
segdat1  dw 0                           ;segmento DSEG1
segdat2  dw 0                           ;segmento DSEG2
seg_dat  db 0                           ;bandera de segmento
modo     db 0                           ;modo de operacion
getword  dw 0                           ;salida de getkey
aux      dw 0                           ;variable auxiliar
revers   db 70H                         ;video reverso
normal   db 07H                         ;video normal

;Variables para manejo de ventanas

vntmono  db "DALLAS.VNT",0
varea    db 50*256 DUP(0)
vlin0    db "LINEA0",0
vds5000t db "DS5000T",0
vregcpu  db "REGCPU",0
vband    db "BANDERA",0
vregesp  db "REGESP1",0
vmem1    db "MEM1",0
vmem2    db "MEM2",0
vcod     db "CODIG1",0
vpuerto  db "PUERTOS",0
vpcon    db "PCON",0
vrup     db "RUPTURA",0
vciclo   db "CICLOS",0
vciclo1  db "CICLOS1",0
vdisplay db "DISPLAY",0
vhelp1   db "HELP1",0
vhelp2   db "HELP2",0
vhelp3   db "HELP3",0
lregcpu  db 47 dup (0)
lband    db 5 dup(0)
```

```
lregesp  db 60 dup (0)
lmem1    db 28*4 dup (0)
lmem2    db 28*4 dup (0)
lcod     db 437 dup (0)
lpuerto  db 8*8 dup (0)
lpcon    db 8 dup (0)
lrup     db 64 dup (0)
lciclo   db 5 dup (0)
ldisplay db " ON"

;Variables con mensajes

codigo    db "       C O D I G O        ",0
codig_m   db "ADDR  MNEMO   PARAMTERS  ",0
area1     db "Area 1 :",0
area2     db "Area 2 :",0
inter     db "Interna    ",0
exter     db "Embebed    ",0
emb_ext   db "/RAM",0
emb_int   db "/ROM",0
nec_arc   db "Este programa requiere un archivo de entrada",0
nombarc   db "Ingrese el nombre del archivo --->",0
nec_mod   db "Seleccione Modo de Operaci",162,"n",0
modo?     db "(S) Simulador  (E) Ejecutor Controlado  (C) Carga serial",0
e_arc_ven db "Error en apertura de archivo de ventanas",0
n_arc_ven db "No se encontro archivo de ventanas",0
f_arc_ven db "Falta espacio para archivo de ventanas",0
arc_e_abr db "Error en la apertura del archivo Objeto",0
arc_e_lee db "Error en la lectura del archivo Objeto",0
arc_e_int db "El archivo Objeto no es de formato INTEL",0
dallas    db "DALLAS.HEX",0
tx_f_d    db "< !! Transmitiendo archivo Monitor !! >",0
dal_a     db "Error en la apertura del archivo Monitor",0
dal_l     db "Error en la lectura del archivo Monitor",0
dal_i     db "El archivo Monitor no es de formato INTEL",0
dal_t     db "Error en la transmisi",162,"n del archivo Monitor",0
dal_o     db "Error en la transmisi",162,"n del archivo Objeto",0
err_pc    db "En este modo el m",161,"nimo valor de PC es 0806H",0
er_memi   db "El m",160,"ximo valor es 7FH",0
er_meme   db "El m",160,"ximo valor es 7FFH",0
er_rev    db "No es posible ejecuci",162,"n reversa",0
123d_     db "Acceso a Ventanas del Depurador.",0
123regcpu db "Editor de Registros de la CPU   ",0
123band   db "     Editor de Banderas       ",0
123regesp db "Editor de Registros Especiales",0
123mem1   db "Editor de Area de Memoria 1   ",0
123mem2   db "Editor de Area de Memoria 2   ",0
123timer  db " Editor de Timers y Sbuf      ",0
123puert  db "     Editor de puertos        ",0
123_cod   db "Corre hasta la instrucci",162,"n apuntada por PC",0
123_cor   db "Programa en ejecuci",162,"n. ESC para abandonar",0
cambio_m  db "Cambio de Direcci",162,"n de Memoria",0
123_ciclo db "Ingrese un valor para setear el numero de ciclos",0
123_ins   db "Ingrese el mnemonico de la instrucci",162,"n que desea insertar",0
123_ens   db "Ingrese el mnemonico de la instrucci",162,"n que desea ejecutar",0
124hex1   db "Ingrese un n",163,"mero de 2 bytes --->",0
```

```
l24hex2   db "Ingrese un n",163,"mero de 1 byte  --->",0
l24bin    db "Ingrese un n",163,"mero binario    --->",0
l24asc    db "Ingrese un caracter ascii    --->",0
l24dec    db "Ingrese un n",163,"mero decimal    --->",0
no_ram    db "Esta ",160,"rea es de ROM",0
lis_rup   db "           Procedimiento listo          ",0
hay_rup   db "       Este break point ya existe       ",0
exc_rup   db "Sobrepaso  el # m",160,"ximo de break points ",0
nohay_rup db "        No hay break points             ",0
nohay_nrup db "    No hay ese n",163,"mero de break point    ",0
tec_sal   db "Presione una tecla para continuar      ",0
l23b_rup  db "Ingrese el n",163,"mero de break point que desea borrar",0
l23a_rup  db "Ingrese el n",163,"mero de break point que desea activar/desact.",0
l23c_rup  db "Ingrese el valor de PC donde va el break point",0
pagina    db "PgDn, PgUp muestra p",160,"ginas de c",162,"digo.",0
er_ens    db "El nmemonico ingresado no es correcto. Presione una tecla para"
          db " continuar",0
pan_pres0 db 0dbh,28 dup(0dfh),0dbh,0c4h,0bfh,0
pan_pres1 db 0dbh,28 dup(" "),0dbh," ",0b3h,0
pan_pres2 db 0dbh,28 dup(" "),0dbh," ",0b3h,0
pan_pres3 db 0dbh,28 dup(0dch),0dbh," ",0b3h,0
pan_pres4 db 0c0h,29 dup(0c4h),0d9h,0
pan_pres5 db "Fabio Gonz",160,"lez - C",130,"sar Moreno",0
pan_pres6 db "All Rights Reserved",0
pan_pres7 db "Quito, ECUADOR",0
pan_pres8 db "EPN - 1991",0
pan_pres9 db "DS-5000T",0
pan_presa db "4.01",0
hola0     db 43 dup (0dbh),0
hola1     db "                                                ",2 dup (0b0h),0
hola2     db "                                                ",2 dup (0b0h),0
hola3     db "                                                ",2 dup (0b0h),0
hola4     db "                                                ",2 dup (0b0h),0
hola5     db "                                                ",2 dup (0b0h),0
hola6     db "                                                ",2 dup (0b0h),0
hola7     db "                                                ",2 dup (0b0h),0
hola8     db 43 dup (0b0h),0
hola9     db 2 dup (0b0h),0
din_est   db "breakpoint Dinámico o Estático",0
er_rx     db "Error en la transmisi",162,"n",0
er_tx_c   db "Error en la transmisi",162,"n de Comando."
          db "  Quiere reintentar (S/N)",0
b_file    db "La transmisi",162,"n del archivo fue correcta",0
reint?    db "Quiere volver a intentar? (S/N)",0
sw0       db "Verifique que el switch de carga este en la posici",162,"n 0",0
sw1       db "Verifique que el switch de carga este en la posici",162,"n 1",0
tx_file   db "< !! Transmitiendo archivo Objeto !! >",0
mcon?     db "Quiere escribir el registro MCON (S/N)",0
ing_mcon  db "Ingrese el valor para el registro MCON con el formato  1F",0
encrk     db "Quiere encriptar el programa (S/N)",0
ing_encr  db "Ingrese palabra de encriptamiento con el formato"
          db "  01 23 45 AB EF",0
b_lock    db "Desea setear el bit security lock (S/N)",0
ram_opc?  db "Seleccione la partici",162,"n de Memoria para la RAM Embebida",0
ram_opc   db "(0)1000H  (1)2000H  (2)3000H  (3)4000H  (4)5000H  "
          db "(5)6000H  (6)7000H  (7)8000H",0
```

```
;Varibles para ingreso de datos

atrprev  db 0
dirprev  dw 0
nombre0  db 40,0
nombre   db 41 dup(0)
binario  db 8 dup ("0")
dhex1    db 4,7 dup(0)
dhex2    db 2,4 dup(0)
dbin     db 1,0,0
dasc     db 1,0,0
ins_ens  db 40,0,41 dup (0)
dencr    db 14,0,15 dup (0)
dasc2    db 2,3 dup(0)

;Registros del uc DS5000T
pc       dw 0
r0       db 0
r1       db 0
r2       db 0
r3       db 0
r4       db 0
r5       db 0
r6       db 0
r7       db 0,78h dup (0)

;SFR's

p0       db 0ffh
regsp    db 7
dpl      db 0
dph      db 0,3 dup (0)
pcon     db 0
tcon     db 0
tmod     db 0
tl0      db 0
tl1      db 0
th0      db 0
th1      db 0,0,0
p1       db 0ffh,7 dup (0)
scon     db 0
sbuf     db 0,6 dup (0)
p2       db 0ffh,7 dup (0)
ie       db 0,7 dup (0)
p3       db 0ffh,7 dup (0)
ip       db 0,8 dup (0)
ek0      db 0
ek1      db 0
ek2      db 0
ek3      db 0
ek4      db 0
mcon     db 0f8h
ta       db 55h,8 dup (0)
psw      db 0,15 dup (0)
acc      db 0,15 dup (0)
b        db 0,15 dup (0)
```

```
p0_pin   db 0ffh
p1_pin   db 0ffh
p2_pin   db 0ffh
p3_pin   db 0ffh
sbuf_in  db 0

;Variables Adicionales

dirm1in    dw 0
dirm2in    dw 20h
dirm1ex    dw 0
dirm2ex    dw 0
m1in_ex    db 0
m2in_ex    db 0
handle     dw 0
cod_error  dw 0
cod_acces  db 0          ;codigo de acceso (R/W,R,W)
num_byte   dw 0
psp        dw ?
cod_pc     dw 0
prox_pc    dw 0
pos_cod    db 0
cod_ens    db 3 dup (0)
tabla_brk  db 5*8 dup (0)        ;tabla de break points
rup_act    db 0                  ;# de rupturas
n_ciclos   dw 0
map_mem    dw 0,800h,1000h,1800h,2000h,2800h,3000h,3800h,4000h  ;direcciones
           dw 4800h,5000h,5800h,6000h,6800h,7000h,8000h       ;particion
rom_ram_e  dw 800h
ram_emb_m  dw 7fffh
ram?       db 0
buf_cod    dw 8 dup(0)   ;buffer para guardar inicio de ventana de codigo
cksum      db 0
ramtab     db "2468ACEF"
ramopc     db 0
inst_pc    db 1,2,10h,11h,12h,20h,21h,22h,30h,31h,32h,40h,41h,50h,51h,60h
           db 61h,70h,71h,73h,81h,91h,0a1h,0b1h,0b4h,0b5h,0b6h,0b7h,0b8h
           db 0b9h,0bah,0bbh,0bch,0bdh,0beh,0bfh,0c1h,0d1h,0d5h,0e1h,0f1h
inst_s     db 3 dup(0)

;Comandos para transmision

com_U      db "U",0dh,0                ;Reset del bit de seguridad
com_L      db "L",0dh,0                ;Carga del programa en el uc
com_F      db "F 0 0 8000h",0dh,0      ;Llena de 0 la memoria
com_C      db 2eh,03h,0                ;Control C
com_R      db 0dh,0                   ;Carriage return
com_Z      db "Z",0dh,0                ;Setea bit security lock
com_W      db "W"," ",0,0,0dh,0        ;Escribe registro MCON
com_K      db "K",15 dup(" "),0dh,0    ;Carga bits de encriptamiento

com_S      db "S",0,0,0,0,0            ;Ejecucion fuera del programa
com_P      db "P",0                    ;Ejecucion paso a paso
com_Q      db "Q",0                   ;descarga
com_E      db "E",0,0,0                ;editor
com_I      db "I",0,0,0,0             ;Inserta instrucciones
```

```
com_M       db "M",0,0                      ;descarga memoria interna
com_N       db "N",0,0                      ;descarga memoria externa

;Banderas

b_editor    db 0                           ;bandera del editor
b_ciclo     db 0                            ;bandera de act/des de ciclos
b_dis       db 0                            ;bandera de act/des de display
b_cor_dis   db 0                            ;bandera de act/des de ejecucion
b_mem       dw 0ffh
b_cod_rup   db 0                            ;bandera para mostrar codigo de ruptura


DSEG ENDS


DSEG1 SEGMENT PUBLIC

area_lect   db 0fffffh dup (0)               ;buffer para el archivo obj.

DSEG1 ENDS

DSEG2 SEGMENT PUBLIC

ram_emb     db 8000h dup (0)                ;RAM embebed
ram_embf    db 3 dup(0)

DSEG2 ENDS

        EXTRN   DESENSDET:FAR,INTEL:FAR
        EXTRN   EJECUTOR:FAR,REVERSA:FAR
        EXTRN   COMSER:FAR,COMUNICA:FAR,COM_FILE:FAR,TX_COMAND:FAR
        EXTRN   VEL4800:FAR,VEL9600:FAR
        EXTRN   TX_LOAD:FAR
        EXTRN   ASSEM:FAR
        EXTRN   LONGITUD:FAR, CICLOS:FAR

CSEG SEGMENT PUBLIC

ASSUME CS:CSEG,DS:DSEG,ES:NOTHING

        SBRVNT
        SBRPRO

        PUBLIC  RINIC, EDITOR, CAMM1IN,CAMM1EX,CAMM2IN,CAMM2EX
        PUBLIC  LEE_ARC,NOM_ARC
        PUBLIC  DEF_PASO,DEF_CORRE
        PUBLIC  CREAR_RUP,ACT_RUP,BORRAR_RUP,MOST_RUP
        PUBLIC  HEXABIN,BINAHEX
        PUBLIC  SET_CIC,ACT_CIC,ACT_DIS
        PUBLIC  INSERTA,ENSAMBLA
        PUBLIC  RESET,HELP,ATRAS

RINIC PROC NEAR
;-------------------------------------------------------------------------
;Inicializa los parametros, abre archivo de ventanas , lee archivo de entrada
```

```
;muestra pantalla inicial
;-------------------------------------------------------------------------------
        mov     ax,dseg                 ;ds apunta a segmento de datos
        mov     ds,ax
        mov     segdat,ax               ;segdat = segmento principal
        call    segment1
        mov     segdat1,ax              ;segdat1= segmento de archivo
        call    segment2
        mov     segdat2,ax              ;segdat2= segmento de RAM emb.
        call    present
        idvnt   vntmono,varea,50        ;carga archivo de ventanas
        mov     aux,ax
        cmp     ax,0
        jne     rini1                   ;error con archivo de ventanas
        jmp     rini5
rini1:  lim23
        lim24
        cmp     aux,1                   ;hay archivo de ventanas?
        jne     rini2                   ;si
        print   1,23,e_arc_ven,revers   ;no, muestra mensaje de error
        jmp     rini4
rini2:  cmp     aux,2                   ;es archivo de ventanas?
        jne     rini3                   ;si
        print   1,23,n_arc_ven,revers   ;no, muestra mensaje de error
        jmp     rini4
rini3:  print   1,23,f_arc_ven,revers   ;no hay espacio para el archivo
rini4:  print   1,24,tec_sal,normal
        getkey
        mov     ax,4c00h                ;abandona el programa
        int     21h
rini5:  call    comser                  ;setea puerto serial
        call    lee_param               ;lee parametros ingresados
        mov     bx,offset vlin0 ;presenta linea 0
        mov     ax,ds
        vnt     "a"
        print   0,2,codigo,revers
        print   0,3,codig_m,0ah
        mov     ax,n_ciclos             ;prepara y muestra ventana
        mov     cx,5                    ;de ciclos
        mov     si,offset lciclo
rini6:  mov     byte ptr ds:[si]," "
        inc     si
        loop    rini6
        dec     si
        call    hexadec
        mov     bx,offset vciclo        ;presenta ventana de ciclos
        mov     si,offset lciclo
        mov     ax,ds
        vnt     "a"
        mov     bx,offset vdisplay      ;presenta ventana de display
        mov     si,offset ldisplay
        mov     ax,ds
        vnt     "a"
        ret
RINIC ENDP
```

```
LEE_PARAM PROC NEAR
;-------------------------------------------------------------------------
;Lee y carga los parametros ingresados por el usuario, si el usuario no
;ingresa parametros, desde aqui se indagagan.
;Entrada:       <Nombre del archivo Objeto>
;               <Modo de Operacion>
;Salida:        Modo:   0 Simulacion
;                       1 Ejecucion controlada
;                       2 Carga serial
;-------------------------------------------------------------------------
        mov     ah,62h
        int     21h
        mov     psp,bx                  ;carga valor de PSP
        mov     es,bx                   ;ES apunta al PSP
        mov     di,81h                  ;va a buscar parametros
        mov     cx,7fh
        cmp     byte ptr es:[80h],0      ;hay parametros ingresados?
        jne     leep2                   ;si
leep1:  jmp     leep6                   ;no
leep2:  mov     al," "
        cld
        repe    scasb                   ;encuentra el primer elemento
        cmp     cx,0                    ;diferente de " "
        je      leep1
        dec     di
        inc     cx
        mov     si,di                   ;SI -> origen de parametros
        push    es
        pop     ds                      ;DS apunta al PSP
        mov     ax,dseg
        mov     es,ax                    ;ES segmento de datos
        mov     di,offset nombre         ;DI -> nombre del archivo
leep3:  lodsb                           ;lee caracter del PSP
        cmp     al," "                  ;fin del nombre?
        ja      leep31                  ;no
        jmp     leep4                   ;si
leep31: stosb                           ;guarda caracter en nombre
        dec     cx                       ;parametro correcto ?
        jz      leep1                   ;no
        jmp     leep3                    ;si
leep4:  dec     cx
        cmp     cx,0                    ;ingreso modo ?
        ja      leep42                  ;si
leep41: jmp     leep7                   ;no
leep42: lodsb
        dec     cx                      ;parametro correcto ?
        jz      leep7                    ;no
        mov     bl,ds:[si]               ;si
        cmp     bl," "                  ;parametro correcto ?
        ja      leep41                  ;no
        and     al,0dfh                 ;si
        cmp     al,"S"                  ;simulacion?
        jne     leep43                  ;no
        mov     bl,0                     ;si
        jmp     leep5
leep43: cmp     al,"E"                  ;ejecucion controlada?
```

```
        jne     leep44                  ;no
        mov     bl,1                    ;si
        jmp     leep5
leep44: cmp     al,"C"                  ;carga serial?
        je      leep45                  ;si
        jmp     leep7                    ;no
leep45: mov     bl,2
leep5:  mov     ax,dseg
        mov     ds,ax                   ;DS = segmento de datos
        mov     es,videoseg              ;ES = segmento de video
        mov     modo,bl                 ;carga el modo de programa
        mov     aux,0
        call    lee_arc                 ;lee archivo objeto
        jmp     leep9
leep6:  mov     ax,dseg
        mov     ds,ax                   ;DS = segmento de datos
        mov     es,videoseg              ;ES = segmento de video
        lim23
        print   1,23,nec_arc,revers
        call    nom_arc                 ;nombre del archivo ?
leep7:  mov     ax,dseg
        mov     ds,ax                   ;DS = segmento de datos
        mov     es,videoseg              ;ES = segmento de video
        lim23
        lim24
        print   20,23,nec_mod,revers
        print   10,24,modo?,normal
leep71: getkey                          ;modo de operacion ?
        mov     ax,getword
        and     al,0dfh
        cmp     al,"S"                  ;simulacion?
        jne     leep72                  ;no
        mov     modo,0                  ;si, setea modo en 0
        jmp     leep8
leep72: cmp     al,"E"                   ;ejecucion controlada?
        jne     leep73                  ;no
        mov     modo,1                  ;si, setea modo en 1
        jmp     leep8
leep73: cmp     al,"C"                  ;carga serial?
        jne     leep74                  ;no
        mov     modo,2                  ;si, setea modo en 2
        jmp     leep8
leep74: mov     ah,2                    ;beep
        mov     dl,7
        int     21h
        jmp     leep71
leep8:  mov     aux,0
        call    lee_arc                 ;lee archivo
leep9:  jnc     leepf                   ;error ?
        mov     ax,4c00h                 ;si, sale del programa
        int     21h
leepf:  ret
LEE_PARAM ENDP

LEE_ARC PROC NEAR
;-------------------------------------------------------------------------
```

```
;Lee y carga archivos de depuracion:
;Entrada:       Nombre = Nombre del archivo Objeto
;               Modo = Modo de operacion
;Salida:        Modo 0:  Lee archivo objeto y lo carga en el PC
;               Modo 1:  Lee archivo objeto, lo carga y transmite al uC
;                        Lee archivo monitor, lo carga y transmite al uC
;               Modo 2: Lee archivo objeto y lo descarga hacia el uC
;-------------------------------------------------------------------------
lee1:   guarpan
        abrir   nombre,cod_acces,handle,cod_error
        traepan
        lim24
        cmp     cod_error,0             ;error en apertura de arc. objeto?
        je      lee11                   ;no
        jmp     lee31                   ;si
lee11:  leer    handle,0ffffh,area_lect,num_byte,cod_error
        cmp     cod_error,0             ;error en lectura de arc. objeto?
        je      lee12                   ;no
        jmp     lee32                   ;si
lee12:  mov     bx,handle               ;cierra archivo objeto
        mov     ah,3eh
        int     21h
        push    ds
        mov     ds,segdat1
        mov     si,offset area_lect     ;origen del archivo objeto
        pop     ds
        push    es
        push    ds
        mov     es,segdat2
        mov     ds,segdat2
        mov     di,offset ram_emb       ;limpia ram embebida
        mov     al,0
        mov     cx,07fffh
        rep     stosb
        mov     di,offset ram_emb       ;destino del archivo objeto
        pop     ds
        pop     es
        call    intel                   ;conversion de INTEL a hex
        jnc     lee13                   ;error ?
        jmp     lee33                   ;si
lee13:  cmp     modo,0                  ;Modo de Simulacion ?
        jne     lee131                  ;no
        jmp     lee2                    ;si
lee131: cmp     modo,1                  ;Modo de Ejecucion controlada?
        je      lee14                   ;si
        call    carga                   ;no, descarga archivo objeto al uC
        mov     ax,4c00h                ;sale del programa
        int     21h
;-------------------------------------------------------------------------
;Transmision del archivo objeto
;-------------------------------------------------------------------------
lee14:  lim23
        lim24
        print   1,23,ram_opc?,revers    ;Seleccion de particion
        print   1,24,ram_opc,normal
lee140: getkey                          ;ingresa opcion
```

```
        mov     ax,getword
        sub     al,"0"
        cmp     al,0                    ;opcion < 0 ?
        jb      lee141                  ;si, error
        cmp     al,7                    ;opcion > 7 ?
        ja      lee141                  ;si, error
        mov     si,offset ramtab
        xor     ah,ah
        add     si,ax
        mov     al,ds:[si]
        mov     ramopc,al               ;carga opcion
        jmp     lee142
lee141: mov     ah,2                    ;beep
        mov     dl,7
        int     21h
        jmp     lee140                  ;ingresa nuevamente
lee142: lim23
        lim24
        print   1,23,sw0,revers         ;switch en posicion cero
        print   1,24,tec_sal,normal
        getkey
        lim23
        lim24
        call    vel4800                 ;transmision a 4800 bits/seg
        print   1,23,tx_file,revers
        mov     si,offset com_R         ;envia un CR
        call    tx_comand
        mov     si,offset com_C         ;envia un Control C
        call    tx_comand
        jnc     lee143                  ;error ?
        jmp     lee34                   ;si
lee143: mov     si,offset com_U         ;envia un Reset al bit
        call    tx_comand               ;de seguridad
        jnc     lee144                  ;error ?
        jmp     lee34                   ;si
lee144: mov     si,offset com_F         ;llena de ceros la memoria embebida
        call    tx_comand
        jnc     lee145                  ;error ?
        jmp     lee34                   ;si
lee145: mov     si,offset com_L         ;envia comando de carga
        call    tx_comand
        mov     cx,num_byte
        push    ds
        mov     ds,segdat1
        mov     si,offset area_lect
        pop     ds
        call    com_file                ;transmite archivo objeto al uC
        jnc     lee146                  ;error ?
        jmp     lee34                   ;si
lee146: mov     si,offset com_C         ;envia un Control C
        call    tx_comand
        jnc     lee147                  ;error ?
        jmp     lee34                   ;si
lee147: mov     si,offset com_W         ;escribe MCON
        mov     al,ramopc
        mov     ds:[si+2],al
```

```
        mov     byte ptr ds:[si+3],"8"
        mov     byte ptr ds:[si+4],0dh
        mov     byte ptr ds:[si+5],0
        call    tx_comand
        jnc     lee15                   ;error en transmision del arc. objeto?
        jmp     lee34                   ;si
lee15:  abrir   dallas,cod_acces,handle,cod_error
        lim24
        cmp     cod_error,0             ;error en apertura de archivo monitor?
        je      lee151                  ;no
        jmp     lee35                   ;si
lee151: leer    handle,0ffffh,area_lect,num_byte,cod_error
        cmp     cod_error,0             ;error en lectura del archivo monitor?
        je      lee152                  ;no
        jmp     lee36                   ;si
lee152: mov     bx,handle               ;cierra archivo monitor
        mov     ah,3eh
        int     21h
        push    ds
        mov     ds,segdat1
        mov     si,offset area_lect     ;origen archivo monitor
        pop     ds
        push    ds
        mov     ds,segdat2
        mov     di,offset ram_emb       ;destino archivo monitor
        pop     ds
        call    intel                   ;conversion INTEL a hex
        jnc     lee16                   ;error ?
        jmp     lee37                   ;si
;-------------------------------------------------------------------------------
;Transmision del archivo monitor
;-------------------------------------------------------------------------------
lee16:  lim23
        lim24
        print   1,23,tx_f_d,revers
        mov     si,offset com_L         ;envia comando de carga
        call    tx_comand
        mov     cx,num_byte
        push    ds
        mov     ds,segdat1
        mov     si,offset area_lect
        pop     ds
        call    com_file                ;transmite el archivo monitor
        jnc     lee17                   ;error ?
        jmp     lee38                   ;si
lee17:  lim23
        lim24
        print   1,23,sw1,revers         ;switch posicion uno
        print   1,24,tec_sal,normal
        getkey
        call    vel9600                 ;transmision 9600 bits/seg
lee2:   cmp     aux,0                   ;limpiar pantalla ?
        jne     lee21                   ;no
        mov     ax,0720h                ;si
        xor     di,di
        mov     cx,80*23
```

```
        rep     stosw
lee21:  call    reset
        jnc     lee22                   ;error en transmision de arc. monitor?
        stc                             ;si
        jmp     lee_f
lee22:  clc
        jmp     lee_f
lee31:  lim23                           ;error en apertura
        print   1,23,arc_e_abr,revers    ;del archivo objeto
        jmp     leef
lee32:  lim23                           ;error en lectura
        print   1,23,arc_e_lee,revers    ;del archivo objeto
        jmp     leef
lee33:  lim23                           ;no es formato INTEL
        print   1,23,arc_e_int,revers    ;el archivo objeto
        jmp     leef
lee34:  lim23
        print   1,23,dal_o,revers        ;error en la transmision
        jmp     leef                    ;del archivo objeto
lee35:  lim23
        print   1,23,dal_a,revers        ;error en apertura
        jmp     leef                    ;del archivo monitor
lee36:  lim23
        print   1,23,dal_l,revers        ;error en lectura
        jmp     leef                    ;del archivo monitor
lee37:  lim23
        print   1,23,dal_i,revers        ;no es de formato intel
        jmp     leef                    ;el archivo monitor
lee38:  lim23
        print   1,23,dal_t,revers        ;error en la transmision
leef:   lim24
        print   1,24,tec_sal,normal      ;tecla para salir
        getkey
        stc
lee_f:  ret
LEE_ARC ENDP

NOM_ARC PROC NEAR
;-------------------------------------------------------------------------------
;Ingreso del nombre del archivo a depurarse a traves de teclado
;Salida:        Nombre: Nombre del archivo
;-------------------------------------------------------------------------------
        lim24
        print   1,24,nombarc,normal
        mov     gdreset,1
nom_arc0:       getline 40,24,nombre0   ;ingreso de un string
        mov     gdreset,0
        cmp     gccod,4                 ;ESC ?
        jne     nom_arc1                ;no
        jmp     nom_arcf                ;si, sale
nom_arc1:       cmp     gccod,1         ;string valido ?
        jne     nom_arc0                ;no
nom_arcf:       ret
NOM_ARC ENDP

INICIAL PROC NEAR
```

```
;-------------------------------------------------------------------------
;Actualiza pantalla, presenta ventanas de registros CPU, registros especiales
;timers, banderas, area de memoria 1, area de memoria 2 y puertos
;-------------------------------------------------------------------------
        cmp     modo,1                  ;modo de Ejecucion controlada ?
        jne     inic0                   ;no
        call    descar                   ;descarga estatus del micro
        jnc     inic0
        jmp     inic_e
inic0:  call    mcod                    ;muestra codigo desensamblado
        call    regist                  ;muestra ventana de reg. cpu
        call    flags                   ;muestra ventana de banderas
        call    reg_espc                 ;muestra ventana de reg. esp.
        call    mos_mem1                 ;muestra ventana de mem1
        call    mos_mem2                 ;muestra ventana de mem2
        call    reg_pcon                 ;muestra ventana de reg. PCON
        call    puertos                 ;muestra ventana de puertos
        call    par_memo                 ;determina particion de mem.
        clc
        jmp     inic_f
inic_e: stc
inic_f: ret
INICIAL ENDP


MCOD PROC NEAR
;-------------------------------------------------------------------------
;Prepara y muestra ventana de codigo desensamblado, resalta la instruccion
;apuntada por PC.
;Entrada:       PC
;               COD_PC = direccion de la primera inst. de la ventana
;               PROX_PC = direccion de la ultima inst. de la ventana
;Salida:        Ventana de codigo desensamblado
;-------------------------------------------------------------------------
        push    dx
        push    cx
        push    bx
        mov     ax,pc                   ;verifica si
        cmp     ax,cod_pc                ;pc se encuentra
        jb      mcod1                   ;entre cod_pc
        cmp     ax,prox_pc               ;y prox_pc
        jb      mcod2
mcod1:  mov     cod_pc,ax                ;si no actualiza cod_pc
mcod2:  mov     dx,cod_pc
        mov     bl,19                    ;# lineas a desensamblarse
        mov     bh,4                    ;# de linea inicial de la
        mov     cx,0                     ;ventana de codigo
        mov     di,offset lcod
        push    ds
        mov     ds,segdat2
        mov     si,offset ram_emb        ;origen del codigo
        pop     ds
        add     si,cod_pc                ;SI apunta a direccion
mcod3:  add     si,cx                   ;de inst. a desensamblarse
        add     dx,cx
        cmp     dx,pc                   ;ubica posicion de linea
```

```
        jne     mcod4                   ;apuntada por PC
        mov     pos_cod,bh              ;guarda posicion
mcod4:  inc     bh
        call    desensdet               ;desensambla una instruccion
        add     di,23
        cmp     bl,2                    ;penultima linea ?
        jne     mcod41                  ;no
        push    si
        add     si,cx
        push    ds                      ;si
        mov     ds,segdat2
        sub     si,offset ram_emb
        pop     ds
        mov     prox_pc,si              ;actualiza proxima instruccion
        pop     si
mcod41: dec     bl
        jnz     mcod3                   ;lazo para 19 lineas
        mov     bx,offset vcod          ;muestra ventana
        mov     si,offset lcod          ;de codigo
        mov     ax,ds
        vnt     "a"
        posdir  0,pos_cod               ;resalta linea que
        mov     di,pddir                ;corresponde a la direccion
        add     di,13                   ;apuntada por PC
        mov     al,70h
        mov     cx,20
mcod5:  stosb
        inc     di
        loop    mcod5
        call    codig_rup               ;verifica puntos de ruptura
        pop     bx
        pop     cx
        pop     dx
        ret
MCOD ENDP

CODIG_RUP PROC NEAR
;-------------------------------------------------------------------------
;Verifica si en la ventana de codigo desensamblada hay puntos de ruptura,
;si los hay muestra estos puntos resaltando la direccion correspondiente
;Entrada:       RUP_ACT = numero de puntos de ruptura
;               TABLA_BRK = tabla de puntos de ruptura
;-------------------------------------------------------------------------
        cmp     rup_act,0               ;hay break points
        jne     crup1                   ;si
        jmp     crupf                   ;no, salga
crup1:  mov     bh,4
        mov     dx,cod_pc               ;direccion inicial de codigo
crup2:  push    bx
        mov     bl,rup_act              ;# de break points
        mov     si,offset tabla_brk
        add     si,3
crup3:  mov     ax,ds:[si]              ;lee PC de break point
        cmp     ax,dx                   ;break point = dir. codigo?
        jne     crup4                   ;no, siga buscando
        jmp     crup5                   ;si, encontro break point
```

```
crup4:  add     si,5
        dec     bl
        cmp     bl,0                    ;ultimo break point?
        jne     crup3                   ;no
        pop     bx
        jmp     crup9
crup5:  pop     bx
        cmp     byte ptr ds:[si-2],0     ;break point activo?
        jne     crup6                   ;si
        mov     al,0ah                  ;no, fija atributo intenso
        jmp     crup7
crup6:  mov     al,70h                  ;fija atributo reverso
crup7:  posdir  0,bh
        mov     di,pddir                 ;ubica direccion en pantalla
        inc     di
        mov     cx,5
crup8:  stosb                           ;cambia de atributo a
        inc     di                      ;a direccion con break point
        loop    crup8
crup9:  push    ds
        mov     ds,segdat2
        mov     si,offset ram_emb
        pop     ds
        add     si,dx
        mov     ax,segdat2
        call    longitud                 ;calcula proxima direccion
        add     dx,cx
        inc     bh
        cmp     bh,19                    ;ultima direccion de codigo
        jne     crup2                    ;no
crupf:  ret
CODIG_RUP ENDP


REGIST PROC NEAR
;-------------------------------------------------------------------------
;Prepara y muestra ventana de registros de la cpu
;-------------------------------------------------------------------------
        push    dx
        push    es
        push    ds                      ;ES = segmento de datos
        pop     es
        mov     di,offset lregcpu        ;buffer de ventana de regcpu
        mov     al,acc
        guardreg                         ;carga ACC en hexadecimal
        mov     bl,acc
        call    hexabin                 ;carga ACC en binario
        mov     al,acc
        stosb                           ;carga ACC en ascii
        mov     ax,pc
        mov     cl,al
        mov     al,ah
        guardreg                         ;carga MSByte de PC
        mov     al,cl
        guardreg                         ;carga LSByte de PC
        mov     al,dph
```

```
        guardreg                        ;carga DPH
        mov     al,dpl
        guardreg                        ;carga DPL
        mov     al,regsp
        guardreg                        ;carga registro SP
        mov     al,b
        guardreg                        ;carga registro B
        xor     ah,ah
        mov     al,psw                  ;determina el banco
        and     al,i8h                   ;el banco de registro
        cmp     al,0                    ;banco 1?
        jne     reg1                    ;no
        mov     dl," "                  ;si, carga ( )
        jmp     reg4
reg1:   cmp     al,8                    ;banco 2?
        jne     reg2                     ;no
        mov     dl,"'"                  ;si, carga (')
        jmp     reg4
reg2:   cmp     al,i6                   ;banco 3?
        jne     reg3                    ;no
        mov     dl,22h                  ;si, carga (")
        jmp     reg4
reg3:   mov     dl,"\"                  ;carga (\)
reg4:   mov     si,offset r0
        add     si,ax                   ;direccion del banco
        mov     cx,8                    ;8 bytes del banco de registros
reg5:   mov     al,dl
        stosb                           ;carga indicacion de banco
        lodsb
        guardreg                         ;carga byte del banco de registros
        loop    reg5
        pop     es
        pop     dx
        mov     bx,offset vregcpu        ;muestra ventana de regcpu
        mov     ax,ds
        mov     si,offset lregcpu
        vnt     "a"
        ret
REGIST ENDP

HEXABIN PROC NEAR
;-------------------------------------------------------------------------
;Convierte un byte hexadecimal a un string (8 bytes) correspondiente a su
;valor en binario
;Entrada:       BL = byte hexadecimal
;Salida:        DS:DI = string correspondiente a su valor en binario
;-------------------------------------------------------------------------
        push    cx
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        mov     cx,8
        cld
bini:   mov     ax,"0"
        rcl     bl,1                    ;CY <- bit
        jnc     bin2
```

```
        mov     ax,"1"
bin2:   stosb                           ;guarda bit
        loop    bin1                    ;lazo para 8 bits
        pop     es
        pop     cx
        ret
HEXABIN ENDP

BINAHEX PROC NEAR
;-------------------------------------------------------------------------
;El string de 8 bytes almacenado en binario, es convertido a un byte
;hexadecimal
;Entrada:       DS:SI = String en binario
;Salida:        BL = byte hexadecimal
;-------------------------------------------------------------------------
        push    si
        push    cx
        mov     si,offset binario       ;si apunta inicio de string
        mov     cx,8
        mov     bl,0                    ;inicializacion de bl
hex1:   clc                             ;CY <- 0
        lodsb                           ;al = byte del string
        cmp     al,"0"
        je      hex2
        stc                             ;CY <- 1
hex2:   rcl     bl,1                    ;guarda carry en bl
        loop    hex1                    ;8 bits
        pop     cx
        pop     si
        ret
BINAHEX ENDP

FLAGS PROC NEAR
;-------------------------------------------------------------------------
;Prepara y muestra ventana de banderas
;-------------------------------------------------------------------------
        push    es
        push    ds                      ;ES = segmento de datos
        pop     es
        mov     bl,psw
        mov     di,offset binario
        call    hexabin                 ;binario <- psw
        mov     al,acc
        mov     ah,"0"
        mov     cx,8
        clc
flag1:  rcl     al,1                    ;encuentra paridad del acc
        jnc     flag3
        cmp     ah,"0"
        jne     flag2
        mov     ah,"1"
        jmp     flag3
flag2:  mov     ah,"0"
flag3:  loop    flag1                   ;lazo para 8 bits
        mov     si,offset binario
        mov     ds:[si+7],ah            ;guarda bit de paridad
```

```
        call    binahex
        mov     psw,bl
        mov     si,offset binario       ;carga en lband los datos
        mov     di,offset lband         ;de las banderas a mostrarse
        movsb                           ;c
        movsb                           ;cy
        movsb                           ;f0
        inc     si
        inc     si
        movsb                           ;ov
        inc     si
        movsb                           ;p
        pop     es
        mov     bx,offset vband         ;muestra ventana de banderas
        mov     ax,ds
        mov     si,offset lband
        vnt     "a"
        ret
FLAGS ENDP

REG_ESPC PROC NEAR
;-------------------------------------------------------------------------
;Prepara y muestra ventana de registros especiales y sbuf de entrada y salida
;-------------------------------------------------------------------------

        mov     di,offset lregesp
        mov     bl,mcon                 ;carga MCON en binario
        call    hexabin
        mov     bl,tmod                 ;carga TMOD en binario
        call    hexabin
        mov     bl,ie                   ;carga IE en binario
        call    hexabin
        mov     byte ptr ds:[di-7]," "   ;IE.6 no es de control
        mov     byte ptr ds:[di-6]," "   ;IE.5 no es de control
        mov     bl,tcon                 ;carga TCON en binario
        call    hexabin
        mov     bl,ip                   ;carga IP en binario
        call    hexabin
        mov     byte ptr ds:[di-8],"x"   ;IP.7 no puede ser leido
        mov     byte ptr ds:[di-7]," "   ;IP.6 no es de control
        mov     byte ptr ds:[di-6]," "   ;IP.5 no es de control
        mov     bl,scon                 ;carga SCON en binario
        call    hexabin
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        mov     al,th0                  ;carga TH0
        guardreg
        mov     al,tl0                  ;carga TL0
        guardreg
        mov     al,sbuf_in               ;carga SBUF IN
        guardreg
        mov     al,th1                  ;carga TH1
        guardreg
        mov     al,tl1                  ;carga TL1
        guardreg
```

```
        mov     al,sbuf                 ;carga SBUF OUT
        guardreg
        pop     es
        mov     bx,offset vregesp       ;presenta venta de
        mov     ax,ds                   ;registros especiales
        mov     si,offset lregesp       ;y buffer de tx y rx
        vnt     "a"
        ret
REG_ESPC ENDP

PAR_MEMO PROC NEAR
;-------------------------------------------------------------------------
;Determina la particion de memoria embebida
;Entrda:        MCON = bit de particion
;Salida:        ROM_RAM_E = limite entre ROM y RAM
;-------------------------------------------------------------------------
        push    ax
        push    bx
        push    cx
        push    si
        mov     al,mcon                 ;lee mcon
        test    al,2                    ;mcon.1 es 1?
        je      par_1                    ;no
        mov     rom_ram_e,800h           ;si, particion en 800h
        jmp     par_2
par_1:  mov     cl,4
        shr     al,cl                   ;al = bits de direccion
        and     al,0fh
        mov     bl,2
        mul     bl
        mov     bl,al
        xor     bh,bh                   ;bx = direccion
        mov     si,offset map_mem
        mov     ax,ds:[si+bx]
        mov     rom_ram_e,ax            ;encuentra particion
par_2:; mov     al,mcon
        test    al,8                     ;mcon.3 es 1?
        je      par_3                    ;no
        mov     ram_emb_m,1fffh         ;si, ram embebida maxima = 1fffh
        jmp     par_4
par_3:  mov     ram_emb_m,7ffffh         ;ram embebida maxima = 7ffffh
par_4:  pop     si
        pop     cx
        pop     bx
        pop     ax
        ret
PAR_MEMO ENDP

MOS_MEM1 PROC NEAR
;-------------------------------------------------------------------------
;Prepara y muestra area de memoria 1
;Entrada:       M1IN_EX = 0 memoria interna
;                         1 memoria embebida
;               DIRM1IN = direccion inicial para memoria interna
;               DIRM1EX = direccion inicial para memoria embebida
;-------------------------------------------------------------------------
```

```
        print   27,13,area1,0ah
        mov     di,offset lmem1         ;buffer de ventana de memoria 1
        cmp     m1in_ex,0               ;memoria interna?
        je      mos1                    ;si
        jmp     mos5                    ;no
mos1:   print   35,13,inter,0ah         ;mensaje de interna
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        mov     bx,dirm1in              ;direccion inicial de mem. 1
        mov     si,offset r0
        mov     cx,4                    ;4 filas
mos2:   and     bx,7fh                  ;correccion de dir. inicial
        mov     al,bh
        guardreg                        ;carga direccion inicial alta
        mov     al,bl
        guardreg                        ;carga direccion inicial baja
        push    cx
        push    bx
        mov     cx,8                    ;8 bytes de datos
mos3:   mov     al,[si+bx]
        guardreg                        ;carga byte en hexadecimal
        inc     bx                      ;siguiente direccion
        and     bx,7fh                  ;correccion de direccion
        loop    mos3                    ;lazo para 8 datos
        pop     bx
        mov     cx,8                    ;8 bytes de datos
mos4:   mov     al,[si+bx]
        stosb                           ;carga byte en ascii
        inc     bx                      ;siguiente direccion
        and     bx,7fh                  ;correccion de direccion
        loop    mos4                    ;lazo para 8 datos
        pop     cx
        loop    mos2                    ;lazo para 4 filas
        jmp     mosf
mos5:   print   35,13,exter,0ah         ;mensaje de externa
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        mov     bx,dirm1ex              ;direccion inicial de mem. 1
        push    ds
        mov     ds,segdat2              ;DS = ram embebida
        mov     si,offset ram_emb
        pop     ds
        mov     cx,4                    ;4 filas
mos6:   and     bx,7fffh                ;correccion de dir. inicial
        mov     al,bh
        guardreg                        ;carga direccion  alta
        mov     al,bl
        guardreg                        ;carga direccion baja
        push    cx
        push    bx
        mov     cx,8                    ;8 datos
mos7:   push    ds
        mov     ds,segdat2              ;DS = ram embebida
        mov     al,ds:[si+bx]
```

```
        pop     ds
        guardreg                        ;carga byte en hexadecimal
        inc     bx                      ;siguiente direccion
        and     bx,7fffh                ;correccion de direccion
        loop    mos7                    ;lazo para 8 datos
        pop     bx
        mov     cx,8
mos8:   push    ds
        mov     ds,segdat2              ;DS = ram embebida
        mov     al,ds:[si+bx]
        pop     ds
        stosb                           ;carga byte en ascii
        inc     bx                      ;siguiente direccion
        and     bx,7fffh                ;correccion de direccion
        loop    mos8
        pop     cx
        loop    mos6                    ;lazo para 4 filas
mosf:   pop     es
        mov     bx,offset vmem1         ;presenta ventana de area
        mov     ax,ds                   ;de memoria 1
        mov     si,offset lmem1
        vnt     "a"
        ret
MOS_MEM1 ENDP

MOS_MEM2 PROC NEAR
;-------------------------------------------------------------------------
;Prepara y muestra area de memoria 2
;Entrada:       M2IN_EX = 0 memoria interna
;                         1 memoria embebida
;              DIRM2IN = direccion inicial para memoria interna
;              DIRM2EX = direccion inicial para memoria embebida
;-------------------------------------------------------------------------
        print   27,18,area2,0ah
        mov     di,offset lmem2         ;buffer de ventana de mem. 2
        cmp     m2in_ex,0               ;memoria interna?
        je      mos_1                   ;si
        jmp     mos_5                   ;no
mos_1:  print   35,18,inter,0ah         ;mensaje de interna
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        mov     bx,dirm2in              ;direccion inicial de mem. 2
        mov     si,offset r0
        mov     cx,4                    ;4 filas
mos_2:  and     bx,7fh                  ;correccion de dir. inicial
        mov     al,bh
        guardreg                        ;carga direccion inicial alta
        mov     al,bl
        guardreg                        ;carga direccion inicial baja
        push    cx
        push    bx
        mov     cx,8                    ;8 bytes de datos
mos_3:  mov     al,[si+bx]
        guardreg                        ;carga byte en hexadecimal
        inc     bx                      ;siguiente direccion
```

```
        and     bx,7fh                  ;correccion de direccion
        loop    mos_3                   ;lazo para 8 datos
        pop     bx
        mov     cx,8                    ;8 bytes de datos
mos_4:  mov     al,[si+bx]
        stosb                           ;carga byte en ascii
        inc     bx                      ;siguiente direccion
        and     bx,7fh                  ;correccion de direccion
        loop    mos_4                   ;lazo para 8 datos
        pop     cx
        loop    mos_2                   ;lazo para 4 filas
        jmp     mos_f
mos_5:  print   35,18,exter,0ah         ;mensaje de externa
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        mov     bx,dirm2ex              ;direccion inicial de mem. 2
        push    ds
        mov     ds,segdat2              ;DS = ram embebida
        mov     si,offset ram_emb
        pop     ds
        mov     cx,4                    ;4 filas
mos_6:  and     bx,7fffh                ;correccion de dir. inicial
        mov     al,bh
        guardreg                        ;carga direccion  alta
        mov     al,bl
        guardreg                        ;carga direccion baja
        push    cx
        push    bx
        mov     cx,8                    ;8 datos
mos_7:  push    ds
        mov     ds,segdat2              ;DS = ram embebida
        mov     al,ds:[si+bx]
        pop     ds
        guardreg                        ;carga byte en hexadecimal
        inc     bx                      ;siguiente direccion
        and     bx,7fffh                ;correccion de direccion
        loop    mos_7                   ;lazo para 8 datos
        pop     bx
        mov     cx,8
mos_8:  push    ds
        mov     ds,segdat2              ;ds = ram embebida
        mov     al,ds:[si+bx]
        pop     ds
        stosb                           ;carga byte en ascii
        inc     bx                      ;siguiente direccion
        and     bx,7fffh                ;correccion de direccion
        loop    mos_8
        pop     cx
        loop    mos_6                   ;lazo para 4 filas
mos_f:  pop     es
        mov     bx,offset vmem2         ;presenta ventana de area
        mov     ax,ds                   ;de memoria 2
        mov     si,offset lmem2
        vnt     "a"
        ret
```

```
MOS_MEM2 ENDP

REG_PCON PROC NEAR
;-------------------------------------------------------------------------------
;Prepara y muestra ventana de pcon
;-------------------------------------------------------------------------------
        mov     di,offset lpcon
        mov     bl,pcon
        call    hexabin
        mov     bx,offset vpcon
        mov     ax,ds
        mov     si,offset lpcon
        vnt     "a"
        ret
REG_PCON ENDP

PUERTOS PROC NEAR
;-------------------------------------------------------------------------------
;Prepara y muestra ventana de puertos
;-------------------------------------------------------------------------------
        mov     di,offset lpuerto       ;buffer de ventana de puertos
        mov     bl,p0
        call    hexabin                 ;carga puerto 0 en binario
        mov     bl,p0_pin
        call    hexabin                 ;carga los pines del puerto 0
        mov     bl,p1
        call    hexabin                 ;carga puerto 1 en binario
        mov     bl,p1_pin
        call    hexabin                 ;carga los pines del puerto 1
        mov     bl,p2
        call    hexabin                 ;carga puerto 2 en binario
        mov     bl,p2_pin
        call    hexabin                 ;carga los pines del puerto 2
        mov     bl,p3
        call    hexabin                 ;carga puerto 3 en binario
        mov     bl,p3_pin
        call    hexabin                 ;carga los pines del puerto 3
        mov     bx,offset vpuerto       ;muestra ventana de puertos
        mov     ax,ds
        mov     si,offset lpuerto
        vnt     "a"
        ret
PUERTOS ENDP

EDITOR PROC NEAR
;-------------------------------------------------------------------------------
;Permite modificar parametros que son visibles en pantalla, el orden de
;edicion es:
;1 registros, 2 banderas ,3 registros especiales, 4 timers y sbuf,
;5 area de  memoria 1, 6 area de memoria 2, 7 puertos.
;A traves de getdata se pide el ingreso de datos desde el teclado, segun
;lo ingresado se procesa.
;       TAB             pasa a proxima ventana
;       SHIFT TAB       pasa a ventana anterior
;       ESC             sale del editor
;Entrada:       B_EDITOR = ventana de edicion actual
```

```
;-------------------------------------------------------------------------
        lim23                           ;limpia linea 23
        lim24                           ;limpia linea 24
        print   4,23,123d_,revers       ;mensaje de Edicion
        cmp     b_editor,0              ;determina la ultima ventana
        jne     rd_01                   ;de edicion
        jmp     rd_1                    ;registros cpu
rd_01:  cmp     b_editor,1
        jne     rd_02
        jmp     rd_2                    ;banderas
rd_02:  cmp     b_editor,2
        jne     rd_03
        jmp     rd_3                    ;registros especiales
rd_03:  cmp     b_editor,3
        jne     rd_04
        jmp     rd_4                    ;area de memoria 1
rd_04:  cmp     b_editor,4
        jne     rd_05
        jmp     rd_5                    ;area de memoria 2
rd_05:  cmp     b_editor,5
        jne     rd_06
        jmp     rd_6                    ;timers y sbuf
rd_06:  jmp     rd_7                    ;puertos

;-------------------------------------------------------------------------
;Editor de registros CPU: ACC,PC,DPTR,SP,B,R0,R1,R2,R3,R4,R5,R6 y R7
;-------------------------------------------------------------------------
rd_1:   mov     b_editor,0
        mov     dl,0                    ;reg actual ACC
        mov     gdreset,1
        print   40,23,123regcpu,revers  ;mensaje de editor de reg. cpu
rd_11:  call    rd_10                   ;resalta registro a editar
        cmp     dl,1                    ;registro de 1 o 2 bytes?
        jne     rd_111
        jmp     rd_112
rd_111: cmp     dl,2
        jne     rd_113
rd_112: lim24
        print   1,24,124hex1,normal     ;registro de 2 bytes
        getdata 72,24,dhex1,'h'         ;espera numero de 2 bytes
        mov     al,atrprev
        mov     di,dirprev
        call    rd_102                  ;quita lo resaltado
        jmp     rd_114
rd_113: lim24
        print   1,24,124hex2,normal     ;registro de un byte
        getdata 74,24,dhex2,'h'         ;espera numero de 1 byte
        mov     al,atrprev
        mov     di,dirprev
        call    rd_103                  ;quita lo resaltado
rd_114: cmp     gccod,5
        jb      rd_12
        cmp     gccod,8
        jae     rd_122
        jmp     rd_1f                   ;TAB, SHIFT TAB o ESC
rd_12:  cmp     gccod,3                 ;--->?
```

```
        jne     rd_                     ;no
rd_121: inc     dl                      ;actualiza posicion
        cmp     dl,13
        jne     rd_122
        mov     dl,0
rd_122: jmp     rd_11
rd_13:  cmp     gccod,4                 ;<---?
        jne     rd_14                   ;no
        dec     dl                       ;actualiza posicion
        cmp     dl,0ffh
        jne     rd_131
        mov     dl,12
rd_131: jmp     rd_11
rd_14:  cmp     gccod,1                 ;arriba?
        jne     rd_15                    ;no
        cmp     dl,0                    ;actualiza posicion
        je      rd_141
        cmp     dl,2
        je      rd_142
        cmp     dl,1
        je      rd_143
        dec     dl
        dec     dl
        jmp     rd_11
rd_141: mov     dl,12
        jmp     rd_11
rd_142: mov     dl,11
        jmp     rd_11
rd_143: mov     dl,0
        jmp     rd_11
rd_15:  cmp     gccod,2                  ;abajo?
        jne     rd_16                   ;no
        cmp     dl,12                    ;actualiza posicion
        je      rd_151
        cmp     dl,11
        je      rd_152
        cmp     dl,0
        je      rd_153
        add     dl,2
        jmp     rd_11
rd_151: mov     dl,0
        jmp     rd_11
rd_152: mov     dl,2
        jmp     rd_11
rd_153: mov     dl,1
        jmp     rd_11
rd_16:  mov     gdreset,1
        mov     si,offset com_E
        cmp     dl,1
        je      rd_162
rd_161: cmp     dl,2
        jne     rd_163
rd_162: mov     bx,word ptr dhex1+6      ;bx <- word ingresada
        jmp     rd_acc
rd_163: mov     bl,byte ptr dhex2+4      ;bl <- byte ingresado
rd_acc: cmp     dl,0                    ;ACC?
```

```
        jne     rd_pc                   ;no
        cmp     modo,1                  ;modo de ejecucion controlada?
        je      rd_acc1                 ;si
        mov     acc,bl                  ;edita registro
        jmp     rd_164
rd_acc1:        mov     cx,3
        mov     byte ptr ds:[si+1],1
        mov     ds:[si+2],bl
        call    comunica                ;edicion en el uC
        jmp     rd_164
rd_pc:  cmp     dl,1                    ;PC?
        jne     rd_dp                   ;no
        cmp     modo,1                  ;modo de ejecucion controlada?
        je      rd_pc1                  ;si
        mov     pc,bx                   ;edita registro
        jmp     rd_164
rd_pc1: cmp     bx,806h
        jb      er_pc
        mov     cx,4
        mov     byte ptr ds:[si+1],2
        mov     ds:[si+2],bh
        mov     ds:[si+3],bl
        call    comunica                ;edicion en el uC
        jmp     rd_164
er_pc:  lim24
        print   1,24,err_pc,normal
        getkey
        jmp     rd_164
rd_dp:  cmp     dl,2                    ;DPTR?
        jne     rd_sp                   ;no
        cmp     modo,1                  ;modo de ejecucion controlada?
        je      rd_dp1                  ;si
        mov     dpl,bl                  ;edita registro
        mov     dph,bh
        jmp     rd_164
rd_dp1: mov     cx,4
        mov     byte ptr ds:[si+1],3
        mov     ds:[si+2],bh
        mov     ds:[si+3],bl
        call    comunica                ;edicion en el uC
        jmp     rd_164
rd_sp:  cmp     dl,3                    ;SP?
        jne     rd_b                    ;no
        cmp     modo,1                  ;modo de ejecucion controlada?
        je      rd_sp1                  ;si
        mov     regsp,bl                ;edita registro
        jmp     rd_164
rd_sp1: mov     cx,3
        mov     byte ptr ds:[si+1],4
        mov     ds:[si+2],bl
        call    comunica                ;edicion en el uC
        jmp     rd_164
rd_b:   cmp     dl,4                    ;B?
        jne     rd_r0                   ;no
        cmp     modo,1                  ;modo de ejecucion controlada?
        je      rd_b1                   ;si
```

```
        mov     b,bl                    ;edita registro
        jmp     rd_164
rd_b1:  mov     cx,3
        mov     byte ptr ds:[si+1],5
        mov     ds:[si+2],bl
        call    comunica                ;edicion en el uC
        jmp     rd_164
rd_r0:  cmp     modo,1                  ;modo de ejecucion controlada?
        je      rd_r1                   ;si
        push    bx                      ;edicion del banco de registros
        mov     bl,dl
        sub     bl,5
        mov     bh,0
        mov     di,offset r0            ;origen de RAM de datos interna
        mov     al,psw                  ;encuentra el banco
        and     al,18h                  ;de registros
        xor     ah,ah
        add     di,ax
        add     di,bx                   ;ubicacion del registro
        pop     bx
        mov     ds:[di],bl              ;edita registro
        jmp     rd_164
rd_r1:  mov     cl,dl
        inc     cl
        mov     ds:[si+1],cl
        mov     ds:[si+2],bl
        mov     cx,3
        call    comunica                ;edicion en el uC
rd_164: push    dx
        call    inicial                 ;actualiza pantalla
        pop     dx
        jmp     rd_121
rd_1f:  cmp     gccod,5                 ;TAB?
        jne     rd_1f1                  ;no
        jmp     rd_2                    ;salta a editor de banderas
rd_1f1: cmp     gccod,6                 ;SHIFT TAB?
        jne     rd_1f2                  ;no .
        jmp     rd_7                    ;salta a editor de puertos
rd_1f2: jmp     rd_f                    ;ESC sale del editor

rd_10:  mov     bl,dl
        mov     cl,31                   ;columna inicial
        mov     ch,3
        cmp     bl,0
        je      rd_1011
rd_100: inc     ch                      ;fila lista
        cmp     bl,2
        jbe     rd_101
        sub     bl,2
        jmp     rd_100
rd_101: dec     bl
rd_1011:        mov     al,9
        mul     bl
        add     cl,al                   ;columna lista
        posdir  cl,ch                   ;ubica posicion a resaltarse
        mov     di,pddir
```

```
        inc     di
        mov     al,70h
        mov     ah,es:[di]
        mov     atrprev,ah              ;guarda atributo previo
        mov     dirprev,di              ;guarda direccion previa
        cmp     dl,1
        jb      rd_103
        cmp     dl,2
        ja      rd_103
rd_102: stosb                           ;cambia el atributo de video
        inc     di
        stosb
        inc     di
rd_103: stosb
        inc     di
        stosb
        ret

;-------------------------------------------------------------------------
;Editor de banderas: C,AC,F0,OV,P
;-------------------------------------------------------------------------
rd_2:   mov     b_editor,1
        mov     dl,0                    ;Bandera actual carry
        mov     gdreset,1
        print   40,23,123band,revers
rd_21:  call    rd_20                   ;resalta bandera actual
        lim24
        print   1,24,124bin,normal
        getdata 75,24,dbin,"b"          ;Espera un bit
        mov     al,atrprev
        mov     di,dirprev
        call    rd_201                  ;quita lo resaltado
        cmp     gccod,0                 ;dato valido
        jne     rd_211                  ;no
        mov     bl,byte ptr dbin+2      ;bl <- bit ingresado
        mov     di,offset binario
rd_c:   cmp     dl,0                    ;carry?
        jne     rd_ac                   ;no
        mov     ds:[di],bl              ;edita bandera
        jmp     rd_210
rd_ac:  cmp     dl,1                    ;carry intermedio?
        jne     rd_f0                   ;no
        mov     ds:[di+1],bl            ;edita bandera
        jmp     rd_210
rd_f0:  cmp     dl,2                    ;F0?
        jne     rd_ov                   ;no
        mov     ds:[di+2],bl
        jmp     rd_210
rd_ov:  mov     ds:[di+5],bl            ;edita bandera over flow
rd_210: call    binahex                 ;actualiza PSW
        cmp     modo,1                  ;modo de ejecucion controlada?
        je      rd_psw                  ;si
        mov     psw,bl                  ;carga cambios en psw
        jmp     rd_2100
rd_psw: mov     si,offset com_E         ;edicion en el uC
        mov     cx,3
```

```
        mov     byte ptr ds:[si+1],14
        mov     ds:[si+2],bl
        call    comunica                ;edicion en el uC
rd_2100:        call    inicial          ;actualiza pantalla
        jmp     rd_220
rd_211: cmp     gccod,4
        ja      rd_24
        cmp     gccod,3
        jb      rd_27
rd_22:  cmp     gccod,3                  ;--->?
        jne     rd_23                   ;no
rd_220: inc     dl                      ;actualiza posicion
        cmp     dl,4
        jne     rd_221
        mov     dl,0
rd_221: jmp     rd_21
rd_23:  cmp     gccod,4                 ;<---?
        jne     rd_24                   ;no
        dec     dl                      ;actualiza posicion
        cmp     dl,0ffh
        jne     rd_231
        mov     dl,3
rd_231: jmp     rd_21
rd_24:  cmp     gccod,8
        ja      rd_27
        cmp     gccod,5                 ;TAB?
        jne     rd_25                    ;no
        jmp     rd_3                    ;salta a editor de reg. espec.
rd_25:  cmp     gccod,6                 ;SHIFT TAB?
        jne     rd_26                   ;no
        jmp     rd_1                    ;salta a editor de reg. cpu
rd_26:  jmp     rd_f                    ;ESC?
rd_27:  push    dx                      ;beep
        mov     ah,2
        mov     dl,7
        int     21h
        pop     dx
        jmp     rd_21
rd_20:  mov     cl,47                   ;columna inicial
        mov     al,3
        mul     dl
        add     cl,al                   ;columna lista
        mov     ch,4                    ;fila lista
        posdir  cl,ch
        mov     di,pddir                 ;posicion de la bandera
        inc     di
        mov     al,70h
        mov     ah,es:[di]
        mov     atrprev,ah               ;guarda atributo previo
        mov     dirprev,di               ;guarda direccion previa
rd_201: stosb                           ;pone atributo reverso
        ret

;-------------------------------------------------------------------------
;Editor de registros especiales: MCON,TMOD,IE,TCON,IP,SCON
;-------------------------------------------------------------------------
```

```
rd_3:   mov     b_editor,2
        cmp     modo,1                  ;modo de ejecucion controlada ?
        jne     rd_3a                   ;no
        jmp     rd_36
rd_3a:  mov     dl,0                     ;registro especial actual
        mov     dh,0                    ;mcon7
        mov     gdreset,1
        print   40,23,123regesp,revers
rd_31:  call    rd_30                   ;resalta bit actual
        lim24
        print   1,24,124bin,normal
        getdata 75,24,dbin,"b"          ;espera un bit
        mov     al,atrprev
        mov     di,dirprev
        call    rd_302                   ;quita lo resaltado
        cmp     gccod,5
        jb      rd_32
        cmp     gccod,8
        jae     rd_321
        jmp     rd_36
rd_32:  cmp     gccod,3                 ;--->?
        jne     rd_33                   ;no
rd_320: inc     dl                      ;actualiza posicion
        cmp     dl,16
        jne     rd_321
        mov     dl,0
rd_321: jmp     rd_31
rd_33:  cmp     gccod,4                 ;<---?
        jne     rd_34                   ;no
        dec     dl                      ;actualiza posicion
        cmp     dl,0ffh
        jne     rd_331
        mov     dl,15
rd_331: jmp     rd_31
rd_34:  cmp     gccod,1                 ;arriba?
        jne     rd_35                   ;no
        dec     dh                      ;actualiza posicion
        cmp     dh,0ffh
        jne     rd_341
        mov     dh,2
rd_341: jmp     rd_31
rd_35:  cmp     gccod,2                 ;abajo?
        jne     rd_352                   ;no
        inc     dh                      ;actualiza posicion
        cmp     dh,3
        jne     rd_351
        mov     dh,0
rd_351: jmp     rd_31
rd_352: mov     al,2                    ;binario valido
        mul     dh
        cmp     dl,8
        jb      rd_mcon
        add     al,1
rd_mcon:        cmp     al,0             ;MCON?
        jne     rd_tmod                 ;no
        cmp     dl,3                    ;son bits de paticiones?
```

```
        ja      rd_mc1                  ;no
        test    mcon,2                  ;PAA es 1?
        jne     rd_mc4                   ;si
        jmp     rd_320                  ;no -> no puede ser escrito
rd_mc1: cmp     dl,4                    ;bit RA32/8?
        jne     rd_mc2                  ;no
        jmp     rd_320                  ;si -> no puede ser escrito
rd_mc2: cmp     dl,6                    ;bit PAA?
        jne     rd_mc3                   ;no
        jmp     rd_320                  ;si -> no puede ser escrito
rd_mc3: cmp     dl,7                    ;bit SL?
        jne     rd_mc4                  ;no
        jmp     rd_320                  ;si -> no puede ser escrito
rd_mc4: mov     bl,mcon
        call    rd_303                  ;edicion del bit
        mov     mcon,bl
        jmp     rd_354
rd_tmod:        cmp     al,1            ;TMOD?
        jne     rd_ie                   ;no
        mov     bl,tmod
        call    rd_303                  ;edicion del bit
        mov     tmod,bl
        jmp     rd_354
rd_ie:  cmp     al,2                    ;IE?
        jne     rd_tcon                  ;no
        cmp     dl,1                    ;IE.6?
        jne     rd_ie1                   ;no
        jmp     rd_320                  ;si -> no se edita
rd_ie1: cmp     dl,2                    ;IE.5?
        jne     rd_ie2                   ;no
        jmp     rd_320                  ;si -> no se edita
rd_ie2: mov     bl,ie
        call    rd_303                   ;edicion del bit
        mov     ie,bl
        jmp     rd_354
rd_tcon:        cmp     al,3            ;TCON?
        jne     rd_ip                   ;no
        mov     bl,tcon
        call    rd_303                  ;edicion del bit
        mov     tcon,bl
        jmp     rd_354
rd_ip:  cmp     al,4                    ;IP?
        jne     rd_scon                 ;no
        cmp     dl,3                    ;IP.7 o IP.6 o IP.5?
        jae     rd_ip1                   ;no
        jmp     rd_320                  ;si -> no se edita
rd_ip1: mov     bl,ip
        call    rd_303                  ;edicion del bit
        mov     ip,bl
        jmp     rd_354
rd_scon:        mov     bl,scon         ;SCON
        call    rd_303                  ;edicion del bit
        mov     scon,bl
rd_354: call    inicial
        jmp     rd_320
rd_36:  cmp     gccod,5                 ;TAB?
```

```
        jne     rd_361                  ;no
        jmp     rd_6                    ;salta a editor de timer y sbuf
rd_361: cmp     gccod,6                 ;SHIFT TAB?
        jne     rd_362                  ;no
        jmp     rd_2                    ;salta a editor de banderas
rd_362: jmp     rd_f
rd_30:  mov     cl,51                   ;columna inicial
        mov     ch,7                    ;fila inicial
        add     ch,dh                   ;fila lista
        cmp     dl,8
        jb      rd_301
        add     cl,13
rd_301: add     cl,dl                   ;columna lista
        posdir  cl,ch
        mov     di,pddir                ;posicion del bit
        inc     di
        mov     al,70h
        mov     ah,es:[di]
        mov     atrprev,ah              ;guarda atributo previo
        mov     dirprev,di              ;guarda direccion previa
rd_302: stosb                           ;pone atributo reverso
        ret
rd_303: mov     di,offset binario
        call    hexabin                 ;convierte el byte
        mov     bl,dl                   ;de registro especial
        cmp     bl,8                    ;en una cadena de
        jb      rd_304                  ;8 caracteres binarios
        sub     bl,8
rd_304: mov     bh,0
        push    es
        push    ds                      ;ES = segmento de datos
        pop     es
        mov     di,offset binario
        add     di,bx
        mov     al,byte ptr dbin+2
        stosb                           ;ubica el bit ingresado
        pop     es                      ;en binario
        call    binahex                 ;binario(8) -> bl(1)
        ret

;-------------------------------------------------------------------------
;Editor del area de memoria 1
;-------------------------------------------------------------------------
rd_4:   mov     b_editor,3
        mov     dl,0                    ;direccion inicial
        mov     dh,0
        mov     gdreset,1
        print   40,23,123mem1,revers
rd_41:  call    rd_40                   ;resalta espacio memoria actual
        cmp     dl,8                    ;espacio ascii?
        jae     rd_41as                 ;si
        jmp     rd_411                  ;no
rd_41as:        lim24
        print   1,24,124asc,normal
        getdata 75,24,dasc,"a"          ;mem1 ascii
        mov     al,atrprev
```

```
        mov     di,dirprev
        call    rd_403                  ;quita lo resaltado
        cmp     m1in_ex,0                ;memoria interna?
        jne     rd_410                   ;no
        jmp     rd_412
rd_410: print   35,13,exter,0ah         ;mensaje de embebed
        jmp     rd_412
rd_411: lim24
        print   1,24,l24hex2,normal
        getdata 74,24,dhex2,"h"         ;mem1 hex
        mov     al,atrprev
        mov     di,dirprev
        call    rd_402                  ;quita lo resaltado
        cmp     m1in_ex,0                ;memoria interna
        jne     rd_41a                  ;no
        jmp     rd_412
rd_41a: print   35,13,exter,0ah         ;mensaje de embebed
rd_412: cmp     gccod,5
        jb      rd_42
        cmp     gccod,8
        jae     rd_421                  ;TAB, SHIFT TAB o ESC
        jmp     rd_46
rd_42:  cmp     gccod,3                  ;--->?
        jne     rd_43                   ;no
rd_420: inc     dl                       ;actualiza posicion
        cmp     dl,16
        jne     rd_421
        mov     dl,0
rd_421: jmp     rd_41
rd_43:  cmp     gccod,4                 ;<---?
        jne     rd_44                    ;no
        dec     dl                      ;actualiza posicion
        cmp     dl,0ffh
        jne     rd_431
        mov     dl,15
rd_431: jmp     rd_41
rd_44:  cmp     gccod,1                 ;arriba?
        jne     rd_45                    ;no
        dec     dh                      ;actualiza posicion
        cmp     dh,0ffh
        jne     rd_441
        mov     dh,3
rd_441: jmp     rd_41
rd_45:  cmp     gccod,2                 ;abajo?
        jne     rd_452                   ;no
        inc     dh                      ;actualiza posicion
        cmp     dh,4
        jne     rd_451
        mov     dh,0
rd_451: jmp     rd_41

rd_452: cmp     m1in_ex,0                ;dato valido
        jne     rd_4ex
        mov     bx,dirm1in              ;ram interna
        mov     di,offset r0
        mov     b_mem,07fh              ;direccion max. de mem. int.
```

```
        mov     seg_dat,0
        jmp     rd_4m
rd_4ex: cmp     ram?,0                  ;RAM embebed?
        je      rd_4sr                  ;si
        lim24                           ;no, es ROM
        print   1,24,no_ram,0f0h        ;imposibilidad de edicion
        push    dx
        mov     dl,7                    ;beep
        mov     ah,2
        int     21h
        pop     dx
        getkey
        lim24
        jmp     rd_41
rd_4sr: mov     bx,dirmiex
        push    ds
        mov     ds,segdat2
        mov     di,offset ram_emb
        pop     ds
        mov     ax,ram_emb_m
        mov     b_mem,ax                ;direccion max. de RAM Embebed
        mov     seg_dat,1
rd_4m:  call    guarmem
        call    inicial
        cmp     dl,7
        je      rd_4m1
        cmp     dl,15
        jne     rd_4m4
        mov     dl,8
        jmp     rd_4m2
rd_4m1: mov     dl,0
rd_4m2: inc     dh
        cmp     dh,4
        jne     rd_4m3
        mov     dh,0
rd_4m3: jmp     rd_41
rd_4m4: jmp     rd_420
rd_46:  cmp     gccod,5                 ;TAB?
        jne     rd_461                  ;no
        jmp     rd_5                    ;salta a editor de mem2
rd_461: cmp     gccod,6                 ;SHIFT TAB?
        jne     rd_462                  ;no
        jmp     rd_6                    ;salta a editor de timer y sbuf
rd_462: jmp     rd_f

rd_40:  cmp     miin_ex,0               ;Memoria Interna o Embebed?
        jne     rd_40e
        jmp     rd_40i                  ;Memoria Interna
rd_40e: cmp     dl,8
        jb      rd_40e1
        mov     bl,dl
        sub     bl,8                    ;Memoria ascii
        jmp     rd_40e2
rd_40e1:        mov     bl,dl           ;Memoria hex
rd_40e2:        mov     ax,8
        mul     dh
```

```
        add     al,bl
        add     ax,dirmiex
        and     ax,ram_emb_m
        cmp     ax,rom_ram_e
        jb      rd_40e3
        print   42,13,emb_ext,0ah       ;RAM
        mov     ram?,0
        jmp     rd_40i
rd_40e3:        print   42,13,emb_int,0ah       ;ROM
        mov     ram?,1
rd_40i: mov     ch,14
        add     ch,dh                   ;fila lista
        cmp     dl,8
        jb      rd_401                  ;memi ascii
        mov     cl,48
        add     cl,dl                   ;columna lista
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,7
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        jmp     rd_403

rd_401: mov     cl,32
        mov     al,3                    ;memi hex
        mul     dl
        add     cl,al                   ;columna lista
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,70h
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
rd_402: stosb
        inc     di
rd_403: stosb
        ret
guarmem:        push    es
        push    dx
        cmp     seg_dat,0               ;memoria interna o embebed
        jne     gm0_1
        mov     ax,dseg                 ;interna
        mov     es,ax
        jmp     gm0_2
gm0_1:  mov     ax,dseg2                ;externa
        mov     es,ax
gm0_2:  cmp     dl,8                    ;ascii o hexadecimal?
        jb      gm1_1
        mov     si,offset dasc+2        ;ascii
        sub     dl,8
        jmp     gm1_2
gm1_1:  mov     si,offset dhex2+4       ;hexadecimal
gm1_2:  mov     al,8
```

```
        mul     dh
        add     al,dl
        mov     ah,0                    ;posicion del caracter
        add     ax,bx                    ;direccion del caracter
        and     ax,b_mem                 ;correccion de direccion
        add     di,ax
        movsb                           ;edita en memoria
        pop     dx
        pop     es
        cmp     modo,1                  ;modo de ejecucion controlada?
        jne     gmf                     ;no
        mov     bl,ds:[si-1]            ;si, edicion en el uC
        mov     si,offset com_E
        cmp     seg_dat,0
        jne     r_emb
        mov     cx,4
        mov     byte ptr ds:[si+1],15
        mov     ds:[si+2],al
        mov     ds:[si+3],bl
        call    comunica                ;edicion en el uC
        jmp     gmf
r_emb:  mov     cx,5
        mov     byte ptr ds:[si+1],16
        mov     ds:[si+2],ah
        mov     ds:[si+3],al
        mov     ds:[si+4],bl
        call    comunica                ;edicion en el uC
        call    descar
gmf:    ret

;-------------------------------------------------------------------------
;Editor de area de memoria 2
;-------------------------------------------------------------------------
rd_5:
        mov     b_editor,4
        mov     dl,0                    ;direccion inicial
        mov     dh,0
        mov     gdreset,1
        print   40,23,123mem2,revers
rd_51:  call    rd_50                    ;resalta espacio memoria actual
        cmp     dl,8
        jae     rd_51as
        jmp     rd_511
rd_51as:        lim24
        print   1,24,124asc,normal
        getdata 75,24,dasc,"a"          ;mem2 ascii
        mov     al,atrprev
        mov     di,dirprev
        call    rd_503                  ;quita lo resaltado
        cmp     m2in_ex,0                ;memoria interna?
        jne     rd_510                  ;no
        jmp     rd_512
rd_510: print   35,18,exter,0ah         ;mensaje de embebida
        jmp     rd_512
rd_511: lim24
        print   1,24,124hex1,normal
```

```
        getdata  74,24,dhex2,"h"        ;mem2 hex
        mov      al,atrprev
        mov      di,dirprev
        call     rd_502                 ;quita lo resaltado
        cmp      m2in_ex,0
        jne      rd_51a
        jmp      rd_512
rd_51a: print    35,18,exter,0ah
rd_512: cmp      gccod,5
        jb       rd_52
        cmp      gccod,8
        jae      rd_521
        jmp      rd_56
rd_52:  cmp      gccod,3                ;--->?
        jne      rd_53                  ;no
rd_520: inc      dl                     ;actualiza posicion
        cmp      dl,16
        jne      rd_521
        mov      dl,0
rd_521: jmp      rd_51
rd_53:  cmp      gccod,4                ;<---?
        jne      rd_54                  ;no
        dec      dl                     ;actualiza posicion
        cmp      dl,0ffh
        jne      rd_531
        mov      dl,15
rd_531: jmp      rd_51
rd_54:  cmp      gccod,1                ;arriba?
        jne      rd_55                  ;no
        dec      dh                     ;actualiza posicion
        cmp      dh,0ffh
        jne      rd_541
        mov      dh,3
rd_541: jmp      rd_51
rd_55:  cmp      gccod,2                ;abajo?
        jne      rd_552                 ;no
        inc      dh                     ;actualiza posicion
        cmp      dh,4
        jne      rd_551
        mov      dh,0
rd_551: jmp      rd_51
rd_552: cmp      m2in_ex,0              ;dato valido
        jne      rd_5ex
        mov      bx,dirm2in
        mov      di,offset r0
        mov      b_mem,07fh             ;direccion max. de mem. int.
        mov      seg_dat,0
        jmp      rd_5m
rd_5ex: cmp      ram?,0
        je       rd_5sr
        lim24
        print    1,24,no_ram,0f0h
        push     dx
        mov      dl,7
        mov      ah,2
        int      21h
```

```
        pop     dx
        getkey
        lim24
        jmp     rd_51
rd_5sr: mov     bx,dirm2ex
        push    ds
        mov     ds,segdat2
        mov     di,offset ram_emb
        pop     ds
        mov     ax,ram_emb_m
        mov     b_mem,ax                ;direccion max. de RAM Embebed
        mov     seg_dat,1
rd_5m:  call    guarmem
        call    inicial
        cmp     dl,7
        je      rd_5m1
        cmp     dl,15
        jne     rd_5m4
        mov     dl,8
        jmp     rd_5m2
rd_5m1: mov     dl,0
rd_5m2: inc     dh
        cmp     dh,4
        jne     rd_5m3
        mov     dh,0
rd_5m3: jmp     rd_51
rd_5m4: jmp     rd_520
rd_56:  cmp     gccod,5                 ;TAB?
        jne     rd_561                  ;no
        jmp     rd_7                    ;salta a editor de puertos
rd_561: cmp     gccod,6                 ;SHIFT TAB?
        jne     rd_562                  ;no
        jmp     rd_4                    ;salta a editor de mem1
rd_562: jmp     rd_f
rd_50:  cmp     m2in_ex,0               ;Memoria Interna o Embebed?
        jne     rd_50e
        jmp     rd_50i                  ;Memoria Interna
rd_50e: cmp     dl,8
        jb      rd_50e1
        mov     bl,dl
        sub     bl,8                    ;Memoria ascii
        jmp     rd_50e2
rd_50e1:        mov     bl,dl           ;Memoria hex
rd_50e2:        mov     ax,8
        mul     dh
        add     al,bl
        add     ax,dirm2ex
        and     ax,ram_emb_m
        cmp     ax,rom_ram_e
        jb      rd_50e3
        print   42,18,emb_ext,0ah       ;RAM
        mov     ram?,0
        jmp     rd_50i
rd_50e3:        print   42,18,emb_int,0ah       ;ROM
        mov     ram?,1
rd_50i: mov     ch,19
```

```
        add     ch,dh                   ;fila lista
        cmp     dl,8
        jb      rd_501                   ;mem2 ascii
        mov     cl,48
        add     cl,dl                   ;columna lista
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,7
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
        jmp     rd_503
rd_501: mov     cl,32
        mov     al,3                     ;mem2 hex
        mul     dl
        add     cl,al                   ;columna lista
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,70h
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
rd_502: stosb
        inc     di
rd_503: stosb
        ret
;-------------------------------------------------------------------------------
;Editor de Timers y Sbuf: TH0,TL0,SBUF IN,TH1,TL1, SBUF OUT
;-------------------------------------------------------------------------------
rd_6:   mov     b_editor,5
        cmp     modo,1                  ;modo de ejecucion controlada?
        jne     rd_6a                   ;no
        jmp     rd_65
rd_6a:  mov     dl,0                     ;timer actual, TH0
        mov     dh,0
        print   40,23,123timer,revers
rd_61:  call    rd_60                   ;Resalta opcion actual
        lim24
        print   1,24,124hex2,normal
        getdata 74,24,dhex2,"h"
        mov     al,atrprev
        mov     di,dirprev
        call    rd_605                  ;quita lo resaltado
rd_610: cmp     gccod,1                 ;arriba?
        jne     rd_612                  ;no
        cmp     dh,0                    ;actualiza posicion
        jne     rd_611
        mov     dh,1
        jmp     rd_61
rd_611: mov     dh,0
        jmp     rd_61
rd_612: cmp     gccod,2                 ;abajo?
        jne     rd_62                   ;no
        cmp     dh,0                    ;actualiza posicion
```

```
        jne     rd_613
        mov     dh,1
        jmp     rd_61
rd_613: mov     dh,0
        jmp     rd_61
rd_62:  cmp     gccod,3                 ;--->?
        jne     rd_63                   ;no
rd_620: inc     dl                       ;actualiza posicion
        cmp     dl,3
        jne     rd_621
        mov     dl,0
rd_621: jmp     rd_61
rd_63:  cmp     gccod,4                 ;<---?
        jne     rd_64                   ;no
        dec     dl                      ;actualiza posicion
        cmp     dl,0ffh
        jne     rd_631
        mov     dl,2
rd_631: jmp     rd_61
rd_64:  cmp     gccod,0                 ;dato valido?
        jne     rd_65                    ;no
        mov     bl,byte ptr dhex2+4      ;bl <- byte ingresado
        mov     ax,3
        mul     dh
        add     al,dl                   ;ubica registro apuntado
rd_th0: cmp     al,0                     ;TH0?
        jne     rd_tl0                  ;no
        mov     th0,bl                   ;edita el registro
        jmp     rd_640
rd_tl0: cmp     al,1                    ;TL0?
        jne     rd_sbufi                 ;no
        mov     tl0,bl                   ;edita el registro
        jmp     rd_640
rd_sbufi:       cmp     al,2             ;SBUF IN?
        jne     rd_th1                  ;no
        mov     sbuf_in,bl               ;edita el registro
        or      scon,1                   ;setea RI
        jmp     rd_640
rd_th1: cmp     al,3                    ;TH1?
        jne     rd_tl1                   ;no
        mov     th1,bl                  ;edita el registro
        jmp     rd_640
rd_tl1: cmp     al,4                     ;TL1?
        jne     rd_sbufo                 ;no
        mov     tl1,bl                  ;edita el registro
        jmp     rd_640
rd_sbufo:       mov     sbuf,bl          ;edito SBUF OUT
        or      scon,2                   ;setea TI
rd_640: call    inicial                 ;actualiza pantalla
        jmp     rd_620
rd_65:  cmp     gccod,5                  ;TAB?
        jne     rd_651                  ;no
        jmp     rd_4                     ;salta a editor de mem1
rd_651: cmp     gccod,6                 ;SHIFT TAB?
        jne     rd_652                   ;no
        jmp     rd_3                    ;salta a editor de reg. espec.
```

```
rd_652: cmp     gccod,7                  ;ESC?
        jne     rd_631                  ;no
        jmp     rd_f                     ;sale del editor

rd_60:  mov     cl,51                   ;fila inicial
        mov     ch,11                    ;columna inicial
        add     ch,dh                   ;columna lista
        cmp     dl,0
        jne     rd_601
        jmp     rd_603
rd_601: cmp     dl,1
        jne     rd_602
        add     cl,10                   ;fila lista de TL
        jmp     rd_603
rd_602: add     cl,21                    ;fila lista de sbuf
rd_603: posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,70h
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
rd_605: stosb
        inc     di
        stosb
        ret

;-------------------------------------------------------------------------------
;Editor de puertos: P0,P1,P2,P3
;-------------------------------------------------------------------------------
rd_7:   mov     b_editor,6
        cmp     modo,1                  ;modo de ejecucion controlada?
        jne     rd_7a                   ;no
        jmp     rd_76
rd_7a:  mov     dl,0                    ;puerto actual
        mov     dh,0                    ;puerto 0
        mov     gdreset,1
        print   40,23,123puert,revers
rd_71:  call    rd_70                   ;resalta bit actual
        lim24
        print   1,24,124bin,normal
        getdata 75,24,dbin,"b"
        mov     al,atrprev
        mov     di,dirprev
        call    rd_702                   ;quita el resaltado
        cmp     gccod,5
        jb      rd_72
        cmp     gccod,8
        jae     rd_721
        jmp     rd_76
rd_72:  cmp     gccod,3                  ;--->?
        jne     rd_73                   ;no
rd_720: inc     dl                       ;actualiza posicion
        cmp     dl,8
        jne     rd_721
        mov     dl,0
```

```
rd_721: jmp     rd_71
rd_73:  cmp     gccod,4                 ;<---?
        jne     rd_74                    ;no
        dec     dl                       ;actualiza posicion
        cmp     dl,0ffh
        jne     rd_731
        mov     dl,7
rd_731: jmp     rd_71
rd_74:  cmp     gccod,1                  ;arriba?
        jne     rd_75                   ;no
        dec     dh                      ;actualiza posicion
        cmp     dh,0ffh
        jne     rd_741
        mov     dh,3
rd_741: jmp     rd_71
rd_75:  cmp     gccod,2                  ;abajo?
        jne     rd_752                  ;no
        inc     dh                       ;actualiza posicion
        cmp     dh,4
        jne     rd_751
        mov     dh,0
rd_751: jmp     rd_71
rd_752: cmp     dh,0                     ;P0?
        jne     rd_p1                    ;no
        mov     bl,p0_pin
        mov     bh,p0
        call    rd_703
        jc      rd_754
        mov     p0_pin,bl
        jmp     rd_754
rd_p1:  cmp     dh,1                    ;P1?
        jne     rd_p2                   ;no
        mov     bl,p1_pin
        mov     bh,p1
        call    rd_703
        jc      rd_754
        mov     p1_pin,bl
        jmp     rd_754
rd_p2:  cmp     dh,2                     ;p2?
        jne     rd_p3                   ;no
        mov     bl,p2_pin
        mov     bh,p2
        call    rd_703
        jc      rd_754
        mov     p2_pin,bl
        jmp     rd_754
rd_p3:  mov     bl,p3_pin
        mov     bh,p3
        call    rd_703
        jc      rd_754
        mov     p3_pin,bl
rd_754: call    inicial                  ;actualiza pantalla
        jmp     rd_720
rd_76:  cmp     gccod,5                  ;TAB?
        jne     rd_761                  ;no
        jmp     rd_1                    ;salta a editor de reg. cpu
```

```
rd_761: cmp     gccod,6                 ;SHIFT TAB?
        jne     rd_762                  ;no
        jmp     rd_5                    ;salta a editor de mem2
rd_762: jmp     rd_f
rd_70:  mov     cl,71                   ;columna inicial
        mov     ch,16                   ;fila inicial
        mov     ax,2
        mul     dh
        add     ch,al                   ;fila lista
        add     cl,dl                   ;columna lista
        posdir  cl,ch
        mov     di,pddir
        inc     di
        mov     al,70h
        mov     ah,es:[di]
        mov     atrprev,ah
        mov     dirprev,di
rd_702: stosb
        ret
rd_703: push    bx
        mov     bl,bh
        mov     di,offset binario
        call    hexabin
        xor     bh,bh
        mov     bl,dl
        mov     si,offset binario
        cmp     byte ptr ds:[si+bx],"0"
        jne     rd_704
        pop     bx
        stc                             ;setea carry si no
        jmp     rd_705                  ;se puede escribir
rd_704: pop     bx
        mov     di,offset binario
        call    hexabin
        mov     bl,dl
        xor     bh,bh
        mov     si,offset binario
        mov     al,byte ptr dbin+2
        cmp     dl,2                    ;El bit corresponde a algun
        jb      rd_7041                 ;pin que genera interrupcion
        cmp     dl,5
        ja      rd_7041
        mov     cl,ds:[si+bx]           ;verifica si hay cambio
        cmp     cl,"1"                  ;de 1L a 0L
        jne     rd_7041
        cmp     al,"0"
        jne     rd_7041
        call    interr                  ;cumplir requerimiento de int.
rd_7041:
        mov     byte ptr ds:[si+bx],al  ;ubica el bit ingresado
        call    binahex
        clc                             ;limpio carry si escribi
rd_705: ret
rd_f:   ret                             ;salida del editor
EDITOR ENDP
```

```
INTERR PROC NEAR
;-------------------------------------------------------------------------
;Determina si se edito uno de los pines que producen interrupcion
;-------------------------------------------------------------------------
        push    ax
        push    si
int0:   cmp     dl,5                    ;interrupcion externa 0
        jne     int1
        or      tcon,2                  ;setea IE0
        jmp     interf
int1:   cmp     dl,4                    ;interrupcion externa 1
        jne     t0
        or      tcon,8                  ;setea IE1
        jmp     interf
t0:     cmp     dl,3                    ;interrupcion timer 0
        je      t0_0
        jmp     t1
t0_0:   test    tmod,4                  ;esta en modo contador
        je      inter_f                 ;no
        test    tcon,10h                ;TR0 habilitado?
        je      inter_f                 ;no
        test    tmod,8                  ;GATE es 1?
        je      t0_1                    ;no
        test    p3_pin,4                ;INT0 es 1?
        je      inter_f                 ;no
t0_1:   mov     al,tmod
        and     al,3
t0_m0:  cmp     al,0                    ;modo 0?
        jne     t0_m1                   ;no
        cmp     tl0,1fh                 ;actualizar TH0?
        jne     t0_m0i                  ;no
        cmp     th0,0ffh                ;overflow?
        jne     t0_m00                  ;no
        or      tcon,20h                ;setea TF0
t0_m00: inc     th0                     ;actualiza TH0
        mov     tl0,0                   ;actualiza TL0
        jmp     interf
t0_m0i: inc     tl0                     ;actualiza TL0
inter_f:        jmp     interf
t0_m1:  cmp     al,1                    ;modo 1?
        jne     t0_m2                   ;no
        cmp     tl0,0ffh                ;actualizar TH0?
        jne     t0_m11                  ;no
        cmp     th0,0ffh                ;overflow?
        jne     t0_m10                  ;no
        or      tcon,20h                ;setea TF0
t0_m10: inc     th0                     ;actualiza TH0
t0_m11: inc     tl0                     ;actualiza TL0
        jmp     interf
t0_m2:  cmp     al,2                    ;modo 2?
        jne     t0_m3                   ;no
        cmp     tl0,0ffh                ;overflow
        jne     t0_m20                  ;no
        or      tcon,20h                ;setea TF0
        mov     al,th0
        mov     tl0,al                  ;carga TL0 con TH0
```

```
        jmp     interf
t0_m20: inc     tl0                     ;incrementa TL0
        jmp     interf
t0_m3:  cmp     tl0,0ffh                 ;modo 3
        jne     t0_m30                  ;salta si no hay overflow
        or      tcon,20h                 ;setea TF0
t0_m30: inc     tl0                      ;actualiza TL0
        cmp     th0,0ffh                 ;overflow
        jne     t0_m31                  ;no
        or      tcon,80h                 ;setea TF1
t0_m31: inc     th0                      ;actualiza TH0
        jmp     interf
t1:     test    tmod,40h                 ;esta en modo contador
        je      int_f                    ;no
        test    tcon,40h                 ;TR1 habilitado?
        je      int_f                   ;no
        test    tmod,80h                 ;GATE es 1?
        je      t1_1                     ;no
        test    p3_pin,8                 ;INT1 es 1?
        je      int_f                   ;no
t1_1:   mov     al,tmod
        and     al,30h
t1_m0:  cmp     al,0                    ;modo 0?
        jne     t1_m1                    ;no
        cmp     tl1,1fh                 ;actualizar TH1?
        jne     t1_m01                   ;no
        cmp     th1,0ffh                 ;overflow?
        jne     t1_m00                  ;no
        mov     al,tmod
        and     al,3                     ;Timer 0
        cmp     al,3                    ;en modo 3?
        je      t1_m00                   ;si
        or      tcon,80h                 ;setea TF1
t1_m00: inc     th1                     ;actualiza TH1
        mov     tl1,0                    ;actualiza TL1
        jmp     interf
t1_m01: inc     tl1                     ;actualiza TL1
int_f:  jmp     interf
t1_m1:  cmp     al,10h                  ;modo 1?
        jne     t1_m2                    ;no
        cmp     tl1,0ffh                 ;actualizar TH1?
        jne     t1_m11                   ;no
        cmp     th1,0ffh                 ;overflow?
        jne     t1_m10                  ;no
        mov     al,tmod
        and     al,3                    ;Timer 0
        cmp     al,3                    ;en modo 3?
        je      t1_m10                   ;si
        or      tcon,80h                 ;setea TF1
t1_m10: inc     th1                     ;actualiza TH1
t1_m11: inc     tl1                     ;actualiza TL1
        jmp     interf
t1_m2:  cmp     al,20h                   ;modo 2?
        jne     t1_m3                   ;no
        cmp     tl1,0ffh                 ;overflow
        jne     t1_m20                  ;no
```

```
        mov     al,tmod
        and     al,3                    ;Timer 0
        cmp     al,3                    ;en modo 3?
        je      t1_m200                 ;si
        or      tcon,80h                 ;setea TF1
t1_m200:        mov     al,th1
        mov     tl1,al                  ;carga TL1 con TH1
        jmp     interf
t1_m20: inc     tl1                     ;incrementa TL1
        jmp     interf
t1_m3:
interf: pop     si
        pop     ax
        ret
INTERR ENDP

CAMM1IN PROC NEAR
;-------------------------------------------------------------------------
;Cambia a area de visualizacion de la memoria interna 1
;Entrada:       Byte correspondiente a la direccion inicial
;Salida:        DIRM1IN = byte ingresado
;               M1IN_EX = 0
;-------------------------------------------------------------------------
        lim23
        lim24
        print   1,23,cambio_m,revers
        print   1,24,124hex2,normal
        mov     gdreset,1
cam1_1: getdata 74,24,dhex2,"h"         ;ingreso de direccion
        cmp     gccod,0                 ;dato valido ?
        je      cam1_2                  ;si
        cmp     gccod,7                 ;no, ESC ?
        je      cam1_3                  ;si, sale
        jmp     cam1_1                   ;no
cam1_2: mov     bl,byte ptr dhex2+4      ;carga direccion ingresada
        cmp     bl,07fh                 ;direccion > 7FH ?
        ja      cam1_4                  ;si, error
        xor     bh,bh                   ;no
        mov     dirm1in,bx               ;cambia direccion inicial
        mov     m1in_ex,0
        call    inicial                 ;actualiza pantalla
cam1_3: ret
cam1_4: lim23                            ;error, direccion muy alta
        lim24
        print   1,23,er_memi,revers      ;presenta mensaje de error
        print   1,24,tec_sal,normal
        getkey
        ret
CAMM1IN ENDP

CAMM1EX PROC NEAR
;-------------------------------------------------------------------------
;Cambia a area de visualizacion de la memoria interna 1 a embebida
;Entrada:       Byte correspondiente a la direccion inicial
;Salida:        DIRM1EX = byte ingresado
;               M1IN_EX = 1
```

```
;------------                                    ------------------------------------
        lim23
        lim24
        print   1,23,cambio_m,revers
        print   1,24,124hex1,normal
        mov     gdreset,1
cam2_1: getdata 72,24,dhex1,"h"          ;ingreso de direccion
        cmp     gccod,0                 ;dato valido ?
        je      cam2_2                  ;si
        cmp     gccod,7                  ;no, ESC ?
        je      cam2_3                  ;si, sale
        jmp     cam2_1                  ;no
cam2_2: mov     bx,word ptr dhex1+6      ;carga direccion ingresada
        cmp     bx,07fffh                ;direccion > 7FFFH ?
        ja      cam2_4                  ;si error
        mov     dirm1ex,bx               ;cambia direccion
        mov     m1in_ex,1
        call    inicial                  ;actualiza pantalla
cam2_3: ret
cam2_4: lim23                           ;error, dir. > 7FFFH
        lim24
        print   1,23,er_meme,revers      ;presenta mensaje de error
        print   1,24,tec_sal,normal
        getkey
        ret
CAMM1EX ENDP

CAMM2IN PROC NEAR
;-------------------------------------------------------------------------------
;Cambia a area de visualizacion de la memoria interna 2 a interna
;Entrada:        Byte correspondiente a la direccion inicial
;Salida:         DIRM2IN = byte ingresado
;                M2IN_EX = 0
;-------------------------------------------------------------------------------
        lim23
        lim24
        print   1,23,cambio_m,revers
        print   1,24,124hex2,normal
        mov     gdreset,1
cam3_1: getdata 74,24,dhex2,"h"          ;ingreso de direccion
        cmp     gccod,0                 ;dato valido ?
        je      cam3_2                   ;si
        cmp     gccod,7                 ;no, ESC ?
        je      cam3_3                   ;si, sale
        jmp     cam3_1                  ;no
cam3_2: mov     bl,byte ptr dhex2+4      ;carga direccion ingresada
        cmp     bl,07fh                  ;direccion > 7FH ?
        ja      cam3_4                  ;si ,error
        xor     bh,bh
        mov     dirm2in,bx               ;cambia direccion
        mov     m2in_ex,0
        call    inicial                  ;actualiza pantalla
cam3_3: ret
cam3_4: lim23                           ;error, dir. > 7FH
        lim24
        print   1,23,er_memi,revers      ;presenta mensaje de error
```

```
        print   1,24,tec_sal,normal
        getkey
        ret
CAMM2IN ENDP

CAMM2EX PROC NEAR
;-------------------------------------------------------------------------
;Cambia a area de visualizacion de la memoria interna 2 a embebida
;Entrada:       Byte correspondiente a la direccion inicial
;Salida:        DIRM2EX = byte ingresado
;               M2IN_EX = 1
;-------------------------------------------------------------------------
        lim23
        lim24
        print   1,23,cambio_m,revers
        print   1,24,124hex1,normal
        mov     gdreset,1
cam4_1: getdata 72,24,dhex1,"h"         ;ingreso de direccion
        cmp     gccod,0                 ;dato valido ?
        je      cam4_2                  ;si
        cmp     gccod,7                 ;no, ESC ?
        je      cam4_3                  ;si, sale
        jmp     cam4_1                  ;no
cam4_2: mov     bx,word ptr dhex1+6     ;carga direccion ingresada
        cmp     bx,07fffh               ;direccion > 7FFFH
        ja      cam4_4                  ;si, error
        mov     dirm2ex,bx              ;cambia direccion
        mov     m2in_ex,1
        call    inicial                 ;refresca pantalla
cam4_3: ret
cam4_4: lim23                           ;error, dir. > 7FFFH
        lim24
        print   1,23,er_meme,revers     ;presenta mensaje de error
        print   1,24,tec_sal,normal
        getkey
        ret
CAMM2EX ENDP

DEP_PASO PROC NEAR
;-------------------------------------------------------------------------
;Ejecuta una instruccion dentro del listado del usuario
;actualiza ciclos, si bandera de ciclos esta activa
;Entrada:       PC
;               AUX = 0 Opcion paso
;                   = 1 Opcion corre
;               B_CICLO = 0 conteo de ciclos desactivo
;                       = 1 conteo de ciclos activo
;-------------------------------------------------------------------------
        cmp     modo,1
        jne     dep_0
        call    ciclos                  ;# de ciclos de la inst.
        push    cx
        mov     si,offset com_P
        call    tx_comand               ;ejecuta en el uC
        mov     si,offset buff_rx
        lodsw                           ;lee pch en ascii
```

```
        call    aschex
        mov     bh,al
        lodsw                           ;lee pcl en ascii
        call    aschex
        mov     bl,al
        mov     pc,bx                   ;carga PC en hexadecimal
        pop     cx
        jmp     dep_1
dep_0:  call    ejecutor
dep_1:  cmp     b_ciclo,0               ;activo ciclos?
        je      dep_2                   ;no
        add     cx,n_ciclos             ;si, actualizar ciclos
        mov     n_ciclos,cx
dep_2:  cmp     aux,0                   ;opcion paso?
        je      dep_3                   ;si
        cmp     b_dis,0                 ;no, display en on?
        jne     dep_f                   ;no
dep_3:  cmp     b_ciclo,0               ;ciclos activo
        je      dep_5                   ;no
        mov     ax,n_ciclos             ;si, muestra ventana
        mov     cx,5                    ;de ciclos actualizada
        mov     si,offset lciclo
dep_4:  mov     byte ptr ds:[si]," "
        inc     si
        loop    dep_4
        dec     si
        call    hexadec
        mov     bx,offset vciclo1
        mov     si,offset lciclo
        mov     ax,ds
        vnt     "a"
dep_5:  call    inicial                 ;refresca pantalla
dep_f:  ret
DEP_PASO ENDP

DEP_CORRE PROC NEAR
;-------------------------------------------------------------------------------
;Va ejecutando secuencialmente las instrucciones del programa
;la ejecucion se detiene si:
;               PC es un punto de ruptura
;               Se presiona la tecla ESC
;               Cuando se entra en un lazo reiterativo estando la
;               actualizacion de pantalla desactivada
;Entrada:       Tabla de puntos de ruptura
;               B_DIS = 0 Activada actualizacion de pantalla
;                     = 1 Desactivada actualizacion de pantalla
;-------------------------------------------------------------------------------
        lim23
        lim24
        print   15,23,123_cor,revers
dep_c1: push    pc                      ;guarda PC
        call    dep_paso                ;ejecuta una instruccion
        pop     ax                      ;restablece PC
        cmp     b_dis,0                 ;display en on?
        je      dep_c2                  ;si
        cmp     ax,pc                   ;si, PC anterior = PC actual?
```

```
        jne     dep_c2                  ;no
        jmp     dep_cf                  ;si, fin
dep_c2: mov     ah,0bh                  ;verifica si en el buffer
        int     21h                     ;de teclado hay datos
        cmp     al,0ffh
        jne     dep_c21                 ;no hay dato
        mov     ah,0                    ;lee dato
        int     16h
        cmp     al,1bh                  ;es ESC?
        je      dep_cf                  ;si salga
dep_c21:        cmp     rup_act,0       ;hay break points?
        jne     dep_c3                  ;si
        jmp     dep_c1                  ;no
dep_c3: mov     cl,rup_act              ;cl = # de break points
        mov     si,offset tabla_brk     ;si -> tabla de break points
        add     si,3
dep_c4: mov     ax,ds:[si]
        cmp     ax,pc                   ;break point = pc?
        jne     dep_c5                  ;no
        jmp     dep_c6                  ;si
dep_c5: add     si,5
        dec     cl
        cmp     cl,0
        jne     dep_c4
        jmp     dep_c1
dep_c6: cmp     byte ptr [si-2],1       ;es breakpoint activo
        jne     dep_c1                  ;no
        cmp     byte ptr[si-1],0        ;dinamico?
        jne     dep_cf                  ;no
dep_c7: sub     si,3                    ;si, borrar break point
        mov     di,si                   ;de tabla de breakpoints
        add     si,5
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        add     cl,7
        sub     cl,rup_act              ;# de veces que  desplazar
        cmp     cl,0
        je      dep_c9
dep_c8: lodsb
        dec     al                      ;actualiza # de breakpoint
        stosb
        movsw                           ;activo/des. y dinam./esta.
        movsw                           ;valor de PC
        dec     cl
        jnz     dep_c8
dep_c9: mov     ax,0
        stosb                           ;borra ultima localidad
        stosw                           ;de la tabla
        stosw
        pop     es
        dec     rup_act                 ;actualiza # de rupturas
        call    mcod
dep_cf: cmp     b_dis,0                 ;esta activada la pantalla
        je      dep_cf1
        call    inicial                 ;si refresca la pantalla
```

```
        cmp     b_ciclo,0               ;ciclos activo ?
        je      dep_cf1                 ;no
        mov     ax,n_ciclos             ;refresca ventana de ciclos
        mov     cx,5
        mov     si,offset lciclo
dep_cf0:        mov     byte ptr ds:[si]," "
        inc     si
        loop    dep_cf0
        dec     si
        call    hexadec
        mov     bx,offset vciclo1
        mov     si,offset lciclo
        mov     ax,ds
        vnt     "a"
dep_cf1:        ret
DEP_CORRE ENDP

RUPTURA PROC FAR
;-------------------------------------------------------------------------------
;Prepara y muestra tabla con puntos de ruptura
;Entrada:       RUP_ACT = numero de puntos de ruptura
;               TABLA_BRK = tabla de puntos de ruptura
;-------------------------------------------------------------------------------
        lim23
        lim24
        mov     di,offset lrup          ;borra buffer de ventana de rupturas
        mov     cx,64                   ;8 brkpnt * 8 caracteres/brkpnt
        mov     al,0
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        rep     stosb
        mov     si,offset tabla_brk
        mov     di,offset lrup
        mov     dl,0
        cmp     rup_act,0
        jne     rdr01
        jmp     rdr03
rdr01:  mov     al,5
        mul     dl
        mov     bp,ax
rdr02:  mov     al,ds:[si+bp]
        hexasc
        mov     [di],bh                 ;pone el # de break point
        mov     [di+1],bl
        mov     al,ds:[si+bp+1]
        cmp     al,0
        je      rdr020
        mov     byte ptr [di+2],"a"     ;marca con a si esta activada
        jmp     rdr021
rdr020: mov     byte ptr [di+2],"d"     ;marca con d si desactivo
rdr021: mov     al,ds:[si+bp+2]
        cmp     al,0
        jne     rdr022
        mov     byte ptr [di+3],"D"     ;D si es dinamico
        jmp     rdr023
```

```
rdr022: mov     byte ptr [di+3],"E"     ;E si es estatico
rdr023: mov     al,ds:[si+bp+4]         ;muestra PC
        hexasc
        xchg    bl,bh
        mov     [di+4],bx
        mov     al,ds:[si+bp+3]
        hexasc
        xchg    bl,bh
        mov     [di+6],bx
        inc     dl
        cmp     dl,rup_act
        je      rdr03
        add     di,8
        jmp     rdr01
rdr03:  pop     es
        mov     bx,offset vrup
        mov     ax,ds
        mov     si,offset lrup
        vnt     "a"
        ret
RUPTURA ENDP

CREAR_RUP PROC NEAR
;-------------------------------------------------------------------------
;Permite ingresar un nuevo punto de ruptura
;Numero maximo de puntos de ruptura = 8
;No se puede tener dos puntos de ruptura con el mismo valor de PC
;Los puntos de ruptura pueden ser dinamicos o estaticos
;Salida:         RUP_ACT = numero de puntos de ruptura
;                TABLA_BRK = tabla de puntos de ruptura
;-------------------------------------------------------------------------
        guarpan
        call    ruptura
        print   1,23,123c_rup,revers
        print   37,24,124hex1,normal
        print   0,24,pagina,0ah
        push    pc
        mov     gdreset,1
        mov     b_cod_rup,0
cr_01:  getdata 72,24,dhex1,"h"
        mov     gdreset,0
        cmp     gccod,7                 ;ESC?
        jne     cr_02                   ;no
        jmp     crf1
cr_02:  cmp     gccod,9                 ;PgDn?
        jne     cr_03                   ;no
        cmp     b_cod_rup,8             ;muestra ventana anterior de cod.
        jb      cr_021
        jmp     cr_022
cr_021: mov     si,offset buf_cod
        mov     ax,2
        mul     b_cod_rup               ;encuentra posicion en el
        add     si,ax                   ;buffer
        mov     ax,cod_pc
        mov     word ptr ds:[si],ax     ;guarda en buffer cod_pc
        inc     b_cod_rup               ;actualiza b_cod_rup
```

```
cr_022: mov     ax,prox_pc
        mov     pc,ax
        call    mcod
        jmp     cr_01
cr_03:  cmp     gccod,8                 ;PgUp?
        jne     cr_04                   ;no
        cmp     b_cod_rup,0             ;muestra siguiente ventana de cod.
        jne     cr_031
        jmp     cr_01
cr_031: dec     b_cod_rup
        mov     ax,2
        mul     b_cod_rup
        mov     si,offset buf_cod
        add     si,ax
        mov     ax,word ptr ds:[si]
        mov     pc,ax
        call    mcod
cr_032: jmp     cr_01
cr_04:  cmp     gccod,0                 ;dato valido?
        jne     cr_032                  ;no
        mov     bx,word ptr dhex1+6
        mov     si,offset tabla_brk
        add     si,3
        mov     ch,0
        mov     cl,rup_act
        cmp     cl,0
        je      cr_11
cr_1:   cmp     cl,8
        jb      cr_0
        jmp     cr_3
cr_0:   mov     ax,ds:[si]
        cmp     ax,bx
        jne     cr_00
        jmp     cr_2
cr_00:  add     si,5
        loop    cr_0
cr_11:  mov     di,offset tabla_brk
        mov     ax,5
        mul     rup_act
        inc     rup_act
        add     di,ax
        mov     al,rup_act
        mov     ds:[di],al              ;guarda # ruptura
        mov     byte ptr ds:[di+1],1    ;guarda codigo de act/desact.
        lim23
        lim24
        print   1,23,din_est,revers
cr_110: getkey
        mov     ax,getword
        and     al,0dfh
        cmp     al,"D"                  ;dinamico?
        jne     cr_111
        mov     byte ptr ds:[di+2],0
        jmp     cr_112
cr_111: cmp     al,"E"                  ;estatico?
        jne     cr_110
```

```
        mov     byte ptr ds:[di+2],1
cr_112: mov     ds:[di+3],bx            ;guarda PC
        call    ruptura
        call    mcod
        print   1,23,lis_rup,revers     ;operacion lista
        jmp     crf
cr_2:   print   1,23,hay_rup,revers     ;ya hay ese break point
        jmp     crf
cr_3:   print   1,23,exc_rup,revers     ;excedio # de break points
crf:    print   40,23,tec_sal,revers
        getkey
crf1:   pop     pc
        traepan
        call    inicial
        ret
CREAR_RUP ENDP

ACT_RUP PROC NEAR
;-------------------------------------------------------------------------------
;Activa o desactiva un punto de ruptura
;Entrada:       RUP_ACT = numero de puntos de ruptura
;               TABLA_BRK = tabla de puntos de ruptura
;-------------------------------------------------------------------------------
        guarpan
        call    ruptura
        print   1,23,123a_rup,revers
        print   37,24,124hex2,normal
        print   0,24,pagina,0ah
        push    pc
        mov     gdreset,1
        mov     b_cod_rup,0
ar_01:  getdata 74,24,dhex2,"h"
        cmp     gccod,7                 ;ESC?
        jne     ar_02                   ;no
        jmp     arf1
ar_02:  cmp     gccod,9                 ;PgDn?
        jne     ar_03                   ;no
        cmp     b_cod_rup,8             ;muestra ventana anterior de codigo
        jb      ar_021
        jmp     ar_022
ar_021: mov     si,offset buf_cod
        mov     ax,2
        mul     b_cod_rup               ;encuentro posicion en el
        add     si,ax                   ;buffer
        mov     ax,cod_pc
        mov     word ptr ds:[si],ax     ;guardo en buffer cod_pc
        inc     b_cod_rup               ;actualizo b_cod_rup
ar_022: mov     ax,prox_pc
        mov     pc,ax
        call    mcod
        jmp     ar_01
ar_03:  cmp     gccod,8                 ;PgUp?
        jne     ar_04                   ;no
        cmp     b_cod_rup,0             ;muestra siguiente ventana de cod.
        jne     ar_031
        jmp     ar_01
```

```
ar_031: dec     b_cod_rup
        mov     ax,2
        mul     b_cod_rup
        mov     si,offset buf_cod
        add     si,ax
        mov     ax,word ptr ds:[si]
        mov     pc,ax
        call    mcod
ar_032: jmp     ar_01
ar_04:  cmp     gccod,0                 ;dato valido?
        jne     ar_032                  ;no
        mov     bl,byte ptr dhex2+4
        cmp     bl,0
        je      ar_4
        cmp     rup_act,0               ;hay break points?
        je      ar_3
        cmp     bl,rup_act              ;existe este # de ruptura
        jbe     ar_1
        jmp     ar_4
ar_1:   mov     si,offset tabla_brk
        mov     ax,5
        dec     bl
        mul     bl
        add     si,ax
        inc     si                      ;si apunta al byte de act/desact.
        mov     al,ds:[si]
        cmp     al,0                    ;inversion del byte de act/desact.
        jne     ar_2
        mov     byte ptr ds:[si],1
        jmp     ar_21
ar_2:   mov     byte ptr ds:[si],0
ar_21:  call    ruptura
        call    mcod
        print   1,23,lis_rup,revers     ;operacion lista
        jmp     arf
ar_3:   print   1,23,nohay_rup,revers   ;no hay break points
        jmp     arf
ar_4:   print   1,23,nohay_nrup,revers  ;no hay este break point
arf:    print   40,23,tec_sal,revers
        getkey
arf1:   pop     pc
        traepan
        call    inicial
        ret
ACT_RUP ENDP

BORRAR_RUP PROC NEAR
;-------------------------------------------------------------------------------
;Permite borrar un punto de ruptura
;Entrada:       RUP_ACT = numero de puntos de ruptura
;               TABLA_BRK = tabla de puntos de ruptura
;-------------------------------------------------------------------------------
        guarpan
        call    ruptura
        print   1,23,123b_rup,revers
        print   37,24,124hex2,normal
```

```
        print   0,24,pagina,0ah
        push    pc
        mov     gdreset,1
        mov     b_cod_rup,0
br_01:  getdata 74,24,dhex2,"h"
        cmp     gccod,7                 ;ESC?
        jne     br_02                   ;no
        jmp     brf1
br_02:  cmp     gccod,9                 ;PgDn?
        jne     br_03                   ;no
        cmp     b_cod_rup,8             ;muestra ventana anterior de codigo
        jb      br_021
        jmp     br_022
br_021: mov     si,offset buf_cod
        mov     ax,2
        mul     b_cod_rup               ;encuentra posicion en el
        add     si,ax                   ;buffer
        mov     ax,cod_pc
        mov     word ptr ds:[si],ax     ;guarda en buffer cod_pc
        inc     b_cod_rup               ;actualiza b_cod_rup
br_022: mov     ax,prox_pc
        mov     pc,ax
        call    mcod
        jmp     br_01
br_03:  cmp     gccod,8                 ;PgUp?
        jne     br_04                   ;no
        cmp     b_cod_rup,0             ;muestra siguiente ventana de codigo
        jne     br_031
        jmp     br_01
br_031: dec     b_cod_rup
        mov     ax,2
        mul     b_cod_rup
        mov     si,offset buf_cod
        add     si,ax
        mov     ax,word ptr ds:[si]
        mov     pc,ax
        call    mcod
br_032: jmp     br_01
br_04:  cmp     gccod,0                 ;dato valido?
        jne     br_032                  ;no
        mov     bl,byte ptr dhex2+4
        cmp     bl,0
        je      br_3
        cmp     rup_act,0               ;hay break points?
        je      br_2
        cmp     bl,rup_act              ;existe este # de ruptura
        jbe     br_1
        jmp     br_3
br_1:   mov     di,offset tabla_brk
        mov     ax,5
        dec     bl
        mul     bl
        add     di,ax                   ;SI = posicion del break point
        mov     si,di
        add     si,5                    ;SI = posicion brkpt + 4
        push    es
```

```
        push    ds                      ;ES = segmento de datos
        pop     es
        mov     cl,7
        sub     cl,bl                   ;CL = # de veces que hay que mover
        cmp     cl,0
        je      br_12
br_11:  lodsb
        dec     al                      ;actualiza # de break point
        stosb
        movsw                           ;codigo de act/desact. y din./est.
        movsw                           ;valor de PC
        dec     cl
        jnz     br_11
br_12:  mov     ax,0                    ;los ultimos valores del buffer
        stosb                           ;se cargan con 0
        stosw
        stosw
        pop     es
        dec     rup_act
        call    ruptura
        call    mcod
        print   1,23,lis_rup,revers
        jmp     brf
br_2:   print   1,23,nohay_rup,revers   ;no hay break points
        jmp     Brf
br_3:   print   1,23,nohay_nrup,revers  ;no hay este break point
brf:    print   40,23,tec_sal,revers
        getkey
brf1:   pop     pc
        traepan
        call    inicial
        ret
BORRAR_RUP ENDP

MOST_RUP PROC NEAR
;-------------------------------------------------------------------------------
;Muestra ventana con tabla de puntos de ruptura
;-------------------------------------------------------------------------------
        guarpan
        call    ruptura
        print   1,23,lis_rup,revers
        print   40,23,tec_sal,revers
        getkey
        traepan
        ret
MOST_RUP ENDP

HEXADEC PROC NEAR
;-------------------------------------------------------------------------------
;Convierte un numero hexadecimal a decimal
;Entrada:        AX = numero hexadecimal
;Salida:         DS:SI = string con numero decimal
;-------------------------------------------------------------------------------
        mov     cx,10
hexdc1: mov     dx,0
        div     cx
```

```
        add     dl,"0"
        mov     ds:[si],dl
        dec     si
        cmp     ax,0
        jne     hexdc1
hexdcf: ret
HEXADEC ENDP

SET_CIC PROC NEAR
;-------------------------------------------------------------------------
;Setea a 0 el conteo de ciclos
;-------------------------------------------------------------------------
        mov     n_ciclos,0              ;setea el numero de ciclos
        mov     ax,n_ciclos
        mov     cx,5
        mov     si,offset lciclo
setc1:  mov     byte ptr ds:[si]," "    ;carga " " en el buffer
        inc     si                      ;de la ventana ciclos
        loop    setc1
        dec     si
        call    hexadec
        cmp     b_ciclo,0               ;ciclos activo ?
        je      setc2                   ;no
        mov     bx,offset vciclo1       ;si, ventana ciclos activa
        jmp     setc3
setc2:  mov     bx,offset vciclo        ;ventana ciclos desactivada
setc3:  mov     si,offset lciclo        ;presenta ventana
        mov     ax,ds
        vnt     "a"
        ret
SET_CIC ENDP

ACT_CIC PROC NEAR
;-------------------------------------------------------------------------
;Activa / desactiva conteo de ciclos
;Entrada:        B_CICLO = 0 desactivo
;                        = 1 activo
;Salida:         B_CICLO cambiado
;-------------------------------------------------------------------------
        mov     ax,n_ciclos
        mov     cx,5
        mov     si,offset lciclo
actc1:  mov     byte ptr ds:[si]," "
        inc     si
        loop    actc1
        dec     si
        call    hexadec
        cmp     byte ptr b_ciclo,0      ;ciclos activados ?
        jne     actc2                   ;si
        mov     byte ptr b_ciclo,1      ;no, activa ciclos
        mov     bx,offset vciclo1
        jmp     actc3
actc2:  mov     byte ptr b_ciclo,0      ;desactiva ciclos
        mov     bx,offset vciclo
actc3:  mov     si,offset lciclo
        mov     ax,ds
```

```
        vnt     "a"
        ret
ACT_CIC ENDP

ACT_DIS PROC NEAR
;-------------------------------------------------------------------------
;Activa / desactiva refresco de pantalla en ejecucion corre
;Entrada:       B_DIS = 0 desactivo
;                     = 1 activo
;Salida:        B_DIS cambiado
;-------------------------------------------------------------------------
        cmp     byte ptr b_dis,0        ;display activo ?
        jne     act_d1          ;si
        mov     byte ptr b_dis,1        ;no, activa display
        mov     si,offset ldisplay      ;prepara ventana de display
        mov     byte ptr ds:[si],"0"
        inc     si
        mov     byte ptr ds:[si],"F"
        inc     si
        mov     byte ptr ds:[si],"F"
        jmp     act_d2
act_d1: mov     byte ptr b_dis,0        ;desactiva display
        mov     si,offset ldisplay      ;prepara ventana de display
        mov     byte ptr ds:[si]," "
        inc     si
        mov     byte ptr ds:[si],"0"
        inc     si
        mov     byte ptr ds:[si],"N"
act_d2: mov     bx,offset vdisplay      ;muestra ventana de display
        mov     si,offset ldisplay
        mov     ax,ds
        vnt     "a"
        ret
ACT_DIS ENDP

PRESENT PROC NEAR
;-------------------------------------------------------------------------
;Despliega pantalla de presentacion del depurador
;-------------------------------------------------------------------------
        print   18,4,hola0,normal
        print   18,5,hola1,normal
        print   18,6,hola2,normal
        print   18,7,hola3,normal
        print   18,8,hola4,normal
        print   18,9,hola5,normal
        print   18,10,hola6,normal
        print   18,11,hola7,normal
        print   18,12,hola0,normal
        print   61,12,hola9,normal
        print   20,13,hola8,normal

        print   26,16,pan_pres5,normal
        print   31,17,pan_pres6,normal
        print   33,18,pan_pres7,normal
        print   35,19,pan_pres8,normal
        print   26,16,pan_pres5,normal
```

```
        print   31,17,pan_pres6,normal
        print   33,18,pan_pres7,normal
        print   35,19,pan_pres8,normal
        ret
PRESENT ENDP

INSERTA PROC NEAR
;-------------------------------------------------------------------------
;Espera el ingreso de una instruccion por el usuario, la ensambla si no
;hay error la inserta en la direccion apuntado por PC, se sobrepone al
;codigo anterior.
;Entrada:       Instruccion ingresada
;-------------------------------------------------------------------------
        lim23
        lim24
        print   1,23,123_ins,revers
        mov     gdreset,1
ins1:   getline 1,24,ins_ens            ;pide mnemonico
        mov     gdreset,0
        cmp     gccod,4                 ;ESC?
        jne     ins2
        jmp     ins_f
ins2:   cmp     gccod,1                 ;dato valido
        jne     ins1
        mov     di,offset cod_ens       ;destino del ensamblado
        mov     si,offset ins_ens+2     ;origen del mnemonico
        call    assem
        jc      ins_er                  ;salta si hay error
        mov     si,offset cod_ens
        mov     ax,dseg
        call    longitud                ;CX = longitud de inst.
        mov     bx,cx
        push    ds
        mov     ds,segdat2
        mov     di,offset ram_emb       ;destino ram embebed
        pop     ds
        add     di,pc
        mov     si,offset cod_ens       ;fuente codigo ensamblado
        push    es
        mov     es,segdat2              ;ES = segmento de ram emb
        rep     movsb
        pop     es
        cmp     modo,1                  ;ejecucion controlada ?
        jne     ins3                    ;no
        mov     di,offset com_I         ;si
        mov     ds:[di+1],bl            ;longitud de la instruccion
        mov     si,offset cod_ens       ;SI -> codigo ensamblado
        add     di,2                    ;DI -> string de comando
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        mov     cx,bx
        rep     movsb                   ;codigo -> comando
        pop     es
        mov     si,offset com_I         ;origen del comando
        mov     cx,bx
```

```
        add     cx,2                    ;# de bytes de transmision
        call    comunica                ;transmision de comando
ins3:   call    inicial                 ;refresco de pantalla
        jmp     ins_f
ins_er: lim23                           ;instruccion invalida
        print   1,23,er_ens,revers
        getkey
ins_f:  ret
INSERTA ENDP

ENSAMBLA PROC NEAR
;-------------------------------------------------------------------------------
;Espera el ingreso de una instruccion por el usuario, la ensambla si no
;hay error, la ejecuta fuera del listado del programa del usuario, altera
;el PC solo si la instruccion es de salto
;Entrada:       Instruccion ingresada
;Salida:        PC es alterado si la instruccion es de salto
;-------------------------------------------------------------------------------
        lim23
        lim24
        print   1,23,123_ens,revers     ;presenta mensaje pidiendo
        mov     gdreset,1               ;una instruccion
ens1:   getline 1,24,ins_ens
        mov     gdreset,0
        cmp     gccod,4                 ;ESC?
        jne     ens2
        jmp     ens_f
ens2:   cmp     gccod,1                 ;string valido
        jne     ens1
        mov     di,offset cod_ens       ;DI = destino del ensamblado
        mov     si,offset ins_ens       ;SI+2 = fuente del mnemonico
        add     si,2
        call    assem                   ;ensambla instruccion
        jnc     ens21                   ;error ?
        jmp     ens_er                  ;si
ens21:  cmp     modo,1                  ;ejecucion controlada ?
        jne     ens3                    ;no
        jmp     ens4                    ;si
ens3:   mov     bl,cod_ens
        mov     cx,41                   ;instrucciones de salto
        mov     si,offset inst_pc
ens31:  lodsb
        cmp     al,bl                   ;instruccion de salto ?
        je      ens32                   ;si
        loop    ens31                   ;no, siga buscando
        push    ds                      ;fin de busqueda
        mov     ds,segdat2
        mov     di,offset ram_embf      ;destino ram embebida
        pop     ds
        push    es
        mov     es,segdat2              ;ES = ram embebida
        mov     si,offset cod_ens       ;fuente codigo ensamblado
        mov     cx,3
        rep     movsb                   ;mueve codigo
        pop     es
        push    pc
```

```
        mov     pc,8000h                ;PC = direccion donde se
        call    ejecutor                ;encuentra la instruccion
        pop     pc
        jmp     ensf
ens32:  push    ds
        mov     ds,segdat2
        mov     di,offset ram_emb
        pop     ds
        add     di,pc                   ;fuente de codigo
        mov     dx,di                   ;pc antes de ejecucion
        push    es
        mov     es,segdat2              ;ES = ram embebida
        mov     si,offset inst_s        ;destino de codigo
        mov     cx,3
ens33:  mov     al,es:[di]              ;salva codigo de PC
        mov     ds:[si],al
        inc     si
        inc     di
        loop    ens33
        mov     di,dx
        mov     si,offset cod_ens
        mov     cx,3                    ;carga instruccion ingresada
        rep     movsb                   ;en el programa
        call    ejecutor
        mov     di,dx                   ;direccion anterior de PC
        mov     si,offset inst_s        ;origen de codigo salvado
        mov     cx,3
        rep     movsb                   ;recupera codigo de PC
        pop     es
        jmp     ensf
ens4:   mov     si,offset cod_ens
        mov     ax,dseg
        call    longitud
        mov     bx,cx                   ;BX = long. instruccion
        mov     si,offset cod_ens       ;SI -> codigo ensamblado
        mov     di,offset com_S+2       ;DI -> string de comando
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        rep     movsb                   ;codigo -> string de comando
        pop     es
        mov     si,offset com_S         ;origen de comando S
        mov     ds:[si+1],bl            ;longitud de instruccion
        mov     cx,bx
        add     cx,2
        call    comunica                ;transmite comando
ensf:   call    inicial                 ;refresca pantalla
        jmp     ens_f
ens_er: lim23                           ;instruccion invalida
        print   1,23,er_ens,revers
        getkey
ens_f:  ret
ENSAMBLA ENDP

CARGA PROC NEAR
;-------------------------------------------------------------------------------
```

```
;Transmite el archivo con el programa del usuario hacia el microcontrolador
;Permite configurar la particion de memoria y asegurar el programa contra
;copia no autorizada
;Entrada:        Archivo Objeto
;-------------------------------------------------------------------------------
load0:  lim23
        lim24
        print   1,23,sw0,revers
        print   1,24,tec_sal,normal
        getkey
        call    vel4800                 ;transmision a 4800 bits/seg
        lim23
        lim24
        print   1,23,tx_file,revers
        mov     si,offset com_R         ;envia un CR
        call    tx_comand
        mov     si,offset com_C         ;envia un Control C
        call    tx_comand
        jnc     load_1
        jmp     err_1
load_1: mov     si,offset com_U         ;envia un Reset al bit
        call    tx_comand               ;de seguridad
        jnc     load1
        jmp     err_1
load1:  mov     si,offset com_F         ;llena de ceros la mem.
        call    tx_comand
        jnc     load2
        jmp     err_1
load2:  mov     si,offset com_L         ;envia comando de carga
        call    tx_comand
        mov     cx,num_byte
        push    ds
        mov     ds,segdat1
        mov     si,offset area_lect
        pop     ds
        call    com_file                ;transmite el archivo
load3:  mov     si,offset com_C         ;envia un Control C
        call    tx_comand
        lim23
        lim24
        print   1,23,mcon?,revers
mcon0:  getkey
        mov     ax,getword
        and     al,0dfh
        cmp     al,"S"                  ;Escribir registro MCON
        je      s_mcon                  ;si
        jmp     n_mcon
s_mcon: lim23
        print   1,23,ing_mcon,revers
        mov     gdreset,1
mcon1:  getdata 1,24,dasc2,"a"
        cmp     gccod,7                 ;ESC?
        jne     mcon2                   ;no
        jmp     encr?
mcon2:  cmp     gccod,0                 ;dato valido
        jne     mcon1
```

```
        mov     si,offset com_W
        mov     al,dasc2+2
        mov     ds:[si+2],al
        mov     al,dasc2+3
        mov     ds:[si+3],al
        call    tx_load
        jc      e_mcon                  ;error en la transmision
        jmp     encr?                   ;no
e_mcon: lim23
        lim24
        print   1,23,er_rx,revers       ;muestra mensaje
        print   1,24,tec_sal,normal
        getkey
        jmp     loadf                   ;sale
n_mcon: cmp     al,"N"                  ;No escribir registro MCON
        je      encr?                   ;si
        mov     ah,2                    ;beep
        mov     dl,7
        int     21h
        jmp     mcon0
encr?:  lim23
        lim24
        print   1,23,encrk,revers
encr0:  getkey
        mov     ax,getword
        and     al,0dfh
        cmp     al,"S"                  ;Encriptar
        je      s_encr                  ;si
        jmp     n_encr                  ;no
s_encr: lim23
        print   1,23,ing_encr,revers
        mov     gdreset,1
encr1:  getline 1,24,dencr
        cmp     gccod,4                 ;ESC?
        jne     encr2
        jmp     lock?
encr2:  cmp     gccod,1                 ;dato valido?
        jne     encr1
        mov     di,offset com_K+2
        mov     si,offset dencr+2
        mov     cx,14
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        rep     movsb                   ;carga palabra ingresada en com_K
        pop     es
        mov     si,offset com_K
        call    tx_load
        jc      e_encr                  ;error en la transmision
        jmp     lock?                   ;no
e_encr: lim23
        lim24
        print   1,23,er_rx,revers       ;muestra mensaje
        print   1,24,tec_sal,normal
        getkey
        jmp     loadf                   ;sale
```

```
n_encr: cmp     al,"N"                  ;No encriptar
        je      lock?                   ;si
        mov     ah,2                    ;beep
        mov     dl,7
        int     21h
        jmp     encr0
lock?:  lim23
        lim24
        print   1,23,b_lock,revers
lock0:  getkey
        mov     ax,getword
        and     al,0dfh
        cmp     al,"S"                  ;Setear bit security lock
        jne     n_lock                  ;no
        mov     si,offset com_Z
        call    tx_comand
        jmp     bien1
n_lock: cmp     al,"N"                  ;No encriptar
        je      bien1                   ;si
        mov     ah,2                    ;beep
        mov     dl,7
        int     21h
        jmp     lock0
bien1:  lim23
        lim24
        print   1,23,b_file,revers      ;bien la transmision
        print   1,24,tec_sal,normal
        getkey
        jmp     loadf
err_1:  lim23
        print   1,23,er_rx,revers
        print   1,24,reint?,normal
rein?:  getkey
        mov     ax,getword
        and     al,0dfh
        cmp     al,"S"                  ;Reintentar
        jne     n_rein                  ;no
        jmp     load0
n_rein: cmp     al,"N"                  ;No reintentar
        je      loadf                   ;si
        mov     ah,2                    ;beep
        mov     dl,7
        int     21h
        jmp     rein?
loadf:  ret
CARGA ENDP

DESCAR PROC NEAR
;-------------------------------------------------------------------------
;Mediante comunicacion serial obtiene el estatus del microcontrolador
;       PC
;       Registros CPU
;       Banderas
;       Area de Memoria 1
;       Area de Memoria 2
;       Puertos
```

```
;-------------------------------------------------------------------------------
        jmp     des1
des0:   lim24
        print   1,24,sw1,normal
        getkey
des1:   mov     si,offset com_0         ;Descarga de PC, Registros CPU
        call    tx_comand               ;banderas y puertos
        jnc     des2
        jmp     des9
des2:   mov     cksum,0                 ;check sum
        mov     si,offset buff_rx
        lodsw                           ;lee numero de bytes
        call    aschex
        add     cksum,al
        lodsw                           ;lee pch en ascii
        call    aschex
        add     cksum,al
        mov     bh,al
        lodsw                           ;lee pcl en ascii
        call    aschex
        add     cksum,al
        mov     bl,al
        mov     pc,bx                   ;carga pc en hexadecimal
        lodsw                           ;lee acc en ascii
        call    aschex
        add     cksum,al
        mov     acc,al                  ;carga acc en hexadecimal
        lodsw                           ;lee dph en ascii
        call    aschex
        add     cksum,al
        mov     dph,al                  ;carga dph en hexadecimal
        lodsw                           ;lee dpl en ascii
        call    aschex
        add     cksum,al
        mov     dpl,al                  ;carga dpl en hexadecimal
        lodsw                           ;lee b en ascii
        call    aschex
        add     cksum,al
        mov     b,al                    ;carga b en hexadecimal
        lodsw                           ;lee psw en ascii
        call    aschex
        add     cksum,al
        mov     psw,al                  ;carga psw en hexadecimal
        lodsw                           ;lee sp en ascii
        call    aschex
        add     cksum,al
        mov     regsp,al                ;carga sp en hexadecimal
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        mov     di,offset r0
        xor     ah,ah
        mov     al,psw
        and     al,18h
        add     di,ax                   ;determina banco de registros
        mov     cx,8
```

```
des21:  lodsw                           ;lee registro en ascii
        call    aschex
        add     cksum,al
        stosb                           ;carga registro en hexadecimal
        loop    des21
        pop     es
        lodsw                           ;lee mcon en ascii
        call    aschex
        add     cksum,al
        mov     mcon,al                 ;carga mcon en hexadecimal
        lodsw                           ;lee p0 en ascii
        call    aschex
        add     cksum,al
        mov     p0,al                   ;carga p0 en hexadecimal
        lodsw                           ;lee p1 en ascii
        call    aschex
        add     cksum,al
        mov     p1,al                   ;carga p1 en hexadecimal
        lodsw                           ;carga p2 en ascii
        call    aschex
        add     cksum,al
        mov     p2,al                   ;carga p2 en hexadecimal
        lodsw                           ;lee p3 en ascii
        call    aschex
        add     cksum,al
        mov     p3,al                   ;carga p3 en hexadecimal
        neg     cksum                   ;calculo de check sum
        lodsw                           ;lee check sum
        call    aschex
        cmp     al,cksum                ;check sum coincide ?
        je      des3                    ;si, siga
        mov     ah,02
        mov     dl,7
        int     21h
        jmp     des1                    ;no, vuelva a transmitir
des3:   cmp     mlin_ex,0
        je      des4
        jmp     des5
des4:   mov     si,offset com_M         ;Descarga de memoria interna 1
        mov     cx,2
        mov     ax,dirmlin
        mov     ds:[si+1],al
        call    comunica
        jnc     des41
        jmp     des9
des41:  mov     si,offset buff_rx
        mov     cksum,0                 ;inicializacion de check sum
        lodsw                           ;lee numero de bytes
        call    aschex
        add     cksum,al
        mov     cx,32
        mov     ax,dirmlin
        xor     ah,ah
        push    es
        push    ds
        pop     es
```

```
des42:  and     al,7fh
        mov     di,offset r0
        add     di,ax
        push    ax
        lodsw                           ;lee byte en ascii
        call    aschex
        add     cksum,al
        stosb                           ;carga byte en hexadecimal
        pop     ax
        inc     al
        loop    des42
        pop     es
        neg     cksum                   ;calculo de check sum
        lodsw                           ;lee check sum
        call    aschex
        cmp     al,cksum                ;check sum coincide ?
        je      des43                   ;si, siga
        mov     ah,02
        mov     dl,7
        int     21h
        jmp     des4                    ;no, vuelva a transmitir
des43:  jmp     des6
des5:   mov     si,offset com_N         ;Descarga de memoria embebida 1
        mov     cx,3
        mov     ax,dirm1ex
        mov     ds:[si+1],ah
        mov     ds:[si+2],al
        call    comunica
        jnc     des51
        jmp     des9
des51:  mov     si,offset buff_rx
        mov     cksum,0                 ;inicializacion de check sum
        lodsw
        call    aschex
        add     cksum,al
        mov     cx,32
        mov     ax,dirm1ex
        push    es
        mov     es,segdat2
des52:  and     ax,7fffh
        mov     di,offset ram_emb
        add     di,ax
        push    ax
        lodsw                           ;lee byte en ascii
        call    aschex
        add     cksum,al
        stosb                           ;carga byte en hexadecimal
        pop     ax
        inc     ax
        loop    des52
        pop     es
        neg     cksum                   ;calcu1o de check sum
        lodsw                           ;lee check sum
        call    aschex
        cmp     al,cksum                ;check sum coincide ?
        je      des6                    ;si, siga
```

```
        mov     ah,02
        mov     dl,7
        int     21h
        jmp     des5                    ;no, vuelva a transmitir
des6:   cmp     m2in_ex,0
        je      des7
        jmp     des8
des7:   mov     si,offset com_M         ;Descarga de memoria interna 1
        mov     cx,2
        mov     ax,dirm2in
        mov     ds:[si+1],al
        call    comunica
        jnc     des71
        jmp     des9
des71:  mov     si,offset buff_rx
        mov     cksum,0                 ;inicializacion de check sum
        lodsw
        call    aschex
        add     cksum,al
        mov     cx,32
        mov     ax,dirm2in
        xor     ah,ah
        push    es
        push    ds
        pop     es
des72:  and     al,7fh
        mov     di,offset r0
        add     di,ax
        push    ax
        lodsw                           ;lee byte en ascii
        call    aschex
        add     cksum,al
        stosb                           ;carga byte en hexadecimal
        pop     ax
        inc     al
        loop    des72
        pop     es
        neg     cksum                   ;calculo de check sum
        lodsw                           ;lee check sum
        call    aschex
        cmp     al,cksum                ;check sum coincide ?
        je      des73                   ;si, siga
        mov     ah,02
        mov     dl,7
        int     21h
        jmp     des7                    ;no, vuelve a transmitir
des73:  jmp     desf1
des8:   mov     si,offset com_M         ;Descarga de memoria embebida 2
        mov     cx,3
        mov     ax,dirm2ex
        mov     ds:[si+1],ah
        mov     ds:[si+2],al
        call    comunica
        jc      des9
        mov     si,offset buff_rx
        mov     cksum,0                 ;inicializa check sum
```

```
        lodsw
        call    aschex
        add     cksum,al
        mov     cx,32
        mov     ax,dirm2ex
        push    es
        mov     es,segdat2
des81:  and     ax,7fffh
        mov     di,offset ram_emb
        add     di,ax
        push    ax
        lodsw                           ;lee byte en ascii
        call    aschex
        add     cksum,al
        stosb                           ;carga byte en hexadecimal
        pop     ax
        inc     ax
        loop    des81
        pop     es
        neg     cksum                   ;calculo de check sum
        lodsw                           ;lee check sum
        call    aschex
        cmp     al,cksum                ;check sum coincide ?
        je      des82                   ;si, siga
        int     21h
        mov     ah,02
        mov     dl,7
        int     21h
        jmp     des8                    ;no, vuelve a transmitir
des82:  jmp     desf1
des9:   lim24
des91:  print   1,24,er_tx_c,normal
        getkey
        mov     ax,getword
        and     al,0dfh
        cmp     al,"S"                  ;Reintentar
        jne     des92                   ;no
        jmp     des0
des92:  cmp     al,"N"                  ;No reintentar
        je      desf                    ;si
        mov     ah,2
        mov     dl,7
        int     21h
        jmp     des91
desf:   stc
        jmp     desf2
desf1:  clc
desf2:  ret
DESCAR ENDP

ASCHEX PROC NEAR
;-------------------------------------------------------------------------
;Convierte  el ascii de AX en hexadecimal en al
;Entrada:        AH = menos signicativo
;                AL = mas significativo
;-------------------------------------------------------------------------
```

```
        push    cx
        cmp     al,"9"
        ja      asc1
        sub     al,"0"
        jmp     asc2
asc1:   sub     al,"A"
        add     al,10
asc2:   mov     cl,4                    ;desplazamiento a la
        shl     al,cl                   ;izquierda 4 veces
        cmp     ah,"9"
        ja      asc3
        sub     ah,"0"
        jmp     asc4
asc3:   sub     ah,"A"
        add     ah,10
asc4:   or      al,ah
        pop     cx
        ret
ASCHEX ENDP

RESET PROC NEAR
;-------------------------------------------------------------------------
;Resetea la CPU
;-------------------------------------------------------------------------
        cmp     modo,1                  ;ejecucion controlada ?
        jne     rest1                   ;no
        jmp     rest2                   ;si
rest1:  mov     pc,0
        mov     cx,0ffh
        mov     di,offset r0
        mov     al,0
        push    es
        push    ds
        pop     es                      ;ES = segmento de datos
        rep     stosb                   ;llena con 0 area interna
        pop     es
        mov     p0,0ffh
        mov     p1,0ffh
        mov     p2,0ffh
        mov     p3,0ffh
        mov     p0_pin,0ffh
        mov     p1_pin,0ffh
        mov     p2_pin,0ffh
        mov     p3_pin,0ffh
        mov     mcon,0f8h
        mov     ta,55h
        mov     sbuf_in,0
rest2:  call    inicial                 ;refresca pantalla
        ret
RESET ENDP

HELP PROC NEAR
;-------------------------------------------------------------------------
;Presenta ventana con arbol de opciones del depurador
;-------------------------------------------------------------------------
        guarpan
```

```
        mov     ax,0720h
        mov     cx,25*80                ;limpia toda la pantalla
        mov     di,0
        rep     stosw
        mov     bx,offset vhelp1
        mov     ax,ds
        vnt     "a"
        mov     bx,offset vhelp2
        mov     ax,ds
        vnt     "a"
        mov     bx,offset vhelp3
        mov     ax,ds
        vnt     "a"
        getkey
        traepan
        ret
HELP ENDP

ATRAS PROC NEAR
;-------------------------------------------------------------------------------
;Permite ejecutar reversamente una instruccion
;-------------------------------------------------------------------------------
        cmp     modo,0                  ;simulacion ?
        je      atr1                    ;si
        jmp     atrf
atr1:   call    reversa
        jc      atr2                    ;error?
        call    inicial                 ;no
        jmp     atrf
atr2:   lim23                           ;mensaje de error
        print   1,23,er_rev,revers
        mov     ah,2                    ;beep
        mov     dl,7
        int     21h
        getkey
atrf:   ret
ATRAS ENDP

SEGMENT1 PROC NEAR
;-------------------------------------------------------------------------------
;Determina el valor del segmento DSEG1
;-------------------------------------------------------------------------------
        ASSUME DS:DSEG1
        mov     ax,dseg1
        ret
SEGMENT1 ENDP

SEGMENT2 PROC NEAR
;-------------------------------------------------------------------------------
;Determina el valor del segmento DSEG2
;-------------------------------------------------------------------------------
        ASSUME DS:DSEG2
        mov     ax,dseg2
        ret
SEGMENT2 ENDP
```

```
;*************************************
;*********     RUTVNT.ASM     **********
;*************************************
;**  Rutinas para manejo de ventanas  **
;*************************************
;-------------------------------------------------------------------------
;Macro    Parametros                    Funcion
;-------------------------------------------------------------------------
;hexasc                                 hex en AL ==> ascii en BX
;idvnt    dirasc,dirres,longres         identifica archivo y area de ventanas
;popall                                 pop a todo
;posdir   x,y                           calcula direccion en memoria de video
;pushall                                push a todo
;vnt      nombre,tipo                   muestra o actualiza una ventana
;sbrvnt                                 subrutinas de los otros macros
;-------------------------------------------------------------------------

;******   HEXASC   ***************
;El byte en AL se convierte en dos bytes ASCII en BX.
hexasc    macro
          call     rha
          endm

;******   IDVNT    ***************
;Este macro recibe el nombre del archivo de ventanas en un ASCIIZ que
;inicia en DIRASC. Abre el archivo, trae todos los grupos principales
;al area que empieza en DIRRES y guarda DIRRES y LONGRES en el indice.
;En AX salen codigos de error: 00 todo bien, 01 error al abrir archivo,
;02 si se encuentra que no es un archivo de ventanas, y
;03 error por falta de espacio (LONGRES= 6 grupos para indice + # de
;grupos principales + 1 grupo para una tabla de residencia actual + al
;menos 4 grupos para secundarios, a menos que todas las ventanas sean
;principales). Nota: Un grupo=256 bytes.
idvnt     macro    dirasc,dirres,longres
          push     dx
          push     di
          push     bx
          mov      dx,offset dirasc
          mov      di,offset dirres
          mov      bl,longres
          call     riv
          pop      bx
          pop      di
          pop      dx
          endm

;******   POPALL   ***************
popall    macro
          pop      ax
          pop      bx
          pop      cx
          pop      dx
          pop      si
          pop      di
          pop      bp
          pop      ds
          pop      es
          endm
```

```
;******   POSDIR   ***************
;Convierte la posicion (X,Y) en una direccion en
;PDDIR para acceso directo a la memoria de video.
posdir     macro     x,y
           push      cx
           mov       cl,x
           mov       ch,y
           call      rpd
           pop       cx
           endm

;******   PUSHALL  ***************
pushall    macro
           push      es
           push      ds
           push      bp
           push      di
           push      si
           push      dx
           push      cx
           push      bx
           push      ax
           endm

;******   VNT      ***************
;BX apunta al nombre de la ventana. Si existe y es principal, sus
;grupos en el area de residentes son los mismos que tenia en el
;archivo. Si es secundaria ve si es residente, caso en que sus grupos
;estan en la entrada en el indice. Si no es residente, se la trae al
;area de residentes y se la marca como tal, borrando la residencia
;de las ventanas sobre las que esta ventana se coloque. Muestra la
;ventana y donde se encuentren variables se las leen del area apuntada
;por AX:SI. Existen dos TIPOs de llenado: 'h', hexadecimal, c/byte
;se muestra en dos variables de la ventana y 'a', ASCII, c/byte se
;muestra como tal en pantalla.
;En AX sale un codigo de error: 0, todo bien; 1, nombre no existe.
vnt        macro     tipo
           mov       tipvnt,tipo
           call      rvnt
           endm

;******   SBRVNT   ***************
sbrvnt     macro

DSEG SEGMENT PUBLIC
tabha      db        '0','1','2','3','4','5','6','7','8','9'
           db        'A','B','C','D','E','F'
dirind     dw        0                    ;direccion del indice
pddir      dw        0
segvar     dw        0                    ;segmento donde estan las variables
offvar     dw        0                    ;offset de la variable actual
tipvnt     db        0                    ;tipo: 'h', 'a' o 't'
dirent     dw        0                    ;direccion de entrada de vent. actual
numgrp     db        0                    ;# de grupos que faltan por traer
columna    db        0                    ;columna escrita actualmente
linea      db        0                    ;linea escrita actualmente
```

```
numbyte   dw      0                   ;# de byte mostrado del grupo actual

DSEG ENDS

rha:      push    ax
          push    cx
          push    dx
          mov     bx,offset tabha
          mov     dl,al
          and     al,0f0h             ;nibble mas significativo
          mov     ah,0
          mov     cl,i6               ;lo paso a la parte menos signif.
          div     cl
          xlat                        ;traduzco a ASCII
          mov     ch,al
          mov     al,dl
          and     al,0fh              ;nibble menos significativo
          xlat                        ;Traduzco a ASCII
          mov     cl,al
          mov     bx,cx
          pop     dx
          pop     cx
          pop     ax
          ret

riv:      push    si
          push    cx
          mov     si,bx
          mov     ah,3dh              ;abro archivo
          mov     al,0
          int     21h
          jnc     riv1                ;error.
          mov     ax,1
          jmp     rivf
riv1:     mov     bx,ax               ;handle
          mov     ah,3fh              ;leo entradas 0, 1 y 2.
          mov     dx,di
          mov     cx,72
          int     21h
          mov     [di+13],bx          ;guardo handle
          mov     bx,si
          mov     [di+12],bl          ;guardo LONGRES
          mov     [di+10],di          ;guardo DIRRES
          mov     dirind,di
          mov     bx,24               ;chequeo paridad del archivo
          mov     cl,8
          mov     al,0
riv11:    add     al,[di+bx]
          inc     bx
          dec     cl
          jnz     riv11
          cmp     al,[di+bx]
          je      riv12
          mov     ax,2                ;paridad de ventanas no se cumple
          jmp     rivf
riv12:    mov     bx,36
```

```
        mov     cl,32
        mov     al,0
riv13:  add     al,[di+bx]
        inc     bx
        dec     cl
        jnz     riv13
        cmp     al,[di+bx]
        je      riv2
        mov     ax,2                ;paridad de grupos no se cumple
        jmp     rivf
riv2:   mov     al,[di+9]           ;# de grupos principales
        cmp     al,[di+1]           ;= total de grupos?
        je      riv21
        add     al,4                ;+4 grupos minimo de secundarias
riv21:  add     al,6                ;+6 de indice
        inc     al                  ;+1 para tabla de residencia
        cmp     al,[di+12]
        jbe     riv3
        mov     ax,3                ;muy poco espacio
        jmp     rivf
riv3:   mov     dx,di               ;todo bien, leo resto del indice
        add     dx,72
        mov     cx,1536-72
        mov     bx,[di+13]          ;handle
        mov     ah,3fh
        int     21h
riv4:   mov     dx,di               ;leo todos los principales
        add     dx,1536
        mov     ch,[di+9]           ;CX=# de grupos principales*256
        mov     cl,0                ;= # de bytes por leer.
        mov     ah,3fh
        int     21h
        mov     ch,[di+9]           ;preparo numero de proximo grupo
        add     ch,6                ;a usarse de area de residentes
        mov     [di+15],ch
        mov     ch,[di+8]           ;copio los # de grupos en el archivo
        add     di,48               ;a los grupos en memoria para
        mov     bp,di               ;las CH ventanas principales
riv5:   add     bp,24               ;y las marco como residentes
        mov     di,bp
        mov     byte ptr[di+8],3
        mov     bx,[di+16]
        mov     [di+20],bx
        mov     bx,[di+18]
        mov     [di+22],bx
        dec     ch
        jnz     riv5
        mov     ax,0
rivf:   pop     cx
        pop     si
        ret

rpd:    push    ax
        push    cx
        mov     al,160              ;80*2 bytes por linea
        mul     ch
```

```
          add       cl,cl               ;+ 2 veces el # de columna
          mov       ch,0
          add       ax,cx
          mov       pddir,ax
          pop       cx
          pop       ax
          ret


rvnt:     pushall
          mov       segvar,ax
          mov       offvar,si
          mov       di,dirind           ;direccion del indice
          mov       dl,[di+2]           ;ultima ventana valida
          add       di,48               ;voy a buscar el nombre en las entradas
          push      es                  ;del indice.
          push      ds
          pop       es
          mov       dh,2
          mov       bp,di
rvnt1:    add       bp,24
          mov       di,bp               ;ubico entrada indicada por
          inc       dh                  ;DH
          mov       si,bx
          mov       cx,9                ;nombre de 8 bytes
rvnt10:   mov       ah,es:[di]          ;leo 1 byte del nombre de una ventana
          inc       di
          lodsb                         ;leo 1 byte del nombre buscado
          dec       cx
          jz        rvnt11              ;nombres iguales?
          cmp       al,ah
          jne       rvn10               ;nombres diferentes
          cmp       al,0                ;iguales?
          je        rvnt11
          jmp       rvnt10
rvn10:    cmp       dh,dl               ;busco en todas las ventanas?
          jbe       rvnt1               ;no, sigo
          pop       es                  ;si, nombre no existe,
          mov       numbyte,1           ;error = 1
          jmp       rvntf
rvnt11:   pop       es
          mov       di,dirind
          mov       [di+6],dh           ;guardo # ventana actual
          mov       al,24
          mul       dh                  ;ubico entrada actual y
          add       di,ax               ;DS:DI apunta a ventana actual
          mov       dirent,di
          mov       dh,[di+8]           ;veo si es P o S.
          cmp       dh,3
          jne       rvnt2               ;S.
          jmp       rvnt4               ;P.
rvnt2:    cmp       dh,2                ;secundaria. Reside?
          jne       rvnt20
          jmp       rvnt4               ;si.
rvnt20:   mov       dl,[di+9]           ;no, ubicar la ventana.
          mov       byte ptr[di+8],2    ;doy residencia a la ventana
```

```
            mov     si,dirind           ;DS:SI apunta al indice
            mov     ah,[si+12]          ;ubico tabla de residencia
            dec     ah                  ;AX tiene el offset de la tabla
            mov     al,0                ;respecto del inicio del indice
            mov     bp,si
            add     bp,ax               ;DS:BP apunta a la tabla
            mov     cl,[si+15]          ;CL = # grupo que va a ser dado
            mov     bx,16               ;a la ventana actual
rvnt21:     cmp     byte ptr[di+bx],0   ;termino ventana?
            je      rvnt3               ;si.
            mov     [di+bx+4],cl
            push    bp                  ;no, leer valor en tabla
            add     bp,cx
            mov     al,ds:[bp]
            cmp     al,[si+6]           ;No quitarse la residencia
            je      rvnt22              ;a si misma.
            cmp     al,0                ;grupo estaba libre?
            je      rvnt22              ;si.
            push    si                  ;no, quito residencia a la ventana
            mov     ah,24
            mul     ah
            add     si,ax
            mov     byte ptr[si+8],0
            pop     si
rvnt22:     mov     al,[si+6]           ;ventana actual
            mov     ds:[bp],al          ;a la tabla de residencia
            pop     bp
            inc     cl                  ;preparo proximo grupo a usarse
            mov     ah,[si+12]
            sub     ah,2
            cmp     cl,ah
            jbe     rvnt23
            mov     ah,[si+9]           ;termino area de residentes
            add     ah,6                ;==> voy a inicio de area de
            mov     cl,ah               ;secundarias
rvnt23:     mov     [si+15],cl
            inc     bl                  ;ya reviso cuatro grupos?
            cmp     bl,20
            jne     rvnt21              ;no, volver.
rvnt3:      mov     bx,16
rvnt31:     mov     dh,[di+bx]          ;leo # de grupo
            cmp     dh,0                ;termino?
            je      rvnt4               ;si.
            mov     dl,0                ;no. CX:DX=offset desde
            mov     cx,0                ;inicio de archivo
            mov     al,0
            mov     ah,42h              ;LSEEK
            push    bx
            mov     bx,[si+13]
            int     21h
            pop     bx
            mov     dh,[di+bx+4]        ;DS:DX apunta a buffer de lectura
            mov     dl,0                ;Leo un grupo del disco
            add     dx,si
            mov     cx,256
            mov     ah,3fh              ;READ
```

```
          push    bx
          mov     bx,[si+13]
          int     21h
          pop     bx
          inc     bl
          cmp     bl,20               ;ya leyo 4 grupos?
          jne     rvnt31              ;no, sigo leyendo.
rvnt4:    mov     ax,[di+12]          ;esq. sup. izq.
          mov     linea,ah
          mov     columna,al
          mov     al,[di+9]           ;# de grupos
          mov     numgrp,al
          mov     numbyte,0           ;# de byte actual
          mov     si,dirind
          mov     ah,[di+20]          ;AX posicion primer grupo
          mov     bp,di
          mov     al,0
          add     si,ax               ;SI apunta al grupo
          posdir  columna,linea
          mov     di,pddir
rvnt41:   lodsw
          cmp     al,1                ;variable?
          je      rvnt42              ;si.
          stosw                       ;no, mostrar caracter.
          jmp     rvnt44
rvnt42:   push    ds
          mov     bx,offvar           ;leo una variable
          mov     ds,segvar
          mov     al,[bx]
          pop     ds
          inc     offvar
          cmp     tipvnt,'h'          ;ventana hexadecimal?
          je      rvnt43              ;si.
          stosw                       ;no, ventana ascii.
          jmp     rvnt44
rvnt43:   hexasc                      ;AL se convierte en ascii en BX
          mov     al,bh
          stosw                       ;muestro mitad mas significativa.
          lodsw                       ;Leo atributo siguiente.Supongo
          mov     al,bl               ;que AL=1 (otra variable) y
          stosw                       ;muestro la otra mitad.
          inc     columna
          mov     dx,si               ;Con DX vere si hubo cambio de grupo.
          call    rvnt0               ;ve si grupo actual ya termino.
          cmp     dx,si               ;Hubo cambio de grupo?
          je      rvn431              ;no, SI apunta correctamente.
          add     si,2                ;si, hago que SI apunte bien.
rvn431:   cmp     al,1                ;termino la ventana?
          je      rvnt5               ;si, salir.
rvnt44:   call    rvnt0               ;no, seguir.
          cmp     al,1                ;termino ventana?
          je      rvnt5               ;si.
          inc     columna
          mov     al,ds:[bp+14]       ;termino linea?
          cmp     al,columna
          jae     rvnt41              ;no, continuar.
```

```
          inc     linea               ;si, nueva linea.
          mov     al,ds:[bp+12]       ;primera columna.
          mov     columna,al
          posdir  columna,linea
          mov     di,pddir
          mov     al,ds:[bp+15]       ;termino ventana?
          cmp     al,linea
          jb      rvnt5
          jmp     rvnt41              ;no, continuar.
rvnt5:    mov     numbyte,0           ;si, salir.
rvntf:    popall
          mov     ax,numbyte
          ret
rvnt0:    push    bp                  ;varia contadores y ve si es necesario
          add     numbyte,2           ;empezar otro grupo. Altera AX.
          cmp     numbyte,100h        ;termino un grupo?
          jb      rvnt02              ;no.
          dec     numgrp              ;si, un grupo menos por mostrar
          cmp     numgrp,0  ;termino?
          jne     rvnt01              ;no.
          mov     al,1                ;si, salgo con AL=1.
          jmp     rvnt03
rvnt01:   mov     ah,numgrp           ;leo un # de grupo
          mov     al,ds:[bp+9]
          sub     al,ah
          mov     ah,0
          add     bp,ax
          mov     ah,ds:[bp+20]       ;AX=posicion de nuevo grupo
          mov     al,0
          mov     numbyte,0
          mov     si,dirind
          add     si,ax               ;SI apunta a nuevo grupo
rvnt02:   mov     al,0                ;AL=0 ==> no termina aun
rvnt03:   pop     bp
          ret

          endm
```

```
;*******************************
;***       MACPRO.ASM        ***
;*******************************
;-------------------------------------------------------------------------
;Macro    Parametros                    Funcion
;-------------------------------------------------------------------------
;getchar                                lee un ASCII
;getdata x,y,buff,tipo                  lee un bin, hex o dec
;getkey                                 recibe dato del teclado (BIOS)
;getline buff                           lee un string del teclado
;guardreg                               guarda un # hex en memoria como ascii
;guarpan                                guarda pantalla
;lim23                                  limpia la linea 23
;lim24                                  limpia la linea 24
;print    x,y,dir,atrib                 pone un texto en pantalla
;rtodo                                  pop todo
;stodo                                  push todo
;sbrpro                                 subrutinas de los otros macros
;traepan                                trae pantalla
;-------------------------------------------------------------------------

;******   GETCHAR  ***************
;Lee en GCASCII un ASCII del teclado o buffer de macros. En GCCOD
;sale el siguiente codigo: 0, hay dato valido en GCASCII. 1, ENTER. 2,
;TAB. 3, SHIFT TAB. 4, ESC. 5, alguna funcion o tecla especial, cuyo
;ASCII extendido sale en GCASCII. Para que ENTER, TAB, SHIFT TAB y
;ESC sean aceptados como datos normales, se debe presionar antes F10.
getchar   macro
          call     rgch
          endm

;******   GETDATA  ***************
;Lee datos del teclado o buffer de macros en BUFF. En BUFF se tiene:
;# de bytes pedidos, # de datos leidos, datos, # convertido en hex
;(4 bytes). Hace eco en pantalla en (X,Y). De acuerdo al TIPO
;recibe datos en bin ('b'), hex ('h'), dec ('d'), La longitud de
;BUFF debe ser adecuada: bin, cualquiera; hex, 2 0 4 bytes; dec, 3
;o 5 bytes. Si se presiona DEL o BS se borra el ultimo dato ingresado.
;ENTER es esperado para confirmar un dato y convertirlo en hex. En
;GCCOD sale un codigo si ocurre lo siguiente: 0, hay un dato correcto al
;final de BUFF. 1, flecha arriba. 2, flecha abajo. 3, flecha derecha.
;4, flecha izquierda. 5, TAB. 6, SHIFT TAB. 7, ESC. 8, PgUp. 9, PgDn.
;Cualquier otro dato es procesado para ver si es valido incluirlo en el
;buffer. Se produce un BEEP si un dato es invalido o al llegar a los
;limites. Si al ingreso a la subrutina GDRESET es 1, buffer y variables
;son inicializadas. Si es 0, no lo son, y se puede seguir recibiendo
;el mismo dato. Los numeros binarios no son convertidos en hex.
getdata   macro    x,y,buff,tipo
          push     bx
          push     cx
          push     dx
          mov      bx,offset buff
          mov      cl,x
          mov      ch,y
          mov      dl,tipo
          call     rgd
          pop      dx
          pop      cx
```

```
          pop       bx
          endm

;******   GETKEY   ****************
;Espera hasta recibir un dato del teclado y lo guarda en GETWORD
getkey    macro
          call      rgk
          mov       ax,getword
          endm

;******   GETLINE  ****************
;Lee del teclado un buffer BUFF (ASCII), que tiene este formato: Numero
;de bytes pedidos, numero de bytes realmente ingresados, texto. Para
;edicion se acepta Backspace o Delete. ENTER para terminar el texto.
;En GCCOD sale un codigo: 1, se digito ENTER (buffer valido). 2, TAB.
;3, SHIFT TAB. 4, ESC y 5, especial. Este macro usa GETCHAR para
;recibir datos. En  X, Y recibe la posicion en que debe hacer eco.
getline   macro     x,y,buff
          push      si
          push      cx
          mov       cl,x
          mov       ch,y
          mov       si,offset buff
          call      rgl
          pop       cx
          pop       si
          endm

;******   GUARDREG         ****************
;El # hexadecimal en al es convertido a ascii y guardado en la localidad
;apuntada por di
guardreg  macro
          call      rgr
          endm

;******    GUARPAN ****************
guarpan   macro
          call      rgp
          endm

;****** LIM23       ****************
lim23     macro
          call      rl23
          endm

;****** LIM24       ****************
lim24     macro
          call      rl24
          endm

;******   PRINT    ****************
;Escribe el texto apuntado por DIR en la pantalla a partir de
;(X,Y) con atributo ATRIB. El texto debe terminar con un 0.
print     macro     x,y,dir,atrib
          posdir    x,y
          push      si
```

```
          push      ax
          mov       si,offset dir
          mov       ah,atrib
          call      rprt
          pop       ax
          pop       si
          endm

;******   RTODO     ***************
rtodo     macro
          pop       ax
          pop       bx
          pop       cx
          pop       dx
          pop       si
          pop       di
          pop       bp
          pop       ds
          pop       es
          endm

;******   STODO     ***************
stodo     macro
          push      es
          push      ds
          push      bp
          push      di
          push      si
          push      dx
          push      cx
          push      bx
          push      ax
          endm

;*******TRAEPAN     **********
traepan   macro
        call    rtp
          endm

;******   SBRPRO    ***************
sbrpro    macro

DSEG SEGMENT PUBLIC
gccod     db        0                   ;codigo de tecla especial pulsada
gdreset   db        1                   ;GETDATA inicializa variables?
gdtab     db        '0123456789ABCDEFabcdef'
gcascii   db        0                   ;resultado de getchar
rglx      db        0                   ;x,y para getline
rgly      db        0
DSEG ENDS

rgch:     push      ax
          push      dx
rgch1:    mov       dl,1
          getkey                        ;leo dato de teclado o buffer
rgch11:   mov       ax,getword
```

```
          cmp     al,0              ;es tecla especial?
          je      rgch3             ;si.
rgch2:    cmp     al,0dh            ;ENTER?
          je      rgchf
          inc     dl
          cmp     al,9              ;TAB?
          je      rgchf
          inc     dl
          inc     dl
          cmp     al,1bh            ;ESC?
          je      rgchf
          mov     dl,0              ;es un ASCII valido.
          mov     dh,al
          jmp     rgchf
rgch3:    mov     dl,5
          mov     dh,ah
          cmp     ah,68             ;F10?
          je      rgch31            ;si.
          cmp     ah,15             ;SHIFT TAB?
          jne     rgchf
          mov     dl,3
          jmp     rgchf
rgch31:   mov     dl,0
          getkey
rgch32:   mov     ax,getword
          mov     dh,al
          cmp     al,1bh            ;ESC?
          je      rgchf
          cmp     al,0dh            ;ENTER?
          je      rgchf
          cmp     al,9              ;TAB?
          je      rgchf
          jmp     rgch1
rgchf:    mov     gccod,dl
          mov     gcascii,dh
          pop     dx
          pop     ax
          ret

rgd:      stodo
          cmp     gdreset,1         ;inicializo variables?
          jne     rgd2              ;no.
          mov     dh,[bx]           ;# de datos pedidos.
          mov     al,0
          mov     [bx+1],al
          mov     si,2
          mov     al,'0'
rgd1:     mov     [si+bx],al        ;inicializo buffer
          inc     si
          dec     dh
          jnz     rgd1
          push    cx
          posdir  cl,ch
          mov     di,pddir
          mov     cl,[bx]
          mov     ch,0
```

```
        mov     al,0b1h         ;primero borro
rgd_1:  stosb
        inc     di
        loop    rgd_1
        pop     cx
rgd2:   getkey
        mov     gccod,0
rgd21:  mov     ax,getword
        cmp     al,0            ;tecla especial?
        jne     rgd4
rgd3:   mov     gccod,1
        cmp     ah,72           ;flecha arriba?
        je      rgd3f
        inc     gccod
        cmp     ah,80           ;flecha abajo?
        je      rgd3f
        inc     gccod
        cmp     ah,77           ;->?
        je      rgd3f
        inc     gccod
        cmp     ah,75           ;<-?
        je      rgd3f
        add     gccod,2
        cmp     ah,15           ;SHIFT TAB?
        je      rgd3f
        add     gccod,2
        cmp     ah,73           ;PGUP?
        je      rgd3f
        inc     gccod
        cmp     ah,81           ;PGDN?
        je      rgd3f
rgd3_f: cmp     ah,83           ;DEL?
        jne     rgd2
        jmp     rgdd
rgd3f:  jmp     rgdf
rgd4:   cmp     al,8            ;BS?
        jne     rgd411
        jmp     rgdd
rgd411: cmp     al,0dh          ;ENTER?
        jne     rgd412
        jmp     rgde
rgd412: cmp     al,9            ;TAB?
        jne     rgd413
        mov     gccod,5
        jmp     rgdf
rgd413: cmp     al,1bh          ;ESC?
        jne     rgd42
        mov     gccod,7
        jmp     rgdf
rgd42:  cmp     dl,'b'          ;binario?
        jne     rgd432
rgd43:  cmp     al,'0'          ;es '0' o '1'?
        je      rgd431
        cmp     al,'1'
        je      rgd431
        jmp     rgdb            ;dato no valido, beep.
```

```
rgd431:  jmp     rgd46
rgd432:  cmp     dl,'a'              ;asc ii?
         je      rgd431
rgd44:   cmp     dl,'h'              ;hex?
         jne     rgd45
         mov     dh,0                ;veo si dato es correcto.
         mov     si,offset gdtab
rgd441:  cmp     al,[si]             ;consta en la tabla?
         je      rgd442
         inc     si
         inc     dh
         cmp     dh,22               ;acabo la tabla?
         jne     rgd441
         jmp     rgdb                ;no fue valido.
rgd442:  jmp     rgd46               ;'abcdef' = 'ABCDEF'
rgd45:   mov     dh,0                ;decimal. Veo si dato es correcto.
         mov     si,offset gdtab
rgd451:  cmp     al,[si]             ;consta en la tabla?
         je      rgd452
         inc     si
         inc     dh
         cmp     dh,10               ;acabo la tabla?
         jne     rgd451
         jmp     rgdb                ;no fue valido.
rgd452:  mov     al,[bx]
         cmp     al,[bx+1]           ;buffer lleno?
         jne     rgd4521
         jmp     rgdb
rgd4521: push    dx
         push    cx
         push    dx
         mov     si,bx
         add     si,3
         cmp     byte ptr[bx],3      ;3 bytes (0-255)?
         je      rgd455
rgd453:  mov     ax,10000
         mov     cl,[si]
         mov     ch,0
         sub     cl,30h
         mul     cx
         cmp     dx,0                ;<FFFF?
         je      rgd454
         jmp     rgd45o              ;overflow, beep.
rgd454:  mov     di,ax
         inc     si
         mov     ax,1000
         mov     cl,[si]
         sub     cl,30h
         mul     cx
         add     di,ax
         jnc     rgd455
         jmp     rgd45o              ;overflow
rgd455:  inc     si
         mov     ax,100
         mov     cl,[si]
         sub     cl,30h
```

```
            mul     cx
            add     di,ax
            jnc     rgd4551
            jmp     rgd45o              ;overflow
rgd4551:    cmp     byte ptr[bx],3
            jne     rgd456
            cmp     ah,0
            je      rgd456
            jmp     rgd45o              ;overflow
rgd456:     inc     si
            mov     ax,10
            mov     cl,[si]
            sub     cl,30h
            mul     cx
            add     di,ax
            jnc     rgd4561
            jmp     rgd45o              ;overflow
rgd4561:    cmp     byte ptr[bx],3
            jne     rgd457
            cmp     ah,0
            je      rgd457
            jmp     rgd45o              ;overflow
rgd457:     pop     dx
            mov     cl,dh
            push    dx
            add     di,cx
            jnc     rgd4571
            jmp     rgd45o              ;overflow
rgd4571:    cmp     byte ptr[bx],3
            jne     rgd45f
            cmp     ah,0
            je      rgd45f
rgd45o:     pop     dx                  ;overflow
            pop     cx
            pop     dx
            jmp     rgdb
rgd45f:     inc     si
            mov     [si],di
            pop     dx
            pop     cx
            pop     dx
rgd46:      mov     ax,getword
            mov     dh,al
            call    rgd0                ;DH=#, BX-->inicio buffer
            cmp     al,1                ;exceso de datos?
            jne     rgd5
rgdb:       push    dx                  ;BEEP
            mov     ah,2
            mov     dl,7
            int     21h
            pop     dx
            jmp     rgd2
rgd5:       call    rgd01
            jmp     rgd2
rgdd:       cmp     byte ptr[bx+1],0    ;DELETE. Buffer vacio?
            je      rgdb
```

```
          call      rgd02
          call      rgd01
          jmp       rgd2
rgde:     cmp       dl,'h'              ;hex?
          je        rgd6                ;a los binarios no los transformo y
          jmp       rgdf                ;los dec. llegan ya transformados.
rgd6:     mov       si,bx               ;convierto en hex.
          mov       dl,[bx]
          mov       dh,0
          add       si,2
          add       si,dx
          cmp       dl,2
          je        rgd62
rgd61:    mov       al,[bx+2]           ;convierto MSB
          call      rgd03
          mov       cl,4
          shl       dh,cl
          mov       [si+1],dh
          mov       al,[bx+3]
          call      rgd03
          add       [si+1],dh           ;MSB transformado
          add       bx,2
rgd62:    mov       al,[bx+2]           ;convierto LSB
          call      rgd03
          mov       cl,4
          shl       dh,cl
          mov       [si],dh
          mov       al,[bx+3]
          call      rgd03
          add       [si],dh
rgdf:     rtodo
          ret

rgd0:     push      bx
          push      cx
          mov       al,[bx]             ;pone DH en el buffer apunt. por BX
          cmp       al,[bx+1]           ;ya no hay espacio?
          mov       al,1
          je        rgd0f
          inc       byte ptr[bx+1]      ;incremento contador
          mov       si,bx
          mov       bx,1
          cmp       byte ptr[si],1
          je        rgd0n
rgd01:    mov       al,[bx+si+2]        ;rota a la izquierda n-1 veces
          mov       [bx+si+1],al        ;donde n=# de datos pedidos.
          inc       bl
          cmp       bl,[si]
          jb        rgd01
rgd0n:    mov       [bx+si+1],dh        ;incluyo nuevo dato.
          mov       al,0
rgd0f:    pop       cx
          pop       bx
          ret
rgd01:    push      cx                  ;presento buffer en pantalla
          push      dx
```

```
          posdir    cl,ch
          mov       di,pddir
          mov       bp,di
          mov       cl,[bx]
          mov       ch,0
          mov       al,0b1h             ;primero borro
rgd011:   stosb
          inc       di
          loop      rgd011
          mov       al,[bx+1]
          mov       cl,[bx]
          sub       cl,al
          mov       si,bx
          add       si,2
          add       si,cx
          mov       di,bp
          add       cx,cx
          add       di,cx
          mov       cl,[bx+1]
          cmp       cl,0
          je        rgd013
rgd012:   lodsb                       ;muestro el numero
          stosb
          inc       di
          loop      rgd012
rgd013:   pop       dx
          pop       cx
          ret
rgd02:    push      bx                  ;rota a la derecha
          dec       byte ptr[bx+1]      ;disminuyo contador
          mov       si,bx
          mov       bl,[si]
          mov       bh,0
          cmp       bl,1
          je        rgd022
rgd021:   mov       al,[bx+si]
          mov       [bx+si+1],al
          dec       bx
          cmp       bl,1
          jne       rgd021
rgd022:   mov       byte ptr[bx+si+1],'0'
          pop       bx
          ret
rgd03:    mov       di,offset gdtab     ;Busco AL en GDTAB
          mov       dh,0                ;y traigo ubicacion en DH.
rgd031:   cmp       al, [di]
          je        rgd032
          inc       di
          inc       dh
          cmp       dh,22
          jne       rgd031
rgd032:   cmp       dh,16
          jb        rgd03f
          sub       dh,6
rgd03f:   ret
```

```
rgk:    stodo
        mov     ah,0
        int     16h
        mov     getword,ax
        rtodo
        ret

rgl:    push    di              ;recibo un string desde teclado
        push    cx              ;SI apunta al inicio del buffer
        push    bx
        mov     rglx,cl
        mov     rgly,ch
        cmp     gdreset,1       ;inicializo el buffer?
        jne     rgla
        mov     di,si           ;limpia el buffer
        inc     di
        mov     cl,[si]
        mov     ch,0            ;CX=# car.
        inc     cx
        mov     ax,ds
        mov     es,ax
        mov     al,0
        rep     stosb
        mov     es,videoseg
rgla:   mov     bh,0
rgl1:   mov     cl,[si+1]
        call    curs24          ;pongo cursor
        getchar                 ;recibo un dato del teclado
        mov     al,atrprev      ;quito cursor
        mov     di,dirprev
        stosb
        mov     cl,[si]         ;num. maximo de caracteres
        mov     bl,[si+1]       ;num. actual de caracteres
        cmp     gccod,5         ;car. especial?
        je      rgl2
        cmp     gccod,1         ;ENTER?
        je      rgl3
        cmp     gccod,0         ;ascii normal?
        je      rgl11
        jmp     rgl3            ;TAB, SHIFT TAB y ESC: salir.
rgl11:  mov     al,gcascii
        cmp     al,8            ;Backspace?
        je      rgld
        cmp     bl,cl           ;fin del buffer?
        je      rglb
        mov     [bx+si+2],al    ;no, guardo dato
        mov     cl,bl           ;eco a pantalla
        add     cl,rglx
        mov     ch,rgly
        posdir  cl,ch
        mov     di,pddir
        mov     al,gcascii
        stosb
        inc     byte ptr[si+1]
        jmp     rgl1
rglb:   mov     dl,7            ;BEEP
```

```
            mov     ah,2
            int     21h
            jmp     rgl1
rgl2:       cmp     gcascii,83          ;DELETE?
            je      rgld
            jmp     rgl3
rgld:       cmp     bl,0                ;puedo seguir borrando?
            je      rglb                ;no,beep
            dec     byte ptr[si+1]
            mov     byte ptr[bx+si+1],0
            mov     cl,bl               ;borro en pantalla
            add     cl,rglx
            dec     cl
            mov     ch,rgly
            posdir  cl,ch
            mov     di,pddir
            mov     al,' '
            stosb
            jmp     rgl1
rgl3:       pop     bx
            pop     cx
            pop     di
            ret

curs24:     push    cx
            push    di
            add     cl,rglx
            mov     ch,rgly             ;encuentro posicion del cursor
            posdir  cl,ch               ;altero AX,DI
            mov     di,pddir
            inc     di
            mov     ah,es:[di]
            mov     atrprev,ah
            mov     dirprev,di
            mov     al,revers
            stosb
            pop     di
            pop     cx
            ret

rprt:       push    di
            mov     di,pddir            ;direccion de inicio
rprt1:      lodsb
            cmp     al,0                ;termino?
            je      rprt2
            stosw                       ;AL tiene el ASCII, AH el atributo
            jmp     rprt1
rprt2:      pop     di
            ret

rgr:        push    bx
            hexasc
            mov     ax,bx
            xchg    al,ah
            stosw
            pop     bx
```

```
        ret

rgp:    stodo
        xor     si,si
        mov     di,offset panbuf
        mov     bx,ds
        mov     es,bx
        mov     ds,videoseg
        mov     cx,3680
        rep     movsb
        rtodo
        ret

rl23:   push    di
        push    cx
        mov     di,23*160               ;DI = linea 23
        mov     cx,80                   ;80 columnas
        mov     ax,7020h                ;video reverso, espacio en blanco
        rep     stosw
        pop     cx
        pop     di
        ret

rl24:   push    di
        push    cx
        mov     di,24*160               ;DI = linea 24
        mov     cx,80                   ;80 columnas
        mov     ax,0720h                ;video normal, espacio en blanco
        rep     stosw                   ;limpia linea 24
        pop     cx
        pop     di
        ret

rtp:    stodo
        xor     di,di                   ;DI = 0
        mov     es,videoseg             ;ES = segmento de video
        mov     si,offset panbuf        ;SI = buffer de pantalla
        mov     cx,3680
        rep     movsb                   ;recupera datos de pantalla
        rtodo
      ret

        endm
```

```
;**************************************
;*********     MACARC.ASM     **********
;**************************************

;MACROS PARA ARCHIVOS

ABRIR     MACRO fichero,acceso,handle,error
          LOCAL bien,fin
;-------------------------------------------------------------------------------
;fichero = cadena ASCIIZ con unidad, sendero y nombre del fichero
;acceso = codigo de acceso (ent)(1 byte)(variable,registro,valor inmediato)
;           0 - solo lectura
;           1 - solo escritura
;           2 - lectura y escritura
;handle = Identificador fichero (file handle)(sal)(1 palabra)(variable)
;error = numero del error (sal)(1 palabra)(variable)
;          si no hay error, error = 0
;          los errores posibles son 2,4,5 y 12
;-------------------------------------------------------------------------------
          push   ax
          push   dx

          lea    dx,fichero
          mov    al,acceso

          mov    ah,3dh
          int    21h

          jnc    bien

          mov    error,ax
          jmp    fin

bien:     mov    handle,ax
          mov    error,0

fin:    pop    dx
          pop    ax

          ENDM

LEER      MACRO handle,nbytes1,area,nbytes2,error
          LOCAL bien,fin
;-------------------------------------------------------------------------------
;handle = identificador fichero (file handle)(ent)(1 palabra)
;           (variable o registro BX).
;nbytes1 = numero de bytes a leer (ent)(1 palabra)(variable,registro CX,
;            valor inmediato)
;area = area de lectura (sal)(variable)
;nbytes2 = numero de bytes leidos (sal)(1 palabra)(variable)
;            Si es cero, se ha tratado de leer a partir del fin de fichero
;error = numero del error (sal)(1 palabra)(variable)
;          si no hay error, error = 0
;          los errores posibles son 5 y 6
;-------------------------------------------------------------------------------
          push   ax
          push   bx
          push   cx
```

```
;*****************************************************************************
;                              ENS.ASM
;*****************************************************************************
;
          EXTRN PC:WORD

DSEG SEGMENT PUBLIC

transit             db        32 dup (?)          ;String a ensamblar

;Pseudocodigos de los mnemotecnicos del DS5000T:

mnemo               db        'NOp',0,'PJMp',1,'JMp',2,'Rr',3
                    db        'INc',4,'BBc',10H,'PCALl',11H,'CALl',12H
                    db        'RRc',13H,'DEc',14H,'Bb',20H,'REt',22H
                    db        'Rl',23H,'ADd',24H,'BNb',30H,'RETi',32H
                    db        'RLc',33H,'ADc',34H,'Bc',40H,'Or',42H
                    db        'BNc',50H,'ANd',52H,'Bz',60H,'XOr',62H
                    db        'BNz',70H,'LDr',74H,'BRa',80H,'LDc',83H
                    db        'DIv',84H,'LDb',92H,'SBc',94H,'MUl',0A4H
                    db        'BCOm',0B2H,'COm',0B3H,'CBNe',0B4H,'PUSh',0C0H
                    db        'BCLr',0C2H,'CLr',0C3H,'SWAp',0C4H,'EXh',0C5H
                    db        'PUL1',0D0H,'BSEt',0D2H,'DAa',0D4H,'DBNz',0D5H
                    db        'EXHd',0D6H,'LDx',0E0H
                    db        0FFH

;Direcciones de los bytes directamente direccionables:

direct_byte         db        'P0',80H,'Sp',81H,'DPl',82H,'DPh',83H
                    db        'PCOn',87H,'TCOn',88H,'TMOd',89H,'TL0',8AH
                    db        'TL1',8BH,'TH0',8CH,'TH1',8DH,'P1',90H
                    db        'SCOn',98H,'SBUf',99H,'P2',0A0H,'Ie',0A8H
                    db        'P3',0B0H,'Ip',0B8H,'EK0',0C1H,'EK1',0C2H
                    db        'EK2',0C3H,'EK3',0C4H,'EK4',0C5H,'MCOn',0C6H
                    db        'Ta',0C7H,'PSw',0D0H,'ACc',0E0H,'b',0F0H
                    db        0FFH

;Direcciones de los bits directamente direccionables:

direct_bit          db        'P0.0',80H,'P0.1',81H,'P0.2',82H,'P0.3',83H
                    db        'P0.4',84H,'P0.5',85H,'P0.6',86H,'P0.7',87H
                    db        'P1.0',90H,'P1.1',91H,'P1.2',92H,'P1.3',93H
                    db        'P1.4',94H,'P1.5',95H,'P1.6',96H,'P1.7',97H
                    db        'P2.0',0A0H,'P2.1',0A1H,'P2.2',0A2H,'P2.3',0A3H
                    db        'P2.4',0A4H,'P2.5',0A5H,'P2.6',0A6H,'P2.7',0A7H
                    db        'P3.0',0B0H,'P3.1',0B1H,'P3.2',0B2H,'P3.3',0B3H
                    db        'P3.4',0B4H,'P3.5',0B5H,'P3.6',0B6H,'P3.7',0B7H
                    db        'IT0',88H,'IE0',89H,'IT1',8AH,'IE1',8BH
                    db        'TR0',8CH,'TF0',8DH,'TR1',8EH,'TF1',8FH
                    db        'Ri',98H,'Ti',99H,'RB8',9AH,'TB8',9BH
                    db        'REn',9CH,'SM2',9DH,'SM1',9EH,'SM0',9FH
                    db        'EX0',0A8H,'ET0',0A9H,'EX1',0AAH,'ET1',0ABH
                    db        'Es',0ACH,'Ea',0AFH,'RXd',0B0H,'TXd',0B1H
                    db        'INT0',0B2H,'INT1',0B3H,'T0',0B4H,'T1',0B5H
                    db        'Wr',0B6H,'Rd',0B7H,'PX0',0B8H,'PT0',0B9H
                    db        'PX1',0BAH,'PT1',0BBH,'Ps',0BCH,'RWt',0BFH
                    db        'p',0D0H,'Ov',0D2H,'RS0',0D3H,'RS1',0D4H
                    db        'F0',0D5H,'Ac',0D6H,'Cy',0D7H
```

```
                db      'ACC.0',0E0H,'ACC.1',0E1H,'ACC.2',0E2H,'ACC.3',0E3H,
                db      'ACC.4',0E4H,'ACC.5',0E5H,'ACC.6',0E6H,'ACC.7',0E7H,
                db      'B.0',0F0H,'B.1',0F1H,'B.2',0F2H,'B.3',0F3H
                db      'B.4',0F4H,'B.5',0F5H,'B.6',0F6H,'B.7',0F7H
                db      'PSW.0',0D0H,'PSW.1',0D1H,'PSW.2',0D2H,'PSW.3',0D3H,
                db      'PSW.4',0D4H,'PSW.5',0D5H,'PSW.6',0D6H,'PSW.7',0D7H,
                db      'IP.0',0B8H,'IP.1',0B9H,'IP.2',0BAH,'IP.3',0BBH
                db      'IP.4',0BCH,'IP.7',0BFH
                db      'IE.0',0A8H,'IE.1',0A9H,'IE.2',0AAH,'IE.3',0ABH
                db      'IE.4',0ACH,'IE.7',0AFH
                db      'SCON.0',98H,'SCON.1',99H,'SCON.2',9AH,'SCON.3',9BH
                db      'SCON.4',9CH,'SCON.5',9DH,'SCON.6',9EH,'SCON.7',9FH
                db      'TCON.0',88H,'TCON.1',89H,'TCON.2',8AH,'TCON.3',8BH
                db      'TCON.4',8CH,'TCON.5',8DH,'TCON.6',8EH,'TCON.7',8FH
                db      0FFH

;Pseudocodigos de los registros del DS5000T:

registros       db      'R0',20h,'R1',21h,'R2',22h,'R3',23h
                db      'R4',24h,'R5',25h,'R6',26h,'R7',27h
                db      'a',28h,'DPTr',29h,'Ab',2AH,'c',2BH
                db      0FFH

;Pseudocodigos de los registros indices del DS5000T:

index_reg       db      '@R0',40h,'@R1',41h,'@DPTr',42h,'@A+DPTr',43h
                db      '@A+Pc',44h
                db      0FFH

;Instrucciones que no tienen parametros:

sin_param       db      0,22H,32H,0D4H

;Instrucciones que tienen un solo parametro:

un_param        db      1,2,3,4,11H,12H,13H,14H
                db      23H,33H,40H,50H,60H,70H,80H,84H
                db      0A4H,0B2H,0B3H,0C0H,0C2H,0C3H,0C4H,0D0H
                db      0D2H

NEsp            equ     10      ;Numero maximo de espacios entre parametros

DSEG ENDS


CSEG SEGMENT
ASSUME CS:CSEG,DS:DSEG,ES:NOTHING

DELSPC  PROC    NEAR
;-----------------------------------------------------------------
;Este procedimiento salta los espacios y los TABs en el string por
;ensamblar y deja SI apuntando al primer byte que no sea un espacio.
;
;Entrada:        DS:SI   apunta al punto del string desde el cual
;                        se desea proceder.
```

```
;Salida:          DS:SI     apunta al primer byte que no es un espacio.
;                 CY        es 1 si hay más de NEsp espacios (error), en
;                           cuyo caso DS:SI no es alterado
;-----------------------------------------------------------------
          push    ax
          push    cx
          push    bp
          mov     cx,NEsp             ;máximo NEsp espacios
          mov     bp,si
busc:     lodsb
          or      al,al
          jz      errdel              ;si se llega al final del string
          cmp     al,' '              ;hay un espacio?
          je      spac                ;si, saltarlo
          cmp     al,9                ;hay un TAB?
          jne     encont              ;no, hecho
spac:     loop    busc                ;saltar el espacio y ver sig. car.

errdel:   mov     si,bp
          stc
          jmp     findl
encont:   dec     si                  ;apuntar al carácter válido
          clc
findl:    pop     bp
          pop     cx
          pop     ax
          ret                         ;retornar

DELSPC    ENDP

ENCONTRAR         PROC      NEAR
;-----------------------------------------------------------------------
;Este procedimiento encuentra el pseudocódigo del mnemotécnico, byte
;directamente direccionable, registro o registro indice que
;forma parte de una instruccion del DS5000T
;
;Entrada:         DS:SI apunta al string, que debe terminar en 0.
;                 ES:DI es la dirección en donde se pondrá el pseudocódigo.
;                 BX    contiene el offset del inicio de la tabla donde se
;                       va a buscar
;
;Salida:          DS:SI apunta al final del mnemotécnico + 2 en el string si
;                       este fue encontrado (pues debe terminar con espacio)
;                 ES:DI apunta al final del pseudocódigo + 1
;                 CY    es 1 si el mnemotécnico es inválido o si hay error
;                       como en DELSPC , en cuyo caso DS:SI y ES:DI no son
;                       alterados (excepto si había espacios desde DS:SI)
;                 AX    contiene el pseudocódigo (AL=0, AH= cod.)
;-----------------------------------------------------------------------
          call    delspc              ;encontrar el mnemotécnico
          jc      fin2                ;en caso de error, terminar
          push    bx
          push    cx
          push    dx
          push    es
          push    di
```

```
        mov     ax,dseg
        mov     es,ax               ;hacer que ES:DI apunte a la tabla
        mov     di,bx
        cmp     bx,offset mnemo     ;se busca un mnemotecnico?
        je      si_mnemo            ;si
        cmp     bx,offset registros         ;no,se busca un registro?
        je      si_reg              ;si
        cmp     bx,offset index_reg         ;no,se busca un registro indice?
        je      si_reg              ;si
        mov     cl,1                ;marcar que se busca un byte
        jmp     no_reg              ;directamente direccionable
fin2:
        jmp     finenc2
si_mnemo:
        mov     cl,2                ;marcar que se busca un mnemo
        jmp     no_reg
si_reg:
        xor     cl,cl               ;marcar que se busca un registro
no_reg:
        mov     dx,si               ;preservar el comienzo del mnemotécnico
busca:
        lodsb                       ;carácter del string en AL
        mov     ah,es:[di]          ;carácter de la tabla en AH
        cmp     ah,'a'              ;es una minúscula, implicando fin de mnemo?
        jnb     finmnemo            ;si
        cmp     ah,'+'              ;no,es un signo +?
        je      nofinmnemo          ;si,implica que no finaliza el mnemo.
        cmp     ah,'9'              ;es un digito?
        jbe     finmnemo            ;si,implica fin de mnemo
        jmp     nofinmnemo
finmnemo:
        cmp     ah,0ffh             ;es el fin de la tabla?
        jne     uff                 ;no, probar el siguiente mnemo de la tabla
        stc                         ;si, el código era inválido pues no
        mov     si,dx               ;constaba en la tabla, asi que se
        jmp     finenc              ;debe generar un bit de error
uff:
        mov     bl,1                ;indicar fin de mnemo haciendo BL = 1
        and     ah,0dfh             ;hacer mayúscula a AH
        jmp     sigcomp             ;comparar el ultimo carácter
nofinmnemo:
        xor     bl,bl               ;indicar que todavía no hay fin de mnemo
sigcomp:
        cmp     al,'+'
        je      sig1
        and     al,0dfh             ;hacer mayúscula a AL
sig1:
        cmp     al,ah               ;son iguales los dos caracteres?
        jne     noeseste            ;no, intentar con el siguiente mnemo
        or      bl,bl               ;si, era el ultimo carácter del mnemo?
        jz      puedeser            ;no, seguir comparando
        lodsb                       ;comparación exitosa, verificar que
        cmp     al,' '              ;un espacio, TAB o 0  despues del mnemo
        jbe     sies
        cmp     cl,2
```

```
          je        noeseste            ;si no, intentar con el siguiente mnemo
          cmp       al,','
          jne       noeseste
sies:     dec       si                  ;apuntando a este
          mov       ah,es:[di+1]        ;encontrado, obtener el pseudocódigo
          pop       di
          pop       es
          or        cl,cl               ;es registro o index_reg?
          jz        si_regist           ;si,guardarlo como byte
          xor       al,al
          stosw                         ;guardarlo en su formato (word)
          clc
          jmp       finenc1
si_regist:
          mov       al,ah
          stosb                         ;guardarlo como byte
          xor       al,al
          clc
          jmp       finenc1
puedeser:
          inc       di                  ;apuntar al siguiente carácter de la tabla
          jmp       busca
noeseste:
          mov       ah,es:[di]          ;avanzar hasta el siguiente mnemo de
          inc       di                  ;la tabla buscando el fin del mnemo actual
          cmp       ah,'a'
          jnb       siga
          cmp       ah,'9'
          jbe       siga
          jmp       noeseste
siga:
          inc       di                  ;saltar el código
          mov       si,dx               ;restaurar comienzo del mnemo en el string
          jmp       busca               ;repetir
finenc:
          pop       di
          pop       es
finenc1:
          pop       dx
          pop       cx
          pop       bx
finenc2:ret                           ;retornar

ENCONTRAR           ENDP

ENCONTRAR_BIT       PROC      NEAR
;-----------------------------------------------------------------------
;Este procedimiento encuentra el pseudocódigo de un bit directamente
;direccionable.
;
;Entrada:           DS:SI apunta al string, que debe terminar en 0.
;                   ES:DI es la dirección en donde se pondrá el pseudocódigo.
;
;Salida:            DS:SI apunta al final del mnemotécnico + 2 en el string si
;                         este fue encontrado (pues debe terminar con espacio)
```

```
;                   ES:DI apunta al final del pseudocódigo + 1
;                   CY    es 1 si el mnemotécnico es inválido o si hay error
;                         como en DELSPC , en cuyo caso DS:SI y ES:DI no son
;                         alterados (excepto si había espacios desde DS:SI)
;                   AX    contiene el pseudocódigo (AL=0, AH= cod.)
;-------------------------------------------------------------------------
            call    delspc              ;encontrar el mnemotécnico
            jc      fin_b               ;en caso de error, terminar
            cmp     byte ptr [si],'9'
            jb      fin_b
            push    bx
            push    cx
            push    dx
            push    es
            push    di
            mov     ax,dseg
            mov     es,ax               ;hacer que ES:DI apunte a la tabla de
            mov     di,offset direct_bit ;bits directamente direccionables
            jmp     siga_b
fin_b:
            jmp     finenc_b
siga_b:     mov     dx,si               ;preservar el comienzo del mnemotécnico
busca2:
            lodsb                       ;carácter del string en AL
            mov     ah,es:[di]          ;carácter de la tabla en AH
            cmp     byte ptr [di+1],80h ;es una minúscula, implicando fin de mnemo?
            jae     finbit2   ;no
            jmp     nofinbit2
finbit2:
            cmp     ah,0ffh             ;es el fin de la tabla?
            jne     uff2                ;no, probar el siguiente mnemo de la tabla
            stc                         ;si, el código era inválido pues no
            mov     si,dx               ;constaba en la tabla, asi que se
            jmp     finenc_b2           ;debe generar un bit de error
uff2:
            mov     bl,1                ;indicar fin de mnemo haciendo BL = 1
            cmp     ah,'a'
            jb      sig13
            and     ah,0dfh             ;hacer mayúscula a AH
sig13:      jmp     sigcomp2            ;comparar el ultimo carácter
nofinbit2:
            xor     bl,bl               ;indicar que todavía no hay fin de mnemo
sigcomp2:
            cmp     al,'a'
            jb      sig12
            and     al,0dfh             ;hacer mayúscula a AL
sig12:
            cmp     al,ah               ;son iguales los dos caracteres?
            jne     noeseste2           ;no, intentar con el siguiente mnemo
            or      bl,bl               ;si, era el ultimo carácter del mnemo?
            jz      puedeser2           ;no, seguir comparando
            lodsb                       ;comparación exitosa, verificar que
            cmp     al,' '              ;un espacio, TAB o 0  despues del mnemo
            jbe     sies2
            cmp     al,','
```

```
numah:  cmp     al,'A'              ;lo mismo con AL
        jb      numal
        sub     al,'A'-'9'-1
numal:  sub     ax,'00'             ;pasar de ascii a numérico
        cmp     al,16
        jae     errhex
        cmp     ah,16
        jae     errhex
        mov     cl,4
        shl     al,cl
        or      al,ah               ;condensar el número
        clc
        jmp     finhex
errhex: stc
finhex: pop     cx
        ret                         ;retornar

ASCAHEX ENDP

ESLETR? PROC    NEAR
;-----------------------------------------------------------------------
;Este procedimiento determina si el carácter contenido en AL es un dígito
;o una letra, en cuyo caso CY = 0, y de lo contrario, CY = 1.
;AL es alterado.
;-----------------------------------------------------------------------
        cmp     al,'0'
        jb      noeslt
        cmp     al,'9'
        jbe     sieslt
        and     al,0dfh             ;mayúscula
        cmp     al,'A'
        jb      noeslt
        cmp     al,'Z'
        jbe     sieslt
noeslt: stc
        ret                         ;retornar
sieslt: clc
        ret                         ;retornar

ESLETR? ENDP

        PUBLIC  ESHEX?
ESHEX?  PROC    NEAR
;-----------------------------------------------------------------------
;Este procedimiento determina si el string al que apunta DS:SI es un número
;hexadecimal, en cuyo caso encuentra su valor.
;
;Entrada:       DS:SI apunta al string, excluyendo signos +, - o espacios
;
;Salida:        AL  = 16 si el número es de 1 o 2 digitos, 32 en otro caso
;               AH  = longitud del string del número
;               BX  = valor del número
;               CY  = 1 si lo apuntado por DS:SI no es un número o si es un
;                     número de mas de 32 bits
;-----------------------------------------------------------------------
        push    dx
```

```
        xor     bx,bx
trycero:
        cmp     byte ptr[si+bx],'0' ;buscar ceros iniciales
        jne     noceroin
        inc     bx                  ;contarlos en BX
        jmp     trycero
noceroin:
        mov     dh,bl               ;número de ceros en DH
        or      bx,bx
        jz      noproblem           ;si no hay ceros iniciales
        mov     ah,[si+bx]
        mov     al,'0'
        call    ascahex             ;si el string era solo ceros, ya no debe
        jnc     noproblem           ;haber mas caracteres hexadecimales válidos
        mov     ax,16
        xor     bx,bx               ;salir con cero
        jmp     finnum
noproblem:
        mov     al,[si+bx+1]
        call    esletr?
        jc      undig
        mov     al,[si+bx+2]
        call    esletr?
        jc      dosdig
        mov     al,[si+bx+3]
        call    esletr?
        jc      tresdig

        mov     al,[si+bx+4]
        call    esletr?             ;tiene más de 4 dígitos?
        jc      N4Dig               ;no, OK
        stc                         ;si, error
        jmp     errnum
N4Dig:
        mov     ax,[si+bx]          ;el número tiene 4 dígitos
        call    ascahex
        jc      errnum
        mov     dl,al
        mov     ax,[si+bx+2]
        call    ascahex
        jc      errnum
        mov     bl,al
        mov     bh,dl
        mov     ax,420h             ;AL=32, AH=4
        jmp     finnum
tresdig:
        mov     al,'0'              ;número de tres dígitos
        mov     ah,[si+bx]
        call    ascahex
        jc      errnum
        mov     dl,al
        mov     ax,[si+bx+1]
        call    ascahex
        jc      errnum
        mov     bl,al
        mov     bh,dl
```

```
          mov       ax,320h             ;AL=32, AH=3
          jmp       finnum
dosdig:
          mov       ax,[si+bx]          ;número de dos digitos
          call      ascahex
          jc        errnum
          mov       bl,al
          xor       bh,bh
          mov       ax,210h             ;AL=16, AH=2
          jmp       finnum
undig:
          mov       al,'0'
          mov       ah,[si+bx]
          call      ascahex
          jc        errnum
          mov       bl,al
          xor       bh,bh
          mov       ax,110h             ;AL=16, AH=1
finnum:
          add       ah,dh               ;incluir los ceros en la longitud
          cmp       ah,2
          jbe       longbyt
          mov       al,20h
longbyt:
          clc
errnum:
          pop       dx
          ret                           ;retornar

ESHEX?    ENDP

ENCHEX    PROC      NEAR
;-----------------------------------------------------------------------------
;Este procedimiento determina si lo apuntado por DS:SI es un número
;hexadecimal, en cuyo caso encuentra su valor y lo coloca en ES:DI después
;del código correspondiente.
;
;Entrada:           DS:SI     apunta al string que representa al número
;                   ES:DI     apunta a donde deba ir el código
;
;Salida:            DS:SI     es ajustado
;                   ES:DI     es ajustado
;                   CL        es 1 si byte, es 2 si word
;                   CY        set si hay error, en cuyo caso DS:SI y ES:DI no son
;                             alterados
;-----------------------------------------------------------------------------
          push      ax
          push      bx
          push      dx
          call      delspc
          jc        finenchex
          xor       dx,dx               ;marcar que no hay #
          mov       al,[si]
          cmp       al,'#'
          je        si_num
          call      delspc
```

```
          jc        finenchex
          jmp       decode
si_num:
          xor       al,al               ;es rel (sin simbolo #)
          not       dl                  ;marcar que es #
          inc       si
          call      delspc
          jc        finenchex
decode:
          call      eshex?
          jc        finenchex
          or        dl,dl               ;es #?
          jz        no_num              ;no
          mov       al,1                ;es data (con simbolo #)
          jmp       sigal
no_num:
          mov       al,0
sigal:
          mov       dl,ah               ;longitud del string del número
          add       si,dx               ;actualizar SI
          or        al,80h              ;código de dato de un byte
          cmp       bx,100h             ;longitud de 2 o menos?
          jb        unbyte              ;si
          mov       cl,2
          or        al,20h              ;no, poner código de word
          cmp       bx,7ffh
          jbe       once_bits
          or        al,02h
once_bits:
          stosb
          mov       ax,bx
          stosw
          jmp       noerrhex
unbyte:
          mov       cl,1
          mov       ah,bl
          stosw
noerrhex:
          clc
finenchex:
          pop       dx
          pop       bx
          pop       ax
          ret                           ;retornar

ENCHEX    ENDP


ENSPASO1  PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento ejecuta el primer paso de compilación, es decir, la
;traducción de un string alfanumérico a seudocódigos que serán transformados
;a código verdadero por otro procedimiento.
;
;Entrada:           DS:SI      apunta al string por compilarse, que debe terminar
;                              en 0.
```

```
;                   ES:DI     apunta al lugar donde se desea el seudocódigo
;
;Salida:            DS:SI     es alterado
;                   ES:DI     es alterado
;                   CY        set si hay error
;-------------------------------------------------------------------------
          push      ax
          push      bx
          push      cx
          push      dx
          mov       bx,offset mnemo
          call      encontrar           ;lo primero debe ser un mnemotécnico
          jc        errpaso1
          mov       dx,ds               ;preservar DS
          mov       bx,dseg
          mov       ds,bx
          mov       bx,offset sin_param
          mov       cx,4                ;averiguar si el mnemotécnico es uno de los
lone?:    cmp       ah,[bx]             ;que no aceptan argumentos
          je        esloner             ;si
          inc       bx
          loop      lone?
          jmp       hayarg
esloner:
          mov       ds,dx
          mov       byte ptr es:[di-2],1          ;marcarlo como loner
                                        ;apuntar al punto inmediatamente despues
          call      delspc              ;del mnemotécnico y buscar el fin de string
          jnc       errpaso1            ;si hay mas caracteres válidos, error

bienpaso1:                              ;proceso sin error
          mov       al,0ffh
          stosb
          clc
          jmp       finpaso1
errpaso1:
          stc                           ;hubo un error
finpaso1:
          pop       dx
          pop       cx
          pop       bx
          pop       ax
          ret                           ;retornar

hayarg:
          mov       bx,offset un_param
          mov       cx,25               ;averiguar si el mnemotécnico es uno de los
one?:
          cmp       ah,[bx]             ;que aceptan un solo argumento
          je        esoner              ;si
          inc       bx
          loop      one?
          jmp       mas_arg
esoner:
          mov       cl,1
          mov       byte ptr es:[di-2],2          ;marcarlo como oner
```

```
BUSCAR_ARG          PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento busca los argumentos que deben acompanar al mnemotecnico
;
;Entrada:           DS:SI apunta al string
;                   ES:DI es la direccion donde se pondra el pseudocodigo de los
;                         argumentos
;
;Salida:            DS:SI es modificado como en ENCONTRAR
;                   ES:DI es modificado como en ENCONTRAR
;                   CY    es 1 si no se encontro un argumento
;-------------------------------------------------------------------------

          push      bx
          push      cx
          mov       bx,offset registros         ;buscar primer argumento
          call      encontrar                   ;es un registro?
          jnc       fin_busc                    ;si
          mov       bx,offset index_reg         ;no
          call      encontrar                   ;es un registro indice?
          jnc       fin_busc                    ;si
          mov       bx,offset direct_byte       ;no
          call      encontrar          ;es un byte directamente direccionable?
          jc        no_byte                     ;no
          mov       byte ptr es:[di-2],61h      ;si, marcarlo como tal
          jmp       fin_busc
no_byte:
          call      encontrar_bit      ;es un bit directamente direccionable?
          jc        no_bit                      ;no
          mov       byte ptr es:[di-2],60h      ;si, marcarlo como tal
          jmp       fin_busc
no_bit:
          call      enchex                      ;es una direccion?
          jc        no_rel                      ;no
          jmp       fin_busc                    ;si, finalizar la busqueda
no_rel:
          lodsb
          cmp       al,'#'                      ;hay el simbolo #?
          jne       compl?                      ;no
          call      enchex                      ;es un dato valido?
          jc        err_busc                    ;no, error
          cmp       cl,1                        ;si, marcarlo
          je        dir_byte
          or        byte ptr es:[di-2],1
          jmp       fin_busc
dir_byte:
          or        byte ptr es:[di-1],1
          jmp       fin_busc
compl?:
          dec       si
          lodsb
          cmp       al,'/'                      ;hay el simbolo /?
          jne       err_busc                    ;no, error
          call      enchex                      ;es un dato valido?
```

```
          jc        bit?                            ;no
          cmp       cl,1                            ;si, marcarlo
          jne       err_busc
          or        byte ptr es:[di-1],2
          jmp       fin_busc
bit?:
          call      encontrar_bit                   ;es un bit complementado?
          jc        err_busc                        ;no, error
          mov       byte ptr es:[di-2],82h          ;si, marcarlo
          jmp       fin_busc
err_busc:
          stc                                       ;hubo un error
fin_busc:
          pop       cx
          pop       bx
          ret                                       ;retornar

BUSCAR_ARG          ENDP

TIPO      PROC      NEAR
;-------------------------------------------------------------------------------
;Este procedimiento determina el tipo de pseudocódigo al que apunta DS:SI y
;proporciona información útil sobre dicho pseudocódigo.
;
;Entrada:           DS:SI     apunta al inicio del pseudocódigo
;
;Salida:            DS:SI     apunta al siguiente pseudocódigo
;                   CY        1 ssi encontró un FFh, fin de instrucción
;
;                   DL =      0 si es un mnemotécnico, en cuyo caso
;                             AL = código del mnemotécnico
;                             AH = información sobre loners (sin parametros),
;                                  oners (con un parametro), pinst y cbne
;
;                   DL =      1 si es un registro, en cuyo caso
;                             AL = 0000Regs
;
;                   DL =      2 si es un registro indice, en cuyo caso
;                             AL = 00000Reg
;
;                   DL =      3 si es un Memoria, en cuyo caso
;                             AL = codigo del bit o byte directly direccionable
;                             AH = informacion sobre bit o byte
;
;                   DL =      4 si es un dato de 8 bits, en cuyo caso
;                             AL = dato
;                             AH = informacion sobre data o rel y complemento
;
;                   DL =      5 si es un dato de 16 u 11 bits, en cuyo caso
;                             AX = dato
;                             BL = informacion sobre data o rel y 16 u 11 bits
;-------------------------------------------------------------------------------
          push      cx
          lodsb
          cmp       al,0ffh             ;fin de instrucción?
          je        fininst
```

```
          mov       ch,ah
          mov       ah,al
          mov       cl,5
          shr       al,cl               ;obtener el tipo
          mov       dl,al               ;en DL
          jz        esinstr
          dec       al
          jz        esregis
          dec       al
          jz        esregind
          dec       al
          jz        esmemoria
          dec       al
          jz        esdata8b
          xchg      ah,bl
          and       bl,3
          lodsw                         ;es dato de 16 u 11 bits
          jmp       rdytipo
esdata8b:
          and       ah,3
          lodsb                         ;es dato de byte
          jmp       rdytipo
esinstr:
          lodsb                         ;es mnemotécnico
          jmp       rdytipo
esregis:
          mov       al,ah               ;es registro
          and       al,0fh
          mov       ah,ch
          jmp       rdytipo
esregind:
          mov       al,ah               ;es registro indice
          and       al,7
          mov       ah,ch
          jmp       rdytipo
esmemoria:
          and       ah,1                ;es memoria
          lodsb
rdytipo:
          clc                           ;proceso sin error
          jmp       fintipo
fininst:
          stc                           ;hubo un error
fintipo:
          pop       cx
          ret                           ;retornar

TIPO      ENDP

JMPSH?    PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento determina si la distancia entre AX y DI esta entre
;+127 y -128, en cuyo caso pone esta distancia en AX y CY = 0; de lo
;contrario, CY = 1.
;-------------------------------------------------------------------------
          push      di
```

```
          mov     di,pc
          sub     ax,di
          cmp     ax,127              ;es la distancia menor que +127?
          jg      notshrt             ;no
          cmp     ax,-128             ;si, es la distancia menor que -128?
          jl      notshrt             ;no
          clc                         ;si
          pop     di
          ret                         ;retornar
notshrt:  stc
          pop     di
          ret                         ;retornar

JMPSH?    ENDP

ESBIT?    PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento determina si el pseudocodigo corresponde a un bit
;directamente direccionable, en base a los parametros generados por la
;subrutina TIPO.
;Entrada:          DL = tipo de pseudocodigo
;                  AX = dato
;Salida:           AL = codigo del bit directamente direccionable
;                  CY = 1 si el pseudocodigo no corresponde a un bit
;-------------------------------------------------------------------------
          cmp     dl,3                ;es memoria?
          jne     no_mem              ;no
          cmp     ah,0
          jne     erresbit
          jmp     finesbit
no_mem:
          cmp     dl,4                ;es dato de 8 bits?
          jne     erresbit  ;no
          cmp     ah,0
          jne     erresbit
          cmp     al,0bfh
          jbe     finesbit
          cmp     al,0d0h
          jb      erresbit
          test    al,8
          jnz     erresbit
          jmp     finesbit
erresbit:
          stc
          ret                         ;retornar
finesbit:
          clc
          ret                         ;retornar

ESBIT?    ENDP

ESBYTE?   PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento determina si el pseudocodigo corresponde a un byte
;directamente direccionable, en base a los parametros generados por la
;subrutina TIPO.
```

```
;Entrada:          DL = tipo de pseudocodigo
;                  AX = dato
;Salida:           AL = codigo del byte directamente direccionable
;                  CY = 1 si el pseudocodigo no corresponde a un byte
;-----------------------------------------------------------------------
          cmp     dl,1                ;es un reg. R0 a R7?
          jne     no_reg1             ;no
          cmp     al,7
          jbe     finesbyte           ;no
no_reg1:
          cmp     dl,3                ;es memoria?
          jne     no_memo             ;no
          cmp     ah,1
          jne     erresbyte
          jmp     finesbyte
no_memo:
          cmp     dl,4                ;es dato de 8 bits?
          jne     erresbyte           ;no
          cmp     ah,0
          jne     erresbyte
          jmp     finesbyte
erresbyte:
          stc
          ret                         ;retornar
finesbyte:
          clc
          ret                         ;retornar

ESBYTE?   ENDP

ESREL?    PROC    NEAR
;-----------------------------------------------------------------------
;Este procedimiento determina si el pseudocodigo corresponde a una
;direccion en base a los parametros generados por la subrutina TIPO.
;Entrada:          DL = tipo de pseudocodigo
;                  BL,AX = dato
;Salida:           AX = codigo de la direccion
;                  CY = 1 si el pseudocodigo no corresponde a una direccion
;-----------------------------------------------------------------------
          cmp     dl,4                ;es dato?
          jb      erresrel  ;no
          je      dato_8b
          test    bl,1
          jnz     erresrel
          jmp     finesrel
dato_8b:
          test    ah,3                ;es dato de 8 bits?
          jnz     erresrel  ;no
          xor     ah,ah
          jmp     finesrel
erresrel:
          stc
          ret                         ;retornar
finesrel:
          clc
          ret                         ;retornar
```

```
ESREL?    ENDP

ESDATA?   PROC     NEAR
;-------------------------------------------------------------------------
;Este procedimiento determina si el pseudocodigo corresponde a un data(#)
;de 8 bits, en base a los parametros generados por la subrutina TIPO.
;Entrada:          DL = tipo de pseudocodigo
;                  AX = dato
;Salida:           AX = codigo del data(#)
;                  CY = 1 si el pseudocodigo no corresponde a un data(#)
;-------------------------------------------------------------------------
          cmp     dl,4
          jne     erresdata
          cmp     ah,1
          jne     erresdata
          jmp     finesdata
erresdata:
          stc
          ret                         ;retornar
finesdata:
          clc
          ret                         ;retornar

ESDATA?   ENDP

ENSPASO2  PROC     NEAR
;-------------------------------------------------------------------------
;Este procedimiento ejecuta el segundo paso de compilación, es decir, la
;traducción de los pseudocódigos a un código verdadero.
;
;Entrada:          DS:SI     apunta al pseudocódigo
;                  ES:DI     apunta al lugar donde se desea el código
;
;Salida:           DS:SI     es alterado
;                  ES:DI     es ajustado
;                  CX        longitud del código
;                  CY        set si hay error, en cuyo caso los registros
;                            quedan inalterados
;-------------------------------------------------------------------------
          push    ax                  ;preservar registros
          push    bx
          push    dx
          push    bp
          push    si
          push    di
          push    cx
          xor     cx,cx               ;longitud inicial = 0
startp2:
          call    tipo                ;determinar tipo de instruccion
          jc      er2
          test    ah,1                ;tiene parametros?
          jz      noeslon             ;si
          stosb                       ;no, generar codigo
          inc     cx                  ;instruccion de 1 byte
          jmp     bienpaso2           ;finalizar
```

```
noeslon:
          test      al,0fh              ;es un salto condicional?
          jnz       nojcond             ;no
          cmp       al,80h
          ja        nojcond
saltorel:                               ;si
          stosb                         ;generar codigo del nombre
          test      ah,2                ;solo un parametro (oner)?
          jz        no_oner             ;no
          call      tipo                ;si, es valido el parametro?
          jc        er2
          call      esrel?              ;lo siguiente es data?
          jc        er2                 ;no, error
          sub       ax,2                ;calcular distancia relativa
          call      jmpsh?              ;y determinar si el salto es posible
          jc        er2
          stosb                         ;generar codigo
          add       cx,2                ;instruccion de 2 bytes
          jmp       bienpaso2           ;finalizar
er2:
          jmp       errp2
no_oner:                                ;es una instruccion con mas de un parametro
          call      tipo                ;es valido el primer parametro?
          jc        er2                 ;no
          call      esbit?              ;si, es un bit directamente direccionable?
          jc        er2
          stosb                         ;generar codigo
          call      tipo                ;es valido el segundo parametro?
          jc        er2                 ;no
          call      esrel?              ;si, es una direccion de 8 bits?
          jc        er2                 ;no
          sub       ax,3                ;si
          call      jmpsh?              ;es un salto corto?
          jc        er2                 ;no
          stosb                         ;generar codigo
          add       cx,3                ;instruccion de 3 bytes
          jmp       bienpaso2           ;finalizar
nojcond:
          cmp       al,2                ;JMP?
          je        sijmp               ;si
          cmp       al,12h              ;CALL?
          jne       nojcall             ;no
sijmp:                                  ;si
          stosb                         ;generar codigo del nombre
          call      tipo                ;es valido el parametro?
          jc        er2                 ;no
          cmp       dl,4                ;si, es data?
          jb        nodata              ;no
          je        data8b              ;si
          test      bl,1
          jnz       er2
          jmp       continue6
data8b:
          test      ah,3
          jnz       er2
          xor       ah,ah
```

```
continue6:
          xchg      al,ah
          stosw                         ;generar codigo
          mov       cx,3                ;instruccion de 3 bytes
          jmp       bienpaso2           ;finalizar
nodata:
          cmp       byte ptr [di-1],12h
          je        er2
          cmp       dl,2                ;es registro indice?
          jne       er2                 ;no
          cmp       al,3                ;es @A+DPTR?
          jne       er2                 ;no
          mov       byte ptr [di-1],73h         ;generar codigo
          inc       cx                  ;instruccion de 1 byte
          jmp       bienpaso2           ;finalizar
er2c:
          jmp       errp2
nojcall:
          cmp       al,3                ;es RR A?
          je        rotar               ;si
          cmp       al,13h              ;es RRC A?
          je        rotar               ;si
          cmp       al,23h              ;es RL A?
          je        rotar               ;si
          cmp       al,33h              ;es RLC A?
          jne       norotar             ;no
rotar:
          stosb                         ;generar codigo del nombre
          call      tipo                ;es valido el parametro?
          jc        er2c                ;no
          cmp       dl,1                ;es el acumulador;
          jne       er2c                ;no
          cmp       al,8
          jne       er2c                ;no
          inc       cx                  ;si, instruccion de 1 byte
          jmp       bienpaso2           ;finalizar
norotar:
          cmp       al,0d5h             ;DBNZ?
          jne       nodbnz              ;no
          stosb                         ;generar codigo del nombre
          call      tipo                ;es valido el primer argumento?
          jc        er2c                ;no
          call      esbyte?             ;es un byte directamente direccionable?
          jc        no_byte1  ;no
          stosb                         ;generar codigo
          call      tipo                ;es valido el segundo parametro?
          jc        er2c                ;no
          call      esrel?              ;si, es una direccion de 8 bits?
          jc        er2c                ;no
          sub       ax,3
          call      jmpsh?              ;es un salto corto?
          jc        er2c                ;no
          stosb                         ;si, generar codigo
          add       cx,3                ;instruccion de 3 bytes
          jmp       bienpaso2           ;finalizar
no_byte1:
```

```
        cmp     dl,1                ;es un reg. R0 a R7?
        jne     er2c                ;no
        cmp     al,7
        ja      er2c                ;no
        add     al,0d8h             ;si, generar codigo
        mov     byte ptr [di-1],al
        call    tipo                ;es valido el segundo parametro?
        jc      er2c                ;no
        call    esrel?              ;si, es una direccion de 8 bits?
        jc      er2c                ;no, error
        sub     ax,2                ;si, calcular distancia relativa
        call    jmpsh?              ;y determinar si el salto es posible
        jc      er2c
        stosb                       ;generar codigo
        add     cx,2                ;instruccion de 2 bytes
        jmp     bienpaso2           ;finalizar
nodbnz:
        cmp     al,0b4h             ;CBNE?
        jne     nocbne              ;no
        stosb                       ;generar codigo del nombre
        call    tipo                ;es valido el primer parametro?
        jc      er2a                ;no
        cmp     dl,1                ;si, es registro?
        jne     no_regist           ;no
        cmp     al,8                ;si, es un registro valido?
        ja      er2a                ;no
        je      acumul              ;si, es el acumulador
        add     al,0b8h             ;generar codigo
        mov     byte ptr [di-1],al
        call    tipo                ;es valido el segundo parametro
        jc      er2a                ;no
        call    esdata?             ;si, es un numero(#) de 8 bits?
        jc      er2a                ;no
        stosb                       ;si, generar codigo
        jmp     continue1
acumul:
        call    tipo                ;es valido el segundo parametro?
        jc      er2a                ;no
        call    esdata?             ;si, es un numero(#) de 8 bits?
        jc      noesdata  ;no
        stosb                       ;generar codigo
        jmp     continue1
noesdata:
        call    esbyte?             ;es un byte directamente direccionable?
        jc      er2a                ;no
        inc     byte ptr [di-1]
        stosb                       ;generar codigo
        jmp     continue1
no_regist:
        cmp     dl,2                ;es reg. indice?
        jne     er2a                ;no
        cmp     al,1                ;si, es un registro indice valido?
        ja      er2a                ;no
        add     al,0b6h             ;si, generar codigo
        mov     byte ptr [di-1],al
        call    tipo                ;es valido el segundo parametro?
```

```
          je        reg_dptr            ;es el reg. DPTR
          cmp       al,8                ;es el acumulador?
          je        continue4           ;si
          add       al,4                ;no, generar codigo
          add       byte ptr [di-1],al
          jmp       continue4
reg_dptr:
          cmp       byte ptr [di-1],4   ;generar codigo
          jne       er2b
          mov       byte ptr [di-1],0a3h
          jmp       continue4
no_registi:
          cmp       dl,2                ;es reg. indice?
          jne        no_regind          ;no
          cmp       al,1
          ja        er2b                ;no es registro indice valido
          add       al,2                ;generar codigo
          add       byte ptr [di-1],al
          jmp       continue4
no_regind:
          call      esbyte?             ;es un byte directamente direccionable?
          jc        er2b                ;no
          stosb                         ;si, generar codigo
          inc       byte ptr [di-2]
          mov       cx,2                ;instruccion de 2 bytes
          jmp       bienpaso2           ;finalizar
continue4:
          inc       cx                  ;instruccion de 1 byte
          jmp       bienpaso2           ;finalizar
er2b:
          jmp       errp2
noincdec:
          cmp       al,24h              ;ADD?
          je        si_add              ;si
          cmp       al,34h              ;ADC?
          je        si_add              ;si
          cmp       al,94h              ;SBC?
          jne       noaddadc            ;no
si_add:
          stosb                         ;generar codigo del nombre
          call      tipo                ;es valido el primer parametro?
          jc        er2b                ;no
          cmp       dl,1                ;si, es el acumulador?
          jne       er2b                ;no, error
          cmp       al,8
          jne       er2b                ;no, error
seg_param:
          call      tipo                ;es valido el segundo parametro?
          jc        er2b                ;no
          cmp       dl,1                ;es registro?
          jne       no_regist2          ;no
          cmp       al,7
          ja        er2b                ;no es un reg. valido
          add       al,4
          add       byte ptr [di-1],al  ;generar codigo
          jmp       continue5
```

```
no_regist2:
          cmp       dl,2                ;es reg. indice?
          jne       no_regind1          ;no
          cmp       al,1
          ja        er2b                ;no es reg. ind. valido
          add       al,2
          add       byte ptr [di-1],al  ;generar codigo
          jmp       continue5
no_regind1:
          call      esdata?             ;es un numero(#) de 8 bits?
          jc        noesdata1           ;no
          stosb                         ;si, generar codigo
          jmp       bp2
noesdata1:
          call      esbyte?             ;es un byte directamente direccionable?
          jc        er2b                ;no
          stosb                         ;si, generar codigo
          inc       byte ptr [di-2]
bp2:
          mov       cx,2                ;instruccion de 2 bytes
          jmp       bienpaso2           ;finalizar
continue5:
          inc       cx                  ;instruccion de 1 byte
          jmp       bienpaso2           ;finalizar
noaddadc:
          cmp       al,42h              ;OR?
          je        si_or               ;si
          cmp       al,52h              ;AND?
          je        si_or               ;si
          cmp       al,62h              ;XOR?
          jne       noandxor  ;no
si_or:
          stosb                         ;generar codigo     del nombre
          call      tipo                ;es valido el primer parametro?
          jc        er2d                ;no
          cmp       dl,1                ;si, es el acumulador?
          jne       no_acum             ;no
          cmp       al,8
          jne       no_acum             ;no
          add       byte ptr [di-1],2   ;generar codigo
          jmp       seg_param
no_acum:
          call      esbyte?             ;es un byte directamente direccionable?
          jc        no_direct           ;no
          stosb                         ;si, generar codigo
          call      tipo                ;es valido el segundo parametro?
          jc        er2d                ;no
          cmp       dl,1                ;si, es registro?
          jne       no_acum2  ;no
          cmp       al,8
          je        bp2a                ;no es un reg. valido
no_acum2:
          call      esdata?             ;es un numero(#) de 8 bits?
          jc        er2d                ;no
          inc       byte ptr [di-2]     ;si, generar codigo
          stosb
```

```
          inc     cx                  ;instruccion de 1 byte
bp2a:
          add     cx,2                ;instruccion de 2 bytes
          jmp     bienpaso2           ;finalizar
no_direct:
          cmp     byte ptr [di-1],62h
          je      er2d
          cmp     dl,1                ;es carry?
          jne     er2d                ;no
          cmp     al,11
          jne     er2d                ;no
          call    tipo                ;es valido el segundo parametro?
          jc      er2d                ;no
          call    esbit?              ;si, es un bit directamente direccionable?
          jc      no_bit1             ;no
          stosb                       ;si, generar codigo
          mov     byte ptr [di-2],72h
          jmp     bp2a
no_bit1:
          cmp     dl,4
          jne     er2d
          cmp     ah,2
          jne     er2d
          stosb                       ;generar codigo
          mov     byte ptr [di-2],0a0h
          jmp     bp2a
er2d:
          jmp     errp2
noandxor:
          cmp     al,83h              ;LDC?
          jne     noldc               ;no
          stosb                       ;generar codigo del nombre
          call    tipo                ;es valido el primer parametro?
          jc      er2d                ;no
          cmp     dl,1                ;si, es el acumulador?
          jne     er2d                ;no
          cmp     al,8
          jne     er2d                ;no
          call    tipo                ;si, es valido el segundo parametro?
          jc      er2d                ;no
          cmp     dl,2                ;si, es reg. indice?
          jne     er2d                ;no
          cmp     al,4                ;es @A+PC?
          je      bp2b                ;si
          cmp     al,3                ;es @A+DPTR?
          jne     er2d                ;no
          mov     byte ptr [di-1],93h ;generar codigo
bp2b:
          inc     cx                  ;instruccion de 1 byte
          jmp     bienpaso2           ;finalizar
noldc:
          cmp     al,92h              ;LDB?
          jne     noldb               ;no
          stosb                       ;generar codigo del nombre
          call    tipo                ;es valido el primer parametro
          jc      er2d                ;no
```

```
        call    esbit?          ;si, es un bit directamente direccionable?
        jc      noesbit         ;no
        stosb                   ;si, generar codigo
        call    tipo            ;es valido el segundo parametro?
        jc      er2d            ;no
        cmp     dl,1            ;si, es carry?
        jne     er2d            ;no
        cmp     al,11
        jne     er2d            ;no
        inc     cx              ;instruccion de 1 byte
        jmp     bp2b
noesbit:
        cmp     dl,1            ;es carry?
        jne     er2d            ;no
        cmp     al,11
        jne     er2d            ;no
        call    tipo            ;es valido el segundo parametro?
        jc      er2d            ;no
        call    esbit?          ;si, es un bit directamente direccionable?
        jc      er2d            ;no
        stosb                   ;si, generar codigo
        mov     byte ptr [di-2],0a2h
        inc     cx              ;instruccion de 1 byte
        jmp     bp2b
er2e:
        jmp     errp2
noldb:
        cmp     al,84h          ;DIV?
        je      si_div          ;si
        cmp     al,0a4h         ;MUL?
        jne     nomuldiv        ;no
si_div:
        stosb                   ;generar codigo del nombre
        call    tipo            ;es valido el parametro?
        jc      er2e            ;no
        cmp     dl,1            ;si, es AB?
        jne     er2e            ;no
        cmp     al,10
        jne     er2e            ;no
        jmp     bp2b
nomuldiv:
        cmp     al,0b2h         ;BCOM?
        je      si_bcom         ;si
        cmp     al,0c2h         ;BCLR?
        jne     nobclr          ;no
si_bcom:
        stosb                   ;generar codigo del nombre
        call    tipo            ;es valido el parametro?
        jc      er2e            ;no
        call    esbit?          ;si, es un bit directamente direccionable?
        jc      er2e            ;no
        stosb                   ;si, generar codigo
        jmp     bp2c
nobclr:
        cmp     al,0c0h         ;PUSH?
        je      si_push         ;si
```

```
        cmp     al,0d0h             ;PULL?
        jne     nopull              ;no
si_push:
        stosb                       ;generar codigo del nombre
        call    tipo                ;es valido el parametro?
        jc      er2e                ;no
        call    esbyte?             ;si, es un byte directamente direccionable?
        jc      er2e                ;no
        stosb                       ;si, generar codigo
        jmp     bp2c
nopull:
        cmp     al,0b3h             ;COM?
        je      si_com              ;si
        cmp     al,0c3h             ;CLR?
        jne     noclr
si_com:
        stosb                       ;generar codigo del nombre
        call    tipo                ;es valido el parametro?
        jc      er2e                ;no
        cmp     dl,1                ;si, es A o CY?
        jne     er2e                ;no
        cmp     al,11               ;si, es CY?
        jne     no_cy               ;no
        jmp     bp2d
no_cy:
        cmp     al,8                ;es A?
        jne     er2e                ;no
        cmp     byte ptr [di-1],0b3h ;COM?
        je      com_a                   ;si
        mov     byte ptr [di-1],0e4h ;no, generar codigo de CLR
        jmp     bp2c
com_a:
        mov     byte ptr [di-1],0f4h ;generar codigo de COM
bp2c:
        add     cx,2                ;instruccion de 2 bytes
        jmp     bienpaso2           ;finalizar
er2f:
        jmp     errp2
noclr:
        cmp     al,0c4h             ;SWAP?
        jne     noswap              ;no
        stosb                       ;si, generar codigo
        call    tipo                ;es valido el parametro?
        jc      er2f                ;no
        cmp     dl,1                ;si, es acumulador?
        jne     er2f                ;no
        cmp     al,8
        jne     er2f                ;no
bp2d:
        inc     cx                  ;instruccion de 1 byte
        jmp     bienpaso2           ;finalizar
noswap:
        cmp     al,0c5h             ;EXH?
        jne     noexh               ;no
        stosb                       ;generar codigo del nombre
        call    tipo                ;es valido el primer parametro?
```

```
          jc      er2f              ;no
          cmp     dl,1              ;si, es el acumulador;
          jne     er2f              ;no
          cmp     al,8
          jne     er2f              ;no
          call    tipo              ;es valido el segundo parametro?
          jc      er2f              ;no
sig_cheq:
          cmp     dl,1              ;es registro?
          jne     no_regist3        ;no
          cmp     al,7
          ja      er2f              ;no es un reg. valido
          add     al,3
          add     byte ptr [di-1],al ;generar codigo
          jmp     bp2d
no_regist3:
          cmp     dl,2              ;es reg. indice?
          jne     no_regind2        ;no
          cmp     al,1
          ja      er2f              ;no es reg. ind. valido
          add     al,1
          add     byte ptr [di-1],al  ;generar codigo
          jmp     bp2d
no_regind2:
          call    esbyte?           ;es un byte directamente direccionable?
          jc      er2f              ;no
          stosb                     ;si, generar codigo
          jmp     bp2c
noexh:
          cmp     al,0d2h           ;BSET?
          jne     nobset            ;no
          stosb                     ;generar codigo del nombre
          call    tipo              ;es valido el parametro?
          jc      er2f              ;no
          call    esbit?            ;si, es un bit directamente direccionable?
          jc      noesbit1          ;no
          stosb                     ;si, generar codigo
          jmp     bp2c
noesbit1:
          cmp     dl,1              ;es carry
          jne     er2f              ;no
          cmp     al,11
          jne     er2f              ;no
          inc     byte ptr [di-1]
          jmp     bp2d
nobset:
          cmp     al,0d6h           ;EXHD?
          jne     noexhd            ;no
          stosb                     ;generar codigo del nombre
          call    tipo              ;es valido el primer parametro?
          jc      er2g              ;no
          cmp     dl,1              ;si, es el acumulador?
          jne     er2g              ;no
          cmp     al,8
          jne     er2g              ;no
          call    tipo              ;es valido el segundo parametro?
```

```
        jc      er2g                ;no
        cmp     dl,2                ;si, es reg. indice?
        jne     er2g                ;no
        cmp     al,0                ;es @R0?
        je      bp2e                ;si
        cmp     al,1                ;es @R1?
        jne     er2g                ;no
        inc     byte ptr [di-1]
bp2e:
        inc     cx                  ;instruccion de 1 byte
        jmp     bienpaso2           ;finalizar
noexhd:
        cmp     al,0e0h             ;LDX?
        jne     noldx               ;no
        stosb                       ;generar codigo del nombre
        call    tipo                ;es valido el primer parametro?
        jc      er2g                ;no
        cmp     dl,1                ;si, es acumulador?
        jne     no_acum3  ;no
        cmp     al,8
        jne     no_acum3  ;no
        call    tipo                ;es valido el segundo parametro
        jc      er2g                ;no
        cmp     dl,2                ;si, es registro indice?
        jne     er2g                ;no
        cmp     al,2
        ja      er2g                ;no es reg. indice valido
        je      esdptr1
        add     al,0e2h
        mov     byte ptr [di-1],al ;generar codigo
esdptr1:
        jmp     bp2e
no_acum3:
        cmp     dl,2                ;es registro indice?
        jne     er2g                ;no
        cmp     al,2
        ja      er2g                ;no es reg. indice valido
        je      esdptr
        add     al,0f2h
        mov     byte ptr [di-1],al  ;generar codigo
        jmp     ldx_1
esdptr:
        mov     byte ptr [di-1],0f0h ;generar codigo
ldx_1:
        call    tipo                ;es valido el segundo parametro?
        jc      er2g                ;no
        cmp     dl,1                ;si, es acumulador?
        jne     er2g                ;no
        cmp     al,8
        jne     er2g                ;no
        jmp     bp2e
er2g:
        jmp     errp2
noldx:
        cmp     al,74h              ;LDR?
        jne     er2g
```

```
          stosb                         ;generar codigo del nombre
          call      tipo                ;es valido el primer parametro?
          jc        er2g                ;no
          cmp       dl,1                ;es acumulador?
          jne       no_acum4  ;no
          cmp       al,8
          jne       no_acum4  ;no
          call      tipo                ;es valido el segundo parametro?
          jc        er2g                ;no
          call      esdata?             ;si, es un numero(#) de 8 bits?
          jc        no_data             ;no
          stosb                         ;si, generar codigo
          jmp       bp2c
no_data:
          mov       byte ptr [di-1],0e5h ;generar codigo
          jmp       sig_cheq
no_acum4:
          call      esbyte?             ;es un byte directamente direccionable?
          jc        no_byte2  ;no
          stosb                         ;si, generar codigo
          mov       byte ptr [di-2],85h
          call      tipo                ;es valido el segundo parametro?
          jc        er2g                ;no
          cmp       dl,1                ;si, es registro?
          jne       no_regist4          ;no
          cmp       al,8
          ja        er2g                ;no es un reg. valido
          je        acum
          add       al,3
          jmp       continue7
no_regist4:
          cmp       dl,2                ;es reg. indice?
          jne       no_regind3          ;no
          cmp       al,1
          ja        er2g                ;no es reg. ind. valido
          add       al,1
continue7:
          add       byte ptr [di-2],al
          jmp       bp2c
no_regind3:
          call      esbyte?             ;es un byte directamente direccionable?
          jc        no_direct1          ;no
          push      si
          push      di
          pop       si
          dec       si
          movsb                         ;generar codigo
          pop       si
          mov       byte ptr [di-2],al
bp2f:
          add       cx,3
          jmp       bienpaso2           ;finalizar
no_direct1:
          call      esdata?             ;es un numero(#) de 8 bits?
          jc        er2g                ;no
          stosb                         ;si, generar codigo
```

```
          mov     byte ptr [di-3],75h
          jmp     bp2f
acum:
          mov     byte ptr [di-2],0f5h ;generar codigo
          jmp     bp2c
no_byte2:
          cmp     dl,1                 ;es DPTR?
          jne     no_dptr              ;no
          cmp     al,9
          jne     no_dptr              ;no
          call    tipo                 ;es valido el segundo parametro?
          jc      er2i                 ;no
          call    esdata?              ;si, es un numero(#) de 8 bits?
          jc      no_data1  ;no
          xor     ah,ah                ;si
          jmp     continue8
er2i:
          jmp     errp2
no_data1:
          cmp     dl,5
          test    bl,1
          jz      errp2
continue8:
          xchg    al,ah
          stosw                        ;generar codigo
          mov     byte ptr [di-3],90h
          jmp     bp2f
er2h:
          add     sp,6
          jmp     errp2
no_dptr:
          push    ax                   ;guardar primer parametro
          push    bx
          push    dx
          call    tipo                 ;es valido el segundo parametro?
          jc      er2h                 ;no
          call    esdata?              ;si, es un numero(#) de 8 bits?
          jc      no_data2  ;no
          stosb                        ;si, generar codigo
          mov     byte ptr [di-2],76h
          inc     cx                   ;instruccion de 1 byte
          jmp     continue9
no_data2:
          call    esbyte?              ;es un byte directamente direccionable?
          jc      no_byte3  ;no
          stosb                        ;si, generar codigo
          mov     byte ptr [di-2],0a6h
          inc     cx                   ;instruccion de 1 byte
          jmp     continue9
no_byte3:
          cmp     dl,1                 ;es acumulador?
          jne     er2h                 ;no
          cmp     al,8
          jne     er2h                 ;no
          mov     byte ptr [di-1],0f6h ;generar codigo
continue9:
```

```
          pop     dx                  ;recuperar primer parametro
          pop     bx
          pop     ax
          cmp     dl,1                ;es registro?
          jne     no_regist5          ;no
          cmp     al,7
          ja      errp2               ;no es un reg. valido
          add     al,2
          jmp     finpaso2
no_regist5:
          cmp     dl,2                ;es reg. indice?
          jne     errp2               ;no
          cmp     al,1
          ja      errp2               ;no es reg. ind. valido
finpaso2:
          test    cx,1                ;es una instruccion de 1 byte?
          jz      un_byte             ;si
          add     byte ptr [di-2],al  ;no, ajustar codigo
          inc     cx
          jmp     bienpaso2           ;finalizar
un_byte:
          add     byte ptr [di-1],al  ;ajustar codigo
          inc     cx                  ;instruccion de 1 byte
          jmp     bienpaso2           ;finalizar
errp2:
          pop     cx                  ;hubo un error
          pop     di
          pop     si
          stc
          jmp     finp2
bienpaso2:
          add     sp,6
          clc                         ;no hubo error
finp2:
          pop     bp                  ;restaurar registros
          pop     dx
          pop     bx
          pop     ax
          ret                         ;retornar

ENSPASO2  ENDP

          PUBLIC  ASSEM
ASSEM     PROC    FAR
;-------------------------------------------------------------------------
;Este es el ensamblador, que llama a los pasos 1 y 2 de compilación.
;
;Entrada:          DS:SI     apunta al string por compilarse
;                  ES:DI     apunta al lugar donde se desea el código
;
;Salida:           DS:SI     es alterado
;                  ES:DI     es alterado
;                  CX        longitud del código
;                  CY        set si hay error
;-------------------------------------------------------------------------
          push    es
```

```
;*****************************************************************************
;                                    DESENS.ASM
;*****************************************************************************
;
IF1
        INCLUDE  MACDESEN.ASM
ENDIF
;
        PUBLIC   OPCODE
        EXTRN    RAM_EMB:BYTE,SEGDAT2:WORD,SEGDAT1:WORD

DSEG SEGMENT PUBLIC

;Mnemotecnicos del DS5000T:

oper00_04           db        'NOP,PJMP,JMP,RR A ,INC A,'
oper05_0f           db        'INC,'
oper10_14           db        'BBC,PCALL,CALL,RRC A,DEC A,'
oper15_1f           db        'DEC,'
oper20_23           db        'BB,RET,RL A ,'
oper24_2f           db        'ADD A,'
oper30_33           db        'BNB,RETI,RLC A,'
oper34_3f           db        'ADC A,'
oper40_43           db        'BC,OR,'
oper44_4f           db        'OR A,'
oper50_53           db        'BNC,AND,'
oper54_5f           db        'AND A,'
oper60_63           db        'BZ,XOR,'
oper64_6f           db        'XOR A,'
oper70_73           db        'BNZ,OR C,JMP,'
oper74_7f           db        'LDR,'
oper80_84           db        'BRA,AND C,LDC A,DIV AB,'
oper85_8f           db        'LDR,'
oper90_93           db        'LDR DPTR,LDB,LDC A,'
oper94_9f           db        'SBC A,'
opera0_a5           db        'OR C,LDB C,INC DPTR,MUL AB,????,'
opera6_af           db        'LDR,'
operb0_b3           db        'AND C,BCOM,COM C,'
operb4_bf           db        'CBNE,'
operc0_c4           db        'PUSH,BCLR,CLR C,SWAP A,'
operc5_cf           db        'EXH A,'
operd0_d7           db        'PULL,BSET,BSET C,DAA,DBNZ,EXHD A,'
operd8_df           db        'DBNZ,'
opere0_e4           db        'LDX A,CLR A,'
opere5_ef           db        'LDR A,'
operf0_f4           db        'LDX @DPTR,LDX,COM A,'
operf5_ff           db        'LDR,'

;Grupos de instrucciones que son desensambladas por una misma rutina:

casos               db        04H,0FH,14H,1FH,23H,2FH,33H,3FH
                    db        43H,4FH,53H,5FH,63H,6FH,73H,7FH
                    db        84H,8FH,93H,9FH,0A3H,0AFH,0B3H,0BFH
                    db        0C4H,0CFH,0D7H,0DFH,0E4H,0EFH,0F4H,0FFH

;Direcciones de las rutinas para hacer el desensamblado:

direcc              dw        offset caso00_04,  offset caso05_0f
```

```
                dw      offset caso10_14,  offset caso15_1f
                dw      offset caso20_23,  offset caso24_2f
                dw      offset caso30_33,  offset caso34_3f
                dw      offset caso40_43,  offset caso44_4f
                dw      offset caso50_53,  offset caso54_5f
                dw      offset caso60_63,  offset caso64_6f
                dw      offset caso70_73,  offset caso74_7f
                dw      offset caso80_84,  offset caso85_8f
                dw      offset caso90_93,  offset caso94_9f
                dw      offset casoa0_a5,  offset casoa6_af
                dw      offset casob0_b3,  offset casob4_bf
                dw      offset casoc0_c4,  offset casoc5_cf
                dw      offset casod0_d7,  offset casod8_df
                dw      offset casoe0_e4,  offset casoe5_ef
                dw      offset casof0_f4,  offset casof5_ff

index_reg       db      '@R0,@R1,@A+DPTR,@A+PC,@DPTR,'          ;reg. indices
reg             db      'R0,R1,R2,R3,R4,R5,R6,R7,'              ;registros

registros       db      'R0,R1,R2,R3,R4,R5,R6,R7,'              ;bancos de
                db      "R0',R1',R2',R3',R4',R5',R6',R7',"      ;registros
                db      'R0",R1",R2",R3",R4",R5",R6",R7",'
                db      'R0\,R1\,R2\,R3\,R4\,R5\,R6\,R7\,'

;Registros de funciones especiales del DS5000T:

sfr             db      80h,'P0',81h,'SP',82h,'DPL',83h,'DPH'
                db      87h,'PCON',88h,'TCON',89h,'TMOD',8Ah,'TL0'
                db      8Bh,'TL1',8Ch,'TH0',8Dh,'TH1',90h,'P1'
                db      98h,'SCON',99h,'SBUF',0A0h,'P2',0A8h,'IE'
                db      0B0h,'P3',0B8h,'IP',0C1h,'EK0',0C2h,'EK1'
                db      0C3h,'EK2',0C4h,'EK3',0C5h,'EK4',0C6h,'MCON'
                db      0C7h,'TA',0D0h,'PSW',0E0h,'ACC',0F0h,'B'
                db      '$'

;Bits directamente direccionables:

bits            db      'P0.0,P0.1,P0.2,P0.3,P0.4,P0.5,P0.6,P0.7,'
                db      'IT0,IE0,IT1,IE1,TR0,TF0,TR1,TF1,'
                db      'P1.0,P1.1,P1.2,P1.3,P1.4,P1.5,P1.6,P1.7,'
                db      'RI,TI,RB8,TB8,REN,SM2,SM1,SM0,'
                db      'P2.0,P2.1,P2.2,P2.3,P2.4,P2.5,P2.6,P2.7,'
                db      'EX0,ET0,EX1,ET1,ES,IE.5,IE.6,EA,'
                db      'RXD,TXD,INT0,INT1,T0,T1,WR,RD,'
                db      'PX0,PT0,PX1,PT1,PS,IP.5,IP.6,RWT,'

bits1           db      0D0H,'P',0D1H,'PSW.1',0D2H,'OV',0D3H,'RS0'
                db      0D4H,'RS1',0D5H,'F0',0D6H,'AC',0D7H,'CY'
                db      0E0H,'ACC.0',0E1H,'ACC.1',0E2H,'ACC.2',0E3H,'ACC.3'
                db      0E4H,'ACC.4',0E5H,'ACC.5',0E6H,'ACC.6',0E7H,'ACC.7'
                db      0F0H,'B.0',0F1H,'B.1',0F2H,'B.2',0F3H,'B.3'
                db      0F4H,'B.4',0F5H,'B.5',0F6H,'B.6',0F7H,'B.7'
                db      '$'

string          db      5 dup (?)
Sdata           db      5 dup (?)
```

```
Srel                db      5 dup (?)
Opcode              db      3 dup (0)
sdata?              db      0
string?             db      0
srel?               db      0
regsi               dw      0
cksum               db      0

;Modos de direccionamiento del DS5000T:

modos1              db      02,03,04,05,07,0fh

dir_modos1          dw      offset direct,      offset direct_data
                    dw      offset data,        offset direct
                    dw      offset indexado,    offset registro

modos2              db      04,05,07,0fh

dir_modos2          dw      offset data,        offset direct_data
                    dw      offset index_data,  offset reg_data

modos3              db      05,07,0fh

dir_modos3          dw      offset data_rel,    offset index_data_rel
                    dw      offset reg_data_rel

DSEG ENDS

cr                  equ     0dh     ;carry return
lf                  equ     0ah     ;line feed

          EXTRN     ASCAHEX:FAR
CSEG SEGMENT
ASSUME CS:CSEG,DS:DSEG,ES:NOTHING

          PUBLIC    DESENS
DESENS    PROC      NEAR
;-----------------------------------------------------------------
;Este es el procedimiento principal. Genera un string desde ES:DI
;terminado en 0, desensamblado de la instruccion apuntada por DS:SI.
;Preserva todos los registros que no transfieren parametros.
;Entrada:           DS:SI apunta al codigo de maquina de la instruccion
;                   ES:DI apunta al lugar donde ira la instruccion
;                       desensamblada
;Salida:  cadena de caracteres a partir de [ES:DI]
;                   CX =  longitud de la instruccion
;-----------------------------------------------------------------
          pushf                         ;Preservar banderas y registros
          push      bx
          push      ds
          push      si
          push      di

          cld
          mov       ax,dseg
          push      es
```

```
          push    di
          mov     es,ax               ;ES apunta a DSEG
          push    ds
          mov     ds,segdat2          ;DS apunta a DSEG2
          mov     di,offset opcode    ;DI apunta a un buffer para
          mov     cx,3                ;3 bytes de codigo
          rep     movsb               ;transferir instruccion desde DSEG2 a DSEG
          pop     ds                  ;DS apunta a DSEG
          pop     di                  ;DI = lugar donde ira la instr.desens.
          pop     es                  ;ES apunta a DSEG
          mov     si,offset opcode    ;SI apunta al codigo de la instr.
          mov     al,[si]             ;leer primer byte de la instruccion
          and     al,1fh
          cmp     al,11h              ;es PCALL?
          je      proc1               ;si
          cmp     al,01h              ;no, es PJMP?
          je      proc2               ;si
          mov     al,[opcode]         ;leer primer byte de la instruccion
          jmp     short siga
proc1:    p_inst  oper10_14           ;desensamblar instruccion PCALL
          mov     cx,2                ;es de 2 bytes
          jmp     short findes        ;finalizar
proc2:    p_inst  oper00_04           ;desensamblar instruccion PJMP
          mov     cx,2                ;es de 2 bytes
          jmp     short findes        ;finalizar
siga:     push    di                  ;preservar DI
          mov     di,offset casos     ;escoger la rutina apropiada para
          xor     bx,bx               ;desensamblar la instruccion
buscop:   scasb                       ;se la encontro?
          jbe     encontrado          ;si
          inc     bx                  ;no, continuar la busqueda
          jmp     short buscop
encontrado:       pop       di        ;restaurar DI
          shl     bx,1
          add     bx,offset direcc
          xor     cx,cx
          call    [bx]                ;llamar a esa subrutina
findes:
          pop     di                  ;restaurar registros
          pop     si
          pop     ds
          pop     bx
          popf
          ret                         ;retornar

DESENS    ENDP

          PUBLIC  DESENSDET
DESENSDET         PROC      FAR
;-------------------------------------------------------------------------
;Este procedimiento genera un string a partir del codigo de maquina apuntado
;por DS:SI, desde ES:DI, terminado en 0, consistente en:
;         - cuatro caracteres que representan al valor inicial de SI en hex.
;         - Cinco carateres del mnemotecnico de la instruccion.
;         - Once caracteres de parametros del mnemotecnico.
;Entrada:          DS:SI apunta al codigo de maquina de la instruccion
```

```
;                   ES:DI apunta al lugar donde ira la cadena de caracteres
;                      generada
;Salida:  cadena de caracteres a partir de [ES:DI]
;                   CX =  longitud de la instruccion
;-------------------------------------------------------------------------
          push      di                  ;preservar registros
          push      es
          mov       ax,dseg
          mov       es,ax               ;ES apunta al segmento DSEG
          cld
          mov       ax,si               ;calcular el valor de PC donde se inicia
          sub       ax,offset ram_emb   ;la instruccion
          mov       word ptr regsi,ax
          push      ax                  ;generar 4 caracteres que representan el
          mov       al,ah               ;valor de PC donde se inicia la instruccion,
          call      convhex             ;en hexadecimal
          pop       ax
          dec       di
          call      convhex
          dec       di
          call      desens              ;desensamblar la instruccion
          push      ax                  ;preservar registros y formatear la cadena
          push      bx                  ;de caracteres para presentarla en pantalla
          push      cx
          push      dx
          push      si
          push      ds
          mov       bx,di               ;DI=comienzo del mnemotecnico
          mov       cx,5                ;buscar un 0 hasta el quinto caracter
          mov       al,0
          repne     scasb
          jne       conparam            ;si no se lo encuentra, hay parametros
          add       cx,14               ;poner 14 espacios al final del unico mnemo
          mov       al,' '
          dec       di
          rep       stosb
          mov       al,0                ;poner un 0 al final
          stosb
          jmp       findesdet
conparam:
          push      bx
          mov       di,bx               ;recuperar el comienzo del mnemotecnico
          mov       cx,6                ;buscar un espacio
          mov       al,' '
          repne     scasb
          cmp       cx,1                ;en que posicion esta el espacio?
          Jb        sexta               ;si CX = 0, sexta posicion
          je        ponspcfin           ;si CX = 1, quinta posicion
          mov       cx,di
          mov       ax,es               ;ci CX >= 2, cuarta o menos.
          Mov       ds,ax               ;en este caso, mover los parametros a la
          sub       di,bx               ;derecha
          mov       dx,5
          sub       dx,di               ;DX = posiciones por mover a la derecha
          mov       di,bx
          mov       bx,cx               ;BX = comienzo de los parametros
```

```
          mov       cx,24
          mov       al,0
          repne     scasb               ;buscar el fin de string
          dec       di
          mov       si,di
          add       di,dx
          std                           ;direccion inversa
cicder:   lodsb
          stosb
          cmp       si,ox               ;se llego al comienzo de los parametros?
          jae       cicder              ;mover el string
          mov       al,' '
cicspc:   cmp       di,si
          je        ponspcfin
          stosb
          jmp       short cicspc
sexta:    mov       ax,es               ;mover los parametros una posicion a la izq.
          mov       ds,ax
          mov       si,di
          dec       di
cicizq:   lodsb
          stosb
          cmp       al,0                ;fin del string?
          jne       cicizq              ;no, continuar moviendo a la izquierda
ponspcfin:        cld                   ;incrementar punteros en instr. con strings
          pop       bx
          mov       di,bx
          mov       cx,20               ;buscar el fin del string
          mov       al,0
          repne     scasb
          dec       di
          mov       al,' '              ;completar formato con espacios
          rep       stosb
          write     0                   ;indicar fin de la instruccion desensamblada
findesdet:
          pop       ds                  ;restaurar registros
          pop       si
          pop       dx
          pop       cx
          pop       bx
          pop       ax
          pop       es
          pop       di
          ret                           ;retornar

DESENSDET         ENDP
;
;Fin de procedimiento
;
;Procedimientos adicionales
;
          SUBRUT                        ;Macro que  contiene las subrutinas usadas en
                                        ;este modulo
```

```
;-------------------------------------------------------------------------
;Cada una de las siguientes rutinas desensambla un grupo de instrucciones
;del DS5000T como se explica en el modulo MACDESEN.ASM
;-------------------------------------------------------------------------
;
CAS000_04           PROC      NEAR
          caso_0    0,2,3,oper00_04
CAS000_04           ENDP


CAS005_0F           PROC      NEAR
          caso_1    oper05_0f,' '
CAS005_0F           ENDP


CAS010_14           PROC      NEAR
          caso_00   10h,12h,13h,oper10_14
CAS010_14           ENDP


CAS015_1F           PROC      NEAR
          caso_1    oper15_1f,' '
CAS015_1F           ENDP


CAS020_23           PROC      NEAR
          caso_2    20h,22h,oper20_23
CAS020_23           ENDP


CAS024_2F           PROC      NEAR
          caso_3    oper24_2f
CAS024_2F           ENDP


CAS030_33           PROC      NEAR
          caso_2    30h,32h,oper30_33
CAS030_33           ENDP


CAS034_3F           PROC      NEAR
          caso_3    oper34_3f
CAS034_3F           ENDP


CAS040_43           PROC      NEAR
          caso_4    40h,oper40_43
CAS040_43           ENDP


CAS044_4F           PROC      NEAR
          caso_3    oper44_4f
CAS044_4F           ENDP


CAS050_53           PROC      NEAR
```

```
          caso_4    50h,oper50_53
CASO50_53           ENDP


CASO54_5F           PROC      NEAR
          caso_3    oper54_5f
CASO54_5F           ENDP


CASO60_63           PROC      NEAR
          caso_4    60h,oper60_63
CASO60_63           ENDP


CASO64_6F           PROC      NEAR
          caso_3    oper64_6f
CASO64_6F           ENDP


CASO70_73           PROC      NEAR
          caso_5    70h,72h,oper70_73,' ',2
CASO70_73           ENDP


CASO74_7F           PROC      NEAR
          caso_6    oper74_7f
CASO74_7F           ENDP


CASO80_84           PROC      NEAR
          caso_7    80h,82h,83h,oper80_84,',',3
CASO80_84           ENDP


CASO85_8F           PROC      NEAR
          caso_8    oper85_8f
CASO85_8F           ENDP


CASO90_93           PROC      NEAR
          caso_9    90h,92h,oper90_93,',',2
CASO90_93           ENDP


CASO94_9F           PROC      NEAR
          caso_3    oper94_9f
CASO94_9F           ENDP


CASOA0_A5           PROC      NEAR
          caso_a    0a0h,0a2h,0a3h,0a4h,opera0_a5
CASOA0_A5           ENDP


CASOA6_AF           PROC      NEAR
          caso_b    opera6_af
```

```
CASOA6_AF           ENDP


CASOB0_B3           PROC      NEAR
          caso_c    0b0h,0b2h,operb0_b3
CASOB0_B3           ENDP


CASOB4_BF           PROC      NEAR
          caso_d    operb4_bf
CASOB4_BF           ENDP


CASOC0_C4           PROC      NEAR
          caso_e    0c0h,0c2h,0c3h,operc0_c4
CASOC0_C4           ENDP


CASOC5_CF           PROC      NEAR
          caso_1    operc5_cf,',',
CASOC5_CF           ENDP


CASOD0_D7           PROC      NEAR
          caso_f    0d0h,0d2h,0d3h,0d4h,0d5h,operd0_d7
CASOD0_D7           ENDP


CASOD8_DF           PROC      NEAR
          caso_b    operd8_df
CASOD8_DF           ENDP


CASOE0_E4           PROC      NEAR
          caso_10   0e0h,0e3h,opere0_e4,',',4
CASOE0_E4           ENDP


CASOE5_EF           PROC      NEAR
          caso_11   opere5_ef
CASOE5_EF           ENDP


CASOF0_F4           PROC      NEAR
          caso_12   0f0h,0f3h,operf0_f4
CASOF0_F4           ENDP


CASOF5_FF           PROC      NEAR
          caso_13   operf5_ff
CASOF5_FF           ENDP


CSEG ENDS
;Fin de Codigo de programa
```

```
;*****************************************************************************
;                              MACDESEN.ASM
;*****************************************************************************
;
;-----------------------------------------------------------------------------
;Este archivo contiene los macros y subrutinas utilizados en el modulo
;DESENS.ASM para desensamblar las instrucciones del DS5000T
;-----------------------------------------------------------------------------
;
CASO_0    MACRO    N1,N2,N3,DIR
          LOCAL    OPER0,OPER1,OPER2,FIN
;-----------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - NOP; JMP addr16; RR A; INC A
;Parametros:       N1, N2, N3 primeros bytes de codigo de las instrucciones
;                             que se quiere desensamblar
;                  DIR = lugar del segmento de datos donde se encuentran
;                        los mnemonicos de las instrucciones a desensamblar
;-----------------------------------------------------------------------------
          cmp      al,n1              ;es la 1a. instruccion del grupo?
          je       oper0              ;si, desensamblarla
          inc      cx                 ;no, CX apunta al mnemonico adecuado
          inc      cx
          cmp      al,n2              ;es la 2a. instruccion del grupo?
          je       oper1              ;si, desensamblarla
          inc      cx                 ;no, CX apunta al mnemonico adecuado
          cmp      al,n3              ;es la 3a. instruccion del grupo?
          je       oper2              ;si, desensamblarla
          inc      cx                 ;no, CX apunta al mnemonico adecuado
          desens2  dir                ;desensamblar la 4a. instruccion del grupo
          jmp      fin                ;finalizar
oper0:    leer     dir                ;desensamblar la 1a. instruccion del grupo
          mov      cx,1               ;el codigo es de 1 byte
          jmp      fin                ;finalizar
oper1:    desens1  dir                ;desensamblar la 2a. instruccion del grupo
          jmp      fin                ;finalizar
oper2:    desens2  dir                ;desensamblar la 3a. instruccion del grupo
          jmp      fin                ;finalizar
fin:      write    0                  ;finalizar
          ret                         ;retornar

          ENDM

CASO_00   MACRO    N1,N2,N3,DIR
          LOCAL    OP1,OP2,OPER0,OPER1,OPER2,FIN
;-----------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - BBC bit,rel; CALL addr16; RRC A; DEC A
;Parametros:       N1, N2, N3 primeros bytes de codigo de las instrucciones
;                             que se quiere desensamblar
;                  DIR = lugar del segmento de datos donde se encuentran
;                        los mnemonicos de las instrucciones a desensamblar
;-----------------------------------------------------------------------------
          cmp      al,n1              ;es la 1a. instruccion del grupo?
          je       oper0              ;si, desensamblarla
          inc      cx                 ;no, CX apunta al mnemonico adecuado
          inc      cx
          cmp      al,n2              ;es la 2a. instruccion del grupo?
```

```
          je        op1                 ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n3               ;es la 3a. instruccion del grupo?
          je        op2                 ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          desens2   dir                 ;desensamblar la 4a. instruccion del grupo
          jmp       fin                 ;finalizar
op1:      jmp       oper1
op2:      jmp       oper2
oper0:    desens10  dir                 ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
oper1:    desens1   dir                 ;desensamblar la 2a. instruccion del grupo
          jmp       fin                 ;finalizar
oper2:    desens2   dir                 ;desensamblar la 3a. instruccion del grupo
          jmp       fin                 ;finalizar
fin:      write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_1    MACRO     DIR,CAR
;-------------------------------------------------------------------------
;Este macro permite desensamblar los siguientes grupos de instrucciones:
;         - INC direct; INC @Ri; INC Rn
;         - DEC direct; DEC @Ri; DEC Rn
;         - EXH A,direct; EXH A,@Ri; EXH A,Rn
;Parametros:       DIR = lugar del segmento de datos donde se encuentran
;                        los mnemonicos de las instrucciones a desensamblar
;                  CAR = caracter desensamblada
;-------------------------------------------------------------------------
          leer      dir                 ;obtener el mnemonico
          write     car                 ;obtener un parametro
          mov       al,[opcode]         ;leer el primer byte
          det_modo_dir modos1,dir_modos1 ;modo de direccionamiento
          transfer string               ;transferir un parametro
          write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_2    MACRO     N1,N2,DIR
          LOCAL     OPER0,OPER1,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar los siguientes grupos de instrucciones:
;         - BB  bit,rel; RET ; RL A
;         - BNB bit,rel; RETI; RLC A
;Parametros:       N1, N2 primeros bytes de codigo de las instrucciones que se
;                         quiere desensamblar
;                  DIR = lugar del segmento de datos donde se encuentran
;                        los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          cmp       al,n1               ;es la 1a. instruccion del grupo?
          je        oper0               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n2               ;es la 2a. instruccion del grupo?
          je        oper1               ;si, desensamblarla
```

```
          inc       cx                  ;no, CX apunta al mnemonico adecuado
oper1:    desens2   dir                 ;desensamblar 2a. o 3a. instr. del grupo
          jmp       fin                 ;finalizar
oper0:    desens10 dir                  ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
fin:      write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_3    MACRO     DIR
          LOCAL     NO_DIRECT,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar los siguientes grupos de instrucciones:
;         - ADD A,#data; ADD A,direct; ADD A,@Ri; ADD A,Rn
;         - ADC A,#data; ADC A,direct; ADC A,@Ri; ADC A,Rn
;         - OR  A,#data; OR  A,direct; OR  A,@Ri; OR  A,Rn
;         - AND A,#data; AND A,direct; AND A,@Ri; AND A,Rn
;         - XOR A,#data; XOR A,direct; XOR A,@Ri; XOR A,Rn
;         - SBC A,#data; SBC A,direct; SBC A,@Ri; SBC A,Rn
;Parametros:       DIR = lugar del segmento de datos donde se encuentran
;                        los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          mov       string?,0
          leer      dir                 ;obtener el mnemonico
          write     ','                 ;escribir una coma entre parametros
          mov       al,[opcode]         ;leer el primer byte
          det_modo_dir modos1,dir_modos1 ;modo de direccionamiento
          cmp       string?,0           ;direccionamiento directo?
          je        no_direct           ;no
          transfer  string              ;si, transferir un parametro
          jmp       fin                 ;finalizar
no_direct:
          write     '#'                 ;escribir el simbolo '#'
          transfer  sdata               ;transferir un parametro
fin:      write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_4    MACRO     N1,DIR
          LOCAL     OPER0,ACUM,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar los siguientes grupos de instrucciones:
;         - BC  rel; OR  direct,A; OR  direct,#data
;         - BNC rel; AND direct,A; AND direct,#data
;         - BZ  rel; XOR direct,A; XOR direct,#data
;Parametros:       N1 primeros bytes de codigo de las instrucciones
;                     que se quiere desensamblar
;                  DIR = lugar del segmento de datos donde se encuentran
;                        los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          mov       sdata?,0
          cmp       al,n1               ;es la 1a. instruccion del grupo?
          je        oper0               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
```

```
          leer      dir                 ;desensamblar 2a. o 3a. instr. del grupo
          write     ' '                 ;poner un espacio
          mov       al,[opcode]         ;leer el primer byte
          det_modo_dir modos1,dir_modos1 ;modo de direccionamiento
          transfer string               ;transferir un parametro
          write     ','                 ;escribir una coma entre los parametros
          cmp       sdata?,0            ;es el segundo parametro #data?
          je        acum                ;no, es el acumulador
          write     '#'                 ;si, escribir el simbolo '#'
          mov       al,[opcode+2]       ;leer el tercer byte
          call      convhex             ;escribirlo como parametro
          jmp       fin                 ;finalizar
oper0:    desens3   dir                 ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
acum:     write     'A'                 ;escribir la letra A como segundo parametro
fin:      write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_5    MACRO     N1,N2,DIR,CAR,N3
          LOCAL     OPER0,OPER1,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - BNZ rel; OR C,bit; JMP @A+DPTR
;Parametros:  N1, N2 primeros bytes de codigo de las instrucciones
;                    que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;             CAR = caracter desensamblada
;             N3  = lugar que ocupa en una tabla un parametro de la instr.
;-------------------------------------------------------------------------
          cmp       al,n1               ;es la 1a. instruccion del grupo?
          je        oper0               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n2               ;es la 2a. instruccion del grupo?
          je        oper1               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          desens6   dir,car,n3          ;desensamblar la 3a. instruccion del grupo
          mov       cx,1                ;el codigo es de 1 byte
          jmp       fin                 ;finalizar
oper0:    desens3   dir                 ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
oper1:    desens5   dir                 ;desensamblar la 2a. instruccion del grupo
fin:      write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_6    MACRO     DIR
          LOCAL     NO_DIRECT,CONT,CONT1,SIGA
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - LDR A,#data; LDR direct,#data; LDR @Ri,#data; LDR Rn,#data
;Parametros:  DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
```

```
;-----------------------------
          mov       string?,0
          leer      dir                 ;obtener el mnemonico
          write     ' '                 ;poner un espacio
          mov       al,[opcode]         ;leer el primer byte
          det_modo_dir modos2,dir_modos2 ;modo de direccionamiento
          cmp       string?,0           ;direccionamiento directo?
          je        no_direct           ;no
          transfer  string              ;si, transferir el primer parametro
          jmp       siga
no_direct:
          write     'A'       ;escribir la letra A como primer parametro
siga:     write     ','                 ;escribir una coma entre parametros
          write     '#'                 ;escribir el simbolo '#'
          cmp       byte ptr [opcode],75h       ;es LDR direct,#data?
          jne       cont      ;no
          mov       al,[si+2]           ;si, leer el tercer byte
          call      convhex             ;escribirlo como segundo parametro
          jmp       cont1
cont:     transfer  sdata               ;transferir el segundo parametro
cont1:    write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_7    MACRO     N1,N2,N3,DIR,CAR,N4
          LOCAL     OPER0,OPER1,OPER2,OP2,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo-de instrucciones:
;         - BRA rel; AND C,bit; LDC A,@A+PC; DIV AB
;Parametros:  N1, N2, N3 primeros bytes de codigo de las instrucciones
;                        que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;             CAR = caracter desensamblada
;             N4  = lugar que ocupa en una tabla un parametro de la instr.
;-------------------------------------------------------------------------
          cmp       al,n1               ;es la 1a. instruccion del grupo?
          je        oper0               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n2               ;es la 2a. instruccion del grupo?
          je        oper1               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n3               ;es la 3a. instruccion del grupo?
          je        op2                 ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          desens2   dir                 ;desensamblar la 4a. instruccion del grupo
          jmp       fin                 ;finalizar
op2:      jmp       oper2
oper0:    desens3   dir                 ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
oper1:    desens5   dir                 ;desensamblar la 2a. instruccion del grupo
          jmp       fin                 ;finalizar
oper2:    desens6   dir,car,n4          ;desensamblar la 3a. instruccion del grupo
          mov       cx,1                ;el codigo es de 1 byte
          jmp       fin                 ;finalizar
```

```
fin:      write    0
          ret                          ;retornar

          ENDM

CASO_8    MACRO    DIR
          LOCAL    ORDEN,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - LDR direct1,direct2; LDR direct,@Ri; LDR direct,Rn
;Parametros:  DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          leer     dir                 ;obtener el mnemonico
          write    ' '                 ;poner un espacio
          mov      al,[opcode]         ;leer el primer byte
          det_modo_dir modos2,dir_modos2 ;modo de direccionamiento
          mov      al,[opcode]         ;leer el primer byte
          cmp      al,85h              ;es LDR direct1,direct2
          je       orden               ;si
          push     di                  ;no
          mov      al,[opcode+1]       ;leer el segundo byte
          mov      di,offset sdata
          trans_byte                   ;1er. parametro tiene nombre especial?
          pop      di
          transfer sdata               ;transferir el primer parametro
          write    ','                 ;escribir una coma entre parametros
          transfer string              ;transferir el segundo parametro
          jmp      fin                 ;finalizar
orden:    push     di
          mov      al,[opcode+2]       ;leer el tercer byte
          mov      di,offset sdata
          trans_byte                   ;1er. parametro tiene nombre especial?
          pop      di
          transfer sdata               ;transferir el primer parametro
          write    ','                 ;escribir una coma entre parametros
          transfer string              ;transferir el segundo parametro
fin:      write    0                   ;finalizar
          ret                          ;retornar

          ENDM

CASO_9    MACRO    N1,N2,DIR,CAR,N3
          LOCAL    OPER0,OPER1,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - LDR DPTR,#data16; LDB bit,C; LDC A,@A+DPTR
;Parametros:  N1, N2 primeros bytes de codigo de las instrucciones
;                       que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;             CAR = caracter desensamblada
;             N3  = lugar que ocupa en una tabla un parametro de la instr.
;-------------------------------------------------------------------------
          cmp      al,n1               ;es la 1a. instruccion del grupo?
          je       oper0               ;si, desensamblarla
```

```
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n2               ;es la 2a. instruccion del grupo?
          je        oper1               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          desens6   dir,car,n3          ;desensamblar la 3a. instruccion del grupo
          mov       cx,1                ;el codigo es de 1 byte
          jmp       fin                 ;finalizar
oper0:    desens1   dir                 ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
oper1:    desens4   dir                 ;desensamblar la 2a. instruccion del grupo
          write     ','                 ;escribir una coma entre parametros
          write     'C'                 ;escribir la letra C como segundo parametro
          mov       cx,2                ;el codigo es de 2 bytes
          jmp       fin                 ;finalizar
fin:      write     0
          ret                           ;retornar

          ENDM

CASO_A    MACRO     N1,N2,N3,N4,DIR
          LOCAL     OPER0,OPER1,OPER2,OP1,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - OR C,/bit; LDB C,bit; INC DPTR; MUL AB
;Parametros:  N1, N2, N3, N4 primeros bytes de codigo de las instrucciones
;                         que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          cmp       al,n1               ;es la 1a. instruccion del grupo?
          je        oper0               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n2               ;es la 2a. instruccion del grupo?
          je        op1                 ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n3               ;es la 3a. instruccion del grupo?
          je        oper2               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n4               ;es la 4a. instruccion del grupo?
          je        oper2               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
oper2:    desens2   dir                 ;desensamblar 3a. o 4a. instr. del grupo
          jmp       fin                 ;finalizar
op1:      jmp       oper1
oper0:    desens7   dir                 ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
oper1:    desens5   dir                 ;desensamblar la 2a. instruccion del grupo
          mov       cx,2                ;el codigo es de 2 bytes
          jmp       fin                 ;finalizar
fin:      write     0
          ret                           ;retornar

          ENDM

CASO_B    MACRO     DIR
          LOCAL     REL,FIN
```

```
;-------------------------------------------------------------------------
;Este macro permite desensamblar los siguientes grupos de instrucciones:
;         - LDR @Ri,direct; LDR Rn,direct
;         - DBNZ Rn,rel
;Parametros:  DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          leer     dir                 ;obtener el mnemonico
          write    ' '                 ;poner un espacio
          mov      al,[opcode]         ;leer el primer byte
          det_modo_dir modos2,dir_modos2 ;modo de direccionamiento
          transfer string              ;transferir el primer parametro
          write    ','                 ;escribir una coma entre parametros
          mov      al,[opcode]         ;leer el primer byte
          cmp      al,0afh             ;es DBNZ Rn,rel?
          ja       rel                 ;si
          mov      al,[opcode+1]       ;no, leer el segundo byte
          push     di
          mov      di,offset string
          trans_byte                   ;2do. parametro tiene nombre especial?
          pop      di
          transfer string              ;transferir el segundo parametro
          jmp      fin                 ;finalizar
rel:      mov      al,[opcode+1]
          call     realadr
fin:      write    0                   ;finalizar
          ret                          ;retornar

          ENDM

CASO_C    MACRO    N1,N2,DIR
          LOCAL    OPER0,OPER1,OP1,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - AND C,/bit; BCOM bit; COM C
;Parametros:  N1, N2 primeros bytes de codigo de las instrucciones
;                    que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          cmp      al,n1               ;es la 1a. instruccion del grupo?
          je       oper0               ;si, desensamblarla
          inc      cx                  ;no, CX apunta al mnemonico adecuado
          cmp      al,n2               ;es la 2a. instruccion del grupo?
          je       op1                 ;si, desensamblarla
          inc      cx                  ;no, CX apunta al mnemonico adecuado
          desens2  dir                 ;desensamblar la 3a. instruccion del grupo
          jmp      fin                 ;finalizar
op1:      jmp      oper1
oper0:    desens7  dir                 ;desensamblar la 1a. instruccion del grupo
          jmp      fin                 ;finalizar
oper1:    desens0  dir                 ;desensamblar la 2a. instruccion del grupo
          jmp      fin                 ;finalizar
fin:      write    0
          ret                          ;retornar
```

```
          ENDM

CASO_D    MACRO     DIR
          LOCAL     NO_ACUM,CONT,ACUM,CONT1
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - CBNE A,#data,rel; CBNE A,direct,rel; CBNE @Ri,#data,rel
;           CBNE Rn,#data,rel
;Parametros:  DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          leer      dir                 ;obtener el mnemonico
          write     ' '                 ;escribir un espacio
          mov       al,[opcode]         ;leer el primer byte
          det_modo_dir modos3,dir_modos3 ;modo de direccionamiento
          cmp       al,0b5h             ;es el primer parametro el acumulador
          ja        no_acum             ;no
          je        acum                ;si
          write     'A'                 ;escribir la letra A como primer parametro
          jmp       cont
acum:     write     'A'                 ;escribir la letra A como primer parametro
          write     ','                 ;escribir una coma entre parametros
          push      di
          mov       al,[opcode+1]       ;leer el segundo byte
          mov       di,offset string
          trans_byte                    ;2do. parametro tiene nombre especial?
          pop       di
          transfer  string              ;transferir el segundo parametro
          jmp       cont1
no_acum:
          transfer  string              ;transferir el primer parametro
cont:     write     ','                 ;escribir una coma entre parametros
          write     '#'                 ;escribir el simbolo #
          transfer  sdata               ;transferir el segundo parametro
cont1:    write     ','                 ;escribir una coma entre parametros
          transfer  srel                ;transferir el tercer parametro
          write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_E    MACRO     N1,N2,N3,DIR
          LOCAL     OPER0,OPER1,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - PUSH direct; BCLR bit; CLR C; SWAP A
;Parametros:  N1, N2, N3 primeros bytes de codigo de las instrucciones
;                        que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          cmp       al,n1               ;es la 1a. instruccion del grupo?
          je        oper0               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n2               ;es la 2a. instruccion del grupo?
          je        oper2               ;si, desensamblarla
```

```
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n3               ;es la 3a. instruccion del grupo?
          je        oper1               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
oper1:    desens2   dir                 ;desensamblar 3a. o 4a. instr. del grupo
          jmp       fin                 ;finalizar
oper0:    desens4   dir                 ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
oper2:    desens0   dir                 ;desensamblar la 2a. instruccion del grupo
fin:      write     0                   ;finalizar
          ret                           ;retornar

          ENDM

CASO_F    MACRO     N1,N2,N3,N4,N5,DIR
          LOCAL     OPER0,OPER1,OPER2,OPER3,OP1,OP2,OP3,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;          - PULL direct; BSET bit; BSET C; DAA; DBNZ direct,rel
;            EXHD A,@Ri
;Parametros:  N1, N2, N3, N4, N5 primeros bytes de codigo de las
;                                instrucciones que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          cmp       al,n1               ;es la 1a. instruccion del grupo?
          je        oper0               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n2               ;es la 2a. instruccion del grupo?
          je        op3                 ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n3               ;es la 3a. instruccion del grupo?
          je        op1                 ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n4               ;es la 4a. instruccion del grupo?
          je        op1                 ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n5               ;es la 5a. instruccion del grupo?
          je        op2                 ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          desens8   dir                 ;desensamblar la sexta instruccion del grupo
          jmp       fin                 ;finalizar
op1:      jmp       oper1
op2:      jmp       oper2
op3:      jmp       oper3
oper0:    desens4   dir                 ;desensamblar la 1a. instruccion del grupo
          jmp       fin                 ;finalizar
oper1:    desens2   dir                 ;desensamblar 3a. o 4a. instr. del grupo
          jmp       fin                 ;finalizar
oper2:    desens11  dir                 ;desensamblar la 5a. instruccion del grupo
          jmp       fin                 ;finalizar
oper3:    desens0   dir                 ;desensamblar la 2a. instruccion del grupo
fin:      write     0                   ;finalizar
          ret                           ;retornar

          ENDM
```

```
CASO_10   MACRO    N1,N2,DIR,CAR,N3
          LOCAL    OPER0,OPER1,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - LDX A,@DPTR; LDX A,@Ri; CLR A
;Parametros:  N1, N2 primeros bytes de codigo de las instrucciones
;                 que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
;                 los mnemonicos de las instrucciones a desensamblar
;             CAR = caracter desensamblada
;             N3  = numero de bytes de codigo
;-------------------------------------------------------------------------
          cmp      al,n1              ;es la 1a. instruccion del grupo?
          je       oper0              ;si, desensamblarla
          cmp      al,n2              ;es la 2a. instruccion del grupo?
          jbe      oper1              ;si, desensamblarla
          inc      cx                 ;no, CX apunta al mnemonico adecuado
          desens2  dir                ;desensamblar la 3a. instruccion del grupo
          jmp      fin                ;finalizar
oper0:    desens6  dir,car,n3         ;desensamblar la 1a. instruccion del grupo
          mov      cx,1               ;el codigo es de 1 byte
          jmp      fin                ;finalizar
oper1:    add      al,04h
          desens8  dir                ;desensamblar la 2a. instruccion del grupo
          jmp      fin                ;finalizar
fin:      write    0
          ret                         ;retornar

          ENDM

CASO_11   MACRO    DIR
          LOCAL    FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - LDR A.direct; LDR A,@Ri; LDR A,Rn
;Parametros:  DIR = lugar del segmento de datos donde se encuentran
;                 los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          desens8  dir                ;desensamblar una instruccion
          mov      al,[opcode]        ;leer el primer byte
          cmp      al,0e5h            ;es LDR A,direct?
          jne      fin                ;no
          mov      cx,2               ;si, el codigo es de 2 bytes
fin:      write    0                  ;finalizar
          ret                         ;retornar

          ENDM

CASO_12   MACRO    N1,N2,DIR
          LOCAL    OPER0,OPER1,FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - LDX @DPTR,A; LDX @Ri,A; COM A
;Parametros:  N1, N2 primeros bytes de codigo de las instrucciones
;                 que se quiere desensamblar
;             DIR = lugar del segmento de datos donde se encuentran
```

```
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          cmp       al,n1               ;es la 1a. instruccion del grupo?
          je        oper0               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          cmp       al,n2               ;es la 2a. instruccion del grupo?
          jbe       oper1               ;si, desensamblarla
          inc       cx                  ;no, CX apunta al mnemonico adecuado
          desens2   dir                 ;desensamblar la 3a. instruccion del grupo
          jmp       fin                 ;finalizar
oper0:    desens2   dir                 ;desensamblar la 1a. instruccion del grupo
          write     ','                 ;escribir una coma entre parametros
          write     'A'                 ;escribir una "A" como segundo parametro
          mov       cx,1                ;el codigo es de 1 byte
          jmp       fin                 ;finalizar
oper1:    add       al,04h
          desens9   dir                 ;desensamblar la 2a. instruccion del grupo
          jmp       fin                 ;finalizar
fin:      write     0
          ret                           ;retornar

          ENDM

CASO_13   MACRO     DIR
          LOCAL     FIN
;-------------------------------------------------------------------------
;Este macro permite desensamblar el siguiente grupo de instrucciones:
;         - LDR direct,A; LDR @Ri,A; LDR Rn,A
;Parametros:  DIR = lugar del segmento de datos donde se encuentran
;                   los mnemonicos de las instrucciones a desensamblar
;-------------------------------------------------------------------------
          desens9   dir                 ;desensamblar una instruccion del grupo
          mov       al,[opcode]         ;leer el primer byte de la instruccion
          cmp       al,0f5h             ;es LDR direct,A?
          jne       fin                 ;no, finalizar
          mov       cx,2                ;el codigo es de 2 bytes
fin:      write     0                   ;finalizar
          ret                           ;retornar

          ENDM

DESENS0   MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con un parametro (bit directamente
;direccionable) y codigo objeto de dos bytes.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          write     ' '                 ;escribir un espacio
          trans_bit                     ;el parametro tiene nombre especial?
          mov       cx,2                ;el codigo es de 2 bytes

          ENDM

DESENS1   MACRO     OPER
```

```
          LOCAL     CONT,CONT1
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con parametros de 16 bits y codigo
;objeto de tres bytes.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          cmp       al,90h              ;es LDR DPTR,#data16?
          jb        cont                ;no
          write     ','                 ;escribir una coma entre parametros
          write     '#'                 ;escribir el simbolo #
          jmp       cont1
cont:     write     ' '                 ;escribir un espacio
cont1:    mov       al,[si+1]           ;obtener un dato de 16 bits y escribirlo
          call      convhex   ;como parametro
          mov       al,[si+2]
          dec       di
          call      convhex
          mov       cx,3                ;el codigo es de 3 bytes

          ENDM

DESENS2   MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones sin parametros o con un parametro
;(A, AB, DPTR o C) y codigo objeto de un byte.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          mov       cx,1                ;el codigo es de 1 byte

          ENDM

DESENS3   MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones de salto relativo, con un parametro
;y codigo objeto de dos bytes.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          write     ' '                 ;escribir un espacio
          mov       al,[si+1]
          mov       cx,2                ;el codigo es de 2 bytes
          call      realadr             ;calcular la direccion del salto

          ENDM

DESENS4   MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con un parametro (byte directamente
;direccionable) y codigo objeto de dos bytes.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
```

```
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          write     ' '                 ;escribir un espacio
          push      di
          mov       al,[si+1]           ;obtener el parametro
          mov       di,offset string
          trans_byte                    ;el parametro tiene nombre especial?
          pop       di
          transfer string               ;transferir el parametro
          mov       cx,2                ;el codigo es de 2 bytes

          ENDM


DESENS5   MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con dos parametro (C y un bit
;directamente direccionable) y codigo objeto de dos bytes.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          write     ','                 ;escribir una coma entre parametros
          trans_bit                     ;1er. parametro tiene nombre especial?
          mov       cx,2                ;el codigo es de 2 bytes

          ENDM

DESENS6   MACRO     OPER,CAR,N-
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con direccionamiento indexado
;(@A+DPTR, @DPTR, @A+PC) y codigo objeto de "N" bytes.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
;             CAR = caracter desensamblada
;             N   = numero de bytes de codigo
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          write     car                 ;escribir caracter de la instr. desens.
          mov       cx,n                ;la instruccion tiene n bytes
          leer      index_reg           ;obtener un parametro

          ENDM

DESENS7   MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con dos parametro (C y el
;complemento de un bit directamente direccionable) y codigo objeto
;de dos bytes.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          write     ','                 ;escribir una coma entre parametros
          write     '/'                 ;escribir el simbolo /
          trans_bit                     ;1er. parametro tiene nombre especial?
```

```
          mov       cx,2                ;el codigo es de 2 bytes

          ENDM

DESENS8   MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con dos parametros (el primero es el
;acumulador) y codigo objeto de un byte.
;Parametros:  OPER = lugar del segmento de datos donde se encuentra
;                    el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          push      ax
          leer      oper                ;obtener el mnemonico
          write     ','                 ;escribir una coma entre parametros
          pop       ax
          det_modo_dir modos1,dir_modos1 ;modo de direccionamiento
          transfer string               ;transferir un parametro
          mov       cx,1                ;el codigo es de 1 byte

          ENDM

DESENS9   MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con dos parametros (el segundo es el
;acumulador) y codigo objeto de un byte.
;Parametros:       OPER = lugar del segmento de datos donde se encuentra
;                         el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          push      ax
          leer      oper                ;obtener el mnemonico
          write     ' '                 ;escribir un espacio
          pop       ax
          det_modo_dir modos1,dir_modos1 ;modo de direccionamiento
          transfer string               ;transferir un parametro
          write     ','                 ;escribir una coma entre parametros
          write     'A'                 ;escribir una A como parametro
          mov       cx,1                ;el codigo es de 1 byte

          ENDM

DESENS10  MACRO     OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con dos parametros (un bit
;directamente direccionable y una direccion de salto relativo) y codigo
;objeto de tres bytes.
;Parametros:       OPER = lugar del segmento de datos donde se encuentra
;                         el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          leer      oper                ;obtener el mnemonico
          write     ' '                 ;escribir un espacio
          trans_bit                     ;1er. parametro tiene nombre especial?
          write     ','                 ;escribir una coma entre parametros
          mov       al,[si+2]
          mov       cx,3                ;el codigo es de 3 bytes
          call      realadr             ;calcular la direccion del salto
```

```
          ENDM

DESENS11  MACRO    OPER
;-------------------------------------------------------------------------
;Este macro desensambla instrucciones con dos parametros (un byte
;directamente direccionable y una direccion de salto relativo) y codigo
;objeto de tres bytes.
;Parametros:       OPER = lugar del segmento de datos donde se encuentra
;                         el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          leer     oper              ;obtener el mnemonico
          write    ' '               ;escribir un espacio
          push     di
          mov      al,[si+1]         ;obtener un parametro
          mov      di,offset string
          trans_byte                 ;el parametro tiene nombre especial?
          pop      di
          transfer string            ;transferir un parametro
          write    ','               ;escribir una coma entre parametros
          mov      al,[si+2]
          mov      cx,3              ;el codigo es de 3 bytes
          call     realadr           ;calcular la direccion de salto

          ENDM

LEER      MACRO    DIR
;-------------------------------------------------------------------------
;Este macro permite transferir un elemento de un arreglo alfanumerico de
;un lugar a otro de la memoria.
;Parametros:       DIR = direccion de inicio del arreglo alfanumerico
;-------------------------------------------------------------------------
          push     si
          mov      si,offset dir     ;SI apunta al inicio del arreglo
          call     readarray         ;transferir el elemento
          pop      si

          ENDM

TRANSFER  MACRO    DIR
;-------------------------------------------------------------------------
;Este macro permite transferir una cadena de caracteres de un lugar a
;otro de la memoria.
;Parametros:       DIR = direccion de inicio de la cadena
;-------------------------------------------------------------------------
          push     cx
          xor      cx,cx             ;preparar transferencia del elemento
          leer     dir               ;transferir el elemento
          pop      cx

          ENDM

WRITE     MACRO    CARACTER
;-------------------------------------------------------------------------
;Este macro permite escribir un caracter en la memoria del computador
;-------------------------------------------------------------------------
          mov      al,caracter       ;AL = caracter a escribir
```

```
          stosb                         ;escribir el caracter en [ES:DI]

          ENDM

P_INST    MACRO     OPER
;-------------------------------------------------------------------------
;Este macro permite desensamblar las instrucciones PJMP y PCALL.
;Parametros:       OPER = lugar del segmento de datos donde se encuentra
;                         el mnemonico de la instruccion a desensamblar
;-------------------------------------------------------------------------
          push      bx                  ;preservar registros
          push      ax
          and       ax,01               ;preparar lectura del mnemonico
          mov       cx,ax
          leer      oper                ;obtener el mnemonico
          write     ' '                 ;escribir un espacio
          mov       bl,[opcode]
          call      cod3bit             ;calcular la dir. de transfer. de control
          mov       ax,regsi            ;de programa
          add       ax,2
          xchg      ah,al
          and       al,0f8h
          or        al,bl
          call      convhex             ;colocar esa direccion como parametro
          mov       al,[si+1]
          dec       di
          call      convhex
          write     0                   ;finalizar
          pop       ax
          pop       bx

          ENDM


DET_MODO_DIR        MACRO     MODO,DIR_MODO
          LOCAL     BUSQ,ENCUENT
;-------------------------------------------------------------------------
;Este procedimiento permite determinar el modo de direccionamiento
;usado en una instruccion.
;
;Entrada: MODO = tabla de bytes que definen el modo de direccionamiento
;         DIR_MODO = tabla de inicio de las rutinas para procesar
;                    cada modo de direcionamiento
;         AL = primer byte del codigo
;Salida:  string y sdata = modo de direccionamiento
;         string? Y sdata? = banderas
;         CX = longitud de instruccion
;-------------------------------------------------------------------------
          push      ax                  ;preservar registros
          push      si
          push      di
          push      bx

          and       al,0fh              ;preparar busqueda del modo de
          mov       di,offset modo      ;direccionamiento
          xor       bx,bx
```

```
busq:     scasb                       ;modo de direccionamiento utilizado?
          jbe       encuent           ;si, procesarlo
          inc       bx                ;no, pasar al siguiente modo
          jmp       busq              ;continuar la busqueda
encuent:
          shl       bx,1
          add       bx,offset dir_modo ;preparar salto a rutina que procese
          call      [bx]              ;el modo de direccionamiento utilizado

          pop       bx                ;restaurar registros
          pop       di
          pop       si
          pop       ax

          ENDM

TRANS_BIT           MACRO
          LOCAL     MAYOR7F,MAYOR_BF,NO_ENC,FIN
;-------------------------------------------------------------------------
;Este macro permite obtener el nombre de aquellos bits directamente
;direccionables que tienen un nombre especial.
;Entrada:           AL = direccion del bit que se quiere analizar
;Salida:            Desde [ES:DI] se coloca el nombre o la direccion del bit
;-------------------------------------------------------------------------
          push      bx                ;preservar registros
          push      cx
          push      si

          mov       al,[si+1]         ;leer segundo byte de la instruccion
          xor       ah,ah
          cmp       al,7fh            ;es un bit que tenga nombre especial?
          ja        mayor7f           ;puede ser
          mov       bl,8              ;no, colocar como parametro la dir. del
          div       bl                ;bit con formato XX.Y donde XX es el byte
          add       al,20h            ;al que pertenece el bit y Y es la
          call      convhex           ;posicion del bit en el byte
          dec       di
          write     '.'
          xchg      al,ah
          add       al,30h
          stosb
          write     'H'
          jmp       fin               ;finalizar
mayor7f:
          cmp       al,0bfh           ;es un bit que tenga nombre especial
          ja        mayor_bf          ;puede ser
          mov       cl,[opcode+1]     ;si, entonces buscar el nombre del bit
          xor       ch,ch             ;en la tabla (bits) donde estan definidos
          sub       cx,80h            ;esos nombres
          mov       si,offset bits
          call      readarray
          jmp       fin               ;finalizar
mayor_bf:
          cmp       al,0d0h           ;es un bit que tenga nombre especial?
          jb        no_enc            ;no
          test      byte ptr [opcode+1],8 ;tiene un nombre espaecial?
```

```
          jnz       no_enc              ;no
          mov       si,offset bits1     ;si, entonces buscar el nombre del bit
          call      readsfr             ;en una tabla (bits1) donde estan definidos
          jc        no_enc              ;esos nombres
          jmp       fin                 ;finalizar
no_enc:   mov       ah,al               ;colocar como parametro la dir. del bit
          and       al,0f8h             ;con formato XX.Y donde XX es el byte al
          sub       ah,al               ;que pertenece el bit y Y es la posicion
          call      convhex             ;del bit en el byte
          dec       di
          write     '.'
          xchg      al,ah
          add       al,30h
          stosb
          write     'H'
fin:
          pop       si                  ;restaurar registros
          pop       cx
          pop       bx

          ENDM

TRANS_BYTE          MACRO
          LOCAL     MAYOR31,NO_ENC,BIENSFR,FINSFR
;-------------------------------------------------------------------------
;Este macro permite obtener el nombre de aquellos bytes directamente
;direccionables que tienen un nombre especial.
;Entrada:            AL = direccion del byte que se quiere analizar
;Salida:             Desde [ES:DI] se coloca el nombre o la direccion del byte
;-------------------------------------------------------------------------
          push      ax                  ;preservar registros
          push      cx
          push      di
          push      si

          push      ax
          push      di
          cmp       al,1fh              ;es un byte que tenga nombre especial?
          ja        mayor31             ;puede ser
          xor       ah,ah               ;no, buscar si es un registro
          mov       cx,ax
          mov       si,offset registros
          call      readarray
          jmp       biensfr
mayor31:
          cmp       al,80h              ;es un byte que tenga nombre especial?
          jb        no_enc              ;no
          mov       si,offset sfr       ;si, entonces buscar el nombre del byte
          call      readsfr             ;en un tabla (sfr) donde estan definidos
          jc        no_enc              ;esos nombres
          jmp       biensfr
no_enc:   pop       di                  ;el byte no tiene nombre especial, colocar
          pop       ax                  ;como parametro la direccion del byte
          call      convhex
          jmp       finsfr              ;finalizar
biensfr:
```

```
          add       sp,4
finsfr:   write     ','                 ;escribir una coma al final del parametro

          pop       si                  ;restaurar registros
          pop       di
          pop       cx
          pop       ax

          ENDM

SUBRUT    MACRO

READARRAY PROC NEAR
;-------------------------------------------------------------------------
;Este procedimiento lee un elemento de un arreglo alfanumerico, de direccion
;especificada por DS:SI. El elemento se coloca desde ES:DI, y DI se acomoda
;para que apunte a la direccion siguiente al ultimo caracter del elemento.
;Los separadores entre elementos deben ser comas (,).
;
;Entrada: CX = numero del elemento en el arreglo.
;
;Salida:  DI = direccion del ultimo caracter transferido + 1
;-------------------------------------------------------------------------
          push      ax                  ;preservar registros
          push      cx
          push      si

          or        cx,cx               ;primer elemento del arreglo?
          jz        encont              ;si
busccoma:
          inc       si                  ;buscar el elemento
          cmp       byte ptr[si],','    ;separador?
          Jne       busccoma            ;no
          loop      busccoma            ;si, decrementar numero de elemento
          inc       si                  ;apuntar al elemento encontrado
encont:   lodsb                         ;leer un caracter
          cmp       al,','              ;separador?
          je        finread             ;si, finalizar transferencia
          stosb                         ;no, transferir
          jmp       encont              ;leer siguiente caracter
finread:
          pop       si                  ;restaurar registros
          pop       cx
          pop       ax
          ret                           ;retornar

READARRAY           ENDP

READSFR   PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento permite leer el nombre de un SFR o bit directamente
;direccionable desde un arreglo alfanumerico que empieza en DS:SI.
;El nombre se coloca desde ES:DI, y DI se acomoda como en readarray
;
;Entrada:            AL = direccion del bit o SFR
;
```

```
;Salida:          DI = direccion del ultimo caracter transferido + 1
;                 CY = 1 si no se encontro el nombre en la tabla
;-------------------------------------------------------------------------
          push    ax                  ;preservar registros
          push    si
busca:    cmp     al,byte ptr[si]     ;elemento separador?
          je      encontr             ;si
          inc     si                  ;no, continuar la busqueda
          cmp     byte ptr [si],'$'   ;fin de la tabla?
          je      no_encontr          ;si, finalizar
          jmp     busca               ;no, continuar la busqueda
encontr:
          inc     si                  ;apuntar al siguiente carater
enconti:
          lodsb                       ;leer ese caracter
          cmp     al,80h              ;elemento separador?
          jae     bien_rsfr           ;si
          cmp     al,'$'              ;no, es el fin de la tabla?
          je      bien_rsfr           ;si
          stosb                       ;transferir el caracter
          jmp     enconti             ;seguir con el siguiente caracter
no_encontr:
          stc                         ;no se encontro un nombre en la tabla
          jmp     fin_rsfr            ;finalizar
bien_rsfr:
          clc                         ;se encontro un nombre en la tabla
fin_rsfr:
          pop     si                  ;restaurar registros
          pop     ax
          ret                         ;retornar

READSFR   ENDP

          PUBLIC  CONVHEX
CONVHEX   PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento genera un string de 2 caracteres desde ES:DI que
;representan hexadecimalmente al numero en AL. DI se acomoda como en el
;procedimiento readarray.
;
;Entrada:         AL = numero por convertir.
;
;Salida:          DI = direccion del ultimo caracter generado + 1
;-------------------------------------------------------------------------
          push    ax                  ;preservar registros
          push    cx
          mov     ah,al
          and     ah,0fh              ;4 lsb en ah
          mov     cl,4
          shr     al,cl               ;4 msb en al
          or      ax,3030h            ;convertir a ascii
          cmp     al,'9'
          jbe     numal
          add     al,'A'-'9'-1
numal:    cmp     ah,'9'
          jbe     numah
```

```
          add       ah,'A'-'9'-1
numah:    stosw
          write     'H'                 ;escribir una H en los param. de la instr.
          pop       cx                  ;restaurar registros
          pop       ax
          ret                           ;retornar

CONVHEX   ENDP

COD3BIT   PROC      NEAR
;-----------------------------------------------------------------------
;Este procedimiento obtiene el numero de 3 bits de los bits 5, 6 y 7 de
;BL, y lo coloca en BL. Es de utilidad para desensamblar instrucciones
;con direccionamiento de 11 bits (PCALL,PJMP).
;
;Entrada:          BL
;
;salida:  BL = (BL AND 0E0H)/32
;-----------------------------------------------------------------------
          push      cx
          and       bl,0e0h
          mov       cl,5                ;dividir para 32
          shr       bl,cl
          pop       cx
          ret                           ;retornar

COD3BIT   ENDP

REALADR   PROC      NEAR
;-----------------------------------------------------------------------
;Este procedimiento calcula la dirección real de un salto relativo y la
;convierte en un string a partir de ES:DI, con DI como en readarray.
;
;Entrada:          AL = direccion relativa del salto
;          CX = longitud de instrucción
;
;Usa:      CONVHEX
;-----------------------------------------------------------------------
          push      ax
          cbw
          add       ax,regsi
          add       ax,cx               ;direccion real en ax
          push      ax
          mov       al,ah               ;transferir el string
          call      convhex
          dec       di
          pop       ax
          call      convhex
          pop       ax
          ret                           ;retornar

REALADR   ENDP

DIRECT    PROC      NEAR
;-----------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen un
```

```
;parametro que es un byte directamente direcionable.
;Entrada:        SI apunta al codigo de la instruccion
;Salida:         STRING = parametro de la instruccion
;-------------------------------------------------------------------------
        mov     al,[si+1]           ;leer segundo byte de la instruccion
        mov     di,offset string    ;el parametro se guardara en STRING
        trans_byte                  ;el parametro tiene nombre especial?
        mov     cx,2                ;el codigo es de 2 bytes
        mov     string?,1
        ret                         ;retornar

DIRECT  ENDP

DIRECT_DATA     PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen un
;parametro que es un byte directamente direcionable y un parametro que es
;un dato inmediato.
;Entrada:        SI apunta al codigo de la instruccion
;Salida:         SDATA = parametro de la instruccion
;-------------------------------------------------------------------------
        call    direct              ;procesar primer parametro
        mov     di,offset sdata     ;el 2do. parametro se guardara en SDATA
        mov     al,[si+2]           ;leer tercer byte de la instruccion
        call    convhex             ;transformarlo a ASCII
        write   ','                 ;escribir una coma al final del parametro
        mov     cx,3                ;el codigo es de 3 bytes
        mov     sdata?,1
        ret                         ;retornar

DIRECT_DATA     ENDP

DATA    PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen un
;parametro que es un dato inmediato.
;Entrada:        SI apunta al codigo de la instruccion
;Salida:         SDATA = parametro de la instruccion
;-------------------------------------------------------------------------
        mov     di,offset sdata     ;el parametro se guardara en SDATA
        mov     al,[si+1]           ;leer segundo byte de la instruccion
        call    convhex             ;transformarlo a ASCII
        write   ','                 ;escribir una coma al final del parametro
        mov     cx,2                ;el codigo es de 2 bytes
        mov     sdata?,1
        ret                         ;retornar

DATA    ENDP

INDEXADO  PROC  NEAR
;-------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen un
;parametro que utiliza direccionamiento indirecto.
;Entrada:        AL = primer byte de la instruccion
;                SI apunta al codigo de la instruccion
;Salida:         STRING = parametro de la instruccion
```

```
;                                   ---------------------------------------------
          push    si                ;preservar SI
          sub     al,6              ;en AX codigo para buscar el registro
          and     ax,0ffh           ;indice que interviene en la instruccion
          mov     cx,ax             ;transferir ese codigo a CX
          mov     si,offset index_reg ;SI apunta a tabla donde buscar el param.
          mov     di,offset string  ;el parametro se guardara en STRING
          call    readarray         ;obtener el parametro
          write   ','               ;escribir una coma al final del parametro
          mov     cx,1              ;el codigo es de 1 byte
          mov     string?,1
          pop     si                ;restaurar SI
          ret                       ;retornar

INDEXADO  ENDP

REGISTRO  PROC    NEAR
;-------------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen un
;parametro que es un registro.
;Entrada:          AL = primer byte de la instruccion
;                  SI apunta al codigo de la instruccion
;Salida:           STRING = parametro de la instruccion
;-------------------------------------------------------------------------------
          push    si                ;preservar SI
          sub     al,8              ;en AX codigo para buscar el registro
          and     ax,0ffh           ;que interviene en la instruccion
          mov     cx,ax             ;transferir ese codigo a CX
          mov     si,offset reg     ;SI = tabla donde se buscara el parametro
          mov     di,offset string  ;el parametro se guardara en STRING
          call    readarray         ;obtener el parametro
          write   ','               ;escribir una coma al final del parametro
          mov     cx,1              ;el codigo es de 1 byte
          mov     string?,1
          pop     si                ;restaurar SI
          ret                       ;retornar

REGISTRO  ENDP

INDEX_DATA          PROC      NEAR
;-------------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen un
;parametro que utiliza direccionamiento indexado y otro que es un dato
;inmediato.
;Entrada:          SI apunta al codigo de la instruccion
;Salida:           STRING = parametro de la instruccion
;                  SDATA  = parametro de la instruccion
;-------------------------------------------------------------------------------
          call    indexado          ;procesar primer parametro
          call    data              ;procesar segundo parametro
          ret                       ;retornar

INDEX_DATA          ENDP

REG_DATA  PROC      NEAR
;-------------------------------------------------------------------------------
```

```
;Este procedimiento sirve para procesar las instrucciones que tienen un
;parametro que es registro y otro que es un dato inmediato.
;Entrada:        SI apunta al codigo de la instruccion
;Salida:         STRING = parametro de la instruccion
;                SDATA  = parametro de la instruccion
;-------------------------------------------------------------------------
          call    registro          ;procesar primer parametro
          call    data              ;procesar segundo parametro
          ret                       ;retornar

REG_DATA  ENDP

REL       PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen un
;parametro que es una direccion para un salto relativo.
;Entrada:        SI apunta al codigo de la instruccion
;Salida:         SREL = parametro de la instruccion
;-------------------------------------------------------------------------
          mov     di,offset srel    ;el parametro se guardara en SREL
          mov     al,[si+2]         ;leer tercer byte de la instruccion
          mov     cx,3              ;el codigo es de 3 bytes
          call    realadr           ;calcular la direccion del salto
          write   ','               ;escribir una coma al final del parametro
          mov     srel?,1
          ret                       ;retornar

REL       ENDP

DATA_REL  PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen un
;parametro que es un dato inmediato y otro que es una direccion para un
;salto relatuvo.
;Entrada:        SI apunta al codigo de la instruccion
;Salida:         SREL = parametro de la instruccion
;                SDATA  = parametro de la instruccion
;-------------------------------------------------------------------------
          call    data              ;procesar primer parametro
          call    rel               ;procesar segundo parametro
          ret                       ;retornar

DATA_REL  ENDP

INDEX_DATA_REL    PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen tres
;parametros: con direccionamiento indexado, un dato inmediato y una
;direccion para un salto relativo.
;Entrada:        SI apunta al codigo de la instruccion
;Salida:         STRING = parametro de la instruccion
;                SDATA  = parametro de la instruccion
;                SREL   = parametro de la instruccion
;-------------------------------------------------------------------------
          call    indexado  ;procesar primer parametro
          call    data              ;procesar segundo parametro
```

```
          call      rel                 ;procesar tercer parametro
          ret                           ;retornar

INDEX_DATA_REL      ENDP

REG_DATA_REL        PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento sirve para procesar las instrucciones que tienen tres
;parametros: un registro, un dato inmediato y una direccion para un salto
;relativo.
;Entrada:           SI apunta al codigo de la instruccion
;Salida:            STRING = parametro de la instruccion
;                   SDATA  = parametro de la instruccion
;                   SREL   = parametro de la instruccion
;-------------------------------------------------------------------------
          call      registro            ;procesar primer parametro
          call      data                ;procesar segundo parametro
          call      rel                 ;procesar tercer parametro
          ret                           ;retornar

REG_DATA_REL        ENDP

          PUBLIC    INTEL
INTEL     PROC      FAR
;-------------------------------------------------------------------------
;Este procedimiento obtiene los bits de codigo de un archivo con formato
;INTEL y los guarda en memoria.
;Entrada:           DS:SI apunta al inicio del archivo INTEL
;                   ES:DI apunta al lugar donde se guardara el codigo
;Salida:            ES:DI es ajustado
;                   DS:SI es ajustado
;                   CY=1  si error
;-------------------------------------------------------------------------
          push      ds                  ;preservar registros
          push      es
          push      si
          push      di
          mov       es,segdat2          ;ES apunta a DSEG2
          mov       ds,segdat1          ;ES apunta a DSEG1
          lodsb                         ;leer primer caracter
          cmp       al,':'              ;es valido?
          jne       errintel            ;no, error
          dec       si                  ;si
          mov       cx,4000             ;inicializar contador de lineas del archivo
repetir1:
          call      lin_intel           ;procesar una linea del archivo
          jc        errintel            ;hubo un error
          lodsb                         ;leer primer caracter de la siguiente linea
          cmp       al,':'              ;es valido?
          jne       bienintel           ;no, fin del archivo
          dec       si                  ;si
          loop      repetir1            ;procesar la linea
errintel:
          pop       di                  ;hubo un error
          pop       si
          stc                           ;indicar el error
```

```
          jmp       finintel            ;finalizar
bienintel:
          add       sp,4                ;no hubo error
          clc
finintel:
          pop       es                  ;restaurar registros
          pop       ds
          ret                           ;retornar

INTEL     ENDP

LIN_INTEL           PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento permite procesar una linea de un archivo en formato
;INTEL de 8 bits, verificando la sintaxis y los checksum.
;Entrada:           DS:SI apunta al inicio de la linea del archivo INTEL
;                   ES:DI apunta al inicio del buffer donde se desea el
;                         codigo hexadecimal.
;Salida:            DS:SI es ajustado
;                   ES:DI es ajustado
;                   CY=1  si error, en cuyo caso DS:SI y ES:DI permanecen
;                         inalterados
;Usa:               ASCAHEX
;-------------------------------------------------------------------------
          push      bx                  ;preservar registros
          push      cx
          push      si
          push      di
          lodsb
          cmp       al,':'              ;hay los 2 puntos iniciales?
          jne       errlin              ;no
          lodsw
          call      ascahex
          jc        errlin
          mov       cksum,al            ;para chequear checksum
          mov       cl,al               ;poner en CX el numero de bytes de codigo
          xor       ch,ch               ;que hay en la linea INTEL
          lodsw
          call      ascahex
          jc        errlin
          add       cksum,al            ;checksum
          mov       bh,al               ;byte mas significativo del puntero en BH
          lodsw
          call      ascahex
          jc        errlin
          add       cksum,al            ;checksum
          mov       bl,al               ;byte menos significativo del puntero en BH
          add       di,bx               ;DI = lugar donde se guardara el codigo
          lodsw
          call      ascahex
          jc        errlin
          add       cksum,al
          jcxz      sigal
repetir:
          lodsw                         ;procesar los bytes de codigo
          call      ascahex
```

```
;*****************************************************************************
;                          SIMULAR.ASM
;*****************************************************************************
;
IF1
          INCLUDE MACEJEC.ASM
ENDIF
;
;Tabla de variables utilizadas en SIMULAR.ASM y definidas en otros modulos:
;
          EXTRN  PC:WORD,RAM_EMB:BYTE
          EXTRN  R0:BYTE,R1:BYTE,R2:BYTE,R3:BYTE
          EXTRN  R4:BYTE,R5:BYTE,R6:BYTE,R7:BYTE
          EXTRN  P0:BYTE,REGSP:BYTE,DPL:BYTE,DPH:BYTE
          EXTRN  PCON:BYTE,TCON:BYTE,TMOD:BYTE
          EXTRN  TL0:BYTE,TL1:BYTE,TH0:BYTE,TH1:BYTE
          EXTRN  P1:BYTE,SCON:BYTE,SBUF:BYTE,SBUF_IN:BYTE
          EXTRN  P2:BYTE,IE:BYTE,P3:BYTE,IP:BYTE
          EXTRN  EK0:BYTE,EK1:BYTE,EK2:BYTE,EK3:BYTE,EK4:BYTE
          EXTRN  MCON:BYTE,TA:BYTE,PSW:BYTE,ACC:BYTE,B:BYTE
          EXTRN  SEGDAT2:WORD,OPCODE:BYTE
          EXTRN  ROM_RAM_E:WORD,RAM_EMB_M:WORD
          EXTRN  P0_PIN:BYTE,P1_PIN:BYTE,P2_PIN:BYTE,P3_PIN:BYTE

DSEG SEGMENT PUBLIC
;
;Tabla de definicion de los grupos de instrucciones que son
;simuladas por una misma rutina:
;
casos_ejec          db        00,02,03,05,07,0fh,10h,12h
                    db        13h,15h,17h,1fh,20h,22h,23h,25h
                    db        27h,2fh,30h,32h,33h,35h,37h,3fh
                    db        40h,45h,47h,4fh,50h,55h,57h,5fh
                    db        60h,65h,67h,6fh,70h,72h,73h,75h
                    db        77h,7fh,80h,82h,83h,84h,85h,87h
                    db        8fh,90h,92h,93h,95h,97h,9fh,0a0h
                    db        0a2h,0a3h,0a4h,0a7h,0afh,0b0h,0b3h,0b5h
                    db        0b7h,0bfh,0c0h,0c3h,0c4h,0c5h,0c7h,0cfh
                    db        0d0h,0d3h,0d4h,0d5h,0d7h,0dfh,0e0h,0e3h
                    db        0e4h,0e5h,0e7h,0efh,0f0h,0f3h,0f4h,0f5h
                    db        0f7h,0ffh
;
;Tabla de localizacion de las rutinas para simulacion de las
;instrucciones del DS5000T:
;
dir_ejec            dw        offset ejec_00, offset ejec_02
                    dw        offset ejec_03, offset ejec_05
                    dw        offset ejec_07, offset ejec_0f
                    dw        offset ejec_10, offset ejec_12
                    dw        offset ejec_13, offset ejec_15
                    dw        offset ejec_17, offset ejec_1f
                    dw        offset ejec_20, offset ejec_22
                    dw        offset ejec_23, offset ejec_25
                    dw        offset ejec_27, offset ejec_2f
                    dw        offset ejec_30, offset ejec_32
                    dw        offset ejec_33, offset ejec_35
                    dw        offset ejec_37, offset ejec_3f
                    dw        offset ejec_40, offset ejec_45
```

```
                dw      offset ejec_47, offset ejec_4f
                dw      offset ejec_50, offset ejec_55
                dw      offset ejec_57, offset ejec_5f
                dw      offset ejec_60, offset ejec_65
                dw      offset ejec_67, offset ejec_6f
                dw      offset ejec_70, offset ejec_72
                dw      offset ejec_73, offset ejec_75
                dw      offset ejec_77, offset ejec_7f
                dw      offset ejec_80, offset ejec_82
                dw      offset ejec_83, offset ejec_84
                dw      offset ejec_85, offset ejec_87
                dw      offset ejec_8f, offset ejec_90
                dw      offset ejec_92, offset ejec_93
                dw      offset ejec_95, offset ejec_97
                dw      offset ejec_9f, offset ejec_a0
                dw      offset ejec_a2, offset ejec_a3
                dw      offset ejec_a4, offset ejec_a7
                dw      offset ejec_af, offset ejec_b0
                dw      offset ejec_b3, offset ejec_b5
                dw      offset ejec_b7, offset ejec_bf
                dw      offset ejec_c0, offset ejec_c3
                dw      offset ejec_c4, offset ejec_c5
                dw      offset ejec_c7, offset ejec_cf
                dw      offset ejec_d0, offset ejec_d3
                dw      offset ejec_d4, offset ejec_d5
                dw      offset ejec_d7, offset ejec_df
                dw      offset ejec_e0, offset ejec_e3
                dw      offset ejec_e4, offset ejec_e5
                dw      offset ejec_e7, offset ejec_ef
                dw      offset ejec_f0, offset ejec_f3
                dw      offset ejec_f4, offset ejec_f5
                dw      offset ejec_f7, offset ejec_ff
;
;Tabla de instrucciones clasificadas por su longitud:
;
casos_long      db      0,1,2,4,5,0fh,10h,11h
                db      12h,14h,15h,1fh,20h,21h,23h,25h
                db      2fh,30h,31h,33h,35h,3fh,42h,43h
                db      45h,4fh,52h,53h,55h,5fh,62h,63h
                db      65h,6fh,72h,73h,74h,75h,82h,84h
                db      85h,8fh,90h,92h,93h,95h,9fh,0a2h
                db      0a5h,0b2h,0b3h,0bfh,0c2h,0c4h,0c5h,0cfh
                db      0d2h,0d4h,0d5h,0d7h,0dfh,0e0h,0e1h,0e4h
                db      0e5h,0f0h,0f1h,0f4h,0f5h,0ffh
;
;Tabla de longitudes en bytes de las instrucciones:
;
long            db      1,2,3,1,2,1,3,2,3,1,2,1,3,2,1,2
                db      1,3,2,1,2,1,2,3,2,1,2,3,2,1,2,3
                db      2,1,2,1,2,3,2,1,3,2,3,2,1,2,1,2
                db      1,2,1,3,2,1,2,1,2,1,3,1,2,1,2,1
                db      2,1,2,1,2,1
;
;Tabla de instrucciones clasificadas por el numero de ciclos
;de maquina requeridos para su ejecucion:
;
```

```
casos_time      db      0,2,0fh,12h,1fh,22h,2fh,32h
                db      3fh,41h,42h,43h,4fh,51h,52h,53h
                db      5fh,61h,62h,63h,6fh,73h,74h,75h
                db      7fh,83h,84h,93h,9fh,0a1h,0a2h,0a3h
                db      0a4h,0b1h,0b3h,0c1h,0cfh,0d1h,0d4h,0d5h
                db      0d7h,0e3h,0efh,0f3h,0ffh
;
;Tabla de ciclos de maquina requeridos por las instrucciones:
;
time            db      1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2
                db      1,2,1,2,1,2,1,2,1,2,4,2,1,2,1,2
                db      4,2,1,2,1,2,1,2,1,2,1,2,1
;
;Tabla de definicion del patron de bits que debe ser escrito
;para tener acceso al ECC:
;
pattern         db      0,1,0,1,1,1,0,0,1,0,1,0,0,0,1,1
                db      0,0,1,1,1,0,1,0,1,1,0,0,0,1,0,1
                db      0,1,0,1,1,1,0,0,1,0,1,0,0,0,1,1
                db      0,0,1,1,1,0,1,0,1,1,0,0,0,1,0,1

ecc             db      64 dup (33h)        ;64 bits del ECC
reset_ecc       db      64                  ;banderas para simulacion
rec_patt        db      64                  ;del ECC
ecc_point       db      0                   ;puntero del ECC

direct          db      0
flag_ta1        db      0
flag_ta2        db      0
flag_int        db      0
flag_prior      db      0
flag_reti       db      0
flag_mcon       db      0
flag_port0      db      0
flag_port1      db      0
flag_port2      db      0
flag_port3      db      0
revers          db      64*32 dup (0)       ;Buffer para ejecucion reversa
pc_revers       dw      0                   ;PC para ejecucion reversa
count_r         db      0                   ;Contador de instr. en ejec.reversa
flag_tcon       db      0                   ;Bandera usada en ejecucion reversa

DSEG ENDS

CSEG SEGMENT
ASSUME CS:CSEG,DS:DSEG,ES:NOTHING

        PUBLIC EJECUTOR
EJECUTOR PROC FAR
;-----------------------------------------------------------------
;Este procedimiento simula la ejecucion de cualquier instruccion
;del uC DS5000T.
;
;Entrada: PC apunta a la instruccion que se va a simular
;
;Salida:  Son alteradas las variables involucradas en la instruccion
```

```
;           simulada
;           PC = direccion de la siguiente instruccion a ejecutar
;         CX = ciclos requeridos por la instruccion
;-------------------------------------------------------------------
          pushf                         ;preservar banderas y registros
          push      bx
          push      dx
          push      ds
          push      es
          push      si

          push      ds
          pop       es                  ;ES apunta a DSEG
          cld
          push      ds
          mov       ds,segdat2          ;DS apunta a DSEG2
          lea       si,ram_emb          ;SI apunta al programa del usuario
          pop       ds                  ;DS apunta a DSEG
          add       si,pc               ;SI apunta a la instruccion a simular
          lea       di,opcode           ;DI = buffer para guardar la instr.
          mov       cx,3                ;inicializar un contador
          push      ds
          mov       ds,segdat2          ;DS apunta a DSEG2
          rep       movsb               ;transferir instruccion a un buffer
          pop       ds                  ;DS apunta a DSEG
          lea       si,opcode           ;SI apunta al buffer donde esta la instr.
          lodsb
          push      ax                  ;preservar AX
          and       al,1fh
          cmp       al,11h              ;es PCALL?
          je        ejec_1              ;si
          cmp       al,01h              ;no, es PJMP?
          jne       siga_1              ;no
          jmp       ejec_2              ;si
siga_1:   pop       ax                  ;restaurar AX
          jmp       siga
ejec_1:                               ;simulacion de PCALL
          preserv_pc                  ;preservar PC para ejecucion reversa
          preserv_dir 81h,regsp       ;preservar SP para ejecucion reversa
          inc       byte ptr regsp    ;SP = SP+1
          pop       ax
          add       pc,2                ;PC = PC+2
          push      si
          lea       si,r0
          mov       al,regsp
          xor       ah,ah
          add       si,ax               ;SI apunta al stack
          preserv_stack               ;preservar el stack para ejecucion reversa
          mov       ax,pc
          mov       [si],ax             ;guardar PC en el stack
          pop       si
          inc       byte ptr regsp    ;SP = SP+1
          dec       si
          lodsw                         ;leer codigo de la instruccion
          xchg      al,ah               ;calcular la direccion del salto
          mov       cl,5
```

```
          shr       ah,cl
          or        ax,0f800h
          or        pc,07ffh
          and       pc,ax               ;simular salto a subrutina
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          jmp       siga_t
ejec_2:                                 ;simulacion de PJMP
          preserv_pc                    ;preservar PC para ejecucion reversa
          pop       ax
          add       pc,2                ;PC = PC+2
          dec       si
          lodsw                         ;leer codigo de la instruccion
          xchg      al,ah               ;calcular la direccion del salto
          mov       cl,5
          shr       ah,cl
          or        ax,0f800h
          or        pc,07ffh
          and       pc,ax               ;simular el salto
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          jmp       siga_t
siga:                                   ;no es PCALL ni PJMP
          push      di                  ;DI apunta a una tabla de grupos de
          mov       di,offset casos_ejec          ;instrucciones que son simuladas
          xor       bx,bx               ;por la misma rutina
buscop:
          scasb                         ;se encontro la instruccion en la tabla?
          jbe       encontrado          ;si
          inc       bx                  ;no, pasar al siguiente elemento de la tabla
          jmp       buscop              ;continuar la busqueda
encontrado:
          pop       di
          shl       bx,1                ;si se encontro, saltar a la
          add       bx,offset dir_ejec  ;rutina apropiada para simular
          call      [bx]                ;la instruccion
siga_t:
          call      timers              ;simular los TIMERS
          cmp       flag_reti,0         ;se debe simular interrupciones?
          je        siga_i              ;si
          jmp       finejec             ;no
siga_i:

;Procesamiento de INTERRUPCIONES:

          test      ie,80h              ;estan habilitadas las interrupciones?
          jnz       siga_int            ;si
          jmp       finejec             ;no
siga_int:
          test      ip,1fh              ;esta escogida alguna prioridad?
          jz        no_prior            ;no
          jmp       prior               ;si
no_prior:
          cmp       flag_int,1          ;esta atendiendo una int.?
```

```
          jne       no_int              ;no
          jmp       finejec             ;si
no_int:
          preserv_interrup              ;preservar param. alterados por la interrup.
          int_ext0 0                    ;procesar las interrupciones
          int_timer0          0
          int_ext1 0
          int_timer1          0
          int_serial          0
          jmp       finejec
prior:
          cmp       flag_prior,1        ;esta atendiendo una int. prior.?
          jne       no_prior1           ;no
          jmp       finejec             ;si
no_prior1:
          preserv_interrup              ;preservar param. alterados por la interrup.
          test      ip,1                ;EX0 tiene maxima prioridad?
          jz        no_pr_ex0           ;no
          int_ext0            1
no_pr_ex0:
          test      ip,2                ;ET0 tiene maxima prioridad?
          jnz       pr_et0              ;si
          jmp       no_pr_et0           ;no
pr_et0:
          int_timer0          1
no_pr_et0:
          test      ip,4                ;EX1 tiene maxima prioridad?
          jnz       pr_ex1              ;si
          jmp       no_pr_ex1           ;no
pr_ex1:
          int_ext1 1
no_pr_ex1:
          test      ip,8                ;ET1 tiene maxima prioridad?
          jnz       pr_et1              ;si
          jmp       no_pr_t1  ;no
pr_et1:
          int_timer1          1
no_pr_et1:
          int_serial          1
          cmp       flag_int,1          ;esta atendiendo una int.?
          jne       no_int1             ;no
          jmp       finejec             ;si
no_int1:
          int_ext0 0                    ;procesar las interrupciones
          int_timer0          0
          int_ext1 0
          int_timer1          0

finejec:
          mov       flag_reti,0         ;se puede simular interrupciones
          preserv_ciclos                ;preservar los ciclos para ejecucion reversa

          pop       si                  ;restaurar registros y banderas
          pop       es
          pop       ds
          pop       dx
```

```
        pop     bx
        popf
        ret                             ;retornar

EJECUTOR ENDP
;Fin de procedimiento
;
; Procedimientos adicionales
;
        SUBRUT  ;Macro que contiene subrutinas utilizadas
                ;en este modulo.
;
;-------------------------------------------------------------------------
;Las rutinas EJEC_00 hasta EJEC_FF permiten realizar la simulacion de
;las instrucciones del DS5000T.
;Estas rutinas manejan los siguientes parametros:
;Entrada:        AL = primer byte de codigo de la instruccion a simular
;                DS:SI = direccion de los bytes de codigo de la instruccion
;Salida:  Son alteradas las variables correspondientes a los recursos
;                del microcontrolador alterados por la instruccion simulada
;                CX = numero de ciclos de maquina requeridos por la
;                instruccion
;-------------------------------------------------------------------------
;
;-------------------------------------------------------------------------
;Simulacion de:  NOP
;-------------------------------------------------------------------------
ejec_00:
        preserv_pc                      ;preservar PC para ejecucion reversa
        inc     pc                      ;PC = PC+1
        mov     cx,1                    ;requiere 1 ciclo de maquina
        mov     flag_ta1,0              ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                             ;retornar

;-------------------------------------------------------------------------
;Simulacion de:  JMP ADDR16
;-------------------------------------------------------------------------
ejec_02:
        preserv_pc                      ;preservar PC para ejecucion reversa
        lodsw                           ;obtener la direccion del salto
        xchg    ah,al
        mov     pc,ax                   ;simular la instruccion
        mov     cx,2                    ;requiere 2 ciclos de maquina
        mov     flag_ta1,0              ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                             ;retornar

;-------------------------------------------------------------------------
;Simulacion de:  RR A
;-------------------------------------------------------------------------
ejec_03:
        preserv_pc                      ;preservar PC para ejecucion reversa
        preserv_dir 0e0h,acc            ;preservar ACC para ejecucion reversa
        ror     acc,1                   ;simular la instruccion
        inc     pc                      ;PC = PC+1
```

```
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    INC A; INC direct
;-------------------------------------------------------------------------
ejec_05:
          preserv_pc                    ;preservar PC para ejecucion reversa
          cmp       al,04               ;es INC A?
          jne       siga_05             ;no
          jmp       acum_05             ;si
siga_05:
          lodsb                         ;leer segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          lea       si,r0               ;direccionar el operando
          xor       ah,ah
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          inc       byte ptr [si]       ;simular la instruccion
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          add       pc,2                ;PC = PC+2
          jmp       fin_05              ;finalizar
acum_05:
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          inc       byte ptr acc        ;simular INC A
          inc       pc                  ;PC = PC+1
fin_05:   mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    INC @Ri
;-------------------------------------------------------------------------
ejec_07:
          preserv_pc                    ;preservar PC para ejecucion reversa
          dir_index si,6                ;direccionar el operando
          jc        error_07            ;direccion no valida
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          inc       byte ptr [si]       ;simular la instruccion
error_07:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    INC Rn
;-------------------------------------------------------------------------
ejec_0f:
```

```
        preserv_pc                  ;preservar PC para ejecucion reversa
        dir_reg  si,8               ;direccionar el operando
        preserv_dir bl,[si+bx]      ;preservar byte de RAM int. para ej.reversa
        inc     byte ptr[si+bx]     ;simular la instruccion
        inc     pc                  ;PC = PC+1
        mov     cx,1                ;requiere 1 ciclo de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BBC bit,rel
;-------------------------------------------------------------------------
ejec_10:
        preserv_pc                  ;preservar PC para ejecucion reversa
        lodsb                       ;obtener primer operando
        add     pc,3                ;PC = PC+3
        call    checkbit            ;bit =1?
        jnc     fin_10              ;no, finalizar
        xor     dl,dl               ;si, hacer bit = 0
        call    setbit
        lodsb                       ;obtener segundo operando
        cbw
        add     pc,ax               ;simular el salto
fin_10: mov     cx,2                ;requiere 2 ciclos de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    CALL ADDR16
;-------------------------------------------------------------------------
ejec_12:
        preserv_pc                  ;preservar PC para ejecucion reversa
        preserv_dir 81h,regsp       ;preservar SP para ejecucion reversa
        add     pc,3                ;PC = PC+3
        inc     byte ptr regsp      ;SP = SP+1
        push    si
        lea     si,r0
        mov     al,regsp
        xor     ah,ah
        add     si,ax               ;SI apunta al stack
        preserv_stack               ;preservar el stack para ejecucion reversa
        mov     ax,pc
        mov     [si],ax             ;guardar PC en el stack
        pop     si
        inc     byte ptr regsp      ;SP = SP+1
        lodsw                       ;obtener la direccion de la subrutina
        xchg    ah,al
        mov     pc,ax               ;simular el salto a subrutina
        mov     cx,2                ;requiere 2 ciclos de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
```

```
;Simulacion de:    RRC A
;-------------------------------------------------------------------------
ejec_13:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          call      check_cy            ;obtener el estado del carry
          rcr       acc,1               ;simular la instruccion
          jnc       no_carry1
          or        psw,80h             ;setear el carry
          jmp       fin_13              ;finalizar
no_carry1:
          and       psw,7fh             ;borrar el carry
fin_13:   inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    DEC A; DEC direct
;-------------------------------------------------------------------------
ejec_15:
          preserv_pc                    ;preservar PC para ejecucion reversa
          cmp       al,14h              ;es DEC A?
          jne       check_15            ;no
          jmp       acum_15             ;si
check_15:
          lodsb                         ;obtener el operando
          check_dir1                    ;verificar si es algun registro especial
          lea       si,r0               ;direccionar el operando
          xor       ah,ah
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          dec       byte ptr [si]       ;simular la instruccion
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          add       pc,2                ;PC = PC+2
          jmp       fin_15              ;finalizar
acum_15:
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          dec       byte ptr acc        ;simular DEC A
          inc       pc                  ;PC = PC+1
fin_15:   mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    DEC @Ri
;-------------------------------------------------------------------------
ejec_17:
          preserv_pc                    ;preservar PC para ejecucion reversa
          dir_index si,16h              ;direccionar el operando
          jc        error_17            ;direccion no valida
```

```
          mov       cl,al
          preserv_dir cl,[si]         ;preservar byte de RAM int. para ej.reversa
          dec       byte ptr [si]     ;simular la instruccion
error_17:
          inc       pc                ;PC = PC+1
          mov       cx,1              ;requiere 1 ciclo de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    DEC Rn
;-------------------------------------------------------------------------
ejec_1f:
          preserv_pc                  ;preservar PC para ejecucion reversa
          dir_reg   si,18h            ;direccionar el operando
          preserv_dir bl,[si+bx]      ;preservar byte de RAM int. para ej. reversa
          dec       byte ptr[si+bx]   ;simular la instruccion
          inc       pc                ;PC = PC+1
          mov       cx,1              ;requiere 1 ciclo de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BB bit,rel
;-------------------------------------------------------------------------
ejec_20:
          preserv_pc                  ;preservar PC para ejecucion reversa
          lodsb                       ;obtener primer operando
          check_pin1                  ;verificar si es un pin de un puerto
          add       pc,3              ;PC = PC+3
          call      checkbit          ;bit = 1?
          jnc       fin_20            ;no, finalizar
          lodsb                       ;si, obtener segundo operando
          cbw
          add       pc,ax             ;simular el salto
fin_20:   check_pin2
          mov       cx,2              ;requiere 2 ciclos de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    RET
;-------------------------------------------------------------------------
ejec_22:
          preserv_pc                  ;preservar PC para ejecucion reversa
          preserv_dir 81h,regsp       ;preservar SP para ejecucion reversa
          lea       si,r0             ;direccionar el tope del stack
          mov       al,regsp
          xor       ah,ah
          add       si,ax
          dec       si
          mov       ax,[si]           ;recuperar PC del stack
          mov       pc,ax
```

```
          sub       regsp,2             ;SP = SP-2
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    RL A
;-------------------------------------------------------------------------
ejec_23:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          rol       acc,1               ;simular la instruccion
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    ADD A,#data; ADD A,direct
;-------------------------------------------------------------------------
ejec_25:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          xor       ah,ah
          cmp       al,24h              ;es ADD A,#data?
          jne       direct_25           ;no
          sahf                          ;borrar banderas del 8086
          lodsb                         ;obtener segundo operando
          add       acc,al              ;simular la instruccion
          jmp       cont1
direct_25:        lodsb                 ;leer segundo byte de la instruccion
          es_ie_ip?                     ;verificar si es el registro IE o IP
          lea       si,r0               ;direccionar el segundo operando
          add       si,ax
          dir_pines                     ;verificar si es un puerto
          sahf                          ;borrar banderas del 8086
          lodsb                         ;obtener el segundo operando
          add       acc,al              ;simular la instruccion
cont1:    call      setflags            ;dar valor a las banderas del DS5000T
          add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    ADD A,@Ri
;-------------------------------------------------------------------------
ejec_27:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          dir_index si,26h              ;direccionar el segundo operando
```

```
          jc        error_27            ;direccion no valida
          sahf                          ;borrar banderas del 8086
          lodsb                         ;obtener el segundo operando
          add       acc,al              ;simular la instruccion
          call      setflags            ;dar valor a las banderas del DS5000T
          jmp       siga_27
error_27:
          mov       acc,0ffh            ;la direccion del seg. operando no es valida
siga_27:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    ADD A,Rn
;-------------------------------------------------------------------------
ejec_2f:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          dir_reg   si,28h              ;direccionar el segundo operando
          sahf                          ;borrar banderas del 8086
          mov       al,[si+bx]          ;obtener el segundo operando
          add       acc,al              ;simular la instruccion
          call      setflags            ;dar valor a las banderas del DS5000T
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BNB bit,rel
;-------------------------------------------------------------------------
ejec_30:
          preserv_pc                    ;preservar PC para ejecucion reversa
          lodsb                         ;obtener primer operando
          check_pin1                    ;verificar si es un pin de un puerto
          add       pc,3                ;PC = PC+3
          call      checkbit            ;bit = 1?
          jc        fin_30              ;si, finalizar
          lodsb                         ;no, obtener segundo operando
          cbw
          add       pc,ax               ;simular el salto
fin_30:   check_pin2
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    RETI
;-------------------------------------------------------------------------
ejec_32:
```

```
          preserv_pc                ;preservar PC para ejecucion reversa
          preserv_dir 8ih,regsp     ;preservar SP para ejecucion reversa
          preserv_interrup          ;preservar banderas de simul. de interrup.
          lea       si,r0           ;direccionar el tope del stack
          mov       al,regsp
          xor       ah,ah
          add       si,ax
          dec       si
          mov       ax,[si]         ;recuperar PC del stack
          mov       pc,ax
          sub       regsp,2         ;SP = SP-2
          mov       cx,2            ;requiere 2 ciclos de maquina
          mov       flag_ta1,0      ;deshabilitar Timed Access
          mov       flag_ta2,0
          cmp       flag_prior,1    ;habilitar simulacion de
          jne       s_reti          ;interrupciones
          mov       flag_prior,0
          jmp       s_reti1
s_reti:   mov       flag_int,0
s_reti1:
          mov       flag_reti,1
          ret                       ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    RLC A
;-------------------------------------------------------------------------
ejec_33:
          preserv_pc                ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc      ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw      ;preservar PSW para ejecucion reversa
          call      check_cy        ;obtener el estado del carry
          rcl       acc,1           ;simular la instruccion
          jnc       no_carry2
          or        psw,80h         ;setear el carry
          jmp       fin_33          ;finalizar
no_carry2:
          and       psw,7fh         ;borrar el carry
fin_33:   inc       pc              ;PC = PC+1
          mov       cx,1            ;requiere 1 ciclo de maquina
          mov       flag_ta1,0      ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                       ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    ADC A,#data; ADC A,direct
;-------------------------------------------------------------------------
ejec_35:
          preserv_pc                ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc      ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw      ;preservar PSW para ejecucion reversa
          xor       ah,ah
          cmp       al,34h          ;es ADC A,#data?
          jne       direct_35       ;no
          sahf                      ;borrar banderas del 8086
          call      check_cy        ;obtener el estado del carry
          lodsb                     ;obtener segundo operando
```

```
          adc       acc,al              ;simular la instruccion
          jmp       cont2
direct_35:          lodsb               ;leer segundo byte de la instruccion
          es_ie_ip?                     ;verificar si es el registro IE o IP
          lea       si,r0               ;direccionar el segundo operando
          add       si,ax
          dir_pines                     ;verificar si es un puerto
          sahf                          ;borrar banderas del 8086
          call      check_cy            ;obtener el estado del carry
          lodsb                         ;obtener segundo operando
          adc       acc,al              ;simular la instruccion
cont2:    call      setflags            ;dar valor a las banderas del DS5000T
          add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    ADC A,@Ri
;-------------------------------------------------------------------------
ejec_37:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          dir_index si,36h              ;direccionar el segundo operando
          jc        error_37            ;direccion no valida
          sahf                          ;borrar banderas del 8086
          call      check_cy            ;obtener el estado del carry
          lodsb                         ;obtener el segundo operando
          adc       acc,al              ;simular la instruccion
          call      setflags            ;dar valor a las banderas del DS5000T
          jmp       siga_37
error_37:
          mov       acc,0ffh            ;la direccion del seg. operando no es valida
siga_37:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    ADC A,Rn
;-------------------------------------------------------------------------
ejec_3f:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          dir_reg   si,38h              ;direccionar el segundo operando
          sahf                          ;borrar banderas del 8086
          call      check_cy            ;obtener el estado del carry
          mov       al,[si+bx]          ;obtener el segundo operando
          adc       acc,al              ;simular la instruccion
          call      setflags            ;dar valor a las banderas del DS5000T
          inc       pc                  ;PC = PC+1
```

```
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BC rel
;-------------------------------------------------------------------------
ejec_40:
          preserv_pc                    ;preservar PC para ejecucion reversa
          add       pc,2                ;PC = PC+2
          call      check_cy            ;carry = 1?
          jnc       fin_40              ;no, finalizar
          lodsb                         ;si, obtener operando
          cbw
          add       pc,ax               ;simular el salto
fin_40:   mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    OR direct,A; OR direct,#data
;                  OR A,#data;  OR A,direct
;-------------------------------------------------------------------------
ejec_45:
          xor       ah,ah
          preserv_pc                    ;preservar PC para ejecucion reversa
          cmp       al,42h              ;es OR direct,A
          jne       siga_or             ;no
          jmp       ejec0               ;si
siga_or:
          cmp       al,43h              ;es OR direct,#data
          jne       siga_45             ;no
          jmp       ejec1               ;si
siga_45:
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          cmp       al,44h              ;es OR A,#data
          jne       direct_45           ;no
          lodsb                         ;si, obtener el segundo operando
          or        acc,al              ;simular la instruccion
          jmp       cont3
direct_45:
          lodsb                         ;leer el segundo byte de la instruccion
          es_ie_ip?                     ;verificar si es el registro IE o IP
          lea       si,r0               ;direccionar el segundo operando
          add       si,ax
          dir_pines                     ;verificar si es un puerto
          lodsb                         ;obtener el segundo operando
          or        acc,al              ;simular la instruccion
cont3:    add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
          jmp       fin_45              ;finalizar
ejec0:    lodsb                         ;leer segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          lea       si,r0               ;direccionar primer operando
```

```
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]         ;preservar byte de RAM int. para ej.reversa
          mov       al,acc            ;obtener segundo operando
          or        [si],al           ;simular la instruccion
          check_dir2
          check_port                  ;verificar si la instr. fue con un puerto
          jmp       cont3
ejec1:    lodsb                       ;leer segundo byte de la instruccion
          check_dir1                  ;verificar si es algun registro especial
          lea       si,r0             ;direccionar primer operando
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]         ;preservar byte de RAM int. para ej.reversa
          mov       al,[opcode+2]     ;obtener segundo operando
          or        [si],al           ;simular la instruccion
          check_dir2
          check_port                  ;verificar si la instr. fue con un puerto
          add       pc,3              ;PC = PC+3
          mov       cx,2              ;requiere 2 ciclos de maquina
fin_45:   mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    OR A,@Ri
;-------------------------------------------------------------------------
ejec_47:
          preserv_pc                  ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc        ;preservar ACC para ejecucion reversa
          dir_index si,46h            ;direccionar el segundo operando
          jc        error_47          ;direccion no valida
          lodsb                       ;obtener el segundo operando
          or        acc,al            ;simular la instruccion
          jmp       siga_47
error_47:
          mov       acc,0ffh          ;la direccion del seg. operando no es valida
siga_47:
          inc       pc                ;PC = PC+1
          mov       cx,1              ;requiere 1 ciclo de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    OR A,Rn
;-------------------------------------------------------------------------
ejec_4f:
          preserv_pc                  ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc        ;preservar ACC para ejecucion reversa
          dir_reg   si,48h            ;direccionar segundo operando
          mov       al,[si+bx]        ;obtener segundo operando
          or        acc,al            ;simular la instruccion
          inc       pc                ;PC = PC+1
          mov       cx,1              ;requiere 1 ciclo de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
```

```
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BNC rel
;-------------------------------------------------------------------------
ejec_50:
          preserv_pc                    ;preservar PC para ejecucion reversa
          add       pc,2                ;PC = PC+2
          call      check_cy            ;carry = 1?
          jc        fin_50              ;si, finalizar
          lodsb                         ;no, obtener operando
          cbw
          add       pc,ax               ;simular el salto
fin_50:   mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    AND direct,A; AND direct,#data
;                  AND A,#data;  AND A,direct
;-------------------------------------------------------------------------
ejec_55:
          xor       ah,ah
          preserv_pc                    ;preservar PC para ejecucion reversa
          cmp       al,52h              ;es AND direct,A?
          jne       siga_and            ;no
          jmp       ejec2               ;si
siga_and:
          cmp       al,53h              ;es AND direct,#data?
          jne       siga_55             ;no
          jmp       ejec3               ;si
siga_55:
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          cmp       al,54h              ;es AND A,#data?
          jne       direct_55           ;no
          lodsb                         ;si, obtener el segundo operando
          and       acc,al              ;simular la instruccion
          jmp       cont4
direct_55:        lodsb                 ;leer el segundo byte de la instruccion
          es_ie_ip?                     ;verificar si es el registro IE o IP
          lea       si,r0               ;direccionar el segundo operando
          add       si,ax
          dir_pines                     ;verificar si es un puerto
          lodsb                         ;obtener segundo operando
          and       acc,al              ;simular la instruccion
cont4:    add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
          jmp       fin_55              ;finalizar
ejec2:    lodsb                         ;leer segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          lea       si,r0               ;direccionar primer operando
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
```

```
          mov       al,acc              ;obtener segundo operando
          and       [si],al             ;simular la instruccion
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          jmp       cont4
ejec3:    lodsb                         ;leer segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          lea       si,r0               ;direccionar primer operando
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          mov       al,[opcode+2]       ;obtener segundo operando
          and       [si],al             ;simular la instruccion
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          add       pc,3                ;PC = PC+3
          mov       cx,2                ;requiere 2 ciclos de maquina
fin_55:   mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    AND A,@Ri
;-------------------------------------------------------------------------
ejec_57:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          dir_index si,56h              ;direccionar segundo operando
          jc        error_57            ;direccion no valida
          lodsb                         ;obtener segundo operando
          and       acc,al              ;simular la instruccion
          jmp       siga_57
error_57:
          mov       acc,0ffh            ;la direccion del seg. operando no es valida
siga_57:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    AND A,Rn
;-------------------------------------------------------------------------
ejec_5f:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          dir_reg   si,58h              ;direccionar segundo operando
          mov       al,[si+bx]          ;obtener segundo operando
          and       acc,al              ;simular la instruccion
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar
```

```
;
;Simulacion de:    BZ rel
;-------------------------------------------------------------------------
ejec_60:
          preserv_pc                  ;preservar PC para ejecucion reversa
          add       pc,2              ;PC = PC+2
          cmp       acc,0             ;ACC = 0?
          jne       fin_60            ;no, finalizar
          lodsb                       ;si, obtener operando
          cbw
          add       pc,ax             ;simular el salto
fin_60:   mov       cx,2              ;requiere 2 ciclos de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    XOR direct,A; XOR direct,#data
;                  XOR A,#data;  XOR A,direct
;-------------------------------------------------------------------------
ejec_65:
          xor       ah,ah
          preserv_pc                  ;preservar PC para ejecucion reversa
          cmp       al,62h            ;es XOR direct,A?
          jne       siga_xor          ;no
          jmp       ejec4             ;si
siga_xor:
          cmp       al,63h            ;es XOR direct,#data?
          jne       siga_65           ;no
          jmp       ejec5             ;si
siga_65:
          preserv_dir 0e0h,acc        ;preservar ACC para ejecucion reversa
          cmp       al,64h            ;es XOR A,#data?
          jne       direct_65         ;no
          lodsb                       ;si, obtener segundo operando
          xor       acc,al            ;simular la instruccion
          jmp       cont5
direct_65:
          lodsb                       ;leer el segundo byte de la instruccion
          es_ie_ip?                   ;verificar si es el registro IE o IP
          lea       si,r0             ;direccionar segundo operando
          add       si,ax
          dir_pines                   ;verificar si es un puerto
          lodsb                       ;obtener segundo operando
          xor       acc,al            ;simular la instruccion
cont5:    add       pc,2              ;PC = PC+2
          mov       cx,1              ;requiere 1 ciclo de maquina
          jmp       fin_65            ;finalizar
ejec4:    lodsb                       ;obtener segundo operando
          check_dir1                  ;verificar si es algun registro especial
          lea       si,r0             ;direccionar primer operando
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]         ;preservar byte de RAM int. para ej.reversa
          mov       al,acc            ;obtener segundo operando
          xor       [si],al           ;simular la instruccion
```

```
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          jmp       cont5
ejec5:    lodsb                         ;leer el segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          lea       si,r0               ;direccionar primer operando
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          mov       al,[opcode+2]       ;obtener segundo operando
          xor       [si],al             ;simular la instruccion
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          add       pc,3                ;PC = PC+3
          mov       cx,2                ;requiere 2 ciclos de maquina
fin_65:   mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    XOR A,@Ri
;-------------------------------------------------------------------------
ejec_67:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          dir_index si,66h              ;direccionar segundo operando
          jc        error_67            ;direccion no valida
          lodsb                         ;obtener segundo operando
          xor       acc,al              ;simular la instruccion
          jmp       siga_67
error_67:
          mov       acc,0ffh            ;la direccion del seg. operando no es valida
siga_67:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    XOR A,Rn
;-------------------------------------------------------------------------
ejec_6f:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          dir_reg   si,68h              ;direccionar segundo operando
          mov       al,[si+bx]          ;obtener segundo operando
          xor       acc,al              ;simular la instruccion
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BNZ rel
```

```
;-------                                  ---------------------------------
ejec_70:
          preserv_pc                    ;preservar PC para ejecucion reversa
          add       pc,2                ;PC = PC+2
          cmp       acc,0               ;ACC = 0?
          je        fin_70              ;si, finalizar
          lodsb                         ;no, obtener operando
          cbw
          add       pc,ax               ;simular el salto
fin_70:   mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------------
;Simulacion de:    OR C,bit
;-------------------------------------------------------------------------------
ejec_72:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          lodsb                         ;obtener segundo operando
          check_pin1                    ;verificar si es un pin de un puerto
          call      checkbit            ;bit = 1?
          jnc       fin_72              ;no, finalizar
          or        psw,80h             ;si, setear el carry
fin_72:   check_pin2
          add       pc,2                ;PC = PC+2
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------------
;Simulacion de:    JMP @A+DPTR
;-------------------------------------------------------------------------------
ejec_73:
          preserv_pc                    ;preservar PC para ejecucion reversa
          mov       al,dpl              ;calcular la direccion del salto
          mov       ah,dph
          add       al,acc
          adc       ah,0
          mov       pc,ax               ;simular el salto
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------------
;Simulacion de:    LDR A,#data; LDR direct,#data
;-------------------------------------------------------------------------------
ejec_75:
          preserv_pc                    ;preservar PC para ejecucion reversa
          cmp       al,74h              ;es LDR A,#data?
          jne       siga_75             ;no
          jmp       acum_75             ;si
siga_75:
```

```
          lodsb                         ;obtener primer operando
          cmp       al,0c7h             ;es el registro TA?
          jne       siga12              ;no
          jmp       siga11
siga12:   check_dir1                    ;verificar si es algun registro especial
siga13:   mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          jmp       siga1
siga11:   preserv_ta                    ;preservar banderas de TA para ejec. reversa
          xor       dl,dl
          cmp       byte ptr [si],0aah ;el seg. operando es #0AAH?
          jne       flag2?              ;no
          mov       flag_ta1,1          ;si, primer paso para habilitar TA
          mov       flag_ta2,0
          jmp       siga1
flag2?:   cmp       byte ptr [si],055h ;el seg. operando es #55H?
          je        s_1                 ;si
          jmp       siga13              ;no
s_1:      cmp       flag_ta1,1          ;hubo el primer paso para habilitar TA?
          je        s_2                 ;si, habilitar TA
          jmp       siga13              ;no
s_2:      mov       flag_ta2,1
siga1:    lea       di,r0               ;direccionar primer operando
          xor       ah,ah
          add       di,ax
          push      si
          mov       cl,al
          mov       si,di
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          pop       si
          movsb                         ;simular la instruccion
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          add       pc,3                ;PC = PC+3
          mov       cx,2                ;requiere 2 ciclos de maquina
          jmp       fin_75              ;finalizar
acum_75:  lodsb                         ;obtener segundo operando
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          mov       acc,al              ;simular la instruccion
          add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
fin_75:   ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR @Ri,#data
;-------------------------------------------------------------------------
ejec_77:
          preserv_pc                    ;preservar PC para ejecucion reversa
          dir_index di,76h              ;direccionar primer operando
          jc        error_77            ;direccion no valida
          push      si
          mov       cl,al
          mov       si,di
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
```

```
          pop       si
          movsb                       ;simular la instruccion
error_77:
          add       pc,2              ;PC = PC+2
          mov       cx,1              ;requiere 1 ciclo de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR Rn,#data
;-------------------------------------------------------------------------
ejec_7f:
          preserv_pc                  ;preservar PC para ejecucion reversa
          dir_reg   di,78h            ;direccionar primer operando
          push      si
          mov       si,di
          preserv_dir bl,[si+bx]      ;preservar byte de RAM int. para ej.reversa
          pop       si
          lodsb                       ;obtener segundo operando
          mov       [di+bx],al        ;simular la instruccion
          add       pc,2              ;PC = PC+2
          mov       cx,1              ;requiere 1 ciclo de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BRA rel
;-------------------------------------------------------------------------
ejec_80:
          preserv_pc                  ;preservar PC para ejecucion reversa
          add       pc,2              ;PC = PC+2
          lodsb                       ;obtener segundo operando
          cbw
          add       pc,ax             ;simular el salto
          mov       cx,2              ;requiere 2 ciclos de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    AND C,bit
;-------------------------------------------------------------------------
ejec_82:
          preserv_pc                  ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw        ;preservar PSW para ejecucion reversa
          lodsb                       ;obtener segundo operando
          check_pin1                  ;verificar si es un pin de un puerto
          call      checkbit          ;bit = 1?
          jc        fin_82            ;si, finalizar
          and       psw,7fh           ;no, borrar el carry
fin_82:   check_pin2
          add       pc,2              ;PC = PC+2
          mov       cx,2              ;requiere 2 ciclos de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
```

```
        mov     flag_ta2,0
        ret                             ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDC A,@A+PC
;-------------------------------------------------------------------------
ejec_83:
        preserv_pc                      ;preservar PC para ejecucion reversa
        inc     pc                      ;PC = PC+1
        push    ds
        mov     ds,segdat2              ;DS apunta a DSEG2
        lea     si,ram_emb              ;SI = inicio del programa del usuario
        pop     ds
        mov     al,acc                  ;direccionar segundo operando
        xor     ah,ah
        add     si,ax
        add     si,pc
        cmp     si,rom_ram_e
        jae     error_83                ;el operando esta fuera de ROM
        push    ds
        mov     ds,segdat2              ;DS apunta a DSEG2
        lodsb                           ;obtener segundo operando
        pop     ds
        preserv_dir 0e0h,acc            ;preservar ACC para ejecucion reversa
        mov     acc,al                  ;simular la instruccion
error_83:
        mov     cx,2                    ;requiere 2 ciclos de maquina
        mov     flag_ta1,0              ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                             ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    DIV AB
;-------------------------------------------------------------------------
ejec_84:
        preserv_pc                      ;preservar PC para ejecucion reversa
        preserv_dir 0d0h,psw            ;preservar PSW para ejecucion reversa
        mov     al,acc
        xor     ah,ah
        cmp     b,0                     ;division para cero?
        je      overflow                ;si, overflow
        div     b                       ;no, simular la instruccion
        preserv_dir 0e0h,acc            ;preservar ACC para ejecucion reversa
        preserv_dir 0e0h,b              ;preservar B para ejecucion reversa
        mov     acc,al                  ;cociente en el acumulador
        mov     b,ah                    ;residuo en el registro B
        and     psw,7bh                 ;borrar el carry
        jmp     fin_84                  ;finalizar
overflow:
        or      psw,4                   ;setear bandera de overflow
        and     psw,7fh                 ;borrar el carry
fin_84: inc     pc                      ;PC = PC+1
        mov     cx,4                    ;requiere 4 ciclos de maquina
        mov     flag_ta1,0              ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                             ;retornar
```

```
;-------------------------------------------------------------------------
;Simulacion de:    LDR direct1,direct2
;-------------------------------------------------------------------------
ejec_85:
          preserv_pc                    ;preservar PC para ejecucion reversa
          lodsb                         ;obtener segundo operando
          lea       si,r0               ;direccionar segundo operando
          xor       ah,ah
          add       si,ax
          dir_pines                     ;verificar si es un puerto
          mov       al,[opcode+2]       ;leer tercer byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          lea       di,r0               ;direccionar primer operando
          add       di,ax
          push      si
          mov       cl,al
          mov       si,di
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          pop       si
          movsb                         ;simular la instruccion
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          add       pc,3                ;PC = PC+3
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR direct,@Ri
;-------------------------------------------------------------------------
ejec_87:
          preserv_pc                    ;preservar PC para ejecucion reversa
          dir_index di,86h              ;direccionar segundo operando
          jnc       siga_87
          jmp       error_87            ;direccion no valida
siga_87:
          lodsb                         ;leer segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          lea       si,r0               ;direccionar primer operando
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          mov       al,[di]             ;simular la instruccion
          mov       [si],al
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
error_87:
          add       pc,2                ;PC = PC+2
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR direct,Rn
```

```
;------
ejec_8f:
        preserv_pc                  ;preservar PC para ejecucion reversa
        dir_reg  di,88h             ;direccionar segundo operando
        lodsb                       ;leer segundo byte de la instruccion
        check_dir1                  ;verificar si es algun registro especial
        lea      si,r0              ;direccionar primer operando
        add      si,ax
        mov      cl,al
        preserv_dir cl,[si]         ;preservar byte de RAM int. para ej.reversa
        mov      al,[di+bx]         ;simular la instruccion
        mov      [si],al
        check_dir2
        check_port                  ;verificar si la instr. fue con un puerto
        add      pc,2               ;PC = PC+2
        mov      cx,2               ;requiere 2 ciclos de maquina
        mov      flag_ta1,0         ;deshabilitar Timed Access
        mov      flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR DPTR,#data16
;-------------------------------------------------------------------------
ejec_90:
        preserv_pc                  ;preservar PC para ejecucion reversa
        preserv_dir 82h,dpl         ;preservar DPTR para ejecucion reversa
        preserv_dir 83h,dph
        lea      di,dph             ;simular la instruccion
        movsb
        lea      di,dpl
        movsb
        add      pc,3               ;PC = PC+3
        mov      cx,2               ;requiere 2 ciclos de maquina
        mov      flag_ta1,0         ;deshabilitar Timed Access
        mov      flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDB bit,C
;-------------------------------------------------------------------------
ejec_92:
        preserv_pc                  ;preservar PC para ejecucion reversa
        lodsb                       ;obtener primer operando
        call     check_cy           ;carry = 1?
        jnc      no_cy7             ;no
        mov      dl,1               ;si, hacer bit = 1
        jmp      conté
no_cy7: xor      dl,dl              ;hacer bit = 0
conté:  call     setbit
        add      pc,2               ;PC = PC+2
        mov      cx,2               ;requiere 2 ciclos de maquina
        mov      flag_ta1,0         ;deshabilitar Timed Access
        mov      flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
```

```
;Simulacion de:    LDC A,@A+DPTR
;-------------------------------------------------------------------------
ejec_93:
          preserv_pc                    ;preservar PC para ejecucion reversa
          push      ds
          mov       ds,segdat2          ;DS apunta a DSEG2
          lea       si,ram_emb          ;SI apunta al programa del usuario
          pop       ds                  ;DS apunta a DSEG
          mov       al,acc              ;direccionar segundo operando
          xor       ah,ah
          add       si,ax
          lea       di,dpl
          mov       ax,word ptr [di]
          add       si,ax
          cmp       si,rom_ram_e
          jae       error_93            ;el operando esta fuera de ROM
          push      ds
          mov       ds,segdat2          ;DS apunta a DSEG2
          lodsb                         ;obtener segundo operando
          pop       ds                  ;DS apunta a DSEG
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          mov       acc,al              ;simular la instruccion
error_93:
          inc       pc                  ;PC = PC+1
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    SBC A,#data; SBC A,direct
;-------------------------------------------------------------------------
ejec_95:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          xor       ah,ah
          cmp       al,94h              ;es SBC A,#data?
          jne       direct_95           ;no
          sahf                          ;borrar banderas del 8086
          call      check_cy            ;obtener el estado del carry
          lodsb                         ;obtener segundo operando
          sbb       acc,al              ;simular la instruccion
          jmp       cont7
direct_95:          lodsb               ;leer segundo byte de la instruccion
          es_ie_ip?                     ;verificar si es el registro IE o IP
          lea       si,r0               ;direccionar segundo operando
          add       si,ax
          dir_pines                     ;verificar si es un puerto
          sahf                          ;borrar banderas del 8086
          call      check_cy            ;obtener el estado del carry
          lodsb                         ;obtener segundo operando
          sbb       acc,al              ;simular la instruccion
cont7:    call      setflags            ;dar valor a las banderas del DS5000T
          add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
```

```
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    SBC A,@Ri
;-------------------------------------------------------------------------
ejec_97:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          dir_index si,96h              ;direccionar segundo operando
          jc        error_97            ;direccion no valida
          sahf                          ;borrar banderas del 8086
          call      check_cy            ;obtener el estado del carry
          lodsb                         ;obtener segundo operando
          sbb       acc,al              ;simular la instruccion
          call      setflags            ;dar valor a las banderas del DS5000T
          jmp       siga_97
error_97:
          mov       acc,0ffh            ;la direccion del seg. operando no es valida
siga_97:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    SBC A,Rn
;-------------------------------------------------------------------------
ejec_9f:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          dir_reg   si,98h              ;direccionar el segundo operando
          sahf                          ;borrar banderas del 8086
          call      check_cy            ;obtener el estado del carry
          mov       al,[si+bx]          ;obtener segundo operando
          sbb       acc,al              ;simular la instruccion
          call      setflags            ;dar valor a las banderas del DS5000T
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    OR C,/bit
;-------------------------------------------------------------------------
ejec_a0:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          lodsb                         ;obtener segundo operando
          check_pin1                    ;verificar si es un pin de un puerto
          call      checkbit            ;bit = 1?
```

```
          jc        fin_a0              ;si, finalizar
          or        psw,80h             ;no, setear el carry
fin_a0:   check_pin2
          add       pc,2                ;PC = PC+2
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDB C,bit
;-------------------------------------------------------------------------
ejec_a2:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          lodsb                         ;obtener segundo operando
          check_pin1                    ;verificar si es un pin de un puerto
          call      checkbit            ;bit = 1?
          jnc       no_bit              ;no
          or        psw,80h             ;si, setear el carry
          jmp       cont8
no_bit:   and       psw,7fh             ;borrar el carry
cont8:    check_pin2
          add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    INC DPTR
;-------------------------------------------------------------------------
ejec_a3:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 82h,dpl           ;preservar DPTR para ejecucion reversa
          preserv_dir 83h,dph
          lea       si,dpl              ;simular la instruccion
          lodsw
          inc       ax
          lea       di,dpl
          stosw
          inc       pc                  ;PC = PC+1
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    MUL AB
;-------------------------------------------------------------------------
ejec_a4:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          mov       al,acc
          xor       ah,ah
          mul       b                   ;simular la instruccion
```

```
          preserv_dir 0e0h,acc        ;preservar ACC para ejecucion reversa
          preserv_dir 0e0h,b          ;preservar B para ejecucion reversa
          mov       acc,al            ;resultado en ACC y registro B
          mov       b,ah
          jo        si_ovf            ;hubo overflow?
          and       psw,0fbh          ;no, borrar bandera de overflow
          jmp       cont9
si_ovf:   or        psw,4             ;si, setear bandera de overflow
cont9:    and       psw,7fh           ;borrar bandera de carry
          inc       pc                ;PC = PC+1
          mov       cx,4              ;requiere 4 ciclos de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR @Ri,direct
;-------------------------------------------------------------------------
ejec_a7:
          preserv_pc                  ;preservar PC para ejecucion reversa
          dir_index di,0a6h           ;direccionar primer operando
          jnc       siga_a7           ;direccion valida
          jmp       error_a7          ;direccion no valida
siga_a7:
          push      si
          mov       cl,al
          mov       si,di
          preserv_dir cl,[si]         ;preservar byte de RAM int. para ej.reversa
          pop       si
          lodsb                       ;leer segundo byte de la instruccion
          es_ie_ip?                   ;verificar si es el registro IE o IP
          lea       si,r0             ;direccionar segundo operando
          add       si,ax
          dir_pines                   ;verificar si es un puerto
          movsb                       ;simular la instruccion
error_a7:
          add       pc,2              ;PC = PC+2
          mov       cx,2              ;requiere 2 ciclos de maquina
          mov       flag_ta1,0        ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR Rn,direct
;-------------------------------------------------------------------------
ejec_af:
          preserv_pc                  ;preservar PC para ejecucion reversa
          dir_reg   di,0a8h           ;direccionar primer operando
          push      si
          mov       si,di
          preserv_dir bl,[si+bx]      ;preservar byte de RAM int. para ej.reversa
          pop       si
          lodsb                       ;leer segundo byte de la instruccion
          es_ie_ip?                   ;verificar si es el registro IE o IP
          lea       si,r0             ;direccionar segundo operando
          add       si,ax
```

```
          dir_pines                     ;verificar si es un puerto
          lodsb                         ;obtener segundo operando
          mov       [di+bx],al          ;simular la instruccion
          add       pc,2                ;PC = PC+2
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    AND C,/bit
;-------------------------------------------------------------------------
ejec_b0:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          lodsb                         ;obtener segundo operando
          check_pin1                    ;verificar si es un pin de un puerto
          call      checkbit            ;bit = 1?
          jnc       fin_b0              ;no, finalizar
          and       psw,7fh             ;si, borrar el carry
fin_b0:   check_pin2
          add       pc,2                ;PC = PC+2
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BCOM bit, COM C
;-------------------------------------------------------------------------
ejec_b3:
          preserv_pc                    ;preservar PC para ejecucion reversa
          cmp       al,0b3h             ;es COM C?
          je        carry               ;si
          lodsb                         ;no, obtener operando
          call      checkbit            ;bit = 1?
          jnc       set                 ;no, hacer bit = 1
          xor       dl,dl               ;si, hacer bit = 0
          jmp       cont10
set:      mov       dl,1
cont10:   call      setbit              ;simular la instruccion
          add       pc,2                ;PC = PC+2
          jmp       fin_b3              ;finalizar
carry:    preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          call      check_cy            ;carry = 1?
          jnc       set_cy              ;no, setear el carry
          and       psw,7fh             ;si, borrar el carry
          jmp       cont11
set_cy:   or        psw,80h             ;setear el carry
cont11:   inc       pc                  ;PC = PC+1
fin_b3:   mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
```

```
;Simulacion de:    CBNE A,#data,rel; CBNE A,direct,rel
;-------------------------------------------------------------------------
ejec_b5:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          add       pc,3                ;PC = PC+3
          cmp       al,0b4h             ;es CBNE A,#data,rel?
          je        data_b5             ;si
          lodsb                         ;leer segundo byte de la instruccion
          es_ie_ip?                     ;verificar si es el registro IE o IP
          push      si
          lea       si,r0               ;direccionar segundo operando
          xor       ah,ah
          add       si,ax
          dir_pines                     ;verificar si es un puerto
          mov       al,acc              ;obtener primer operando
          cmp       al,[si]             ;primer operando = segundo operando?
          pop       si
          jmp       cont12
data_b5:  lodsb                         ;obtener segundo operando
          cmp       acc,al              ;primer operando = segundo operando?
cont12:   call      check_cbne          ;decidir si debe saltar y setear banderas
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    CBNE @Ri,#data,rel
;-------------------------------------------------------------------------
ejec_b7:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          add       pc,3                ;PC = PC+3
          dir_index di,0b6h             ;direccionar el primer operando
          jc        error_b7            ;direccion no valida
          mov       al,[di]             ;obtener el primer operando
          inc       si                  ;direccionar el segundo operando
          cmp       al,[si-1]           ;primer operando = segundo operando?
          call      check_cbne          ;decidir si debe saltar y setear banderas
error_b7:
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    CBNE Rn,#data,rel
;-------------------------------------------------------------------------
ejec_bf:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          add       pc,3                ;PC = PC+3
          dir_reg   di,0b8h             ;direccionar el primer operando
          mov       al,[di+bx]
          inc       si                  ;direccionar el segundo operando
```

```
          cmp       al,[si-1]           ;primer operando = segundo operando?
          call      check_cbne          ;decidir si debe saltar y setear banderas
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    PUSH direct
;-------------------------------------------------------------------------
ejec_c0:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 81h,regsp         ;preservar SP para ejecucion reversa
          inc       regsp               ;SP = SP+1
          lea       di,r0
          mov       al,regsp
          xor       ah,ah
          add       di,ax               ;DI apunta al tope del stack
          push      si
          mov       si,di
          preserv_stack                 ;preservar el stack para ejecucion reversa
          pop       si
          lodsb                         ;leer segundo byte de la instruccion
          es_ie_ip?                     ;verificar si es el registro IE o IP
          lea       si,r0               ;direccionar operando
          add       si,ax
          dir_pines                     ;verificar si es un puerto
          movsb                         ;simular la instruccion
          add       pc,2                ;PC = PC+2
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BCLR bit; CLR C
;-------------------------------------------------------------------------
ejec_c3:
          preserv_pc                    ;preservar PC para ejecucion reversa
          cmp       al,0c3h             ;es CLR C?
          je        carry1              ;si
          lodsb                         ;obtener operando
          xor       dl,dl               ;hacer bit = 0
          call      setbit
          add       pc,2                ;PC = PC+2
          jmp       fin_c3              ;finalizar
carry1:   preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          and       psw,7fh             ;borrar el carry
          inc       pc                  ;PC = PC+1
fin_c3:   mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    SWAP A
```

```
ejec_c4:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          mov       cl,4                ;simular la instruccion
          rol       acc,cl
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    EXH A,direct
;-------------------------------------------------------------------------
ejec_c5:
          preserv_pc                    ;preservar PC para ejecucion reversa
          lodsb                         ;leer segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          xor       ah,ah
          lea       si,r0               ;direccionar segundo operando
          add       si,ax
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          dir_pines                     ;verificar si es un puerto
          mov       al,[si]             ;obtener segundo operando
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          xchg      acc,al              ;simular la instruccion
          mov       [si],al
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    EXH A,@Ri
;-------------------------------------------------------------------------
ejec_c7:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          dir_index si,0c6h             ;direccionar segundo operando
          jc        error_c7            ;direccion no valida
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          mov       al,[si]             ;obtener el segundo operando
          xchg      acc,al              ;simular la instruccion
          mov       [si],al
          jmp       siga_c7
error_c7:
          mov       acc,0ffh            ;la direccion del seg. operando no es valida
siga_c7:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
```

```
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    EXH A,Rn
;-------------------------------------------------------------------------
ejec_cf:
        preserv_pc                  ;preservar PC para ejecucion reversa
        dir_reg  si,0c8h            ;direccionar segundo operando
        preserv_dir bl,[si+bx]      ;preservar byte de RAM int. para ej.reversa
        mov     al,[si+bx]          ;obtener segundo operando
        preserv_dir 0e0h,acc        ;preservar ACC para ejecucion reversa
        xchg    acc,al              ;simular la instruccion
        mov     [si+bx],al
        inc     pc                  ;PC = PC+1
        mov     cx,1                ;requiere 1 ciclo de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    PULL direct
;-------------------------------------------------------------------------
ejec_d0:
        preserv_pc                  ;preservar PC para ejecucion reversa
        preserv_dir 81h,regsp       ;preservar SP para ejecucion reversa
        lodsb                       ;leer segundo byte de la instruccion
        check_dir1                  ;verificar si es algun registro especial
        lea     di,r0               ;DI apunta al operando
        xor     ah,ah
        add     di,ax
        push    si
        mov     cl,al
        mov     si,di
        preserv_dir cl,[si]         ;preservar byte de RAM int. para ej.reversa
        pop     si
        lea     si,r0
        mov     al,regsp
        add     si,ax               ;SI apunta al tope del stack
        movsb                       ;simular la instruccion
        check_dir2
        check_port                  ;verificar si la instr. fue con un puerto
        dec     regsp               ;decrementar el Stack Pointer
        add     pc,2                ;PC = PC+2
        mov     cx,2                ;requiere 2 ciclos de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    BSET bit; BSET C
;-------------------------------------------------------------------------
ejec_d3:
        preserv_pc                  ;preservar PC para ejecucion reversa
        cmp     al,0d3h             ;es BSET C?
```

```
          je        carry2              ;si
          lodsb                         ;no, obtener operando
          mov       dl,1
          call      setbit              ;hacer bit = 0
          add       pc,2                ;PC = PC+2
          jmp       fin_d3              ;finalizar
carry2:   preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          or        psw,80h             ;setear el carry
          inc       pc                  ;PC = PC+1
fin_d3:   mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    DAA
;-------------------------------------------------------------------------
ejec_d4:
          preserv_pc                    ;preservar PC para ejecucion reversa
          mov       al,acc
          xor       ah,ah
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          preserv_dir 0d0h,psw          ;preservar PSW para ejecucion reversa
          daa                           ;simular la instruccion
          mov       acc,al
          jc        carry3
          and       psw,7fh             ;borrar el carry
          jmp       fin_d4              ;finalizar
carry3:   or        psw,80h             ;setear el carry
fin_d4:   inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    DBNZ direct,rel
;-------------------------------------------------------------------------
ejec_d5:
          preserv_pc                    ;preservar PC para ejecucion reversa
          add       pc,3                ;PC = PC+3
          lodsb                         ;leer segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          xor       ah,ah
          lea       di,r0               ;direccionar primer operando
          add       di,ax
          push      si
          mov       cl,al
          mov       si,di
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          pop       si
          dec       byte ptr [di]       ;decrementar primer operando
          cmp       byte ptr [di],0     ;primer operando = 0?
          je        fin_d5              ;si, finalizar
          lodsb                         ;no, obtener segundo operando
          cbw
```

```
          add       pc,ax               ;simular el salto
fin_d5:   check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    EXHD A,@Ri
;-------------------------------------------------------------------------
ejec_d7:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          dir_index si,0d6h             ;direccionar segundo operando
          jc        error_d7            ;direccion no valida
          mov       cl,al
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          mov       bl,[si]             ;simular la instruccion
          mov       bh,bl
          mov       al,acc
          mov       ah,al
          or        al,0f0h
          or        bl,0f0h
          xchg      al,bl
          or        ah,0fh
          or        bh,0fh
          and       al,ah
          and       bl,bh
          mov       acc,al
          mov       [si],bl
          jmp       siga_d7
error_d7:
          mov       acc,0ffh            ;la direccion del seg. operando no es valida
siga_d7:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    DBNZ Rn,rel
;-------------------------------------------------------------------------
ejec_df:
          preserv_pc                    ;preservar PC para ejecucion reversa
          add       pc,2                ;PC = PC+2
          dir_reg   di,0d8h             ;direccionar primer operando
          push      si
          mov       si,di
          preserv_dir bl,[si+bx]        ;preservar byte de RAM int. para ej.reversa
          pop       si
          dec       byte ptr [di+bx]    ;decrementar primer operando
          cmp       byte ptr [di+bx],0  ;primer operando = 0?
          je        fin_df              ;si, finalizar
          lodsb                         ;no, obtener segundo operando
```

```
          cbw
          add       pc,ax               ;simular el salto
fin_df:   mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDX A,@DPTR
;-------------------------------------------------------------------------
ejec_e0:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          test      mcon,4              ;es una instruccion con el ECC?
          jnz       cont_e0             ;si
          jmp       siga_e0             ;no
cont_e0:
          cmp       dph,0               ;es una direccion valida para acceso al ECC?
          je        cont_e01            ;si
          jmp       error_ecc0          ;no
cont_e01:
          preserv_ecc                   ;preservar banderas usadas con el ECC
          cmp       dpl,4               ;leer el ECC?
          jne       write?              ;no
          cmp       reset_ecc,0         ;si, fue reseteado el ECC?
          je        read?               ;si
          dec       reset_ecc           ;no, ejecutar proceso para reset del ECC
          jmp       error_e0            ;finalizar
read?:    cmp       rec_patt,0          ;fue ingresado el patron de bits?
          je        cont_e02            ;si
          jmp       error_e0            ;no, finalizar
cont_e02:
          cmp       ecc_point,63        ;se han realizado ya 64 lecturas?
          ja        bloqueo             ;si, bloquear el ECC
          cmp       byte ptr [ecc+37],0b3h ;esta funcionando el ECC?
          je        siga_e03            ;no
          call      lee_hora            ;poner en el ECC la hora del computador
          call      lee_fecha           ;poner en el ECC la fecha del computador
siga_e03:
          lea       si,ecc              ;SI apunta al primer bit del ECC
          mov       al,ecc_point
          xor       ah,ah
          add       si,ax               ;SI apunta al bit del ECC que se va a leer
          lodsb                         ;leer un bit del ECC
          mov       acc,al
          inc       ecc_point           ;apuntar al siguiente bit del ECC
          jmp       siga_e01            ;finalizar
bloqueo:
          mov       reset_ecc,64        ;bloquear el ECC
          mov       rec_patt,64
          mov       ecc_point,0
          jmp       siga_e01
write?:   cmp       dpl,1               ;escribir en el ECC
          jbe       cont_e03            ;si
          jmp       error_ecc0          ;no, finalizar
cont_e03:
```

```
            cmp     reset_ecc,0         ;fue reseteado el ECC?
            je      write1?             ;si
            jmp     error_e0            ;no, finalizar
write1?:
            cmp     rec_patt,0          ;fue ingresado el patron de bits?
            jne     pattern?            ;no
            cmp     ecc_point,63        ;si, se han hecho ya 64 escrituras?
            ja      bloqueo1            ;si, bloquear el ECC
            call    check_ecc           ;no, escribir en algun bit no modificable?
            jc      siga_e04            ;si, dejar inalterado el bit
            lea     si,ecc              ;no, escribir el bit
            cmp     dpl,0               ;escribir un cero?
            jne     wr_uno              ;no
            mov     al,ecc_point        ;si, escribir un cero
            xor     ah,ah
            add     si,ax
            mov     byte ptr [si],33h
            jmp     siga_e02
wr_uno:     mov     al,ecc_point        ;escribir un uno
            xor     ah,ah
            add     si,ax
            mov     byte ptr [si],0b3h
siga_e02:
            call    igual_hora          ;setear hora del computador con al ECC
            call    igual_fecha         ;setear fecha del computador con el ECC
siga_e04:
            inc     ecc_point           ;apuntar al siguiente bit del ECC
            jmp     siga_e01            ;finalizar
bloqueo1:
            mov     reset_ecc,64        ;bloquear el ECC
            mov     rec_patt,64
            mov     ecc_point,0
            jmp     error_e0            ;finalizar
pattern?:
            lea     si,pattern          ;escribir el patron de bits
            mov     al,rec_patt
            xor     ah,ah
            add     si,ax
            dec     si
            lodsb
            cmp     al,dpl              ;el bit coincide con el patron?
            jne     bloqueo1            ;no, bloquear el ECC
            dec     rec_patt            ;si, seguir proceso de reconoc. de patron
            jmp     siga_e01            ;finalizar
siga_e0:
            push    ds                  ;acceso normal a RAM embebida
            mov     ds,segdat2          ;DS apunta a DSEG2
            lea     si,ram_emb          ;SI apunta al inicio de la memoria embebida
            pop     ds                  ;DS apunta a DSEG
            lea     di,dpl
            mov     ax,[di]
            add     si,ax
            mov     ax,ram_emb_m
            cmp     si,ax               ;es una direccion valida?
            jae     error_e0            ;no, finalizar
            mov     ax,rom_ram_e        ;si
```

```
          cmp       si,ax               ;es una direccion valida?
          jb        error_e0            ;no, finalizar
          push      ds                  ;si
          mov       ds,segdat2          ;DS apunta a DSEG2
          lodsb                         ;obtener segundo operando
          pop       ds
          mov       acc,al              ;simular la instruccion
          jmp       siga_e01            ;finalizar
error_ecc0:
          mov       acc,0ffh            ;la direccion del seg. operando no es valida
          jmp       siga_e01
error_e0:
          mov       acc,0               ;no esta habilitado el acceso al ECC
siga_e01:
          inc       pc                  ;PC = PC+1
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDX A,@Ri
;-------------------------------------------------------------------------
ejec_e3:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          test      mcon,4              ;es una instruccion con el ECC?
          jz        siga_e3             ;no
          mov       acc,0ffh            ;si, no puede acceder al ECC con esta instr.
          jmp       error_e3            ;finalizar
siga_e3:                                ;acceso normal a RAM embebida
          lea       si,r0               ;direccionar segundo operando
          xor       ah,ah
          sub       al,0e2h
          call      sel_bank
          mov       al,[si+bx]
          push      ds
          mov       ds,segdat2          ;DS apunta a DSEG2
          lea       si,ram_emb          ;SI apunta al inicio de la memoria embebida
          pop       ds                  ;DS apunta a DSEG
          xor       ah,ah
          add       si,ax
          mov       ax,ram_emb_m
          cmp       si,ax               ;es una direccion valida?
          jae       error_e3            ;no, finalizar
          mov       ax,rom_ram_e        ;si
          cmp       si,ax               ;es una direccion valida?
          jb        error_e3            ;no, finalizar
          push      ds                  ;si
          mov       ds,segdat2          ;DS apunta a DSEG2
          lodsb                         ;obtener segundo operando
          pop       ds                  ;DS apunta a DSEG
          mov       acc,al              ;simular la instruccion
error_e3:
          mov       acc,0               ;la direccion del seg. operando no es valida
          inc       pc                  ;PC = PC+1
```

```
        mov     cx,2                ;requiere 2 ciclos de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    CLR A
;-------------------------------------------------------------------------
ejec_e4:
        preserv_pc                  ;preservar PC para ejecucion reversa
        preserv_dir 0e0h,acc        ;preservar ACC para ejecucion reversa
        mov     acc,0               ;simular la instruccion
        inc     pc                  ;PC = PC+1
        mov     cx,1                ;requiere 1 ciclo de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR A,direct
;-------------------------------------------------------------------------
ejec_e5:
        preserv_pc                  ;preservar PC para ejecucion reversa
        preserv_dir 0e0h,acc        ;preservar ACC para ejecucion reversa
        lodsb                       ;leer segundo byte de la instruccion
        es_ie_ip?                   ;verificar si es el registro IE o IP
        xor     ah,ah
        lea     si,r0               ;direccionar segundo operando
        add     si,ax
        dir_pines                   ;verificar si es un puerto
        lodsb                       ;obtener segundo operando
        mov     acc,al              ;simular la instruccion
        add     pc,2                ;PC = PC+2
        mov     cx,1                ;requiere 1 ciclo de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
        ret                         ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR A,@Ri
;-------------------------------------------------------------------------
ejec_e7:
        preserv_pc                  ;preservar PC para ejecucion reversa
        preserv_dir 0e0h,acc        ;preservar ACC para ejecucion reversa
        dir_index si,0e6h           ;direccionar segundo operando
        jc      error_e7            ;direccion no valida
        lodsb                       ;obtener segundo operando
        mov     acc,al              ;simular la instruccion
        jmp     siga_e7
error_e7:
        mov     acc,0ffh            ;la direccion del seg. operando no es valida
siga_e7:
        inc     pc                  ;PC = PC+1
        mov     cx,1                ;requiere 1 ciclo de maquina
        mov     flag_ta1,0          ;deshabilitar Timed Access
        mov     flag_ta2,0
```

```
        ret                             ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR A,Rn
;-------------------------------------------------------------------------
ejec_ef:
        preserv_pc                      ;preservar PC para ejecucion reversa
        preserv_dir 0e0h,acc            ;preservar ACC para ejecucion reversa
        dir_reg  si,0e8h                ;direccionar segundo operando
        mov      al,[si+bx]             ;obtener segundo operando
        mov      acc,al                 ;simular la instruccion
        inc      pc                     ;PC = PC+1
        mov      cx,1                   ;requiere 1 ciclo de maquina
        mov      flag_ta1,0             ;deshabilitar Timed Access
        mov      flag_ta2,0
        ret                             ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDX @DPTR,A
;-------------------------------------------------------------------------
ejec_f0:
        preserv_pc                      ;preservar PC para ejecucion reversa
        test     mcon,4                 ;es una instruccion con el ECC?
        jnz      cont_f02               ;si
        jmp      siga_f0                ;no
cont_f02:
        cmp      dph,0                  ;es una direccion valida para acceso al ECC?
        je       cont_f0                ;si
        jmp      siga_f01               ;no
cont_f0:
        cmp      dpl,1                  ;escribir en el ECC?
        jbe      cont_f01               ;si
        jmp      siga_f01               ;no, finalizar
cont_f01:
        preserv_ecc                     ;preservar banderas usadas con el ECC
        cmp      reset_ecc,0            ;fue reseteado el ECC?
        je       write2?                ;si
        jmp      siga_f01               ;no, finalizar
write2?:
        cmp      rec_patt,0             ;fue ingresado el patron de bits?
        jne      pattern1?              ;no
        cmp      ecc_point,63           ;si, se han hecho ya 64 escrituras?
        ja       bloqueo2               ;si, bloquear el ECC
        call     check_ecc              ;no, escribir en algun bit no modificable?
        jc       siga_f03               ;si, dejar inalterado el bit
        lea      si,ecc                 ;no, escribir el bit
        cmp      dpl,0                  ;escribir un cero?
        jne      wr_uno1                ;no
        mov      al,ecc_point           ;si, escribir un cero
        xor      ah,ah
        add      si,ax
        mov      byte ptr [si],33h
        jmp      siga_f02
wr_uno1:
        mov      al,ecc_point           ;escribir un uno
        xor      ah,ah
```

```
          add       si,ax
          mov       byte ptr [si],0b3h
siga_f02:
          call      igual_hora          ;setear hora del computador con el ECC
          call      igual_fecha         ;setear fecha del computador con el ECC
siga_f03:
          inc       ecc_point           ;apuntar al siguiente bit del ECC
          jmp       siga_f01            ;finalizar
bloqueo2:
          mov       reset_ecc,64        ;bloquear el ECC
          mov       rec_patt,64
          mov       ecc_point,0
          jmp       siga_f01            ;finalizar
pattern1?:
          lea       si,pattern          ;escribir el patron de bits
          mov       al,rec_patt
          xor       ah,ah
          add       si,ax
          dec       si
          lodsb
          cmp       al,dpl              ;el bit coincide con el patron?
          jne       bloqueo2            ;no, bloquear el ECC
          dec       rec_patt            ;si, seguir proceso de reconoc. de patron
          jmp       siga_f01            ;finalizar
siga_f0:
          push      es                  ;acceso normal a RAM embebida
          mov       es,segdat2          ;ES apunta a DSEG2
          lea       di,ram_emb          ;DI apunta al inicio de la memoria embebida
          pop       es                  ;ES apunta a DSEG
          lea       si,dpl              ;direccionar primer operando
          mov       ax,[si]
          add       di,ax
          mov       ax,ram_emb_m
          cmp       di,ax               ;es una direccion valida?
          jae       siga_f01            ;no, finalizar
          mov       ax,rom_ram_e        ;si
          cmp       di,ax               ;es una direccion valida?
          jb        siga_f01            ;no, finalizar
          mov       cx,di               ;si
          push      es
          mov       es,segdat2          ;ES apunta a DSEG 2
          mov       dh,es:[di]
          preserv_mem cx,dh             ;preservar byte de RAM emb. para ej.reversa
          mov       al,acc
          stosb                         ;simular la instruccion
          pop       es                  ;ES apunta a DSEG
siga_f01:
          inc       pc                  ;PC = PC+1
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDX @Ri,A
;-------------------------------------------------------------------------
```

```
ejec_f3:
          preserv_pc                    ;preservar PC para ejecucion reversa
          test      mcon,4              ;es una instruccion con el ECC?
          jz        siga_f3             ;no
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          mov       acc,0ffh            ;si, no puede acceder al ECC con esta instr.
          jmp       error_f3            ;finalizar
siga_f3:                                ;acceso normal a RAM embebida
          lea       si,r0               ;direccionar primer operando
          xor       ah,ah
          sub       al,0f2h
          call      sel_bank
          mov       al,[si+bx]
          push      es
          mov       es,segdat2
          lea       di,ram_emb
          pop       es
          xor       ah,ah
          add       di,ax
          mov       ax,ram_emb_m
          cmp       di,ax               ;es una direccion valida?
          jae       error_f3            ;no
          mov       ax,rom_ram_e        ;si
          cmp       di,ax               ;es una direccion valida?
          jb        error_f3            ;no
          mov       cx,di               ;si
          push      es
          mov       es,segdat2          ;ES apunta a DSEG2
          mov       dh,es:[di]
          preserv_mem cx,dh             ;preservar byte de RAM emb. para ej. reversa
          mov       al,acc              ;obtener segundo operando
          stosb                         ;simular la instruccion
          pop       es                  ;ES apunta a DSEG
error_f3:
          inc       pc                  ;PC = PC+1
          mov       cx,2                ;requiere 2 ciclos de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    COM A
;-------------------------------------------------------------------------
ejec_f4:
          preserv_pc                    ;preservar PC para ejecucion reversa
          preserv_dir 0e0h,acc          ;preservar ACC para ejecucion reversa
          not       acc                 ;simular la instruccion
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR direct,A
;-------------------------------------------------------------------------
```

```
ejec_f5:
          preserv_pc                    ;preservar PC para ejecucion reversa
          lodsb                         ;leer segundo byte de la instruccion
          check_dir1                    ;verificar si es algun registro especial
          xor       ah,ah
          lea       di,r0               ;direccionar primer operando
          add       di,ax
          push      si
          mov       cl,al
          mov       si,di
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          pop       si
          mov       al,acc
          stosb                         ;simular la instruccion
          check_dir2
          check_port                    ;verificar si la instr. fue con un puerto
          add       pc,2                ;PC = PC+2
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR @Ri,A
;-------------------------------------------------------------------------
ejec_f7:
          preserv_pc                    ;preservar PC para ejecucion reversa
          dir_index di,0f6h             ;direccionar primer operando
          jc        error_f7            ;direccion no valida
          push      si
          mov       cl,al
          mov       si,di
          preserv_dir cl,[si]           ;preservar byte de RAM int. para ej.reversa
          pop       si
          mov       al,acc
          stosb                         ;simular la instruccion
error_f7:
          inc       pc                  ;PC = PC+1
          mov       cx,1                ;requiere 1 ciclo de maquina
          mov       flag_ta1,0          ;deshabilitar Timed Access
          mov       flag_ta2,0
          ret                           ;retornar

;-------------------------------------------------------------------------
;Simulacion de:    LDR Rn,A
;-------------------------------------------------------------------------
ejec_ff:
          preserv_pc                    ;preservar PC para ejecucion reversa
          dir_reg   di,0f8h             ;direccionar primer operando
          push      si
          mov       si,di
          preserv_dir bl,[si+bx]        ;preservar byte de RAM int. para ej.reversa
          pop       si
          mov       al,acc
          mov       [di+bx],al          ;simular la instruccion
          inc       pc                  ;PC = PC+1
```

```
;*****************************************************************************
;                          MACEJEC.ASM
;*****************************************************************************
;
;-----------------------------------------------------------------------------
;Este archivo contiene los macros utilizados para realizar la simulacion de
;cualquier instruccion del DS5000T.
;-----------------------------------------------------------------------------
;
DIR_INDEX           MACRO     REG,DIR
                    LOCAL     ERROR,FIN
;-----------------------------------------------------------------------------
;Este macro determina si un registro indice (@R0 o @R1) esta apuntando a
;alguno de los primeros 128 bytes de la RAM interna del DS5000T.
;Parametros:  REG = Registro indice SI o DI
;             DIR = Primer byte del grupo de instrucciones con
;                   direccionamiento indexado que se esta procesando
;Salida:      REG = direccion a la que apunta R0 o R1
;               CY  = 0 si es posible simular la instruccion
;                     1 en otro caso
;-----------------------------------------------------------------------------
          lea       reg,r0              ;REG apunta a R0
          xor       ah,ah
          sub       al,dir              ;determinar si el reg. indice es R0 o R1
          call      sel_bank            ;chequear banco de registros activo
          mov       al,[reg+bx]         ;determinar a donde apunta @R0 o @R1
          cmp       al,80h              ;es una direccion valida?
          jae       error               ;no, error
          add       reg,ax              ;si, REG apunta a esa direccion
          clc                           ;CY = 0
          jmp       fin
error:    stc                           ;CY = 1
fin:
          ENDM

DIR_REG   MACRO     REG,DIR
;-----------------------------------------------------------------------------
;Este macro hace que [REG+BX] apunte a uno de los 32 registros del DS5000T.
;Entrada:     AL  = Primer byte de la instruccion
;Parametros:  REG = Registro indice SI o DI
;               DIR = Primer byte del grupo de instrucciones que se esta
;                     procesando.
;-----------------------------------------------------------------------------
          lea       reg,r0              ;REG apunta a R0
          xor       ah,ah
          sub       al,dir              ;determinar que reg. interviene en la instr.
          call      sel_bank            ;chequear banco de registros activo

          ENDM

CHECK_DIR1          MACRO
          LOCAL     SIGA,S_1,S_2,S_3,S_4,S_5,S_6,S_7,S_8,FIN,FIN1
;-----------------------------------------------------------------------------
;Este macro chequea si en una instruccion con direccionamiento directo
;interviene alguno de los registros especiales del DS5000T.
;Entrada:           AL = direccion del registro
;Salida:            DL = mascara
;                   DL = 0 si no es un registro especial
```

```
;--------------------
        mov     direct,al           ;preservar AL
        cmp     al,0a8h             ;IE?
        jne     siga                ;no
        mov     flag_reti,1         ;si, no se debe simular atencion a interrup.
        jmp     s_6
siga:   cmp     al,0b8h             ;IP?
        jne     s_1                 ;no
        mov     flag_reti,1         ;si, no se debe simular atencion a interrup.
        jmp     s_6
s_1:    cmp     al,99h              ;SBUF?
        jne     s_8                 ;no
        preserv_dir 98h,scon        ;si, preservar SCON para ejecucion reversa
        or      scon,2              ;setear bandera de transmision
        jmp     s_6
s_8:    cmp     al,0c6h             ;MCON?
        jne     s_2                 ;no
        mov     flag_mcon,1         ;si
        mov     dh,mcon             ;preservar MCON
        test    mcon,2              ;PAA = 1?
        jnz     s_3                 ;si
        call    check_ta            ;no, esta habilitado el Timed Access?
        jc      s_4                 ;no
        mov     dl,0f9h             ;si, mascara para MCON (PAA=0, hab.TA)
fin1:   and     dh,dl               ;indicar cuales bits de MCON pueden modif.
        jmp     fin
s_3:    call    check_ta            ;esta habilitado el Timed Access?
        jc      s_5                 ;no
        mov     dl,09h              ;si, mascara para el MCON (PAA=1, hab.TA)
        jmp     fin1
s_4:    mov     dl,0fbh             ;mascara para MCON (PAA=0, deshab.TA)
        jmp     fin1
s_5:    mov     dl,0bh              ;mascara para MCON (PAA=1, deshab.TA)
        jmp     fin1
s_2:    cmp     al,87h              ;PCON?
        jne     s_6                 ;no
        mov     flag_mcon,0         ;si
        mov     dh,pcon             ;preservar PCON
        call    check_ta            ;esta habilitado el Timed Access?
        jc      s_7                 ;no
        mov     dl,30h              ;setear mascara para el PCON (hab.TA)
        jmp     fin1
s_7:    mov     dl,46h              ;setear mascara para el PCON (deshab.TA)
        jmp     fin1
s_6:    xor     dl,dl               ;no interviene un registro especial
fin:    xor     ah,ah
        ENDM

CHECK_DIR2      MACRO
        LOCAL   FIN,ES_PCON
;-------------------------------------------------------------------------
;Este macro permite completar el procesamiento de instrucciones con
;direccionamiento directo en las que intervienen registros especiales del
;DS5000T.
;Entrada:       DL = mascara
;Salida:        MCON o PCON modificados
```

```
;-------------------------------------------------------------------------
          cmp       dl,0                ;fue una instr. con algun reg. especial?
          je        fin                 ;no
          cmp       flag_mcon,0         ;si, fue con el registro MCON?
          je        es_pcon             ;no
          not       dl                  ;modificar unicamente los bits a los que se
          and       mcon,dl             ;tiene acceso en el MCON
          or        mcon,dh
          jmp       fin
es_pcon:
          not       dl                  ;modificar bits a los que se tiene acceso
          and       pcon,dl             ;en el PCON
          or        pcon,dh
fin:
          ENDM

CHECK_PORT          MACRO
          LOCAL     SIGA,S_1,S_2,FIN
;-------------------------------------------------------------------------
;Este macro chequea si en una instruccion con direccionamiento directo
;intervino alguno de los puertos.
;Entrada:           AL = direccion de un byte directamente direccionable
;Salida:            Si es un puerto, se modifican los pines del puerto
;-------------------------------------------------------------------------
          cmp       direct,80h          ;puerto 0?
          jne       siga                ;no
          mov       al,p0               ;si, escribir latch de P0
          mov       p0_pin,al           ;en pines de P0
          jmp       fin
siga:     cmp       direct,90h          ;puerto 1?
          jne       s_1                 ;no
          mov       al,p1               ;si, escribir latch de P1
          mov       p1_pin,al           ;en pines de P1
          jmp       fin
s_1:      cmp       direct,0a0h         ;puerto 2?
          jne       s_2                 ;no
          mov       al,p2               ;si, escribir latch de P2
          mov       p2_pin,al           ;en pines de P2
          jmp       fin
s_2:      cmp       direct,0b0h         ;puerto 3?
          jne       fin                 ;no
          mov       al,p3               ;si, escribir latch de P3
          mov       p3_pin,al           ;en pines de P3
fin:
          ENDM

DIR_PINES           MACRO
          LOCAL     PORT?,SIGA,S_1,S_2,FIN
;-------------------------------------------------------------------------
;Este macro sirve en ciertas instrucciones en que se deben leer los pines
;y no el latch de un puerto; ademas, si en la instruccion interviene el
;registro SBUF, se direcciona al registro de entrada (SBUF IN).
;Entrada:           AL = direccion de un byte directamente direccionable
;Salida:            DS:SI apunta a los pines de un puerto (si es puerto)
;                   Si AL = [SBUF], SI apunta a SBUF IN.
;-------------------------------------------------------------------------
```

```
          cmp       al,099h             ;SBUF?
          jne       port?               ;no, es un puerto?
          lea       si,sbuf_in          ;si, SI apunta a SBUF IN
          preserv_dir 98h,scon          ;preservar SCON para ejecucion reversa
          jmp       fin
port?:    cmp       al,80h              ;puerto 0?
          jne       siga                ;no
          lea       si,p0_pin           ;si, direccionar los pines de P0
          jmp       fin
siga:     cmp       al,90h              ;puerto 1?
          jne       s_1                 ;no
          lea       si,p1_pin           ;si, direccionar los pines de P1
          jmp       fin
s_1:      cmp       al,0a0h             ;puerto 2?
          jne       s_2                 ;no
          lea       si,p2_pin           ;si, direccionar los pines de P2
          jmp       fin
s_2:      cmp       al,0b0h             ;puerto 3?
          jne       fin                 ;no
          lea       si,p3_pin           ;si, direccionar los pines de P3
fin:
          ENDM

ES_IE_IP?           MACRO
          LOCAL     IP?,FIN
;-------------------------------------------------------------------------
;Este macro chequea si en una instruccion con direccionamiento directo
;interviene el registro IE o IP en cuyo caso, si existe una interrupcion,
;no se la debe simular sino hasta despues de la siguiente instruccion.
;Entrada:            AL        = direccion de un byte directamente direccionable
;Salida:             FLAG_RETI = 1 si en la instruccion interviene IE o IP
;-------------------------------------------------------------------------
          cmp       al,0a8h             ;IE?
          jne       ip?                 ;no
          mov       flag_reti,1         ;si, no se debe simular interrupciones
          jmp       fin
ip?:      cmp       al,0b8h             ;IP?
          jne       fin                 ;no
          mov       flag_reti,1         ;si, no se debe simular interrupciones
fin:
          ENDM

CHECK_PIN1          MACRO
          LOCAL     PORT0,PORT1,PORT2,SIGA,S_1,S_2,IP?,SET_FLAG,FIN
;-------------------------------------------------------------------------
;Este macro chequea si en una instruccion interviene un pin de algun puerto
;Entrada:            AL = direccion de un bit directamente direccionable
;Salida:             Si es un pin, se escribe el pin en el latch
;-------------------------------------------------------------------------
          mov       ah,al
          and       ah,0c8h
          cmp       ah,80h              ;es un pin de un puerto?
          je        siga                ;si
          jmp       fin                 ;no
siga:     cmp       al,80h              ;puede ser un pin de P0?
          jae       s_1                 ;si
```

```
        jmp     fin                 ;no
s_1:    cmp     al,88h              ;es un pin de P0?
        jb      port0               ;si
        cmp     al,90h              ;puede ser un pin de P1?
        jae     s_2                 ;si
        jmp     fin                 ;no
s_2:    cmp     al,98h              ;es un pin de P1?
        jb      port1               ;si
        cmp     al,0a0h             ;puede ser un pin de P2?
        jb      fin                 ;no
        cmp     al,0a8h             ;si, es un pin de P2?
        jb      port2               ;si
        cmp     al,0b0h             ;puede ser un pin de P3?
        jb      set_flag            ;no, es un bit de IE?
        cmp     al,0b8h             ;si, es un pin de P3?
        jae     ip?                 ;no
        mov     flag_port3,1        ;si, puerto 3
        mov     ah,p3               ;preservar latch de P3
        mov     direct,ah
        mov     ah,p3_pin           ;escribir los pines de P3
        mov     p3,ah               ;en latch de P3
        jmp     fin
ip?:    cmp     al,0c0h             ;es un bit de IP?
        jb      set_flag            ;si
        jmp     fin                 ;no
port0:  mov     flag_port0,1        ;puerto 0
        mov     ah,p0               ;preservar latch de P0
        mov     direct,ah
        mov     ah,p0_pin           ;escribir los pines de P0
        mov     p0,ah               ;en latch de P0
        jmp     fin
port1:  mov     flag_port1,1        ;puerto 1
        mov     ah,p1               ;preservar latch de P1
        mov     direct,ah
        mov     ah,p1_pin           ;escribir los pines de P1
        mov     p1,ah               ;en latch de P1
        jmp     fin
port2:  mov     flag_port2,1        ;puerto 2
        mov     ah,p2               ;preservar latch de P2
        mov     direct,ah
        mov     ah,p2_pin           ;escribir los pines de P2
        mov     p2,ah               ;en latch de P2
        jmp     fin
set_flag:
        mov     flag_reti,1         ;si es IE o IP no se debe
fin:                                ;simular interrupciones
        ENDM

CHECK_FIN2      MACRO
        LOCAL   SIGA,S_1,S_2,FIN
;-------------------------------------------------------------------------
;Este macro permite completar el procesamiento de instrucciones con bits
;directamente direccionbles.
;-------------------------------------------------------------------------
        cmp     flag_port0,1        ;fue una instr. con un pin de P0?
        jne     siga                ;no
```

```
          mov       ah,direct           ;si, restaurar latch de P0
          mov       p0,ah
          jmp       fin
siga:     cmp       flag_port1,1        ;fue una instr. con un pin de P1?
          jne       s_1                 ;no
          mov       ah,direct           ;si, restaurar latch de P1
          mov       p1,ah
          jmp       fin
s_1:      cmp       flag_port2,1        ;fue una instr. con un pin de P2?
          jne       s_2                 ;no
          mov       ah,direct           ;si, restaurar latch de P2
          mov       p2,ah
          jmp       fin
s_2:      cmp       flag_port3,1        ;fue una instr. con un pin de P3?
          jne       fin                 ;no
          mov       ah,direct           ;si, restaurar latch de P3
          mov       p3,ah
fin:
          ENDM

JMP_INT   MACRO     DIR
;-------------------------------------------------------------------------
;Este macro permite simular la atencion a una interrupcion
;Parametros:  DIR = valor de PC donde se inicia la rutina de atencion a la
;                   interrupcion.
;-------------------------------------------------------------------------
          inc       byte ptr regsp      ;incrementar Stack Pointer
          lea       si,r0               ;SI apunta a R0
          mov       al,regsp
          xor       ah,ah
          add       si,ax               ;SI apunta a SP
          mov       ax,pc
          mov       [si],ax             ;guardar PC en el stack
          inc       byte ptr regsp      ;incrementar Stack Pointer
          mov       pc,dir              ;simular salto a rutina de
          jmp       finejec             ;atencion a una interrupcion

          ENDM

INT_EXT0  MACRO     PRIOR
          LOCAL     NO_TRAN0,NO_EX0
;-------------------------------------------------------------------------
;Este macro simula la atencion a la interrupcion externa 0
;-------------------------------------------------------------------------
          test      ie,1                ;esta habilitada la int. externa 0?
          jz        no_ex0              ;no
          test      tcon,2              ;hay interrupcion externa 0?
          jz        no_ex0              ;no
          test      tcon,1              ;es por transicion?
          jz        no_tran0            ;no
          and       tcon,0fdh           ;si, resetear bandera EX0
no_tran0:
          max_prior? prior              ;es una interrup. de maxima prioridad?
          jmp_int   0003h               ;simular salto a rutina de atencion
no_ex0:                                 ;a interrupcion externa 0
          ENDM
```

```
INT_TIMER0          MACRO     PRIOR
          LOCAL     NO_ET0
;-------------------------------------------------------------------------
;Este macro simula la atencion a la interrupcion del TIMER 0
;-------------------------------------------------------------------------
          test      ie,2                ;esta habilitada la int. de Timer 0?
          jz        no_et0              ;no
          test      tcon,20h            ;hay interrupcion de Timer 0?
          jz        no_et0              ;no
          and       tcon,0dfh           ;si, resetear bandera TF0
          max_prior? prior              ;es una interrup. de maxima prioridad?
          jmp_int   000bh               ;simular salto a rutina de atencion
no_et0:                                 ;a interrupcion de TIMER0
          ENDM

INT_EXT1  MACRO     PRIOR
          LOCAL     NO_TRAN1,NO_EX1
;-------------------------------------------------------------------------
;Este macro simula la atencion a la interrupcion externa 1
;-------------------------------------------------------------------------
          test      ie,4                ;esta habilitada la int. externa 1?
          jz        no_ex1              ;no
          test      tcon,8              ;hay interrupcion externa 0?
          jz        no_ex1              ;no
          test      tcon,4              ;es por transicion?
          jz        no_tran1            ;no
          and       tcon,0f7h           ;si, resetear bandera EX1
no_tran1:
          max_prior? prior              ;es una interrup. de maxima prioridad?
          jmp_int   0013h               ;simular salto a rutina de atencion
no_ex1:                                 ;a interrupcion externa 1
          ENDM

INT_TIMER1          MACRO     PRIOR
          LOCAL     NO_ET1
;-------------------------------------------------------------------------
;Este macro simula la atencion a la interrupcion del TIMER 1
;-------------------------------------------------------------------------
          test      ie,8                ;esta habilitada la int. de Timer 1?
          jz        no_et1              ;no
          test      tcon,80h            ;hay interrupcion de Timer 1?
          jz        no_et1              ;no
          and       tcon,07fh           ;si, resetear bandera TF1
          max_prior? prior              ;es una interrup. de maxima prioridad?
          jmp_int   001bh               ;simular salto a rutina de atencion
no_et1:                                 ;a interrupcion de TIMER1
          ENDM

INT_SERIAL          MACRO     PRIOR
          LOCAL     RX,NO_SER
;-------------------------------------------------------------------------
;Este macro simula la atencion a la interrupcion de comunicacion serial
;-------------------------------------------------------------------------
          test      ie,10h              ;esta habilitada la int. serial?
          jz        no_ser              ;no
```

```
          test      scon,1              ;hay interrupcion de Rx?
          jnz       Rx                  ;si
          test      scon,2              ;hay interrupcion de Tx?
          jz        no_ser              ;no
Rx:       max_prior? prior              ;es una interrup. de maxima prioridad?
          jmp_int   0023h               ;simular salto a rutina de atencion
no_ser:                                 ;a interrupcion de comunicacion serial
          ENDM

MAX_PRIOR?          MACRO     PRIOR
          LOCAL     SIGA,FIN
;-------------------------------------------------------------------------
;Este macro chequea si una interrupcion es de maxima prioridad y de ser
;asi, deshabilita la simulacion de cualquier otra interrupcion
;-------------------------------------------------------------------------
          mov       al,prior
          cmp       al,1                ;es una interrup. de max. prioridad?
          jne       siga                ;no
          mov       flag_prior,1        ;si, no se debe simular la atencion
          jmp       fin                 ;a otra interrupcion
siga:     mov       flag_int,1          ;solo se debe simular la atencion a
fin:                                    ;interrup. de maxima prioridad
          ENDM

LOAD_REG  MACRO     REG,CH?
          LOCAL     ES_UNO,SIGA,LAZO,SIGA1
;-------------------------------------------------------------------------
;Este macro es utilizado en la simulacion del ECC y permite cargar un
;registro de 8 bits del 8086 con informacion para actualizar fecha u hora
;del computador.
;Parametros:  REG = registro de 8 bits del 8086
;             CH? = 1 si REG es el registro CH.
;-------------------------------------------------------------------------
          push      cx                  ;preservar CX
          mov       cx,8                ;inicializar un contador
lazo:     cmp       byte ptr [si],33h   ;es cero el bit del ECC?
          jne       es_uno              ;no
          clc                           ;si, poner un cero en el bit
          rcr       al,1                ;correspondiente de REG
          jmp       siga
es_uno:   stc                           ;poner un uno en el bit
          rcr       al,1                ;correspondiente de REG
siga:     inc       si                  ;apuntar al siguiente bit del ECC
          loop      lazo                ;repetir 8 veces
          mov       ah,ch?
          cmp       ah,1                ;es el registro CH ?
          jne       siga1               ;no
          cmp       byte ptr [si-1],33h ;el ECC esta en formato de 24 horas
          je        siga1               ;si
          and       al,7fh              ;no, hacer cero el bit AM/PM
          cmp       byte ptr [si-3],33h ;es una hora AM?
          je        siga1               ;si
          and       al,1fh              ;no, calcular la hora equivalente
          add       al,12               ;en formato de 24 horas
siga1:
          push      cx                  ;preservar registros
```

```
          push      dx
          mov       dh,al               ;cambiar AL de BCD a hexadecimal
          and       dh,0fh
          mov       cl,4
          shr       al,cl
          mov       dl,10
          xor       ah,ah
          mul       dl
          add       al,dh
          pop       dx                  ;restaurar registros
          pop       cx
          pop       cx
          mov       reg,al              ;cargar REG con la informacion
                                        ;obtenida del ECC
          ENDM

PRESERV_PC          MACRO
;-------------------------------------------------------------------------
;Este macro preserva el valor de PC para realizar la simulacion de
;ejecucion reversa de las instrucciones
;Salida:            PC_REVERS es modificado
;-------  ---------------------------------------------------------------
          push      ax
          push      di
          lea       di,revers           ;DI apunta al buffer para ejecucion reversa
          add       di,pc_revers
          mov       ax,pc               ;preservar PC
          stosw
          add       pc_revers,2         ;actualizar valor de pc_revers
          pop       di
          pop       ax

          ENDM

PRESERV_DIR         MACRO     DIR,VALOR
;-------------------------------------------------------------------------
;Este macro preserva la direccion y el valor de un byte directamente
;direccionable para realizar la simulacion de ejecucion reversa de las
;instrucciones.
;Salida:            PC_REVERS es modificado
;-------------------------------------------------------------------------
          push      ax
          push      di
          lea       di,revers           ;DI apunta al buffer para ejecucion reversa
          add       di,pc_revers
          mov       al,1
          stosb
          mov       al,dir              ;preservar la direccion del byte
          stosb
          mov       al,valor            ;preservar el valor del byte
          stosb
          add       pc_revers,3         ;actualizar valor de pc_revers
          pop       di
          pop       ax

          ENDM
```

```
PRESERV_MEM     MACRO     DIR,VALOR
;-------------------------------------------------------------------------
;Este macro preserva la direccion y el valor de un byte de la RAM
;embebida para realizar la simulacion de ejecucion reversa de las
;instrucciones.
;Salida:         PC_REVERS es modificado
;-------------------------------------------------------------------------
        push    ax
        push    di
        push    es
        push    ds
        pop     es                  ;ES apunta a DSEG
        lea     di,revers           ;DI apunta al buffer para ejecucion reversa
        add     di,pc_revers
        mov     al,2
        stosb
        mov     ax,dir              ;preservar la direccion del byte
        stosw
        mov     al,valor            ;preservar el valor del byte
        stosb
        add     pc_revers,4         ;actualizar valor de pc_revers
        pop     es
        pop     di
        pop     ax

        ENDM

PRESERV_STACK   MACRO
;-------------------------------------------------------------------------
;Este macro preserva el Stack para realizar la simulacion de ejecucion
;reversa de las instrucciones
;Salida:         PC_REVERS es modificado
;-------------------------------------------------------------------------
        push    ax
        push    si
        push    di
        lea     di,revers           ;DI apunta al buffer para ejecucion reversa
        add     di,pc_revers
        mov     al,1                ;preservar el stack para ejecucion reversa
        stosb
        mov     al,regsp
        stosb
        lodsb
        stosb
        mov     al,1
        stosb
        mov     al,regsp
        inc     al
        stosb
        lodsb
        stosb
        add     pc_revers,6         ;actualizar valor de pc_revers
        pop     di
        pop     si
        pop     ax
```

```
          ENDM

PRESERV_ECC       MACRO
;-------------------------------------------------------------------------
;Este macro preserva los registros y banderas que son modificados
;cuando se simula el reloj en tiempo real, para realizar la simulacion
;de ejecucion reversa de las instrucciones.
;Salida:           PC_REVERS es modificado
;-------------------------------------------------------------------------
          push      ax
          push      si
          push      di
          lea       di,revers           ;DI apunta al buffer para ejecucion reversa
          add       di,pc_revers
          mov       al,6
          stosb
          mov       al,ecc_point        ;preservar banderas usadas en la
          stosb                         ;simulacion del ECC
          lea       si,ecc
          xor       ah,ah
          add       si,ax
          movsb
          mov       al,reset_ecc
          stosb
          mov       al,rec_patt
          stosb
          add       pc_revers,5         ;actualizar valor de pc_revers
          pop       di
          pop       si
          pop       ax

          ENDM

PRESERV_INTERRUP  MACRO
;-------------------------------------------------------------------------
;Este macro preserva los registros y banderas que son modificados
;cuando se simula la atencion a una interrupcion, para realizar la
;simulacion de ejecucion reversa de las instrucciones
;Salida:           PC_REVERS es modificado
;-------------------------------------------------------------------------
          push      ax
          push      di
          lea       di,revers           ;DI apunta al buffer para ejecucion reversa
          add       di,pc_revers
          mov       al,4
          stosb
          mov       al,flag_int         ;preservar banderas usadas en la
          stosb                         ;simulacion de interrupciones
          mov       al,flag_prior
          stosb
          mov       al,flag_reti
          stosb
          mov       al,tcon             ;preservar el registro TCON
          stosb
          add       pc_revers,5         ;actualizar valor de pc_revers
          pop       di
```

```
          pop       ax

          ENDM

PRESERV_TA          MACRO
;-------------------------------------------------------------------------
;Este macro preserva las banderas que son modificados por las
;instrucciones con el registro TA, para realizar la simulacion de
;ejecucion reversa de las instrucciones.
;Salida:            PC_REVERS es modificado
;-------------------------------------------------------------------------
          push      ax
          push      di
          lea       di,revers           ;DI apunta al buffer para ejecucion reversa
          add       di,pc_revers
          mov       al,3
          stosb
          mov       al,flag_ta1         ;preservar banderas usadas en la
          stosb                         ;simulacion de instrucciones con TA.
          mov       al,flag_ta2
          stosb
          add       pc_revers,3         ;actualizar valor de pc_revers
          pop       di
          pop       ax

          ENDM

PRESERV_TIMERS      MACRO
;-------------------------------------------------------------------------
;Este macro preserva los TIMERS y el registro TCON para realizar la
;simulacion de ejecucion reversa de las instrucciones
;Salida:            PC_REVERS es modificado
;-------------------------------------------------------------------------
          push      ax
          push      di
          lea       di,revers           ;DI apunta al buffer para ejecucion reversa
          add       di,pc_revers
          mov       al,5
          stosb
          mov       al,tl0              ;preservar TIMER 0 para ejecucion reversa
          stosb
          mov       al,th0
          stosb
          mov       al,tl1              ;preservar TIMER 1 para ejecucion reversa
          stosb
          mov       al,th1
          stosb
          mov       al,tcon
          stosb
          add       pc_revers,6         ;actualizar PC_REVERS
          pop       di
          pop       ax

          ENDM

PRESERV_CICLOS      MACRO
```

```
;-------------------------------------------------------------------------
;Este macro preserva el valor del numero de ciclos requeridos por una
;instruccion y actualiza los valores de PC_REVERS y COUNT_R.
;Salida:          PC_REVERS es modificado
;-------------------------------------------------------------------------
          push     ax
          push     di
          lea      di,revers           ;DI apunta al buffer para ejecucion reversa
          add      di,pc_revers
          mov      al,7                ;preservar los ciclos para ejecucion reversa
          stosb
          mov      ax,cx
          stosb
          mov      al,1
          call     actual_revers       ;actualizar PC_REVERS y COUNT_R
          pop      di
          pop      ax

          ENDM

          SUBRUT   MACRO
;-------------------------------------------------------------------------
;Este macro contiene las subrutinas utilizadas en el ejecutor de
;simulacion
;-------------------------------------------------------------------------

CHECKBIT  PROC     NEAR
;-------------------------------------------------------------------------
;Este procedimiento determina el estado (0 o 1) de alguno de los bits
;directamente direccionables.
;Entrada:         AL = bit directamente direccionable
;Salida:          CY = 1 si el estado del bit es 1
;                      0 en otro caso
;-------------------------------------------------------------------------
          push     si                  ;preservar registros
          push     cx
          push     bx
          push     ax
          xor      ah,ah
          cmp      al,7fh              ;es alguno de los primeros 128 bits?
          ja       mayor7f             ;no
          mov      bl,8                ;si, direccionarlo mediante los
          div      bl                  ;registros AL y AH
          add      al,20h
          jmp      conti
mayor7f:
          mov      ah,al               ;direccionar uno de los siguientes 128 bits
          and      al,0f8h             ;con los registros AL y AH
          sub      ah,al
conti:    mov      cl,ah               ;cargar en CL la posicion del bit en un byte
          inc      cl
          lea      si,r0               ;direccionar el byte al que pertenece el bit
          xor      ah,ah
          add      si,ax
          lodsb                        ;cargar en AL el byte
          clc
```

```
          rcr     al,cl               ;establecer en el carry el estado del bit
          pop     ax                  ;restaurar registros
          pop     bx
          pop     cx
          pop     si
          ret                         ;retornar

CHECKBIT  ENDP

SETBIT    PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento permite dar valores (0 o 1) a alguno de los bits
;directamente direccionables.
;Entrada:          AL = bit directamente direccionable
;                  DL = valor que debera tener el bit
;-------------------------------------------------------------------------
          push    si                  ;preservar registros
          push    cx
          push    bx
          push    ax
          xor     ah,ah
          cmp     al,7fh              ;es alguno de los primeros 128 bits?
          ja      may7f               ;no
          mov     bl,8                ;si, direccionarlo mediante los
          div     bl                  ;registros AL y AH
          add     al,20h
          jmp     conti1
may7f:    mov     ah,al               ;direccionar uno de los siguientes 128 bits
          and     al,0f8h             ;con los registros AL y AH
          sub     ah,al
conti1:   mov     cl,ah               ;cargar en CL la posicion del bit en un byte
          inc     cl
          lea     si,r0               ;direccionar el byte que contiene el bit
          xor     ah,ah
          add     si,ax
          mov     ah,al
          preserv_dir ah,[si]         ;preservar byte de RAM int. para ej.reversa
          cmp     dl,0                ;hay que hacer cero el bit?
          je      clr_bit             ;si
          xor     dl,dl               ;no, escribir un uno en el bit
          stc
          rcl     dl,cl
          or      [si],dl
          jmp     checkport           ;chequear si el bit es un pin de un puerto
clr_bit:
          mov     dl,0ffh             ;escribir un cero en el bit
          clc
          rcl     dl,cl
          and     [si],dl
          jmp     checkport1          ;chequear si el bit es un pin de un puerto
checkport:
          pop     ax
          mov     ah,al
          and     ah,0c8h
          cmp     ah,80h              ;es un pin de un puerto?
          jne     finset1             ;no
```

```
          cmp     al,80h              ;puede ser un pin de P0?
          jb      finset1             ;no
          cmp     al,88h              ;si, es un pin de P0?
          jb      port0               ;si
          cmp     al,90h              ;puede ser un pin de P1?
          jb      finset1             ;no
          cmp     al,98h              ;si, es un pin de P1?
          jb      port1               ;si
          cmp     al,0a0h             ;puede ser un pin de P2?
          jb      finset1             ;no
          cmp     al,0a8h             ;si, es un pin de P0?
          jb      port2               ;si
          cmp     al,0b0h             ;puede ser un pin de P3?
          jb      set_flag            ;no, es un bit de IE
          cmp     al,0b8h             ;si, es un pin de P3?
          jae     ip?                 ;no
          or      p3_pin,dl           ;si, poner un uno en un pin de P3
finset1:
          jmp     finset
ip?:      cmp     al,0c0h             ;es un bit de IP?
          jb      set_flag            ;si
          jmp     finset              ;no
port0:    or      p0_pin,dl           ;poner un uno en un pin de P0
          jmp     finset
port1:    or      p1_pin,dl           ;poner un uno en un pin de P1
          jmp     finset
port2:    or      p2_pin,dl           ;poner un uno en un pin de P2
          jmp     finset
set_flag:
          mov     flag_reti,1         ;si es IE o IP no se debe simular
          jmp     finset              ;las interrupciones
checkporti:
          pop     ax                  ;preservar AX
          mov     ah,al
          and     ah,0c8h
          cmp     ah,80h              ;es un pin de un puerto?
          jne     finset              ;no
          cmp     al,80h              ;puede ser un pin de P0?
          jb      finset              ;no
          cmp     al,88h              ;si, es un pin de P0?
          jb      port_0              ;si
          cmp     al,90h              ;puede ser un pin de P1?
          jb      finset              ;no
          cmp     al,98h              ;si, es un pin de P1?
          jb      port_1              ;si
          cmp     al,0a0h             ;puede ser un pin de P2?
          jb      finset              ;no
          cmp     al,0a8h             ;si, es un pin de P2?
          jb      port_2              ;si
          cmp     al,0b0h             ;puede ser un pin de P3?
          jb      set_flag1           ;no, es un bit de IE
          cmp     al,0b8h             ;si, es un pin de P3?
          jae     ip1?                ;no
          and     p3_pin,dl           ;si, poner un cero en un pin de P3
          jmp     finset
ip1?:     cmp     al,0c0h             ;es un bit de IP?
```

```
          jb        set_flag1           ;si
          jmp       finset              ;no
port_0:   and       p0_pin,dl           ;poner un cero en un pin de P0
          jmp       finset
port_1:   and       p1_pin,dl           ;poner un cero en un pin de P1
          jmp       finset
port_2:   and       p2_pin,dl           ;poner un cero en un pin de P2
          jmp       finset
set_flag1:
          mov       flag_reti,1         ;si es IE o IP no se debe simular
finset:                                 ;las interrupciones
          pop       bx                  ;restaurar los registros
          pop       cx
          pop       si
          ret                           ;retornar

SETBIT    ENDP

SETFLAGS  PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento chequea las banderas del 8086 y de acuerdo a ellas,
;da valor a las banderas simuladas del DS5000T.
;Salida:             PSW = banderas del uP i8086
;-------------------------------------------------------------------------
          push      ax                  ;preservar AX
          lahf                          ;cargar en AH las banderas del 8086
          jo        si_ov               ;hubo overflow?
          and       psw,0fbh            ;no, borrar la bandera de overflow (DS5000T)
          jmp       conti2
si_ov:    or        psw,4               ;si, setear bandera de overflow (DS5000T)
conti2:   test      ah,10h              ;hubo carry auxiliar?
          jz        no_ac               ;no
          or        psw,40h             ;si, setear carry auxiliar (DS5000T)
          jmp       conti3
no_ac:    and       psw,0bfh            ;borrar bandera de carry auxiliar (DS5000T)
conti3:   test      ah,1                ;hubo carry?
          jz        no_cy6              ;no
          or        psw,80h             ;si, setear bandera de carry (DS5000T)
          jmp       fin_sf
no_cy6:   and       psw,7fh             ;borrar bandera de carry (DS5000T)
fin_sf:   pop       ax                  ;restaurar AX
          ret                           ;retornar

SETFLAGS  ENDP

CHECK_CY  PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento determina el estado (0 o 1) del carry.
;Salida:           CY = 1 si el carry simulado es igual a 1
;                       0 en otro caso
;-------------------------------------------------------------------------
          test      psw,80h             ;hay carry del DS5000T?
          jz        no_carry3           ;no
          stc                           ;si, setear bandera de carry del 8086
          jmp       fincheck
no_carry3:
```

```
          clc                         ;borrar bandera de carry del 8086
fincheck:
          ret                         ;retornar

CHECK_CY  ENDP

CHECK_CBNE          PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento da los valores adecuados a las banderas que son
;alteradas por la instruccion CBNE y calcula la direccion a donde debe
;saltar, en caso que deba hacerlo
;Entrada:            DS:SI apunta al tercer byte de la instruccion CBNE
;Salida:             PSW = banderas del uP i8086
;                    PC puede ser alterado
;-------------------------------------------------------------------------
          push      ax                ;preservar AX
          jz        conti4            ;se debe simular un salto?
          pushf                       ;no, preservar banderas
          lodsb                       ;leer tercer byte de CBNE
          cbw
          add       pc,ax             ;calcular la direccion del salto
          popf                        ;restaurar banderas
          jc        set_cy1           ;se debe setear el carry del DS5000T?
conti4:   and       psw,7fh           ;no, borrar el carry del DS5000T
          jmp       fin_cbne
set_cy1:
          or        psw,80h           ;setear carry (DS5000T)
fin_cbne:
          pop       ax                ;restaurar AX
          ret                         ;retornar

CHECK_CBNE          ENDP

CHECK_TA  PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento determina si se puede ejecutar una instruccion que
;necesita ser habilitada mediante el registro TA (0c7h)
;
;Salida:             CY = 0 si esta habilitado el Timed Access
;                         1 en otro caso
;-------------------------------------------------------------------------
          cmp       flag_ta1,0        ;hubo la primera instr. para habilitar TA?
          je        deshab_ta         ;no
          cmp       flag_ta2,0        ;si, hubo la segunda instr. para habil. TA?
          je        deshab_ta         ;no
          clc                         ;si, habilitado TA
          jmp       fin_ta
deshab_ta:
          stc                         ;deshabilitado TA
fin_ta:   ret                         ;retornar

CHECK_TA  ENDP

SEL_BANK  PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento chequea cual es el banco de registros que esta
```

```
;seleccionado.
;Entrada:        AL = registro (R0 a R7) que interviene en la instruccion
;Salida:         BX = direccion del registro de acuerdo al banco activo
;-------------------------------------------------------------------------
          mov     bx,ax             ;cargar en BX un codigo del registro
          mov     al,psw            ;chequear en PSW el banco activo
          and     al,18h
          add     bx,ax             ;calcular la direccion del registro tomando
          xor     bh,bh             ;en cuenta cual es el banco de reg. activo
          ret                       ;retornar

SEL_BANK  ENDP

TIMERS    PROC    NEAR
;-------------------------------------------------------------------------
;Este procedimiento permite simular la operacion de los TIMERS
;-------------------------------------------------------------------------
          push    ax                ;preservar los registros
          push    cx

          push    cx
          test    tmod,4            ;esta en modo timer?
          jne     timer0_f          ;no
          test    tcon,10h          ;TR0 habilitado?
          je      timer0_f          ;no
          preserv_timers            ;preservar TIMERS para ejecucion reversa
          test    tmod,8            ;GATE es 1?
          je      t0_1              ;no
          test    p3_pin,4          ;INT0 es 1?
          je      timer0_f          ;no
t0_1:     mov     al,tmod
          and     al,3
t0_m0:    cmp     al,0              ;modo 0?
          jne     t0_m1             ;no
inc_t00:
          cmp     tl0,1fh           ;actualizar TH0?
          jne     t0_m01            ;no
          cmp     th0,0ffh          ;overflow?
          jne     t0_m00            ;no
          or      tcon,20h          ;setea TF0
t0_m00:   inc     th0               ;actualiza TH0
          mov     tl0,0             ;actualiza TL0
          jmp     timer0f
t0_m01:   inc     tl0               ;actualiza TL0
          loop    inc_t00
timer0_f:
          jmp     timer0f
t0_m1:    cmp     al,1              ;modo 1?
          jne     t0_m2             ;no
inc_t01:
          cmp     tl0,0ffh          ;actualizar TH0?
          jne     t0_m11            ;no
          cmp     th0,0ffh          ;overflow?
          jne     t0_m10            ;no
          or      tcon,20h          ;setea TF0
t0_m10:   inc     th0               ;actualiza TH0
```

```
t0_m11:   inc     tl0             ;actualiza TL0
          loop    inc_t01
          jmp     timer0f
t0_m2:    cmp     al,2            ;modo 2?
          jne     t0_m3           ;no
inc_t02:
          cmp     tl0,0ffh        ;overflow
          jne     t0_m20          ;no
          or      tcon,20h        ;setea TF0
          mov     al,th0
          mov     tl0,al          ;carga TL0 con TH0
          jmp     timer0f
t0_m20:   inc     tl0             ;incrementa TL0
          loop    inc_t02
          jmp     timer0f
t0_m3:    cmp     tl0,0ffh        ;modo 3
          jne     t0_m30          ;salta si no hay overflow
          or      tcon,20h        ;setea TF0
t0_m30:   inc     tl0             ;actualiza TL0
          cmp     th0,0ffh        ;overflow
          jne     t0_m31          ;no
          or      tcon,80h        ;setea TF1
t0_m31:   inc     th0             ;actualiza TH0
          loop    t0_m3
timer0f:
          pop     cx
t1:       test    tmod,40h        ;esta en modo timer?
          jne     time_f          ;no
          test    tcon,40h        ;TR1 habilitado?
          je      time_f          ;no
          test    tcon,10h        ;TR0 habilitado?
          jne     t1_0            ;si, no preservar TIMERS
          preserv_timers          ;preservar TIMERS para ejecucion reversa
t1_0:     test    tmod,80h        ;GATE es 1?
          je      t1_1            ;no
          test    p3_pin,8        ;INT1 es 1?
          je      time_f          ;no
t1_1:     mov     al,tmod
          and     al,30h
t1_m0:    cmp     al,0            ;modo 0?
          jne     t1_m1           ;no
inc_t10:
          cmp     tl1,1fh         ;actualizar TH1?
          jne     t1_m01          ;no
          cmp     th1,0ffh        ;overflow?
          jne     t1_m00          ;no
          mov     al,tmod
          and     al,3            ;Timer 0
          cmp     al,3            ;en modo 3?
          je      t1_m00          ;si
          or      tcon,80h        ;setea TF1
t1_m00:   inc     th1             ;actualiza TH1
          mov     tl1,0           ;actualiza TL1
          jmp     timerf
t1_m01:   inc     tl1             ;actualiza TL1
          loop    inc_t10
```

```
time_f:   jmp     timerf
t1_m1:    cmp     al,10h              ;modo 1?
          jne     t1_m2               ;no
inc_t11:
          cmp     tl1,0ffh            ;actualizar TH1?
          jne     t1_m11              ;no
          cmp     th1,0ffh            ;overflow?
          jne     t1_m10              ;no
          mov     al,tmod
          and     al,3                ;Timer 0
          cmp     al,3                ;en modo 3?
          je      t1_m10              ;si
          or      tcon,80h            ;setea TF1
t1_m10:   inc     th1                 ;actualiza TH1
t1_m11:   inc     tl1                 ;actualiza TL1
          loop    inc_t11
          jmp     timerf
t1_m2:    cmp     al,20h              ;modo 2?
          jne     t1_m3               ;no
inc_t12:
          cmp     tl1,0ffh            ;overflow
          jne     t1_m20              ;no
          mov     al,tmod
          and     al,3                ;Timer 0
          cmp     al,3                ;en modo 3?
          je      t1_m200             ;si
          or      tcon,80h            ;setea TF1
t1_m200:
          mov     al,th1
          mov     tl1,al              ;carga TL1 con TH1
          jmp     timerf
t1_m20:   inc     tl1                 ;incrementa TL1
          loop    inc_t12
          jmp     timerf
t1_m3:
timerf:   pop     cx                  ;restaurar los registros
          pop     ax
          ret                         ;retornar

TIMERS ENDP

CHECK_ECC           PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento permite controlar que no se escriba ningun valor en
;ciertos bits del ECC que no pueden ser modificados.
;Entrada:           ECC_POINT apunta a algun bit del ECC
;Salida:            CY = 1 si no se puede alterar el bit apuntado por ecc_point
;                        0 en otro caso
;-------------------------------------------------------------------------
          cmp     ecc_point,15        ;es el bit 15 del ECC?
          je      no_write            ;si, no se puede alterar ese bit
          cmp     ecc_point,23        ;es el bit 23 del ECC?
          je      no_write            ;si, no se puede alterar ese bit
          cmp     ecc_point,30        ;es el bit 30 del ECC?
          je      no_write            ;si, no se puede alterar ese bit
          cmp     ecc_point,35        ;es el bit 35 del ECC?
```

```
          je        no_write            ;si, no se puede alterar ese bit
          cmp       ecc_point,36        ;es el bit 36 del ECC?
          je        no_write            ;si, no se puede alterar ese bit
          cmp       ecc_point,38        ;es el bit 38 del ECC?
          je        no_write            ;si, no se puede alterar ese bit
          cmp       ecc_point,39        ;es el bit 39 del ECC?
          je        no_write            ;si, no se puede alterar ese bit
          cmp       ecc_point,46        ;es el bit 46 del ECC?
          je        no_write            ;si, no se puede alterar ese bit
          cmp       ecc_point,47        ;es el bit 47 del ECC?
          je        no_write            ;si, no se puede alterar ese bit
          cmp       ecc_point,53        ;es el bit 53 del ECC?
          je        no_write            ;si, no se puede alterar ese bit
          cmp       ecc_point,54        ;es el bit 54 del ECC?
          je        no_write            ;si, no se puede alterar ese bit
          cmp       ecc_point,55        ;es el bit 55 del ECC?
          jne       fin_check
no_write:
          stc                           ;no modificar bit apuntado por ecc_point
          ret                           ;retornar
fin_check:
          clc                           ;se modificar bit apuntado por ecc_point
          ret                           ;retornar

CHECK_ECC         ENDP

WRITE_ECC         PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento permite escribir byte a byte en el ECC
;Entrada:          AL = numero hexadecimal que se quiere escribir en el ECC
;Salida:           ECC modificado
;                  DS:SI apunta al ultimo bit modificado del ECC + 1
;-------------------------------------------------------------------------
          push      ax                  ;preservar los registros
          push      bx
          push      cx
          push      dx

          xor       ah,ah               ;transformar el numero en AL a formato
          mov       dh,10               ;BCD para escribirlo en el ECC
          div       dh
          mov       bl,ah
          xor       ah,ah
          div       dh
          mov       cl,4
          sal       ah,cl
          or        bl,ah               ;BL contiene el numero en formato BCD
          mov       cx,8                ;inicializar un contador

lazo:     clc
          rcr       bl,1                ;hay que escribir un cero en el ECC?
          jc        uno                 ;no
          mov       byte ptr [si],33h   ;si, escribirlo
          jmp       cont
uno:      mov       byte ptr [si],0b3h ;escribir un uno en el ECC
cont:     inc       si                  ;apuntar al siguiente bit del ECC
```

```
          loop      lazo                ;repetir 8 veces

          pop       dx                  ;restaurar registros
          pop       cx
          pop       bx
          pop       ax
          ret                           ;retornar

WRITE_ECC           ENDP

LEE_HORA  PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento lee la hora del computador y coloca ese valor en el ECC
;Salida:             ECC es modificado
;-------------------------------------------------------------------------
          mov       ah,2ch              ;leer la hora del computador
          int       21h
          lea       si,ecc              ;SI apunta al reloj del ECC
          mov       al,dl
          call      write_ecc           ;escribir centesimas de seg. en el ECC
          mov       al,dh
          call      write_ecc           ;escribir segundos en el ECC
          mov       al,cl
          call      write_ecc           ;escribir minutos en el ECC
          mov       al,ch
          cmp       byte ptr [si+7],33h ;ECC en formato de 24 horas?
          je        format_24           ;si
          cmp       al,12               ;no, es una hora PM?
          jae       pm                  ;si
          call      write_ecc           ;no, escribir la hora en el ECC
          mov       byte ptr [si-3],33h ;es una hora AM
          mov       byte ptr [si-1],0b3h ;setear bit de formato AM/PM
          jmp       fin_leeh
pm:       sub       al,12               ;obtener el valor de la hora PM
          call      write_ecc           ;escribir la hora en el ECC
          mov       byte ptr [si-3],0b3h ;es una hora PM
          mov       byte ptr [si-1],0b3h ;setear bit de formato AM/PM
          jmp       fin_leeh
format_24:
          call      write_ecc           ;escribir la hora en el ECC
fin_leeh:           ret                 ;retornar

LEE_HORA  ENDP

LEE_FECHA           PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento lee la fecha del computador y coloca ese valor en el ECC
;Salida:             ECC es modificado
;-------------------------------------------------------------------------
          mov       ah,2ah              ;leer la fecha del computador
          int       21h
          lea       si,ecc              ;SI apunta al calendario del ECC
          add       si,32
          inc       al                  ;obtener dia de la semana
          call      write_ecc           ;escribir dia de la semana en el ECC
          mov       al,dl
```

```
          call      write_ecc           ;escribir el dia del mes en el ECC
          mov       al,dh
          call      write_ecc           ;escribir el mes en el ECC
          mov       ax,cx               ;calcular decenas y unidades de anios
          cmp       ax,2000             ;para colocar ese valor en el ECC
          jae       may_2000
          sub       ax,1900
          jmp       siga_leef
may_2000:
          sub       ax,2000
siga_leef:
          call      write_ecc           ;escribir el anio en el ECC
          ret                           ;retornar

LEE_FECHA         ENDP

IGUAL_HORA        PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento setea la hora del computador de acuerdo al contenido
;del ECC.
;-------------------------------------------------------------------------
          lea       si,ecc              ;SI apunta al reloj del ECC
          load_reg  dl,0                ;cargar en DL las centesimas de segundo
          load_reg  dh,0                ;cargar en DH los segundos
          load_reg  cl,0                ;cargar en CL los minutos
          load_reg  ch,1                ;cargar en CH las horas
          mov       ah,2dh              ;setear la hora del computador
          int       21h
          ret                           ;retornar

IGUAL_HORA        ENDP

IGUAL_FECHA       PROC      NEAR
;-------------------------------------------------------------------------
;Este procedimiento setea la fecha del computador de acuerdo al contenido
;del ECC.
;-------------------------------------------------------------------------
          lea       si,ecc              ;SI apunta al calendario del ECC
          add       si,40               ;cargar en DL el dia del mes
          load_reg  dl,0                ;cargar en DH el mes
          load_reg  dh,0                ;cargar en CL decenas y unidades de anios
          load_reg  cl,0                ;calcular en CX el dato completo del anio
          xor       ch,ch
          cmp       cl,80
          jb        es_2000
          add       cx,1900
          jmp       siga3
es_2000:
          add       cx,2000
siga3:    mov       ah,2bh              ;setear la fecha del computador
          int       21h
          ret                           ;retornar

IGUAL_FECHA       ENDP

          PUBLIC    REVERSA
```

```
          mov       cx,ax
          jmp       bien_rev
error_rev:
          stc                           ;indicar error
          jmp       fin_rev
bien_rev:
          clc                           ;no hubo error
fin_rev:
          mov       flag_tcon,0         ;resetear bandera de recuperacion de TCON

          pop       es                  ;restaurar registros
          pop       di
          pop       si
          pop       dx
          pop       ax
          ret                           ;retornar

REVERSA   ENDP

ACTUAL_REVERS       PROC      NEAR
;-----------------------------------------------------------------------
;Este procedimiento permite actualizar el valor de PC_REVERS y
;de tal manera que apunte al sitio del buffer de ejecucion
;reversa donde se debera guardar o recuperar los parametros afectados
;por la siguiente instruccion que se simule. Ademas se actualiza
;COUNT_R para saber hasta cuantas instrucciones se pueden simular en
;ejecucion reversa
;Entrada:          AL = 1 si se guardaron parametros
;                       0 si se van a recuperar parametros
;Salida:           PC_REVERS es alterado
;                  COUNT_R   es alterado
;-----------------------------------------------------------------------
          and       pc_revers,0ffe0h
          cmp       al,1                ;se guardaron parametros?
          jne       recup               ;no, se recuperaron
          cmp       pc_revers,7e0h      ;pc_revers tiene su valor max.?
          jne       no_max              ;no
          mov       pc_revers,0         ;volver al inicio del buffer de ej.reversa
          jmp       actual1
no_max:   add       pc_revers,20h
actual1:
          cmp       count_r,64          ;count_r tiene su valor maximo?
          je        fin_actual          ;no
          inc       count_r
          jmp       fin_actual
recup:    cmp       pc_revers,0         ;pc_revers tiene su valor min.?
          jne       no_min              ;no
          mov       pc_revers,7e0h      ;volver al final del buffer de ej.reversa
          jmp       actual2
no_min:   sub       pc_revers,20h
actual2:
          cmp       count_r,0           ;count_r tiene su valor minimo?
          je        fin_actual          ;no
          dec       count_r
fin_actual:
          ret                           ;retornar
```

```
ACTUAL_REVERS       ENDP

            PUBLIC   LONGITUD
LONGITUD    PROC     FAR
;-------------------------------------------------------------------
;Este procedimiento obtiene la longitud, en bytes, de una instruccion
;del uC DS5000T.
;
;Entrada:             PC apunta al primer byte de la instruccion
;                     AX = segmento de datos donde esta la instruccion
;Salida:              CX = numero de bytes de la instruccion
;-------------------------------------------------------------------
            push     bx                 ;preservar los registros
            push     si
            push     di
            push     es

            push     ds
            pop      es                 ;ES apunta a DSEG
            cld
            push     ds
            mov      ds,ax              ;DS = segmento de datos indicado por AX
            lodsb                       ;obtener primer byte de codigo dela instr.
            pop      ds                 ;DS apunta a DSEG
            lea      di,casos_long      ;SI apunta a tabla de instr. ordenadas
            xor      bx,bx              ;segun su longitud
busco:      scasb                       ;se encontro la instruccion en la tabla?
            jbe      encont             ;si
            inc      bx                 ;no, pasar al sig. elemento de la tabla
            jmp      busco              ;continuar la busqueda
encont:     lea      si,long            ;si se encontro la instruccion, obtener
            mov      cl,[si+bx]         ;su longitud y colocarla en CX
            xor      ch,ch

            pop      es                 ;restaurar registros
            pop      di
            pop      si
            pop      bx
            ret                         ;retornar

LONGITUD    ENDP

            PUBLIC   CICLOS
CICLOS      PROC     FAR
;-------------------------------------------------------------------
;Este procedimiento obtiene los ciclos que demora la ejecucion de
;una instruccion del uC DS5000T.
;
;Entrada:             PC apunta al primer byte de la instruccion
;
;Salida:              CX = numero de ciclos requeridos por la instruccion
;-------------------------------------------------------------------
            push     bx                 ;preservar registros
            push     si
            push     di
```

```
          push    es

          push    ds
          pop     es                ;ES apunta a DSEG
          cld
          push    ds
          mov     ds,ax             ;DS = segmento de datos indicado por AX
          lodsb                     ;obtener primer byte de codigo dela instr.
          pop     ds                ;DS apunta a DSEG
          lea     di,casos_time     ;SI apunta a tabla de instr. ordenadas
          xor     bx,bx             ;segun los ciclos requeridos para su ejec.
busco_t:
          scasb                     ;se encontro la instruccion en la tabla?
          jbe     encont_t  ;si
          inc     bx                ;no, pasar al sig. elemento de la tabla
          jmp     busco_t           ;continuar la busqueda
encont_t:
          lea     si,time           ;si se encontro la instruccion, obtener
          mov     cl,[si+bx]        ;el numero de ciclos requeridos para su
          xor     ch,ch             ;ejecucion y colocarlo en CX

          pop     es                ;restaurar registros
          pop     di
          pop     si
          pop     bx
          ret                       ;retornar

CICLOS    ENDP

          ENDM
```

```
;*************************************
;*********     COMSER.ASM     **********
;*************************************


          PUBLIC BUFF_RX
          EXTRN  SEGDAT1:WORD

DSEG SEGMENT PUBLIC

uart_addr          dw 0                ;direccion del UART
line_status        dw 0                ;direccion de la linea de estatus
buff_rx    db 4000 dup (0)             ;buffer para recepcion

DSEG ENDS

CSEG SEGMENT
ASSUME CS:CSEG,DS:DSEG,ES:NOTHING

          PUBLIC COMSER,COMUNICA,COM_FILE,TX_COMAND
          PUBLIC TX_LOAD,VEL4800,VEL9600

COMSER PROC FAR
;-------------------------------------------------------------------------------
;Determina la direccion del UART y la direccion de la linea de status
;Salida:        uart_addr = direccion del UART
;               line_status = direccion de la linea de estatus
;-------------------------------------------------------------------------------
          push     es
          mov      ax,40h
          mov      es,ax
          mov      ax,word ptr es:[0]
          mov      uart_addr,ax                ;direccion del UART
          add      ax,5
          mov      line_status,ax              ;direccion de la linea de status
          pop      es
          ret
COMSER ENDP

VEL4800 PROC FAR
;-------------------------------------------------------------------------------
;Setea la comunicacion serial a:
;         8 bits de datos
;         1 bit de parada
;         ninguna paridad
;         4800 bps
;Modo de carga serial
;-------------------------------------------------------------------------------
          push     dx
          mov      al,11000011b
          mov      ah,0
          mov      dx,0
          int      14h
          pop      dx
          ret
VEL4800 ENDP

VEL9600 PROC FAR
```

```
;-------------------------------------------------------------------------
;Setea la comunicacion serial a:
;         8 bits de datos
;         1 bit de parada
;         ninguna paridad
;         9600 bps
;Modo de ejecucion controlada
;-------------------------------------------------------------------------
          push      dx
          mov       al,11100011b
          mov       ah,0
          mov       dx,0
          int       14h
          pop       dx
          ret
VEL9600 ENDP

COMUNICA PROC FAR
;-------------------------------------------------------------------------
;Un string con un comando es transmitido, cx = # bytes del string
;se espera recibir un caracter ">" para salir (Bien la transmision)
;si se supera el tiempo de espera tambien sale (Error en la transmision).
;Entrada:
;         DS:SI = Apunta al string a transmitirse
;            CX = Numero de bytes del string
;Salida:
;         CY = 0 Bien la transmision
;         CY = 1 Error en la transmision
;-------------------------------------------------------------------------
          push      bx
          push      dx
          push      es
          push      ds
          pop       es                          ;es = segmento de datos
          mov       di,offset buff_rx           ;buffer de datos recibidos
          mov       bx,0
comu1:    cmp       cx,0                        ;fin de string?
          je        comu2                       ;si
          mov       dx,line_status              ;no
          in        al,dx
          test      al,20h                      ;buffer de transmision vacio?
          jz        comu2                       ;no
          mov       dx,uart_addr                ;si
          lodsb                                 ;carga caracter
          out       dx,al                       ;transmite caracter
          dec       cx                          ;actualiza # de bytes de tx
          mov       bx,0
comu2:    mov       dx,line_status
          in        al,dx
          test      al,1                        ;hay dato recibido?
          jz        comu3                       ;no
          mov       dx,uart_addr                ;si
          in        al,dx                       ;lee dato
          cmp       al,">"                      ;es el comando promp?
          je        comu4                       ;si, caracter de salida
          stosb                                 ;almacena dato
```

```
          mov     bx,0                    ;resetea tiempo de espera
          jmp     comu1
comu3:    inc     bx                      ;actualiza tiempo de espera
          cmp     bx,0                    ;supero tiempo de espera ?
          jne     comu1                   ;no
          stc                             ;si, setea carry
          jmp     comuf
comu4:    clc                             ;limpia el carry
comuf:    pop     es
          pop     dx
          pop     bx
          ret
COMUNICA ENDP

COM_FILE PROC FAR
;-------------------------------------------------------------------------
;Descarga serialmente un archivo desde el computador personal al uC
;El uC procesa este archivo, si el archivo o la transmision no es correcta
;envia un mensaje al computador.
;         Entrada: CX = Numero de bytes a transmitir
;                       SI = Inicio del archivo
;         Salida:       CY = 0 Bien la transmision
;                       CY = 1 Error en la transmision
;-------------------------------------------------------------------------
          mov     di,offset buff_rx
          push    es
          push    ds
          pop     es                      ;es = segmento de datos
          mov     bx,0                    ;tiempo de espara
file1:    cmp     cx,0                    ;fin del archivo?
          je      file2                   ;si
          mov     dx,line_status          ;no
          in      al,dx
          test    al,20h                  ;buffer de transmision vacio ?
          jz      file2                   ;no
          mov     dx,uart_addr            ;si, prepara dato para transmision
          push    ds
          mov     ds,segdat1
          lodsb
          pop     ds
          out     dx,al                   ;transmite dato
          dec     cx
          mov     bx,0
file2:    mov     dx,line_status
          in      al,dx
          test    al,1                    ;hay dato recibido?
          jz      file3                   ;no
          mov     dx,uart_addr            ;si
          in      al,dx                   ;lee dato y lo guarda
          stosb
          mov     bx,0
          jmp     file1
file3:    inc     bx                      ;actualiza tiempo de espera
          cmp     bx,0                    ;supero tiempo de espera ?
          jne     file1                   ;no
          mov     si,offset buff_rx       ;si
```

```
          cmp       si,di                         ;se recibio datos ?
          je        file4                         ;no
          stc                                     ;si, error
          jmp       file5
file4:    clc
file5:    pop       es
          ret
COM_FILE ENDP

TX_COMAND PROC FAR
;-------------------------------------------------------------------------------
;Un string con un comando terminado en 0 es transmitido, se espera recibir
;un caracter ">" para salir (Bien la transmision), si se supera el tiempo
;de espera tambien sale (Error en la transmision).
;Entrada:
;         DS:SI = Apunta al string a transmitirse
;Salida:
;         CY = 0 Bien la transmision
;         CY = 1 Error en la transmision
;-------------------------------------------------------------------------------
          push      dx
          push      cx
          push      bx
          push      es
          push      ds
          pop       es                            ;es = segmento de datos
          mov       di,offset buff_rx
          mov       cl,0                          ;bandera de transmision
tx1:      cmp       cl,0                          ;hay datos para transmitir
          jne       tx3                           ;no
          mov       dx,line_status                ;si
          in        al,dx
          test      al,20h                        ;buffer de tx vacio?
          jz        tx3                           ;no
          lodsb                                   ;carga byte
          cmp       al,0                          ;fin del comando?
          jne       tx2                           ;no
          mov       cl,1                          ;cambia bandera de transmision
          jmp       tx3
tx2:      mov       dx,uart_addr
          out       dx,al                         ;transmite caracter
          mov       bx,0
tx3:      mov       dx,line_status
          in        al,dx
          test      al,1                          ;hay dato recibido?
          jz        tx4                           ;no
          mov       dx,uart_addr                  ;si
          in        al,dx                         ;lee el caracter recibido
          cmp       al,">"                        ;es el comando promp?
          je        tx5                           ;si, salida
          stosb                                   ;guardar byte
          mov       bx,0                          ;resetea tiempo de espera
          jmp       tx1
tx4:      inc       bx                            ;actualiza tiempo de espera
          cmp       bx,0                          ;supero tiempo de espera?
          jne       tx1                           ;no
```

```
          stc                                   ;setea carry
          jmp       txf
tx5:      clc                                   ;limpia el carry
txf:      pop       es
          pop       bx
          pop       cx
          pop       dx
          ret
TX_COMAND ENDP

TX_LOAD PROC FAR
;-------------------------------------------------------------------------
;Un string con un comando terminado en 0 es transmitido, este string es cargado
;en el micro y procesado por el software interno
;Entrada:
;         DS:SI = Apunta al string a transmitirse
;Salida:
;         CY = 0    Bien la transmision
;         CY = 1    AX = 0 No hay respuesta a la transmision
;                   AX = 1 Mensaje de error recibido
;-------------------------------------------------------------------------
          push      dx
          push      cx
          push      bx
          push      es
          push      ds
          pop       es                          ;es = segmento de datos
          mov       di,offset buff_rx
          mov       cl,0                        ;bandera de tx
load1:    cmp       cl,0                        ;hay datos para transmitir
          jne       load3                       ;no
          mov       dx,line_status              ;si, buffer de tx vacio?
          in        al,dx
          test      al,20h
          jz        load3                       ;no
          lodsb                                 ;carga byte
          cmp       al,0                        ;fin del comando?
          jne       load2                       ;no
          mov       cl,1                        ;cambia bandera de tx
          jmp       load3
load2:    mov       dx,uart_addr
          out       dx,al                       ;transmite byte
          mov       bx,0
load3:    mov       dx,line_status
          in        al,dx
          test      al,1                        ;hay dato recibido?
          jz        load4                       ;no
          mov       dx,uart_addr                ;si
          in        al,dx                       ;lee el caracter recibido
          cmp       al,">"                      ;es el comando promp?
          je        load5                       ;bien tx
          stosb                                 ;guardar byte
          mov       bx,0                        ;resetea tiempo de espera
          jmp       load1
load4:    inc       bx                          ;actualiza tiempo de espera
          cmp       bx,0                        ;supero tiempo de espera?
```

```
          jne       load1                   ;no
load5:    mov       si,offset buff_rx       ;analisis de los bytes recibidos
          cmp       si,di
          je        load9
load6:    lodsb                             ;carga byte de buffer de rx
          cmp       al,"E"                  ;byte es E?
          jne       load7                   ;no
          cmp       si,di                   ;si, fin de datos recibidos?
          je        load0                   ;si
          lodsb                             ;no, carga siguiente byte
          cmp       al,":"                  ;byte es :?
          je        load8                   ;si, error
          dec       si                      ;no
          jmp       load6                   ;sigue buscando
load7:    cmp       si,di                   ;fin de datos recibidos
          je        load0                   ;si
          jmp       load6                   ;no, sigue buscando
load8:    stc                               ;setea el carry
          mov       ax,1                    ;ax = 1
          jmp       loadf
load9:    stc                               ;setea el carry
          mov       ax,0                    ;ax = 0
          jmp       loadf
load0:    clc                               ;limpia el carry
loadf:    pop       es
          pop       bx
          pop       cx
          pop       dx
          ret
TX_LOAD ENDP

CSEG ENDS

END COMSER
;
```

```
;*************************************
;*********      DALLAS.ASM      **********
;*************************************


MCON      EQU      0C6H
TA        EQU      0C7H
;-------------------------------------------------------------------------------
;Variables usadas por el programa monitor
;-------------------------------------------------------------------------------
PCH       EQU      68H              ;PC del usuario
PCL       EQU      69H

REGACC    EQU      6AH              ;Respaldo de registros modificados
REGSP     EQU      6BH              ;dentro del programa monitor
REGB      EQU      6CH
REGR0     EQU      6DH
REGR1     EQU      6EH
REGDPL    EQU      6FH
REGDPH    EQU      70H
REGMCON   EQU      71H
FLAGS     EQU      72H

INST0     EQU      73H              ;buffer para guardar instruccion
INST1     EQU      74H              ;de entrada (3 bytes)
INST2     EQU      75H

CODIG0    EQU      76H              ;buffer para guardar instruccion
CODIG1    EQU      77H              ;del programa del usuario (3 bytes)
CODIG2    EQU      78H

STACK1    EQU      79H              ;variables para guardar stack
STACK2    EQU      7AH              ;del usuario
STACK3    EQU      7BH
STACK4    EQU      7CH
STACK5    EQU      7DH
STACK6    EQU      7EH

BANDERA   EQU      7FH              ;bandera para ejecucion

;-------------------------------------------------------------------------------
          ORG      00H              ;Inicio
          sjmp     start

          ORG      0BH
          ljmp     int_t0           ;Inicio de rutina de servicio
                                    ;de interrupcion T0

          ORG      30H
start:    mov      ta,#0aah         ;Inicializacion del
          mov      ta,#55h          ;Time Access
          mov      pcon,#0          ;Reset del watchdog timer

          mov      ie,#0            ;deshabilitacion de interrupciones
          mov      tmod,#21h        ;T1 = modo 2, T0 = modo 1
          mov      th1,#0feh        ;Seteo de velocidad de transmision
          mov      tl1,#0feh        ;a 9600 bits/seg, 8 bits de datos,
          orl      pcon,#80h        ;1 bit de parada y ninguna paridad
```

```
          mov       scon,#52h
          mov       tcon,#40h           ;activa T1

          mov       pch,#10h            ;PC inicial del programa
          mov       pcl,#0h             ;del usuario

a1:       jnb       ri,$                ;espera recibir serialmente un comando
          clr       ri                  ;limpia bandera de recepcion

          mov       regacc,a            ;guarda los registros
          mov       regb,b              ;que van a ser alterados
          mov       flags,psw           ;por el programa monitor
          mov       regr0,r0
          mov       regr1,r1
          mov       regdpl,dpl
          mov       regdph,dph

          mov       a,sbuf              ;carga en a el comando recibido

com_S?:   cjne      a,#53h,com_P?       ;comando S?
          sjmp      ej_uniq
com_P?:   cjne      a,#50h,com_Q?       ;comando P?
          ljmp      ej_paso
com_Q?:   cjne      a,#51h,com_E?       ;comando Q?
          ljmp      actual
com_E?:   cjne      a,#45h,com_M?       ;comando E?
          ljmp      editor
com_M?:   cjne      a,#4dh,com_N?       ;comando M?
          ljmp      d_memi
com_N?:   cjne      a,#4eh,com_I?       ;comando N?
          ljmp      d_meme
com_I?:   cjne      a,#49h,com_X?       ;comando I?
          ljmp      insert
com_X?:   ljmp      ning

ej_uniq:
;-------------------------------------------------------------------------
;Ejecuta una instruccion fuera del listado de programa del usuario
;si la instruccion es de salto altera el PC del usuario
;Entrada:  SBUF (IN) = numero de bytes de la instruccion
;          SBUF (IN) = codigo de la instruccion (maximo 3 bytes)
;Salida:   SBUF (OUT) = PC del usuario
;-------------------------------------------------------------------------
          jnb       ri,$                ;espera recibir el # de bytes
          clr       ri                  ;de la instruccion
          mov       a,sbuf
          mov       r1,a                ;guarda en r1 el # de bytes

          mov       r0,#inst0           ;r0 = direccion de almacenamiento
          mov       b,r1                ;b  = # de bytes a leer

load_i:   jnb       ri,$                ;espera los bytes de la inst.
          clr       ri
          mov       a,sbuf              ;lee un byte
          mov       @r0,a               ;guarda el byte
          inc       r0                  ;actualiza la direccion de almacenamiento
```

```
          djnz     b,load_i

          mov      b,#41              ;numero de instrucciones de salto
          mov      dptr,#tabla1       ;inicio de tabla con instrucciones de salto
busca:    mov      a,#0
          movc     a,@a+dptr          ;carga instruccion de salto
          cjne     a,inst0,no_es      ;instruccion ingresada mueve PC ?
          mov      bandera,#1         ;si, bandera = 1
          sjmp     fin_bus
no_es:    inc      dptr
          djnz     b,busca            ;fin de busqueda
          mov      bandera,#0         ;instruccion no mueve PC, bandera = 0
fin_bus:  mov      regmcon,mcon
          mov      ta,#0aah           ;setea memoria ROM como RAM
          mov      ta,#55h
          mov      mcon,#00000010b

          mov      a,bandera
          cjne     a,#0,mov_cod       ;instruccion de salto ?
;-------------------------------------------------------------------------------
;Proceso para instrucciones que no son de salto
;-------------------------------------------------------------------------------
          mov      dptr,#paso         ;dptr = direccion destino
          mov      r0,#inst0          ;r0 = direccion fuente
g:        mov      a,@r0              ;carga la instruccion
          movx     @dptr,a            ;fuera del listado del programa
          inc      r0                 ;del usuario
          inc      dptr
          djnz     r1,g

          mov      dptr,#paso         ;carga en #salto la
          mov      r0,dpl             ;direccion paso a la que debe
          mov      a,dph              ;saltar el programa para ejecutar
          mov      dptr,#salto        ;la instruccion
          inc      dptr
          movx     @dptr,a
          inc      dptr
          mov      a,r0
          movx     @dptr,a
          ljmp     g1                 ;salta a preparar la ejecucion
;-------------------------------------------------------------------------------
;Proceso para instrucciones de salto
;-------------------------------------------------------------------------------
mov_cod:  mov      dph,pch            ;guarda codigo del listado del programa
          mov      dpl,pcl            ;del usuario apuntado por PC
          movx     a,@dptr
          mov      codig0,a
          inc      dptr
          movx     a,@dptr
          mov      codig1,a
          inc      dptr
          movx     a,@dptr
          mov      codig2,a

          mov      dph,pch            ;carga en la direccion
          mov      dpl,pcl            ;apuntada por el PC del usuario
```

```
          mov     a,inst0           ;la instruccion ingresada
          movx    @dptr,a
          inc     dptr
          mov     a,inst1
          movx    @dptr,a
          inc     dptr
          mov     a,inst2
          movx    @dptr,a

          mov     dptr,#salto       ;carga en #salto la direccion
          inc     dptr              ;apuntada por el PC del usuario
          mov     a,pch             ;a la que debe saltar el programa
          movx    @dptr,a           ;para ejecutar la instruccion
          inc     dptr
          mov     a,pcl
          movx    @dptr,a
;-------------------------------------------------------------------------------
;Preparacion para ejecucion de instruccion
;-------------------------------------------------------------------------------
g1:       mov     ta,#0aah          ;vuelve a la definicion
          mov     ta,#55h           ;original de seteo de memoria
          mov     mcon,regmcon

prepa:    mov     b,#6
          mov     r0,sp             ;guarda 6 valores de
          dec     r0                ;alrededor del stack
          mov     r1,#stack1        ; I stack6 I
g_stac:   mov     a,@r0             ; I stack5 I
          mov     @r1,a             ; I stack4 I
          inc     r0                ; I stack3 I
          inc     r1                ; I stack2 I <- SP
          djnz    b,g_stac          ; I stack1 I
          mov     regsp,sp

          clr     tf0               ;limpia TF0 y TR0
          clr     tr0
          mov     th0,#0ffh         ;setea TH0 y TL0
          mov     tl0,#0efh
          setb    ea                ;habilita interrupciones
          setb    et0               ;habilita INT T0
          setb    tr0               ;setea TR0

          mov     a,regacc          ;restablece los registros
          mov     b,regb            ;guardados para la ejecucion
          mov     r0,regr0          ;de una instruccion
          mov     r1,regr1
          mov     dpl,regdpl
          mov     dph,regdph
          mov     psw,flags

          ljmp    salto             ;salta a la direccion de ejecucion

ej_paso:
;-------------------------------------------------------------------------------
;Ejecuta la instruccion apuntada por el PC del usuario
;Salida:           SBUF (OUT) = PC del usuario
```

```
;-------------
          mov       regmcon,mcon
          mov       ta,#0aah            ;setea memoria ROM como RAM
          mov       ta,#55h
          mov       mcon,#00000010b

          mov       bandera,#2          ;bandera de ejecucion = 2

          mov       dptr,#salto         ;carga en #salto la direccion
          inc       dptr                ;apuntada por el PC del usuario
          mov       a,pch               ;a la que debe saltar el programa para
          movx      @dptr,a             ;ejecutar la instruccion
          inc       dptr
          mov       a,pcl
          movx      @dptr,a

          mov       ta,#0aah            ;vuelve a la definicion
          mov       ta,#55h             ;original de seteo de memoria
          mov       mcon,regmcon

          ljmp      prepa               ;salta a preparar la ejecucion

;-------------------------------------------------------------------------------
;Salida de ejecucion de instruccion
;-------------------------------------------------------------------------------
siga:     clr       ea                  ;deshabilita interrupciones
          clr       et0

          mov       regacc,a            ;guarda acc
          mov       sp,#stack1

          mov       a,pch
          lcall     hexasc              ;transmite pch
          mov       a,pcl
          lcall     hexasc              ;transmite pcl

          mov       a,regacc            ;recupera acc
          mov       sp,regsp            ;restablece sp

          jnb       ti,$                ;fin del comando
          clr       ti                  ;trnsmite caracter promp ">"
          mov       sbuf,#3eh
          ljmp      a1                  ;regreso a lazo de espera de comando

actual:
;-------------------------------------------------------------------------------
;Descarga serialmente el status del microcontrolador
;Salida:           SBUF (OUT) = PC, ACC, PSW, B, DPTR, SP
;                  SBUF (OUT) = Banco de registros
;                  SBUF (OUT) = MCON
;                  SBUF (OUT) = Puertos
;-------------------------------------------------------------------------------
          mov       regsp,sp            ;guarda sp del usuario
          mov       sp,#stack1          ;fija nuevo stack

          mov       inst0,#0            ;checksum
```

```
        mov     inst0,a         ;actualizacion checksum
        mov     a,r2
        lcall   hexasc          ;transmite r2
        mov     a,r3
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,r3
        lcall   hexasc          ;transmite r3
        mov     a,r4
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,r4
        lcall   hexasc          ;transmite r4
        mov     a,r5
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,r5
        lcall   hexasc          ;transmite r5
        mov     a,r6
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,r6
        lcall   hexasc          ;transmite r6
        mov     a,r7
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,r7
        lcall   hexasc          ;transmite r7
        mov     a,mcon
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,mcon
        lcall   hexasc          ;transmite mcon
        mov     a,p0
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,p0
        lcall   hexasc          ;transmite p0
        mov     a,p1
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,p1
        lcall   hexasc          ;transmite p1
        mov     a,p2
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,p2
        lcall   hexasc          ;transmite p2
        mov     a,p3
        add     a,inst0
        mov     inst0,a         ;actualizacion checksum
        mov     a,p3
        lcall   hexasc          ;transmite p3

        mov     a,inst0         ;calculo de checksum
        cpl     a
```

```
          add       a,#1
          lcall     hexasc              ;transmite checksum

          mov       psw,flags           ;restablece registros alterados
          mov       a,regacc
          mov       sp,regsp

          jnb       ti,$                ;fin del comando
          clr       ti                  ;trnsmite caracter promp ">"
          mov       sbuf,#3eh
          ljmp      a1                  ;regreso a lazo de espera de comando

editor:
;-------------------------------------------------------------------------------
;Edita recursos del microcontrolador
;Entrada:          SBUF (IN) = Codigo de edicion
;                  SBUF (IN) = Byte de edicion
;                  SBUF (IN) = Direccion
;-------------------------------------------------------------------------------
          mov       regmcon,mcon
          anl       mcon,#11111011b     ;direcciona RAM de datos

          jnb       ri,$                ;espera byte codigo
          clr       ri
          mov       a,sbuf              ;carga en a el byte de codigo
e_acc:    cjne      a,#1,e_pc           ;editar acc?
          jnb       ri,$                ;si, espera byte
          clr       ri
          mov       regacc,sbuf         ;edita el acumulador
          ljmp      e_f
e_pc:     cjne      a,#2,e_dptr         ;editar PC?
          jnb       ri,$                ;si, espera byte mas sig.
          clr       ri
          mov       pch,sbuf            ;edita PC mas significativo
          jnb       ri,$                ;espera byte menos sig.
          clr       ri
          mov       pcl,sbuf            ;edita PC menos significativo
          ljmp      e_f
e_dptr:   cjne      a,#3,e_sp           ;editar dptr?
          jnb       ri,$                ;si,espera byte mas significativo
          clr       ri
          mov       regdph,sbuf         ;edita dph
          jnb       ri,$                ;espera byte menos significativo
          clr       ri
          mov       regdpl,sbuf         ;edita dpl
          ljmp      e_f
e_sp:     cjne      a,#4,e_b            ;editar sp?
          jnb       ri,$                ;si, espera byte
          clr       ri
          mov       sp,sbuf             ;edita sp
          ljmp      e_f
e_b:      cjne      a,#5,e_r0           ;editar b?
          jnb       ri,$                ;si, espera byte
          clr       ri
          mov       regb,sbuf           ;edita b
          ljmp      e_f
```

```
e_r0:   cjne    a,#6,e_r1       ;editar r0?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     regr0,sbuf      ;edita r0
        ljmp    e_f
e_r1:   cjne    a,#7,e_r2       ;editar r1?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     regr1,sbuf      ;edita r1
        ljmp    e_f
e_r2:   cjne    a,#8,e_r3       ;editar r2?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     r2,sbuf         ;edita r2
        ljmp    e_f
e_r3:   cjne    a,#9,e_r4       ;editar r3?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     r3,sbuf         ;edita r3
        ljmp    e_f
e_r4:   cjne    a,#10,e_r5      ;editar r4?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     r4,sbuf         ;edita r4
        ljmp    e_f
e_r5:   cjne    a,#11,e_r6      ;editar r5?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     r5,sbuf         ;edita r5
        ljmp    e_f
e_r6:   cjne    a,#12,e_r7      ;editar r6?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     r6,sbuf         ;edita r6
        ljmp    e_f
e_r7:   cjne    a,#13,e_psw     ;editar r7?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     r7,sbuf         ;edita r7
        ljmp    e_f
e_psw:  cjne    a,#14,e_mi      ;editar psw?
        jnb     ri,$            ;si, espera byte
        clr     ri
        mov     flags,sbuf      ;edita psw
        ljmp    e_f
e_mi:   cjne    a,#15,e_me      ;editar memoria interna?
        jnb     ri,$            ;si, espera bytes
        clr     ri
        mov     a,sbuf
        anl     psw,#11100111b  ;banco 0
        cjne    a,#0,noR0       ;no es r0
        jnb     ri,$
        clr     ri
        mov     regr0,sbuf      ;edita en r0 el dato
        ljmp    e_f
nor0:   cjne    a,#1,noR1       ;no es r1
```

```
          jnb       ri,$
          clr       ri
          mov       regr1,sbuf          ;edita en r1 el dato
          ljmp      e_f
noR1:     mov       r0,a                ;lee direccion
          jnb       ri,$
          clr       ri
          mov       a,sbuf              ;lee byte dato
          mov       @r0,a               ;edita en memoria interna
          ljmp      e_f
e_me:
          cjne      a,#16,e_f           ;editar memoria externa?
          jnb       ri,$                ;si, espera bytes
          clr       ri
          mov       dph,sbuf            ;lee direccion mas significativa
          jnb       ri,$
          clr       ri
          mov       dpl,sbuf            ;lee direccion menos significativa
          jnb       ri,$
          clr       ri
          mov       a,sbuf              ;lee byte dato

          mov       regmcon,mcon
          anl       mcon,#11111011b     ;direcciona RAM de datos

          movx      @dptr,a             ;edita en memoria externa

          mov       mcon,regmcon
e_f:      mov       a,regacc            ;restablece registros
          mov       b,regb              ;que fueron guardados
          mov       r0,regr0
          mov       r1,regr1
          mov       dpl,regdpl
          mov       dph,regdph
          mov       psw,flags

          jnb       ti,$                ;fin del comando
          clr       ti                  ;trnsmite caracter promp ">"
          mov       sbuf,#3eh
          ljmp      a1                  ;regreso a lazo de espera de comando

d_memi:
;-------------------------------------------------------------------------
;Descarga informacion de la memoria interna
;Entrada:          SBUF (IN) = direccion inicial de memoria interna
;Salida:           SBUF (OUT) = 32 bytes de memoria interna
;-------------------------------------------------------------------------
          anl       psw,#11100111b      ;setea banco 0
          mov       regr0,r0
          jnb       ri,$                ;espera byte con direccion inicial
          clr       ri
          mov       regsp,sp            ;guarda sp
          mov       sp,#stack1          ;fija nuevo stack

          mov       inst0,#32           ;checksum
          mov       a,#32
```

```
          lcall    hexasc              ;transmite numero total de bytes
          mov      a,sbuf              ;carga direccion inicial
          mov      b,#32
d_memi0:  anl      a,#7fh              ;corrigue direccion
          cjne     a,#0,n_r0           ;es r0?
          mov      stack1,a            ;si, salva acumulador
          mov      a,regr0
          add      a,inst0             ;actualiza checksum
          mov      inst0,a
          mov      a,regr0
          lcall    hexasc              ;transmite r0
          mov      a,stack1            ;recupera acumulador
          sjmp     lazo_m
n_r0:     mov      r0,a                ;corrigue direccion
          mov      stack1,a            ;si, salva acumulador
          mov      a,@r0
          add      a,inst0             ;actualiza checksum
          mov      inst0,a
          mov      a,@r0
          lcall    hexasc              ;transmite byte de memoria
          mov      a,stack1            ;recupera acumulador
lazo_m:   inc      a                   ;actualiza direccion
          djnz     b,d_memi0

          mov      a,inst0             ;calcula checksum
          cpl      a
          add      a,#1
          lcall    hexasc              ;transmite checksum

          mov      sp,regsp            ;recupera sp

          mov      r0,regr0            ;restablece registros
          mov      a,regacc            ;que fueron alterados
          mov      b,regb
          mov      psw,flags

          jnb      ti,$                ;fin del comando
          clr      ti                  ;trnsmite caracter promp ">"
          mov      sbuf,#3eh
          ljmp     a1                  ;regreso a lazo de espera de comando

d_meme:
;-------------------------------------------------------------------------------
;Descarga informacion de la memoria embebida
;Entrada:          SBUF (IN) = direccion inicial de memoria embebida
;Salida:           SBUF (OUT) = 32 bytes de memoria embebida
;-------------------------------------------------------------------------------
          mov      a,mcon
          swap     a
          anl      a,#0fh              ;bits de particion
          mov      dptr,#tabla3
          movc     a,@a+dptr           ;encuentra el byte mas sigf.
          mov      regr0,a             ;de la direccion de particion


          jnb      ri,$                ;espera word de direccion
```

```
        clr     ri
        mov     dph,sbuf            ;carga direccion  mas significativa
        jnb     ri,$
        clr     ri
        mov     dpl,sbuf            ;carga direccion  menos significativa

        mov     regsp,sp            ;guarda sp
        mov     sp,#stack1          ;fija nuevo stack

        mov     regmcon,mcon        ;guarda mcon
        anl     mcon,#11111011b     ;direcciona ram de datos

        mov     inst0,#32           ;checksum
        mov     a,#32
        lcall   hexasc              ;transmite numero de bytes

        mov     b,#32               ;32 datos a transmitirse
d_me0:  clr     c
        mov     a,dph
        subb    a,regr0             ;rom o ram?
        jc      rom                 ;rom
        mov     a,dph               ;ram
        anl     a,#7fh
        mov     dph,a               ;corrigue direccion
        movx    a,@dptr
        add     a,inst0             ;actualiza checksum
        mov     inst0,a
        movx    a,@dptr
        lcall   hexasc              ;transmite byte de ram
        sjmp    d_me1
rom:    mov     a,#0
        movc    a,@a+dptr
        add     a,inst0             ;actualiza checksum
        mov     inst0,a
        mov     a,#0
        movc    a,@a+dptr
        lcall   hexasc              ;transmite byte de rom
d_me1:  inc     dptr
        djnz    b,d_me0

        mov     a,inst0             ;calcula checksum
        cpl     a
        add     a,#1
        lcall   hexasc              ;transmite checksum

        mov     sp,regsp            ;recupera sp

        mov     a,regacc            ;restablece registros
        mov     b,regb              ;que fueron alterados
        mov     r0,regr0
        mov     dpl,regdpl
        mov     dph,regdph
        mov     psw,flags

        jnb     ti,$                ;fin del comando
        clr     ti                  ;transmite caracter promp ">"
```

```
          mov     sbuf,#3eh
          ljmp    a1                  ;regreso a lazo de espera de comando

insert:
;-------------------------------------------------------------------------
;Inserta una instruccion en el listado del programa (sustiye el codigo
;anterior)
;Entrada:          SBUF (IN) = numero de bytes de la instruccion
;                  SBUF (OUT) = codigo de la instruccion
;-------------------------------------------------------------------------
          jnb     ri,$                ;espera recibir el # de bytes
          clr     ri                  ;de la instruccion
          mov     r1,sbuf             ;guarda en r1 el # de bytes

          mov     r0,#inst0           ;r0 = dir. de almacenamiento
          mov     b,r1                ;b  = # de bytes a leer

ins0:     jnb     ri,$                ;espera los bytes de la inst.
          clr     ri
          mov     a,sbuf              ;lee un byte
          mov     @r0,a               ;guarda el byte
          inc     r0                  ;actualiza la direccion de almacenamiento
          djnz    b,ins0

          mov     regmcon,mcon
          mov     ta,#0aah            ;setea memoria ROM como RAM
          mov     ta,#55h
          mov     mcon,#00000010b

          mov     dph,pch             ;dptr = direccion pc (destino)
          mov     dpl,pcl
          mov     r0,#inst0           ;r0 = direccion fuente
ins1:     mov     a,@r0               ;carga la intruccion
          movx    @dptr,a             ;en la direccion apuntada por PC
          inc     r0
          inc     dptr
          djnz    r1,ins1

          mov     ta,#0aah            ;vuelve a la definicion
          mov     ta,#55h             ;original de seteo de memoria
          mov     mcon,regmcon

          mov     a,regacc            ;restablece registros
          mov     b,regb              ;que fueron alterados
          mov     r0,regr0
          mov     r1,regr1
          mov     dpl,regdpl
          mov     dph,regdph
          mov     psw,flags

          jnb     ti,$                ;fin del comando
          clr     ti                  ;trnsmite caracter promp ">"
          mov     sbuf,#3eh
          ljmp    a1                  ;regreso a lazo de espera de comando

ning:
```

```
;-------------------------------------------------------------------------
;Salida para un comando que no es conocido
;-------------------------------------------------------------------------
          mov       a,regacc            ;restablece acc

          jnb       ti,$                ;fin del comando
          clr       ti                  ;trnsmite caracter promp ">"
          mov       sbuf,#3eh
          ljmp      a1                  ;regreso a lazo de espera de comando

hexasc:
;-------------------------------------------------------------------------
;Conversion de un numero hexadecimal a ascii
;Entrada:          ACC = numero hexadecimal
;Salida:           SBUF (OUT) = numero ascii
;-------------------------------------------------------------------------
          push      dpl                 ;salva dptr
          push      dph
          push      acc                 ;salva acc
          mov       dptr,#tabla2        ;origen de tabla de conversion
          swap      a
          anl       a,#0fh              ;nibble mas significativo
          movc      a,@a+dptr           ;acc = ascii correspondiente
          jnb       ti,$
          clr       ti
          mov       sbuf,a              ;transmite el nibble
          pop       acc                 ;recupera byte de transmision
          anl       a,#0fh
          movc      a,@a+dptr           ;acc = ascii correspondiente
          jnb       ti,$
          clr       ti
          mov       sbuf,a              ;transmite el nibble
          pop       dph                 ;recupera dptr
          pop       dpl
          ret

int_t0:
;-------------------------------------------------------------------------
;Rutina de servicio a la interruccion del Timer 0, restablece valores
;del stack alterados, determina el nuevo valor de PC, modifica direccion
;de salida de la interrupcion
;Entrada:          BANDERA = tipo de ejecucion
;                  0         Ejecucion fuera del listado del programa (no salto)
;                  1         Ejecucion fuera del listado del programa (salto)
;                  2         Ejecucion dentro del programa
;-------------------------------------------------------------------------
          clr       ea                  ;deshabilita interrucciones
          mov       regdph,dph          ;guarda registros utilizados
          mov       regdpl,dpl
          mov       regacc,a
          mov       regr0,r0
          mov       regr1,r1
          mov       flags,psw

          mov       a,bandera           ;ejecucion es de salto ?
          cjne      a,#0,act_pc1        ;sio
```

```
          ljmp    no_act              ;no, no actualiza pc
act_pc1:  cjne    a,#1,act_pc2

          mov     regmcon,mcon
          mov     ta,#0aah            ;setea memoria ROM como RAM
          mov     ta,#55h
          mov     mcon,#00000010b

          mov     dph,pch             ;restablece el codigo del usuario
          mov     dpl,pcl             ;en la direccion apuntada por PC
          mov     a,codig0
          movx    @dptr,a
          inc     dptr
          mov     a,codig1
          movx    @dptr,a
          inc     dptr
          mov     a,codig2
          movx    @dptr,a

          mov     ta,#0aah            ;vuelve a la definicion
          mov     ta,#55h             ;original de seteo de memoria
          mov     mcon,regmcon

act_pc2:  mov     r0,sp               ;carga en pch y pcl
          mov     a,@r0               ;la direccion de la proxima
          mov     pch,a               ;instruccion a ejecutarse
          dec     r0
          mov     a,@r0
          mov     pcl,a

no_act:   mov     a,regsp             ;determina el tipo de inst.
          cjne    a,sp,no_ret         ;instruccion ret
          sjmp    ok_ins
no_ret:   inc     a
          cjne    a,sp,no_pop         ;instruccion pop
          sjmp    ok_ins
no_pop:   inc     a
          cjne    a,sp,no_otro        ;instruccion otro
          sjmp    ok_ins
no_otro:  inc     a
          cjne    a,sp,no_push        ;instruccion push
          sjmp    ok_ins
no_push:  inc     a                   ;instruccion call
ok_ins:   clr     c
          subb    a,regsp
          mov     r1,#stack1
          add     a,r1
          mov     r1,a
          mov     r0,sp
          dec     r0
          mov     a,@r1               ;restituye bytes alterados
          mov     @r0,a               ;por la interrupcion
          inc     r0
          inc     r1
          mov     a,@r1
          mov     @r0,a
```

```
        dec     sp                  ;ajusta el stack pointer
        dec     sp

        mov     regsp,sp            ;guarda sp

        mov     sp,#stack6          ;altera sp para salir
        mov     r0,sp               ;de la interrupcion
        mov     dptr,#siga          ;a la direccion siga
        mov     a,dph
        mov     @r0,a
        dec     r0
        mov     a,dpl
        mov     @r0,a

        mov     dph,regdph          ;restablece los registros
        mov     dpl,regdpl          ;alterados
        mov     a,regacc
        mov     r0,regr0
        mov     r1,regr1
        mov     psw,flags

        reti

tablai:
;-------------------------------------------------------------------------------
;Tabla con el codigo de instrucciones que alteran PC (salto)
;-------------------------------------------------------------------------------
        db      1           ;ajmp
        db      2           ;ljmp
        db      10h         ;jbc
        db      11h         ;acall
        db      12h         ;lcall
        db      20h         ;jb
        db      21h         ;ajmp
        db      22h         ;ret
        db      30h         ;jnb
        db      31h         ;acall
        db      32h         ;reti
        db      40h         ;jc
        db      41h         ;ajmp
        db      50h         ;jnc
        db      51h         ;acall
        db      60h         ;jz
        db      61h         ;ajmp
        db      70h         ;jnz
        db      71h         ;acall
        db      73h         ;jmp @a + dptr
        db      81h         ;ajmp
        db      91h         ;acall
        db      0a1h        ;ajmp
        db      0b1h        ;acall
        db      0b4h        ;cjne a,#data,rel
        db      0b5h        ;cjne a,direct,rel
        db      0b6h        ;cjne @ri,#data,rel
        db      0b7h
```

```
          db        0b8h        ;cjne rn,#data,rel
          db        0b9h
          db        0bah
          db        0bbh
          db        0bch
          db        0bdh
          db        0beh
          db        0dfh
          db        0c1h        ;ajmp
          db        0d1h        ;acall
          db        0d5h        ;djnz direct,rel
          db        0e1h        ;ajmp
          db        0f1h        ;acall

tabla2:
;-------------------------------------------------------------------------------
;Tabla para conversion de numeros hexadecimales a ascii
;-------------------------------------------------------------------------------
          db        30h         ;0
          db        31H         ;1
          db        32H         ;2
          db        33H         ;3
          db        34H         ;4
          db        35H         ;5
          db        36H         ;6
          db        37H         ;7
          db        38H         ;8
          db        39H         ;9
          db        41H         ;A
          db        42H         ;B
          db        43H         ;C
          db        44H         ;D
          db        45H         ;E
          db        46H         ;F

tabla3:
;-------------------------------------------------------------------------------
;Tabla con las direcciones de particion
;-------------------------------------------------------------------------------
          db        0
          db        08h
          db        10h
          db        18h
          db        20h
          db        28h
          db        30h
          db        38h
          db        40h
          db        48h
          db        50h
          db        58h
          db        60h
          db        68h
          db        70h
          db        80h
```

B. MANUAL DEL USUARIO:

El programa depurador requiere un archivo que contenga el programa del usuario ensamblado, este archivo debe ser de formato INTEL de 8 bits.

El depurador puede trabajar en 3 modos:

- Modo de Simulación
- Modo de Ejecución Controlada
- Modo de Carga Serial

Modo de Simulación.- Simula en la pantalla la ejecución de un programa del usuario codificado para el microcontrolador DS5000T.

Modo de Ejecución Controlada.- Ejecuta instrucciones del programa del usuario directamente en el microcontrolador, bajo control del computador personal.

Modo de Carga Serial.- Descarga el programa del usuario hacia el microcontrolador DS5000T, con la posibilidad de configurar las condiciones de operación: partición de memoria, encriptamiento,etc.

El paquete está constituido por un conjunto de archivos, los cuales son necesarios para correr el programa principal. Estos archivos son:

DALLAS.EXE Archivo principal.

DALLAS.VNT Archivo de ventanas.

DALLAS.HEX Archivo monitor para el DS5000T.

Se puede ejecutar el programa ingresando el siguiente comando:

DALLAS <UNIDAD:> <CAMINO> <NOMBRE.EXT> <M>

donde:

DALLAS Nombre del programa de depuración.

UNIDAD Unidad donde se encuentra el archivo del usuario

CAMINO Directorio donde se encuentra el programa del usuario

NOMBRE.EXT Nombre y extensión del archivo del usuario.

M Modo de operación:

- S Simulación
- E Ejecución Controlada
- C Carga Serial

Si no se ingresan todas las especificaciones, el programa presenta mensajes para que el usuario las ingrese.

El programa se maneja en base a menúes. Cuando al usuario se le presente un menú puede tomar alguna de estas opciones.

a) Presionar una de las flechas y moverse a una opción cualquiera.

b) Digitar la tecla ENTER entrando con ésto a cumplir la rutina asociada a la opción sobre la que estaba ubicado y luego a recibir el tratamiento adecuado de acuerdo al "tipo de función" de que se trate.

c) Digitar la letra clave de una opción, con lo que entra directamente a esta opción sin necesidad de ubicarse sobre ella. Realiza la rutina asociada y luego recibe el tratamiento adecuado al "tipo de función".

d) Cualquier otra alternativa no es tomada en cuenta.

Una vez ejecutada la rutina asociada a una opción existen varias alternativas (A esto se refieren las palabras "tipo de función" indicadas anteriormente). Estas alternativas son:

a) Se cumple la rutina asociada y se regresa al mismo menú que contiene la opción. Estas rutinas podrían llamarse "normales".

b) Se cumple la rutina asociada y se emerge un nivel en el árbol de opciones, es decir, se transfiere el control al menú "padre" del que contenía la opción. Estas funciones serán llamadas "inversas".

c) Se ejecuta la rutina y se profundiza un nivel, mostrando un nuevo menú, hijo de la opción que fue digitada. Estas opciones son del tipo "rama".

El menú aparece en la penúltima línea de la pantalla (línea 23) y una explicación sobre la opción actual en la última línea (línea 24).

La tecla (ESC), permite siempre salir al nivel superior de menúes. Si se está en el menú principal, ESC significa que se desea salir del programa, en todo caso se hace una pregunta para confirmar esta decisión.

La tecla (F2), presenta una ventana sobre la opción en la que se está ubicado actualmente con las subopciones que ésta posee, si existen. En el momento que aparece esa ventana de subopciones, las flechas hacia arriba y hacia abajo son aceptadas para moverse en ese menú. La explicación que aparece en la línea 24 es la asociada a la subopción en la que está actualmente. Si se presiona ENTER sobre una subopción se ingresa a la opción que la contiene y el cursor queda ubicado sobre la subopción seleccionada, listo para que una nueva presión de la tecla ENTER la mande a ejecutar. El mismo F2 desactiva la ventana de subopciones si no la necesita.

La tecla (F1), presenta una pantalla de ayuda acerca de la manera de manejar los menúes.

Display: Activa o desactiva la actualización de pantalla durante la ejecución del programa en la opción Corre. Si la actualización está desactivada, actualiza la pantalla cuando sale de la opción Corre; esto es, si se detiene en un breakpoint, el usuario detiene la ejecución o cuando el programa esta dentro de un lazo.

Reset: Inicializa la depuración del programa con los valores de reset del microcontrolador.

Ayuda: Presenta una ventana con el árbol de opciones del depurador, acompañadas de una breve explicación.

Salir: Posibilita salir del depurador.

MODO DE SIMULACION

En el modo de simulación, se puede simular en el computador personal todas las características del microcontrolador, y no se requiere equipo adicional.

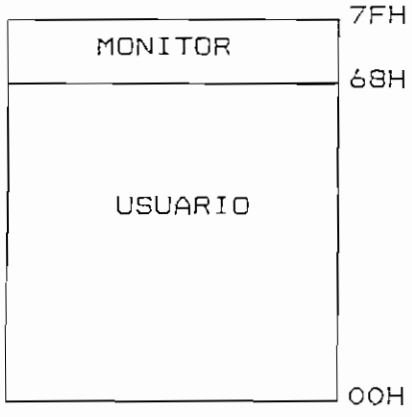
MODO DE EJECUCION CONTROLADA

En el modo de ejecución controlada, se tiene que descargar hacia el microcontrolador el programa monitor y el programa a depurarse. Es necesario por tanto equipo de

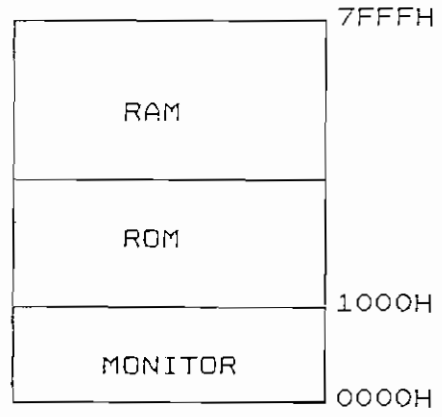
hardware adicional en el que se encuentre el microcontrolador, para lo que se dispone de un kit de desarrollo.

La necesidad de un programa monitor hace que se tenga que utilizar recursos del microcontrolador, los cuales no están a disposición del usuario, a continuación se indican las especificaciones y recursos disponibles para trabajar en este modo.

RAM INTERNA



RAM EMBEBIDA



REGISTROS ESPECIALES

Etiqueta	Dirección	Accesible
B	0F0H	Si
A	0E0H	Si
PSW	0D0H	Si
TA	0C7H	No
MCON	0C6H	Parcialmente (ECE2)
EK4-1	0C5H-0C1H	No
IP	0B8H	No
P3	0B0H	Si
IE	0A8H	No
P2	0A0H	Si
SBUF	099H	No
SCON	098H	No
P1	090H	Si
TH1	08DH	No
TH0	08CH	No
TL1	08BH	No
TL0	0BAH	No
TMOD	089H	No
TCON	088H	No
PCON	087H	No
DPH	083H	Si
DPL	082H	Si
SP	081H	Si
P0	080H	Si

El origen del programa debe comenzar en la dirección 1000h, adicionalmente se puede setear la partición de memoria embebida, sólo al inicio, en la descarga del programa.

MODO DE CARGA SERIAL

En este modo se carga el programa del usuario, para ejecución en tiempo reaal, se puede configurar al microcontrolador (encriptar, particionar memoria y asegurar el software cargado). Se requiere del kit de desarrollo.

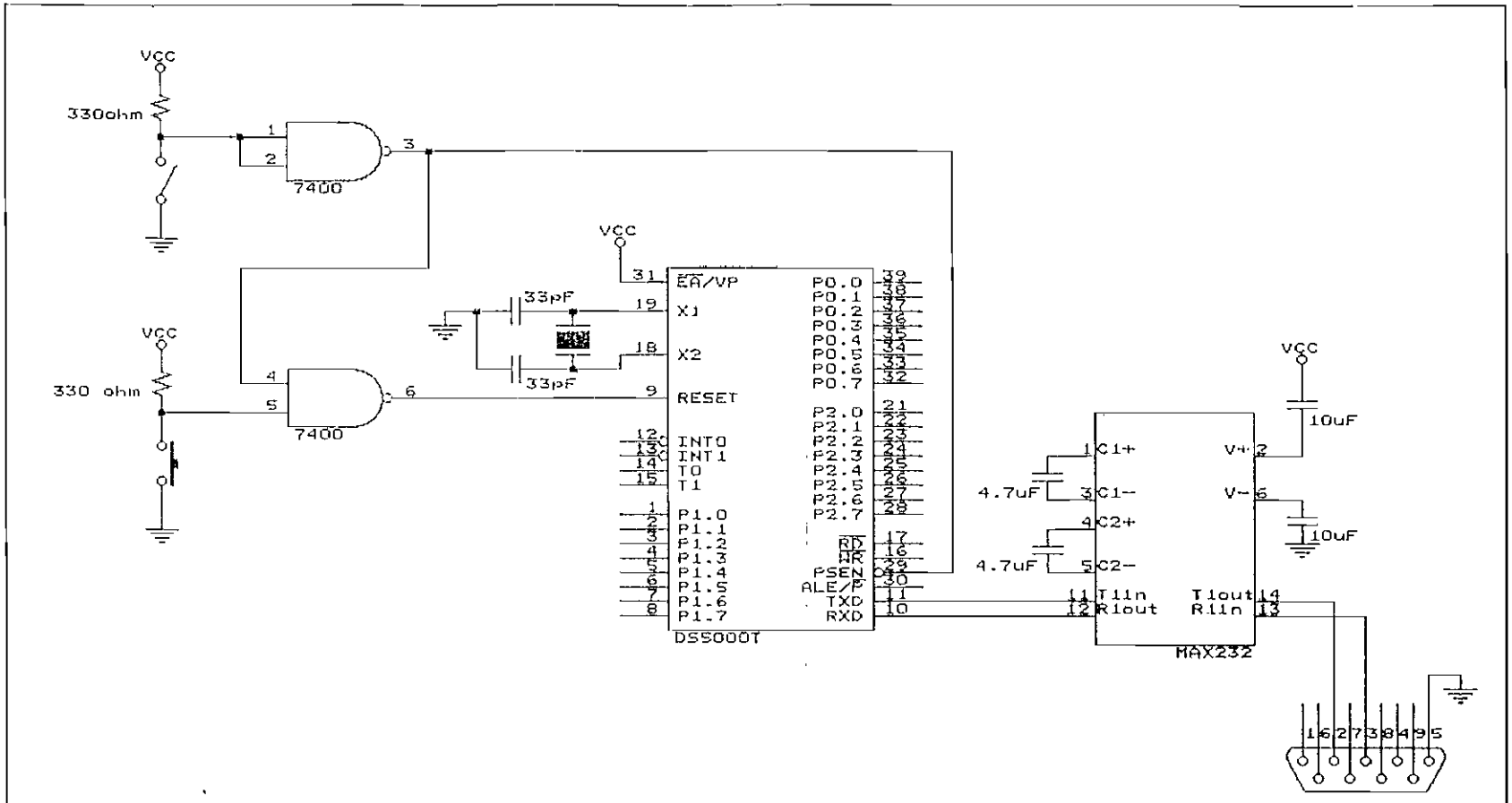


Figura 8.1.- Diagrama circuital del Kit de Desarrollo.

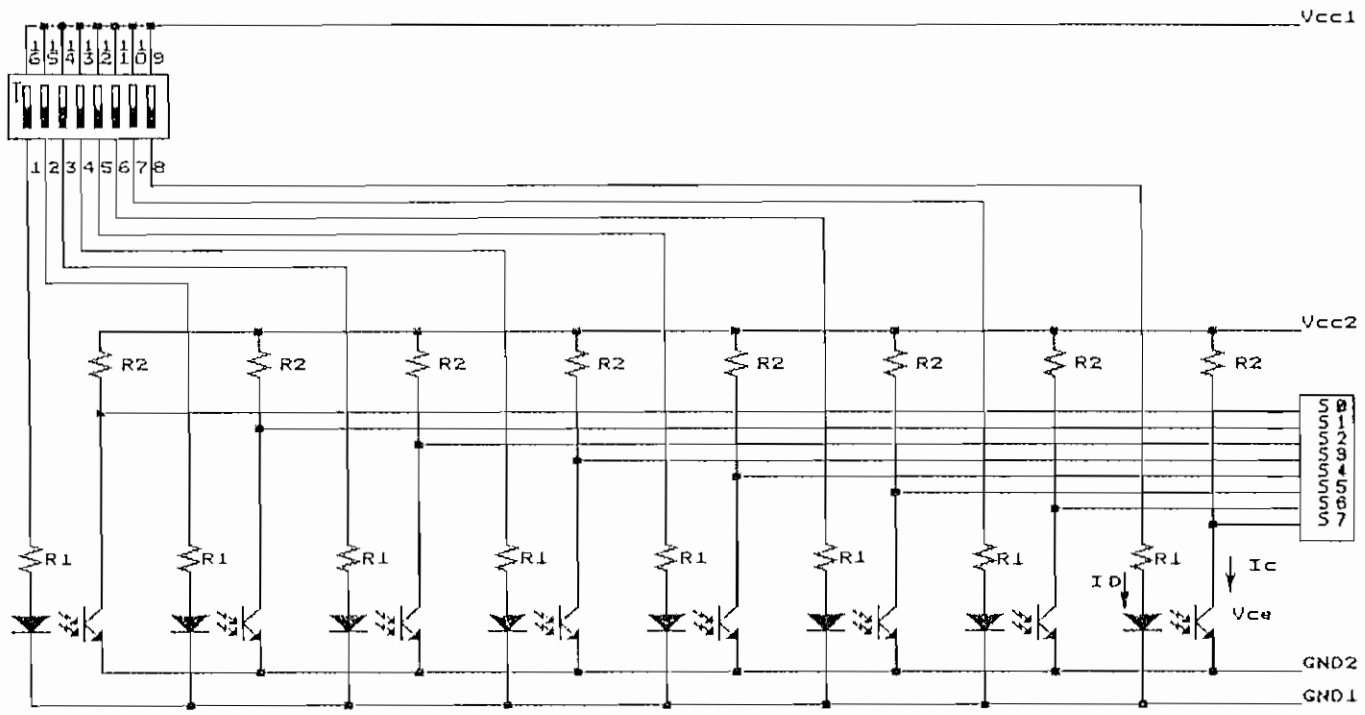


Figura B.2.- Diagrama circuital de los dip_switches.

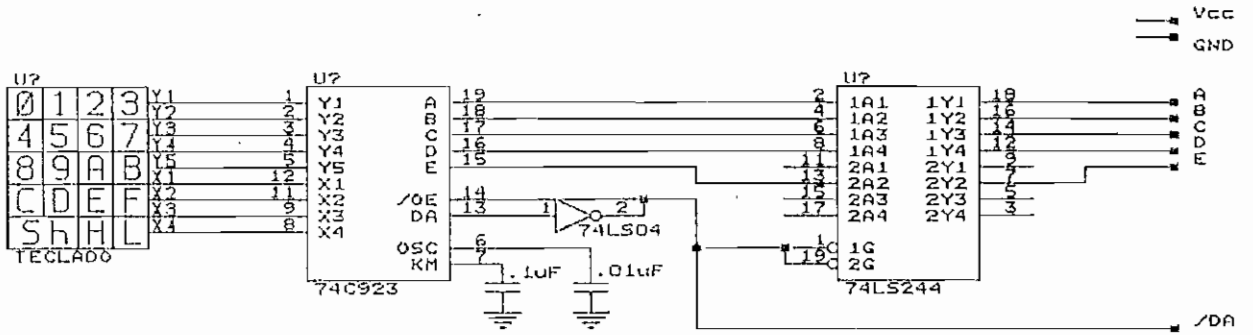


Figura B.3.- Diagrama circuital del teclado.

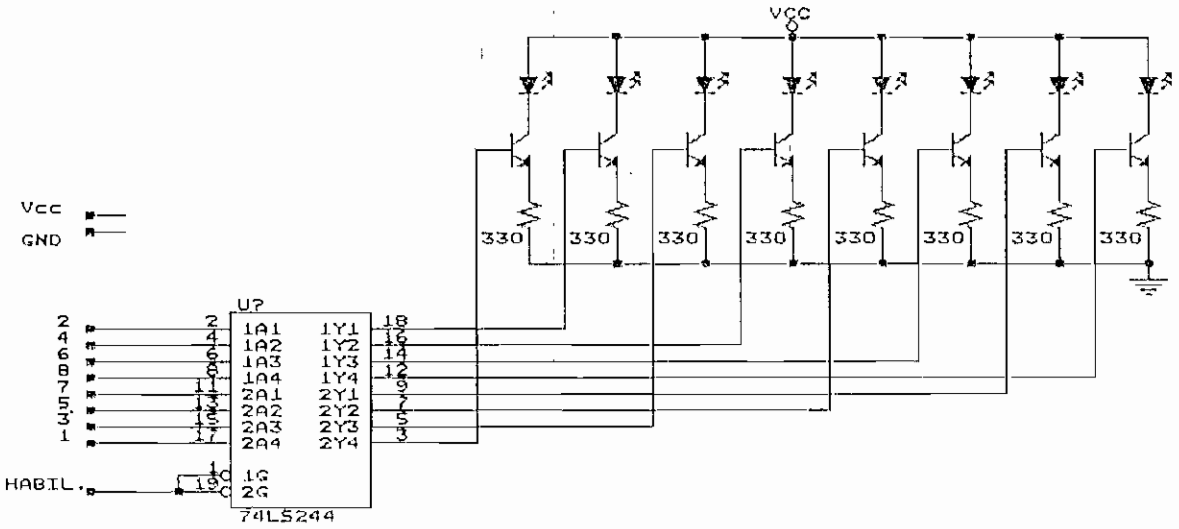


Figura 8.4.- Diagrama circuital de los leds.

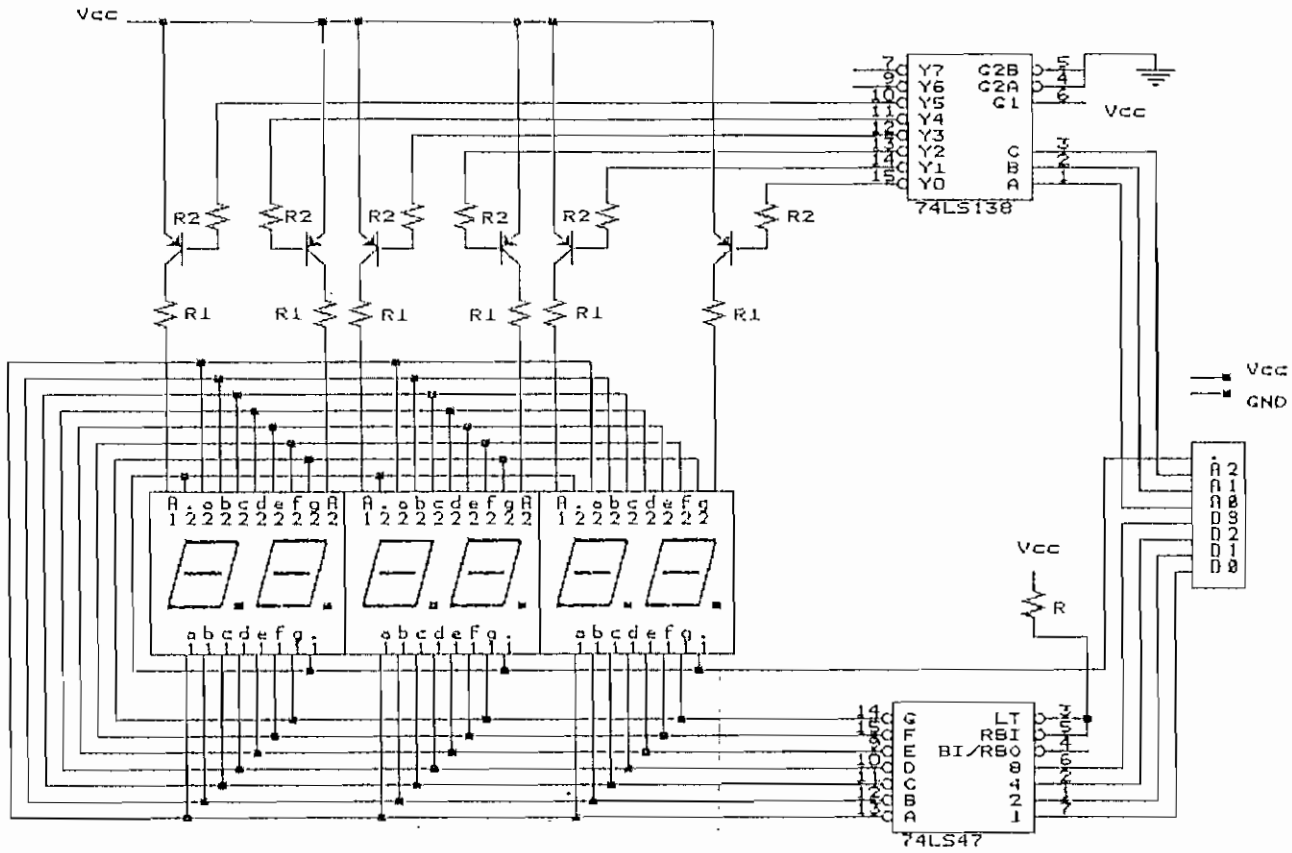


Figura B.5.- Diagrama circuital del arreglo de displays

C. INTERRUPCIONES DEL ROM BIOS Y DOS

INTERRUPCIONES DEL DOS

Nueve son las interrupciones del DOS que se llaman a través de sus números. Cinco de estas interrupciones están listadas en la tabla C.1. Las cuatro interrupciones no mostradas en esta tabla se usan para propósitos especializados: la interrupción 33 (21 hex) se usa para llamar a cada una de las ochenta funciones del DOS, y las interrupciones desde la 34 hasta la 36 son interrupciones de dirección que se usan para señalar subrutinas especiales.

Las funciones universales del DOS, mostradas en la tabla C.2 son llamadas mediante la interrupción 33; el número de la función se sitúa en el registro AH. Las funciones universales pueden ser usadas con cualquier versión del DOS.

Las nuevas llamadas ampliadas a funciones del DOS pueden ser usadas solamente con las versiones DOS 2.00 , o superiores. Son llamadas mediante la interrupción 33 y el número de función se sitúa en el registro AH. La tabla C.3 lista todas las funciones nuevas del DOS, incluyendo aquellas que fueron introducidas con la versión DOS 3.00.

Interrupción Dec	Hex	Descripción
32	20	Terminación de programa: ir a un final normal
37	25	Lectura de disco absoluta.
38	26	Escritura del disco absoluta.
39	27	Terminar permaneciendo residente.
47	2F	Control de la cola de impresión (spooler) (DOS 3.00 y posteriores).

Tabla C.1. Las cinco interrupciones principales del DOS.

Función Dec	Hex	Descripción	Función Dec	Hex	Descripción
0	0	Terminar: fin del programa.	23	17	Renombrar un fichero.
1	1	Entrada del teclado con eco.	25	19	Comunicar la unidad activa.
2	2	Salida al display.	26	1A	Fijar la zona de transferencia del disco.
3	3	Entrada serie.	27	1B	Obtener información FAT de la unidad activa.
4	4	Salida serie.	28	1C	Obtener información FAT de cualquier unidad.
5	5	Salida de impresora.	33	21	Leer un registro de un fichero aleatorio.
6	6	Entrada/salida directa teclado/display.	34	22	Escribir un registro en un fichero aleatorio.
7	7	Entrada directa del teclado sin eco.	35	23	Obtener el tamaño de un fichero.
8	8	Entrada del teclado sin eco.	36	24	Fijar el campo de un registro aleatorio.
9	9	Escribir cadena en pantalla.	37	25	Fijar un vector de interrupción.
10	A	Entrada del teclado con buffer	38	26	Crear un segmento de programa.
11	B	Comprobación del estado de entrada del teclado.	39	27	Leer registros de un fichero aleatorio.
12	C	Borrar el teclado y ejecutar función.	40	28	Escribir registros en un fichero aleatorio.
13	D	Rearrancar el disco.	41	29	Analizar el nombre de un fichero.
14	E	Seleccionar la unidad activa.	42	2A	Obtener la fecha.
15	F	Abrir un fichero.	43	2B	Fijar la fecha.
16	10	Cerrar un fichero.	44	2C	Obtener la hora.
17	11	Buscar un primer fichero.	45	2D	Fijar la hora.
18	12	Buscar el siguiente fichero.	46	2E	Activar/desactivar la verificación de escritura en disco.
19	13	Borrar un fichero.	47	2F	Obtener la dirección del DTA.
20	14	Leer un registro de un fichero secuencial.	48	30	Obtener la versión del DOS.
21	15	Escribir un registro en un fichero secuencial.	49	31	KEEP: Terminar permaneciendo residente avanzado.
22	16	Crear un fichero.	51	33	Obtener/ajutar control break.

Tabla C.2.- Las funciones universales del DOS.

Función Dec	Hex	Descripción	Función Dec	Hex	Descripción
53	35	Obtener el vector de interrupción.	69	45	DUP: Duplicar el número del fichero.
54	36	Obtener el espacio libre del disco.	70	46	CDUP: Duplicación forzada del título.
56	38	Obtener información dependiente del país.	71	47	Obtener el directorio activo.
57	39	MKDIR: Crear directorio.	72	48	Asignar memoria.
58	3A	RMDIR: Borrar directorio.	73	49	Liberar memoria asignada.
59	3B	CHDIR: Cambiar el directorio activo.	74	4A	SETBLOCK: Modificar un bloque de memoria asignada.
60	3C	CREAT: Crear un fichero.	75	4B	EXEC: Cargar/ejecutar un programa.
61	3D	Abrir un fichero.	76	4C	Terminar un proceso.
62	3E	Cerrar un número de fichero.	77	4D	Obtener el código de retorno de un subprograma.
63	3F	Leer desde fichero o dispositivo.	78	4E	FIND FIRST: Empezar la búsqueda de ficheros.
64	40	Escribir en un fichero o dispositivo.	79	4F	Continuar la búsqueda de ficheros.
65	41	Borrar fichero.	84	54	Obtener el estado de verificación.
66	42	Mover el puntero del fichero.	86	56	Renombrar un fichero.
67	43	CHMOD: Obtener/modificar los atributos del fichero.	87	57	Obtener/modificar la fecha y la hora de un fichero.
68	44	IOCTL: Control de entrada/salida para los dispositivos.			
Funciones del DOS 3.00					
89	59	Obtener el código de error ampliado.	91	5B	Crear un fichero nuevo.
90	5A	Crear un fichero transitorio.	92	5C	Bloquear/desbloquear el acceso a un fichero.
			98	62	Obtener la dirección del PSP.

Tabla C.3.- Las funciones nuevas del DOS disponibles con el DOS 2.00 y las versiones posteriores.

INTERRUPCIONES DE LA ROM-BIOS

A continuación se listan brevemente todos los servicios de la ROM-BIOS:

Tema	Interrupción Dec	Interrupción Hex	Servicio (Hex)	Descripción	Modo Específico
Impresión de pantalla	5	5	n/a	Envía los contenidos de la pantalla a la impresora.	
Video	16	10	0	Fija el modo de video.	
Video	16	10	1	Ajusta el tamaño del cursor.	
Video	16	10	2	Posiciona el cursor.	
Video	16	10	3	Lee la posición del cursor.	
Video	16	10	4	Lee la posición del lápiz óptico.	
Video	16	10	5	Elección de la página activa.	
Video	16	10	5 (AL:128)	Obtención de los registros de la página de visualización.	
Video	16	10	5 (AL:129)	Da valores al registro de la página de visualización de la CPU.	
Video	16	10	5 (AL:130)	Da valores al registro de la página de visualización del CRT.	
Video	16	10	5 (AL:131)	Da valores a ambos registros de página de visualización.	
Video	16	10	6	Desplazamiento vertical ascendente (Scroll up).	
Video	16	10	7	Desplazamiento vertical descendente (Scroll down).	
Video	16	10	8	Lee caracter y atributo.	
Video	16	10	9	Escribe caracter y atributo.	
Video	16	10	A	Escribe caracter.	
Video	16	10	B	Elección de la paleta de color.	
Video	16	10	C	Escribe un pixel.	
Video	16	10	D	Lee un pixel.	
Video	16	10	E	Escribir un caracter como TTY.	
Video	16	10	13	Escribir una cadena de caracteres.	AT
Equipo	17	11	n/a	Obtiene la lista de los equipos periféricos.	
Memoria	18	12	n/a	Obtiene el tamaño de la memoria útil (en Kbytes).	
Disco	19	13	0	Reinicializa el sistema del disco.	
Disco	19	13	1	Obtención del estado del disco.	
Disco	19	13	2	Lee sectores del disco.	
Disco	19	13	3	Escribe sectores del disco.	
Disco	19	13	4	Verifica los sectores del disco.	
Disco	19	13	5	Formatea las pistas del disco.	
Disco	19	13	8	Obtiene los actuales parámetros del controlador.	AT
Disco	19	13	9	Inicializa las tablas de parámetros del disco duro.	AT
Disco	19	13	A	Lectura de sectores largos.	AT
Disco	19	13	B	Escritura de sectores largos.	AT
Disco	19	13	C	Busca un cilindro.	AT
Disco	19	13	D	Reinicialización alternativa del disco.	AT
Disco	19	13	10	Comprueba si la unidad está lista.	AT

Tabla C.4.- Breve resumen de los servicios de la ROM-BIOS.

Tema	Interrupción Dec	Hex	Servicio (Hex)	Descripción	Modo Específico
Disco	19	13	11	Recalibrado de la unidad.	AT
Disco	19	13	14	Diagnóstico del controlador.	AT
Disco	19	13	15	Obtención del tipo de disco.	AT
Disco	19	13	16	Cambia el estado del disco.	AT
Disco	19	13	17	Indica el tipo de disco.	AT
Puerto serie	20	14	0	Inicializa los parámetros del puerto serie	
Puerto serie	20	14	1	Envía un caracter.	
Puerto serie	20	14	2	Recibe un caracter.	
Puerto serie	20	14	3	Obtiene el estado del puerto serie.	
Cassette	21	15	0	Activa el motor del cassette.	
Cassette	21	15	1	Desactiva el motor del cassette.	
Cassette	21	15	2	Lee bloques de datos.	
Cassette	21	15	3	Escribe bloques de datos.	
Dispositivos	21	15	80	Dispositivo abierto.	AT
Dispositivos	21	15	81	Dispositivo cerrado.	AT
Dispositivos	21	15	82	Finalización del programa del dispositivo.	AT
Dispositivos	21	15	83	Espera de sucesos.	AT
Joystick	21	15	84	Joystick.	AT
Petición del sistema	21	15	85	Pulsación de la tecla SysReq.	AT
Dispositivos	21	15	86	Espera.	AT
Dispositivos	21	15	87	Mueve bloque.	AT
Memoria	21	15	88	Obtención del tamaño de la memoria ampliada.	AT
Memoria	21	15	89	Conmuta a memoria virtual.	AT
Dispositivos	21	15	90	Bucle de dispositivo ocupado.	AT
Dispositivos	21	15	91	Activación del indicador y realización de interrupción.	AT
Teclado	22	16	0	Lee el siguiente caracter del teclado.	
Teclado	22	16	1	Informa si hay algún caracter preparado en el buffer.	
Teclado	22	16	2	Obtiene el estado de Shift.	
Impresora	23	17	0	Envía un byte a la impresora.	
Impresora	23	17	1	Inicializa la impresora.	
Impresora	23	17	2	Obtiene el estado de la impresora.	
BASIC	24	18	n/a	Pasa el control al BASIC.	
Bootstrap	25	19	n/a	Relanza el ordenador.	
Hora	26	1A	0	Lee la cuenta del reloj.	
Hora	26	1A	1	Da valor a la cuenta del reloj.	
Hora	26	1A	2	Lee la hora del reloj de tiempo real.	AT
Hora	26	1A	3	Fija la hora del reloj de tiempo real.	AT
Hora	26	1A	4	Lee la fecha del reloj de tiempo real.	AT
Hora	26	1A	5	Fija la fecha del reloj de tiempo real.	AT
Hora	26	1A	6	Pone la alarma.	AT
Hora	26	1A	7	Quita la alarma.	AT

Tabla C.4.- Breve resumen de los servicios de la ROM-BIOS.

(Tomado de: NORTON,P.,Guía del Programador para el IBM PC, Ediciones Anaya Multimedia, Madrid, 1987)

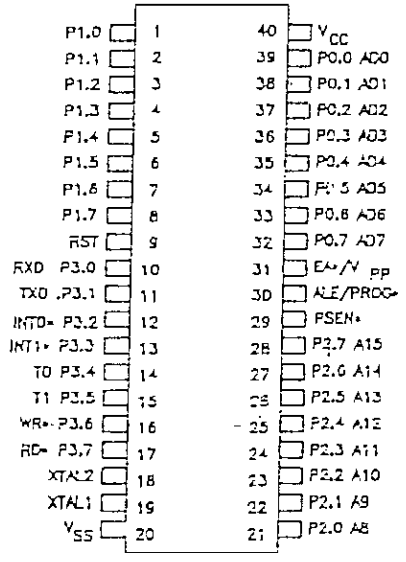
Dallas Semiconductor
Soft Microcontroller

PRELIMINARY
DS5000

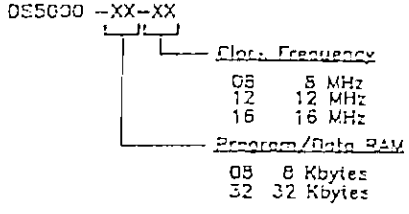
FEATURES

- 8-bit uC adapts to task-at-hand:
  - 8 or 32 Kbytes of high performance nonvolatile RAM for Program and/or RAM for Program and/or Data Memory storage
  - Initial downloading of software in end system via on-chip serial port
  - Capable of modifying its own Program and/or Data Memory in end use
  - 128 internal nonvolatile registers for variable retention
- Crashproof operation:
  - Maintains all nonvolatile resources for 10 years in the absence of V_{CC}
  - Orchestrates orderly shutdown and automatic restart on power up/down
  - Automatic restart on detection of errant software execution
- Software Security Feature:
  - Executes encrypted software to prevent unauthorized disclosure
- On-chip full duplex serial I/O port
- Two on-chip timer/event counters
- 32 parallel I/O lines
- Compatible with industry standard 8051 instruction set and pinout

PIN CONNECTIONS



ORDERING INFORMATION



DESCRIPTION

The DS5000 is a high performance 8-bit CMOS microcontroller that offers "softness" in all aspects of its application. This is accomplished through the comprehensive use of nonvolatile technology to preserve all information in the absence of system V_{CC}. The entire Program/Data Memory space is implemented using high speed, nonvolatile

static CMOS RAM. Two memory size versions are available which offer either 8 Kbytes or 32 Kbytes of NVRAM for Program/Data storage. Furthermore, internal data registers and key configuration registers are also nonvolatile.

A major benefit resulting from its nonvolatility is that the Soft Microcontroller allows Program Memory to be changed at any time, even after the device has been installed in the end system. Additionally, the size of the Program and Data Memory areas in the embedded RAM is variable and can be set either when the application software is initially loaded or by the software itself during execution.

Initial loading of the application software into the DS5000 is possible from either a parallel or serial interface to a host system. This function allows initialization of the nonvolatile areas of the device including Program/Data RAM and the configuration parameters. Serial loading uses the on-chip serial I/O port to accept incoming data from a host computer with an RS232 port, such as a PC-based development system. Not only is it possible to initially boot via the serial port in the end system but any subsequent software reloading can be made at will during system operation without the need for removal of the device.

The softness also provides the ultimate in adaptive system design by allowing either the Data RAM or the Data Registers to retain information in the absence of V_{cc}. As a result, a virtually unlimited number of variables and/or data tables can be updated and maintained over the life of the product, as opposed to their being lost during a power fluctuation. This capability allows software to be developed which updates variables and data tables to reflect the cumulative knowledge of the control system from the time that it was put into service. Consequently, control systems may be given the ability to learn from experience and react by altering processing steps in response to operating conditions which change over extended periods of time.

The DS5000 Soft Microcontroller incorporates control functions which provide crashproof operation when system power is momentarily disrupted, or removed entirely. These functions include the Power Fail Warning interrupt, Automatic Power Down, and Power On Restart. The Power Fail Warning interrupt provides an early warning of a potential power failure so that the operational state of the system may be stored prior to a complete removal of system V_{cc}. The Automatic Power Down feature causes all nonvolatile resources to be sustained at low current from the embedded lithium energy source while system power is removed. When V_{cc} voltage is applied once again, the processor is automatically restarted with an internal flag set indicating that a Power On sequence has just been performed. Regardless of whether the power merely fluctuates or is absent for years, upon its return the Soft Microcontroller has the ability to resume execution when power is reapplied as if the power failure had not occurred at all.

The Soft Microcontroller's tolerance of power cycling provides an alternative for battery powered hand-held systems which typically drain their batteries during periods of idle time when processing is not being performed. On/off power cycling can be employed to cause such systems to consume battery power only during processing to ensure a dramatic reduction of the overall power dissipation.

The DS5000 also provides extensive software security with its unique on-chip software encryption logic. This feature prevents un

authorized individuals from reading and disassembling Program/Data RAM. When activated, the device loads and executes the software in an encrypted form, rendering the contents of the RAM and the execution of the program unintelligible to the outside observer. The encryption algorithm uses an internally stored and protected 40-bit key which is programmed by the user. Any attempt to discover the key value results in its erasure, rendering the contents of the Program/Data RAM useless. In this manner, the investment represented by the resident software is protected from piracy.

The DS5000 incorporates these unique functions in a device which is instruction set and pin compatible with the Industry standard 8051 microcontroller architecture. Development work for new designs based on the DS5000 may be performed utilizing existing development tools and software packages which support the 8051 architecture.

The DS5000 also provides a full complement of I/O functions including two 16-bit event counter/timers, a full duplex serial I/O port capable of asynchronous or synchronous operation, 32 parallel I/O lines, and a watchdog timer. If additional external memory is desired beyond the embedded Program/Data RAM, 18 parallel I/O lines may be assigned to serve the Expanded Bus function.

PIN DESCRIPTION

NOTE: All inverted signal names are denoted with a asterisk (*) as a suffix to the signal name (e.g. INT0*). This convention is followed throughout this document.

V_{cc}, GND	Power Supply inputs.
P0.7-P0.0	Port 0: Bidirectional I/O; open drain These pins also serve the function of:
AD7-AD0	Address/Data Bus: Bidirectional
P1.7-P1.0	Port 1: Bidirectional I/O
P2.7-P2.0	Port 2: Bidirectional I/O These pins also serve the function of:
A15-A8	Address Bus: Outputs
P3.7-P3.0	Port 3: Bidirectional I/O Each of the pins on Port 3 may be selected to serve an alternate function; as described below:
RD* (P3.7)	Expanded Data Memory Read Strobe: Output; active low

WR* (P3.6)	Expanded Data Memory Write Strobe: Output; acti low
T1,T0 (P3.5,P3.4)	Timer/Counter pins: Inputs; active high
INT1*, INT0* (P3.3,P3.2)	External Interrupt pins: Inputs; active low
TXD (P3.1)	Transmit Data: Output
RXD (P3.0)	Receive Data: Input
RST	Reset:. Input; active high
ALE (PROG*)	Address Latch Enable: Output; active high (or Program Byte Enable: Input; active low)
PSEN*	Program Store Enable: Output; active low
EA* (VPP)	External Access Enable: Input; active low (or VPP programming voltage input)
XTAL1, XTAL2	Crystal inputs

INSTRUCTION SET

The DS5000 executes an instruction set which is object code compatible with the industry standard 8051 microcontroller. As a result, software development packages which have been written for the 8051 are compatible with the DS5000 including cross-assemblers, high-level language compilers, and debugging tools.

A complete description for the DS5000 instruction set is available in the DS5000 User's Guide (part # DS5000G).

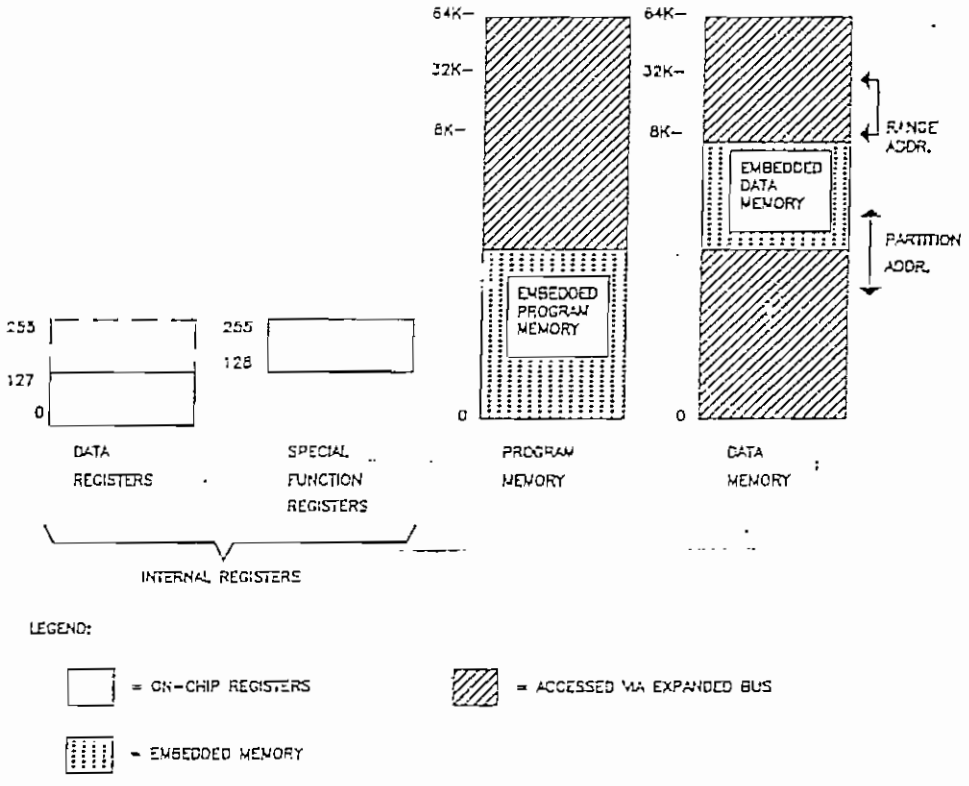
MEMORY ORGANIZATION

Figure 1 illustrates the address spaces which are accessed by the DS5000. As illustrated in the figure, separate address spaces exist for Program and Data Memory.

Since the basic addressing capabilit machine is 16-bits, a maximum of 64 K Program Memory and 64 Kbytes Memory can be accessed by the DS50 The 8K or 32K byte embedded RAM a be used to contain both Program ar Memory.

The Internal Register space is divided parts: Data Registers and Special F Registers. There are a total of 12 Registers including four 8-byte ba working registers (R0-R7). The Function Registers include the CPU ters as well as registers which control and status information for th gram and Data Memory mapping, non- operation, and on-chip I/O functions.

DS5000 LOGICAL ADDRESS SPACES Figure 1



SPECIAL FUNCTION REGISTERS

There are a total of 23 Special Function Registers which have been implemented in the DS5000. Table 5-1 lists each of these along with their respective addresses, reset values, and functional descriptions.

DS5000 SPECIAL FUNCTION REGISTER MAP Table 1

New or Modified Register	Label	Direct Register Address	Reset Value	Bit address-able	Functional Description
	B	0F0H	00H	X	B Register
	A	0E0H	00H	X	Accumulator
	PSW	0D0H	00H	X	Program Status Word
X	TA	0C7H	055H		Timed Access
X	MCON	0C6H	RT		Memory Control
X	IP	0B8H	00H	X	Interrupt Priority Ctl.
	P3	0B0H	0FFH	X	Port 3 Parallel I/O
	IE	0A8H	00H	X	Interrupt Enable Ctl.
	P2	0A0H	0FFH	X	Port 2 Parallel I/O
	SBUF	099H	??		Serial Data Buffer
	SCON	098H	00H	X	Serial Control
	P1	090H	0FFH	X	Port 1 Parallel I/O
	TH1	08DH	00H		Timer 1 High Byte
	TH0	08CH	00H		Timer 0 High Byte
	TL1	08BH	00H		Timer 1 Low Byte
	TL0	08AH	00H		Timer 0 Low Byte
	TMOD	089H	00H		Timer Mode Select
	TCON	088H	00H	X	Timer Control
X	PCON	087H	RT		Power Control
	DPH	083H	00H		Data Pointer High Byte
	DPL	082H	00H		Data Pointer Low Byte
	SP	081H	07H		Stack Pointer
	P0	080H	0FFH	X	Port 0 Parallel I/O

NOTES:

?? indicates that the register value is indeterminate on reset.

RT indicates that the initialization performed on the register is dependent on the type of the reset.

The Power Control (PCON), Interrupt Priority (IP), Memory Control (MCON), and Timed Access (TA) registers represent modifications from the 8051 implementation, as denoted in the above table. The following is a detailed summary of these registers.

POWER CONTROL REGISTER

Label: PCON Register Address: 087H

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	POR	PFW	WTR	EPFW	EWT	STOP	IDL

Bit Description:

PCON.7 SMOD
"Double Baud Rate": When set to a 1, the baud rate is doubled when the serial port is being used in modes 1, 2, or 3.
Initialization: Cleared to a 0 on any reset.
Read Access: Can be read normally at any time.
Write Access: Can be written normally at any time.

PCON.6 POR
"Power On Reset": Indicates that the previous reset was initiated during a Power On sequence.
Initialization: Cleared to a 0 when a Power On Reset occurs. Remains at 0 until it is set to a 1 by software.
Read Access: Can be read normally at any time.
Write Access: Can be written only by using the Timed Access Register.

PCON.5: PFW
"Power Fail Warning": Indicates that a potential power failure is in progress. Set to 1 whenever V_{CC} voltage is below the V_{PFW} threshold. Cleared to a 0 immediately following a read operation of the PCON register. Once set, it will remain set until the read operation occurs regardless of activity on V_{CC}.
Initialization: Cleared to a 0 during a Power On Reset.
Read Access: Can be read normally anytime.
Write Access: Not writeable.

PCON.4: WTR
"Watchdog Timer Reset" Set to a 1 when a reset was issued as a result of a Watchdog Timer timeout. Cleared to 0 immediately following a read of the PCON register
Initialization: Set to a 1 after a Watchdog Timeout Reset. Cleared to a 0 on a No-V_{LI} Power on Reset. Remains unchanged during other types of resets.
Read Access: May be read normally anytime.
Write Access: Cannot be written

PCON.3: EPFW
"Enable Power Fail Interrupt": Used to enable or disable the Power Fail Interrupt. When EPFW is set to a 1 it will be enabled; it will be disabled when EPFW is cleared to a 0.
Initialization: Cleared to a 0 on any type of reset.
Read Access: Can be read normally anytime.
Write Access: Can be written normally anytime.

PCON.2: EWT
"Enable Watch-dog Timer" Used to enable or disable the Watchdog Timeout Reset. The Watch dog Timer is enabled if EWT is set to a 1 and will be disabled if EWT is cleared to a 0.
Initialization: Cleared to a 0 on a No-V_{LI} Power on Reset. Remains unchanged during other types of resets.
Read Access: May be read normally anytime.
Write Access: Can be written only by using the Timed Access register.

PCON.1: STOP
"Stop": Used to invoke the Stop Mode. When set to a 1 program execution will terminate immediately and Stop Mode operation will commence. Cleared to a 0 when program execution resumes following a hardware reset.
Initialization: Cleared to a 0 on any type of reset
Read Access: Can be read anytime.
Write Access: Can be written only by using the Timed Access register.

PCON.0: IDL
"Idle": Used to invoke the Idle Mode. When set to a 1 program execution will be halted and will resume when the Idle bit is cleared to 0 following an interrupt or a hardware reset.
Initialization: Cleared to 0 on any type of reset or interrupt.
Read Access: Can be read normally anytime.
Write Access: Can be written normally anytime.

INTERRUPT PRIORITY REGISTER

Label: IP — Register Address: 0B8H

D7	D6	D5	D4	D3	D2	D1	D0
RWT	–	–	PS	PT1	PX1	PT0	PX0

Bit Description:

IP.7: RWT
"Reset Watch-Timer": When set to a 1 the Watchdog Timer count will be reset, and counting will begin again. The RWT bit will then automatically be cleared again to 0. Writing a 0 into this bit has no effect.
Initialization: Cleared to a 0 on any reset.
Read Access: Cannot be read.
Write Access: Can be written only by using the Timed Access register.

All of the following bits are read/write at any time and are cleared to 0 following any hardware reset.

IP.4: PS
"Serial Port Priority": Programs Serial Port interrupts for high priority when set to 1. Low priority is selected when cleared to 0.

IP.3: PT1
"Timer 1 Priority": Programs Timer 1 interrupt for high priority when set to 1. Low priority is selected when cleared to 0.

IP.2: PX1
"Ext. Int. 1 Priority": Programs External Interrupt 1 for high priority when set to 1. Low priority is selected when cleared to 0.

IP.1: PT0
"Timer 0 Priority": Programs Timer 0 interrupt for high priority when set to 1. Low priority is selected when cleared to 0.

IP.0: PX0
"Ext. Int. 0 Priority": Programs External Interrupt 0 for high priority when set to 1. Low priority is selected when cleared to 0.

MEMORY CONTROL REGISTER

Label: MCON Register Address: 0C6H

D7	D6	D5	D4	D3	D2	D1	D
PA3	PA2	PA1	PA0	RA32/8	ECE2	PAA	—

Bit Description:

MCON.7-4: PA3-0

"Partition Address": Used to select the starting address of Data Memory in Embedded R. Program space lies below the Partition Address.

Selection:

PA3	PA2	PA1	PA0	Partition Address
0	0	0	0	0000H
0	0	0	1	0800H
0	0	1	0	1000H
0	0	1	1	1800H
0	1	0	0	2000H
0	1	0	1	2800H
0	1	1	0	3000H
0	1	1	1	3800H
1	0	0	0	4000H
1	0	0	1	4800H
1	0	1	0	5000H
1	0	1	1	5800H
1	1	0	0	6000H
1	1	0	1	6800H
1	1	1	0	7000H *
1	1	1	1	8000H *

* A 4 Kbyte Increment (not 2 Kbytes) in the Partition Address takes place between bit field values 1110B and 1111B.

Initialization: Set to all 1's on a No V_{LI} Power On Reset or when the Security Loc. t cleared to a 0 from a previous 1 state. These bits are also set to al. 1 when any attempt is made to have them cleared to all 0's with the SL set to a 1 (illegal condition).

Read Access: May be read anytime.
Write Access: PAA bit must = 1 in order to write PA3-0. Timed Access is not required to write to PA3-0 once PAA = 1.

MCON.3: RA32/8
"Range Address": Sets the maximum usable address in Embedded Memory.
RA32/8 = 0 sets Range Address = 1FFFH (8K)
RA32/8 = 1 sets Range Address = 7FFFH (32K)

Initialization: Set to a 1 during a No V_{LI} Power On Reset and when the Security Lock bit (SL) is cleared to a 0 from a previous 1 state. Remains unchanged on all other types of resets.
Read Access: May be read normally anytime.
Write Access: Cannot be modified by the application software; can only be written during Program Load Mode.

MCON.2: ECE2
"Enable Chip Enable 2": Used to enable or disable the CE2* signal to additional Embedded RAM Data Memory space. This bit should always be cleared to 0 in the DS5000 8 and DS5000 32 versions.
Initialization: Cleared to 0 only during a No V_{LI} Power On Reset.
Read Access: Read normally anytime.
Write Access: Can be written normally at any time.

MCON.1: PAA
"Partition Address Access": Used to protect the programming of the Partition Address select bits. PA3-0 cannot be written when PAA = 0. PAA can be written only via the Timed Access register.
Initialization: PAA is cleared only on a No-V_{LI} Power On Reset
Read Access: PAA may be read anytime.
Write Access: The Timed Access register must be used to perform any type of write operation on the PAA bit

TIMED ACCESS REGISTER

Label: TA Register Address: 0C7H

D7	D6	D5	D4	D3	D2	D1	D0

Bit Description:

TAn.n: (All Timed Access bits)

"Timed Access": Used to invoke a Timed Access procedure required to write to any of the Timed Access protected bits including EWT, RWT, STOP, PAA. Timed Access is activated by three sequential write operations as in the example shown below:

```
MOV   0C7H, 0AAH      ; Write 0AAH to TA register
MOV   0C7H, 055H      ; Write 055H to TA register
ORL   IP,#80H         ; Reset Watchdog Timer
```

Initialization: Written with the value of 055H following any type of reset.

Read Access: Cannot be read from the application software

PROGRAM LOAD MODES

The Program Load Modes allow initialization of the embedded Program/Data Memory and nonvolatile Internal Registers. This initialization may be performed in one of two ways:

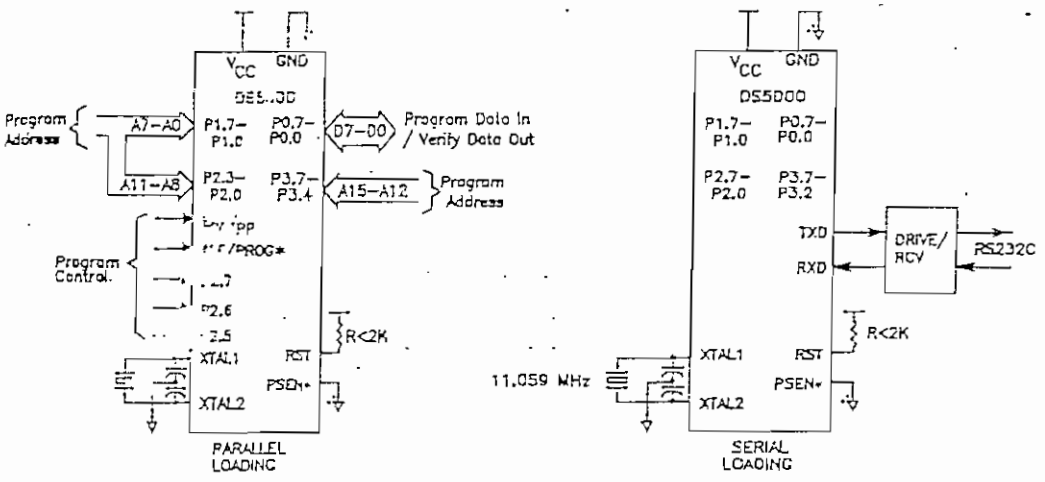
1) Parallel Program Load cycles which perform the initial loading from parallel address/data information presented on the I/O port pins. This mode is timing-set compatible with the 8751H microcontroller programming mode.

2) Serial Program Loading which is capable of performing bootstrap loading of the DS5000. This feature allows the loading of the application program to be delayed until the DS5000 is installed in the end system.

The DS5000 is placed in its Program Load configuration by simultaneously applying a logic 1 to the RST pin and forcing the PSEN* line to a logic 0 level. Immediately following this action, the DS5000 will look for a Parallel Program Load pulse, or a serial ASCII carriage return (0DH) character received at 9600, 2400, 1200, or 300 bps over the serial port.

The hardware configurations used to select these modes of operation are illustrated in Figure 2.

PROGRAM LOADING CONFIGURATIONS Figure 2



The table below summarizes the selection of the available Parallel Program Load cycles. Figure 5 illustrates the timing associated with these cycles.

PARALLEL PROGRAM LOAD CYCLES Table 2

Mode	RST	PSEN*	PROG*	EA*	P2.7	P2.6	P2.5
Program	1	0	0	VPP	1	0	X
Security Set	1	0	0	VPP	1	1	X
Verify	1	X	X	1	0	0	X
Prog Expanded	1	0	0	VPP	0	1	0
Verify Expanded	1	0	1	1	0	1	0
Prog MCON or key registers	1	0	0	VPP	0	1	1
Verify MCON reg	1	0	1	1	0	1	1

The Program Cycle is used to load a byte of data into a register or memory location within the DS5000. The Verify Cycle is used to read this byte back for comparison with the originally loaded value to verify proper loading. The Security Set Cycle may be used to enable and disable the Software Security feature of the DS5000. One may also enter bytes for the MCON register or for the 5 encryption registers using the Program MCON cycle. When using this cycle, the absolute register address must be presented at Ports 1 and 2 as in the normal Program cycle (Port 2 should be 00H). The MCON contents can likewise be verified using the Verify MCON cycle.

When the DS5000 first detects a Parallel Program Strobe pulse or a Security Set Strobe

pulse while in the Program Load Mode following a Power On Reset, the internal hardware of the DS5000 is initialized so that an existing 4 Kbyte program can be programmed into a DS5000 with little or no modification. This initialization automatically sets the Range Address for 8 Kbytes and maps the lowest 4 Kbyte bank of Embedded RAM as Program Memory. The next 4 Kbytes of Embedded RAM are mapped as Data Memory.

In order to program more than 4 Kbytes of program code, the Program/Verify Expanded cycles can be used. Up to 32 Kbytes of program code can be entered and verified. Note that the expanded 32Kbyte Program/Verify cycles take much longer than the normal 4 Kbyte Program/Verify cycles.

A typical parallel loading session would follow this procedure. First, set the contents of the MCON register with the correct range and partition only if using expanded programming cycles. Next, the encryption registers can be loaded to enable encryption of the program/data memory (not required). Then, program the DS5000 using either normal or expanded Program cycles and check the memory contents using Verify cycles. The last operation would be to turn on the security lock feature by either a Security Set cycle or by explicitly writing to the MCON register and setting MCON.0 to a 1.

SERIAL LOAD MODE

The Serial Program Load Mode is the eas fastest, most reliable, and most comp method of initially loading application soft into the DS5000's nonvolatile RAM. Com nication can be performed over a stan asynchronous serial communications port typical application would use a simple RS2 serial interface to program the DS5000 i final production procedure. The hardware figuration which is required for the Serial gram Load Mode is illustrated in Figure 2. *pins 2.7 and 2.6 must be either open or pu high to avoid placing the DS5000 in a par load cycle.* Although an 11.0592 MHz cryst shown in Figure 2, a variety of crystal frequ cies and loader baud rates are supported wi are shown in Table 3. The serial loade designed to operate across a three wire in face from a standard UART. The rece transmit and ground wires are all that necessary to establish communication with DS5000.

The Serial Loader implements an easy-to-u command line interface which allows an ap cation program in an Intel Hex representat to be loaded into and read back from t device. Intel Hex is the typical format wh existing 8051 cross-assemblers output. T serial loader responds to 11 single charac commands which are summarized below:

COMMAND	FUNCTION
C	Return CRC-16 checksum of embedded RAM
D	Dump Intel Hex File
F	Fill Embedded RAM block with constant
K	Load 40-bit Encryption Key
L	Load Intel Hex File
R	Read MCON register
T	Trace (Echo) incoming Intel Hex data
U	Clear Security Lock
V	Verify Embedded RAM with incoming Intel Hex
W	Write MCON register
Z	Set Security Lock

SERIAL LOADER BAUD RATES FOR DIFFERENT CRYSTAL FREQUENCIES*
Table 3

Crystal freq (MHz)	Baud Rate				
	300	1200	2400	4800	9600
16.000000		Y	Y		
15.000000		Y	Y	Y	Y
14.318180		Y	Y	Y	Y
12.000000		Y	Y	Y	
11.059200	Y	Y	Y	Y	Y
11.000000	Y	Y	Y	Y	Y
10.000000		Y	Y	Y	
9.216000	Y	Y	Y	Y	Y
8.000000		Y			
7.372800	Y	Y	Y	Y	Y
6.144000	Y	Y	Y		
6.000000	Y	Y	Y		
5.990400	Y	Y	Y		
5.120000	Y	Y	Y		
5.068800	Y	Y	Y		
5.000000	Y	Y	Y		
4.915200	Y	Y	Y		
4.608000	Y	Y	Y	Y	
4.433620	Y	Y	Y	Y	
4.194300	Y				
4.096000	Y				
4.032000	Y				
3.579545	Y	Y	Y	Y	Y
2.457600	Y	Y			
2.000000	Y				
1.843200	Y	Y	Y	Y	Y

* Y indicates that the baud rate for that particular crystal is supported by the DS5000 serial loader auto-baud detection scheme.

POWER MANAGEMENT

The DS5000 is implemented using CMOS circuitry for low power consumption during full operation. Two software initiated modes are available for further power reduction for times when processing is not required and V_{CC} is at normal operating voltage. These are the Idle and Stop Modes. In addition, internal control circuitry automatically places the the DS5000 in its Data Retention Mode in the absence of V_{CC}.

The c hip nonvolatile control circuitry monitors the V_{CC} for three below nominal operating volt (Figure 3). When the voltage drops be.. .. the Power Fail Warning threshold (V_{PFW}) an interrupt will be generated to signal the procassor of an impending power fail condition. This is to allow time for a service routine to save the operational state of the microcontroller prior to the V_{CC} dropping below the V_{CCmin} threshold. When this occurs, processor operation is automatically terminated by internally halting the clock after the entire circuit has been made ready for the Data Retention Mode. Finally, once V_{CC} voltage drops below the Lithium cell voltage threshold (V_{LI}) power from the embedded lithium cell is applied to place the device in its Data Retention Mode.

When V_{CC} voltage is again applied to the system, an internal Power On Reset cycle is executed without the need for any external components on the RST pin. In addition, internal status is available to distinguish the Power On Reset from other types of resets.

SOFTWARE SECURITY

The Software Security feature is implemented using Address and Data Encryptor circuitry which is present on the DS5000 die. Operation of the Software Security feature is performed by mainpulation of the 40-bit Encryption Key word and the Security Lock bit while in one of the Program Load modes. Encryption operation is first intiated by loading the 40-bit En tion Key word.

When Software Encryption Operation is i fect and the Security Lock is disabled, application software may be initially stor an encrypted form during the initial loadi the device using one of the Program l modes. As the loading is performed, the Encryptor logic transforms the opcode, and, and data byte defined at each me location defined by the software.. Simi the Address Encryptor translates the "log address of each location into an encry address at which the byte is actually st Although each encryptor uses its own rithm for encrypting data, both depend or 40-bit key word which is contained in Encryption Key registers (EK0-4).

As long as the Security Lock remains abled, the actual unencrypted contents o embedded Program/Data RAM may be back for verification while in the Program L mode. Once the contents have been ver the final action performed during the Prog Load Mode should be the enabling of Security Lock bit. From this point on it wi impossible to read back the unencry contents of the Program/Data RAM or contents of the Encryption Key registers

When the application software is execu the Address and Data Encryptors provide opcodes, operands, and data to the CPU execution of the application software can place as normal. This action also takes pl in real time so that no additional delays imposed on the execution time of the ware. Thus, the Software Encryption Op tion is transparent to the application softw

The Software Encryption Operation is disa and the contents of the Encryption Key re

ters are automatically erased whenever the Security Lock bit is cleared to a 0 from a previous 1 condition. This action renders the contents of the embedded Program/Data RAM useless, so that the application software can no longer be correctly interpreted by the DS5000 CPU. Although the contents of the Program/Data RAM can at this point be read back in a Program Load Mode, they cannot be de-encrypted since the original 40-bit key word has been lost.

ADDITIONAL INFORMATION

A complete description for all operational aspects of the DS5000, including an instruction set description, timing details, and electrical specifications is available in the DS5000 User's Guide (part # DS5000G).

DEVELOPMENT SUPPORT

Dallas Semiconductor offers two kit packages for developing and testing user code. The DS5000K Evaluation Kit allows the user to download Intel hex formatted code directly to the DS5000 from a PC-XT/AT or compatible computer. The kit consists of a DS5000-32-08, an interface pod, demo software, and an RS-232 connector that attaches to the COM1 or COM2 serial port of a PC.

The DS5000DK Development Kit consists of an assembler and a real-time in-circuit emulator that interfaces to a PC-XT/AT or compatible. See the DS5000K and DS5000DK data sheets for further details.

SELECTED ELECTRICAL CHARACTERISTICS

The following are selected electrical operating characteristics of the DS5000. A full set of electrical characterics is available in the DS5000 User's Guide.

ABSOLUTE MAXIMUM RATINGS*

Voltage on any pin relative to ground	-0.1 to 7.0V
Operating Temperature	-0 deg. to 70 deg. C
Storage Temperature	-40 deg. C to +70 deg. C
Soldering Temperature	260 deg. C for 10 sec.

* This is a stress rating only and functional operation of the device at these or any other conditions outside of those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

D.C. CHARACTERISTICS

(T_A = 0 deg. C to 70 deg. C; V_{CC} = 5V ±10%)

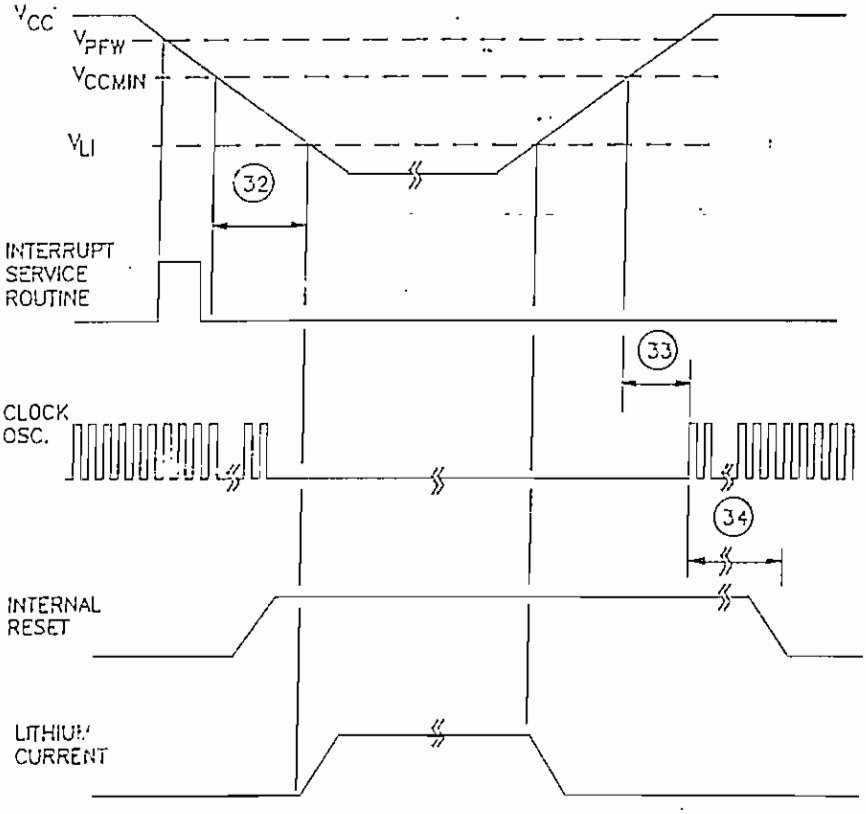
PARAMETER	SYM.	MIN.	TYP.	MAX.	UNITS	NOTES
Stop Mode Current	I_{SM}			80	uA	4
Power Fail Warning Voltage	V_{PFW}	4.15	4.6	4.75	V	
Minimum Operating Voltage	V_{CCmin}	4.05	4.5	4.65	V	
Lithium Supply Voltage	V_{LI}			3.3	V	
Programming Supply Voltage (Parallel Program Mode)	V_{PP}	12.5		13.0	V	
Program Supply Current	I_{pp}		9.2	15	mA	
Operating Current DS5000 8 DS5000 32	I_{CC}		20 25	43.2 48.2	mA	
Idle Mode Current	I_{CC}			6.2	mA	

A.C. CHARACTERISTICS
POWER CYCLING TIMING

(T_A = 0°C to 70°C; V_{CC} = 5V ± 10%)

#	PARAMETER	SYMBOL	MIN.	MAX.	UNITS
32	Slew rate from V_{CCmin} to V_{LImax}	t_F	40		us
33	Crystal start up time	t_{CSU}	(note 5)		
34	Power On Reset Delay	t_{POR}	$2150^{*}4t_{CLK}$		us

POWER CYCLING TIMING DIAGRAM Figure 3



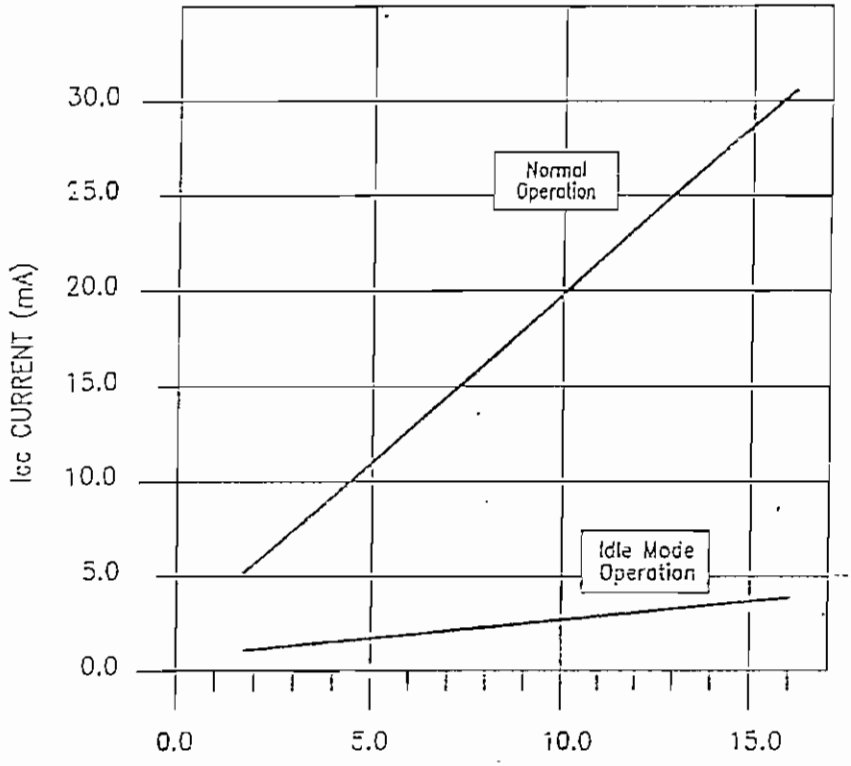
A.C.CHARACTERISTICS (T_A= 0°C to 70°C; V_{CC}= 5V ±10%)
PARALLEL PROGRAM LOAD TIMING:

#	PARAMETER	SYMBOL	MIN.	MAX.	UNITS
40	Oscillator Frequency	$1/t_{CLK}$	1.0	12.0	MHz
41	Address Setup to PROG* low	t_{AVPRL}	0		t_{CLK}
42	Address Hold After PROG* high	t_{PRHAV}	0		t_{CLK}
43	Data Setup to PROG* low	t_{DVPRL}	0		t_{CLK}
44	Data Hold After PROG* low	t_{PRHDV}	0		t_{CLK}
45	P2.7, 2.6, 2.5 Setup to V_{PP}	t_{P2XVP}	0		t_{CLK}
46	V_{PP} Setup to PROG* low	t_{VPHPRL}	0		t_{CLK}
47	V_{PP} Hold After PROG* low	t_{PRHVPL}	0		t_{CLK}
48	PROG* Width low	t_{PRW}	2400		t_{CLK}
49	Data Output from Address Valid	t_{AVDV}		48 1800*	t_{CLK} t_{CLK}
50	P2.7, 2.6 active to data valid	t_{DVP2XA}		48 1800*	t_{CLK} t_{CLK}
51	Data Hold after P2.7, 2.6 inactive	t_{P2XHDI}	0	48 240*	t_{CLK} t_{CLK}
52	Delay to Reset/PSEN* active after Power On	t_{PORPV}	26304		t_{CLK}
53	Reset/PSEN* active (or Verify inactive) to V_{PP} high		1200		t_{CLK}

54	V_{pp} inactive (between Program cycles)		1200		t_{CLK}
55	Verify active time	t_{VFT}	48 2400		t_{CLK} t_{CLK}

* Second set of numbers refer to expanded memory programming up to 32Kbytes.

DS5000 Icc VS FREQUENCY Figure 4



Normal operation is measured using

1) External crystals on XTAL1 and 2
2) All port pins disconnected
3) RST = 0 Volts and EA = VCC.
4) Port performing endless loop writing to internal memory.

Idle mode operation is measured using

1) External clock source at XTAL1; XTAL 2 floating
2) All port pins disconnected
3) RST = 0 Volts and EA = VCC
4) Port set in IDLE mode by software.

NOTES:

1. All voltages are referenced to ground.

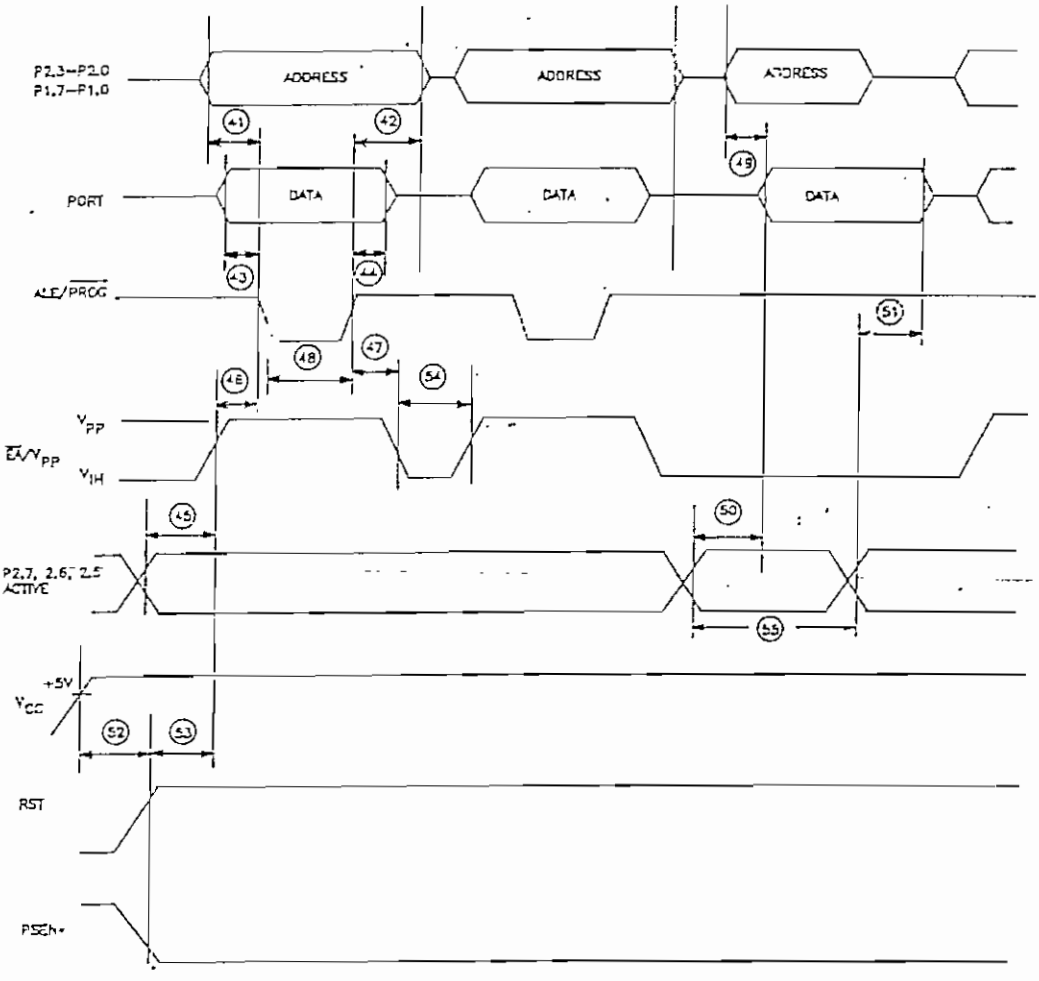
2. Maximum operating I_{CC} is measured with all output pins disconnected; XTAL1 driven with t_{CLKR}, t_{CLKF} = 10 ns, V_{IL} = 0.5V, V_{IH} = 4.5V; XTAL2 disconnected; $\overline{EA}$=RST=PORT0=V_{CC}.

3. Idle Mode I_{CC} is measured with all output pins disconnected; XTAL1 driven with t_{CLKR}, t_{CLKF} = 10 ns, V_{IL} = 0.5V, V_{IH} = 4.5V; XTAL2 disconnected; $\overline{EA}$=RST=PORT0=V_{CC}.

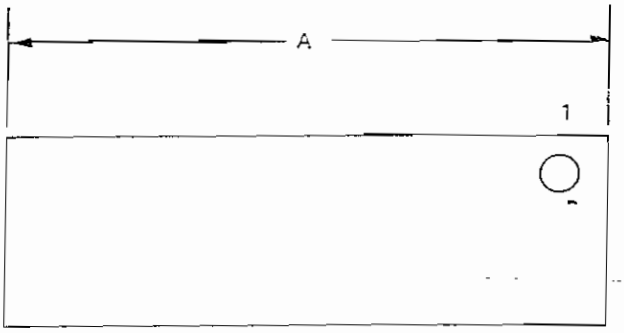
4. Stop Mode I_{CC} is measured with all output pins disconnected; $\overline{EA}$=PORT0=V_{CC}; XTAL2 not connected; RST = V_{SS}.

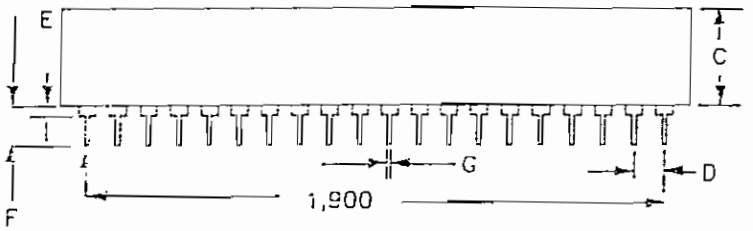
5. Crystal start up time is the time required to get the mass of the crystal into vibrational motion from the time that power is first applied to the circuit until the first clock pulse is produced by the on-chip oscillator. The user should check with the crystal vendor for a worst case spec on this time.

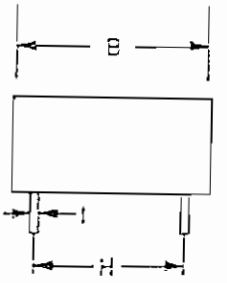
PARALLEL PROGRAM LOAD TIMING Figure 5



DIM.	INCHES	
	MIN.	MAX.
A	2.080	2.100
B	.680	.700
C	.290	.310
D	.090	.110
E	.040	.060
F	.165	.185
G	.016	.020
H	.590	.610
I	.009	.012







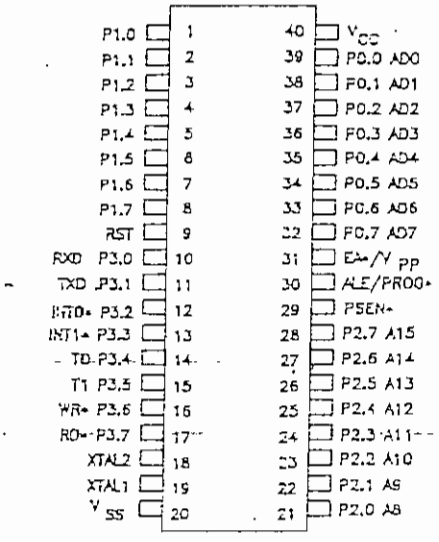
Dallas Semiconductor
Time Microcontroller

PRELIMINARY
DS5000T

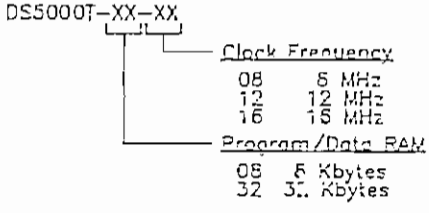
FEATURES

- DS5000 Soft Microcontroller with embedded clock/calendar
- Internal lithium cell preserves clock function in the absence of V_{CC}
- Permits logging of events with time and date stamp
- 8 or 32 Kbytes of embedded nonvolatile program/data RAM
- Program loading via on-chip full-duplex serial port
- User-selectable program/data memory partition
- All 4 ports available for system control
- Resident encryptor protects program from piracy
- Power sequencer and watchdog timer help ensure crash-proof operation
- Compatible with industry standard 8051 instruction set and pinout
- Clock accuracy is better than 2 min/month @25 deg C

PIN CONNECTIONS



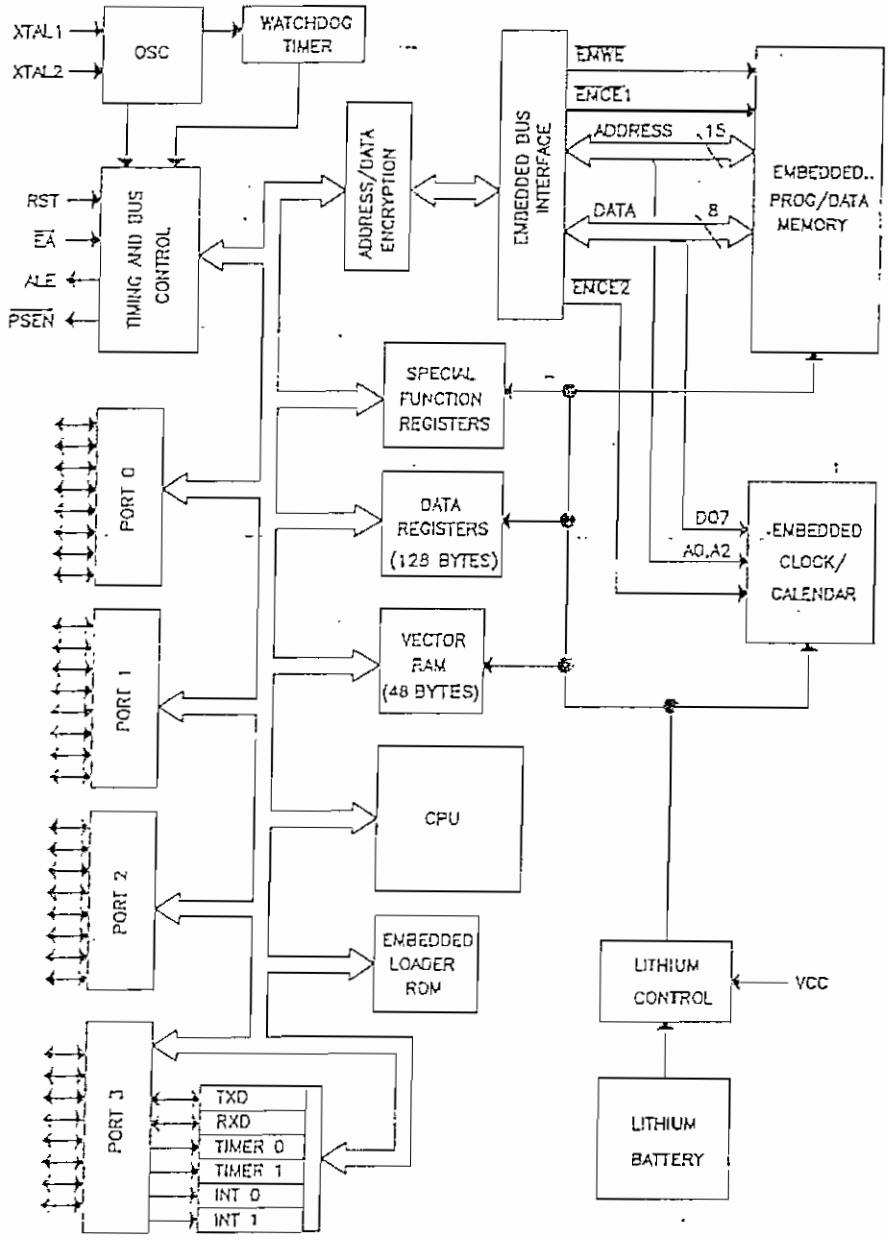
ORDERING INFORMATION



DESCRIPTION

The DS5000T Time Microcontroller offers all the features of the DS5000 with the added benefit of an embedded real-time clock/calendar function. The clock function itself is accessed as though it were a part of the embedded Data RAM so that the 32 I/O pins are free for the application use. With this feature, new and existing microcontrolled systems can now log events, schedule activities and time operations. The combination of DS5000T's "soft" features together with a real-time clock/calendar provides a powerful controller that adapts to the needs of time-driven applications.

DS5000T TIME MICROCONTROLLER BLOCK DIAGRAM Figure 1



ECC COMPARISON REGISTER DESCRIPTION Figure 2

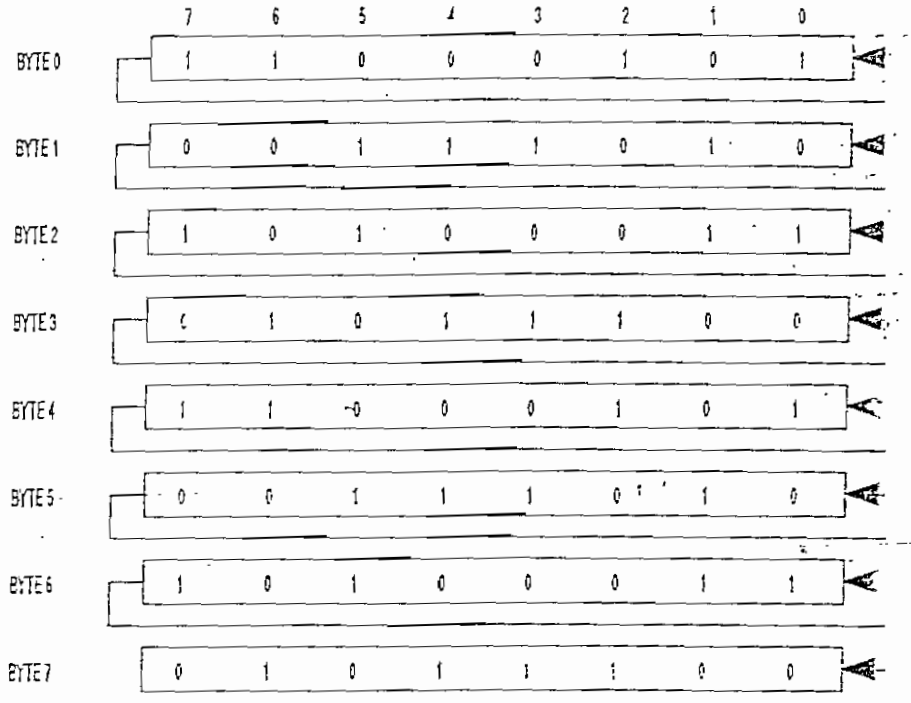


Figure 1 is the block diagram of the DS5000T, illustrating how the embedded program/data memory and the embedded clock/calendar (ECC) are connected to the CPU. The time parameters available are hundreths of seconds, seconds, minutes, hours, days, dates, months and years. Entry to the time function is identical to the serial communication method used by the DS1215 TimeChip. Basically, the time function appears as a slice of embedded data memory that is enabled by the ECE2 bit in the MCON special function register. This bit must be set to a logic 1 in order to select this new data memory map so that the time f appears to the CPU as simply a re memory location.

Aside from the clock/calendar fu the DS5000T operates just lik DS5000 Soft Microcontroller so tha ing DS5000 code is upward com with the DS5000T. Features su nonvolatile data/program me watchdog timer, power sequence software encryption are all preser the DS5000T. Please refer to the D. data sheet and the User's Guide f details on its operation.

EMBEDDED CLOCK/CALENDAR OPERATION

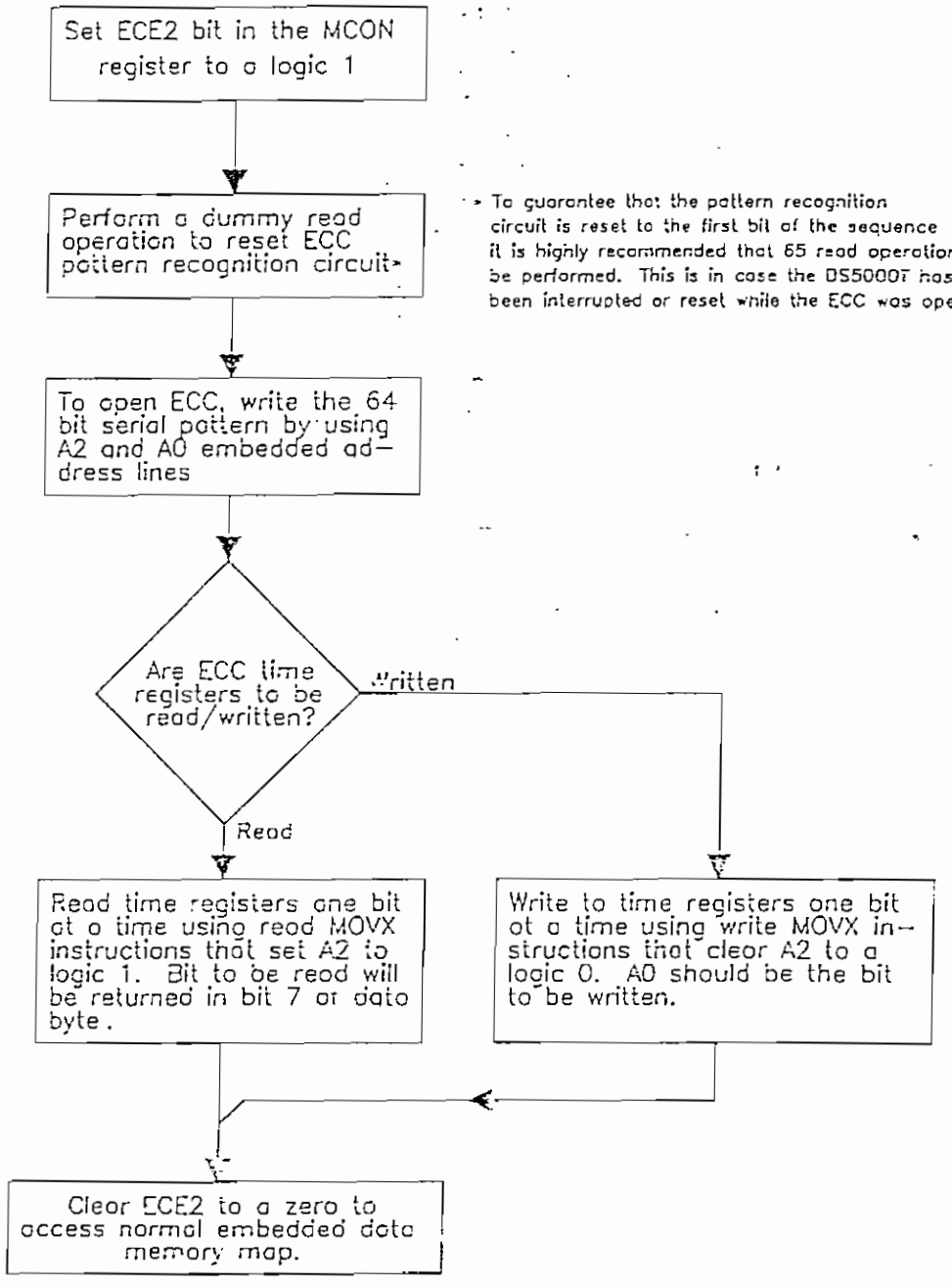
The embedded clock/calendar (ECC) operates much like the DS1215 TimeChip in that a 64 bit serial pattern must be written before time information can be read or written. The ECC has a pattern recognition circuit that checks incoming data to see that if it matches the necessary sequence which is illustrated in Figure 2. After all 64 bits match, the ECC is ready allow the time registers to either be written or read. To reset the pattern recognition circuit so as to look for bit 1 of the sequence, a read cycle must be performed. Then the 64 bits of the recognition sequence must be written to the ECC. If any bit should mismatch, the pattern recognition circuit will halt its operation and only a read cycle will be able to reset this circuit.

Communication to the ECC involves the use of the memory map enabled by bit ECE2 in the MCON register. Once this bit is set to a 1, all embedded memory read/write operations are directed to the ECC address space instead of the normal 8/32 Kbytes of embedded program/data memory. Actual read/write operations use embedded address lines A0 and A2 of the ECE2 memory map. To write a data bit to ECC, a MOVX instruction that forces A2 low and A0 to the state of the bit must be performed. *All other address lines should be set to a logic 0.* Address line A2 can be thought of as the write enable to the ECC and A0 as the write bit. Therefore, to write the 64 bits of the pattern recognition sequence, 64 MOVX instructions must be executed.

To read a data bit from the ECC once the 64-bit pattern has been entered, a read MOVX instruction (MOVX A,@Ri or MOVX A,@DPTR) must be executed that sets A2 to a 1. The data bit desired will be then be returned in bit 7 of the accumulator. Therefore, to retrieve the 8 bytes of time information in the ECC, 64 read MOVX instructions must be executed.

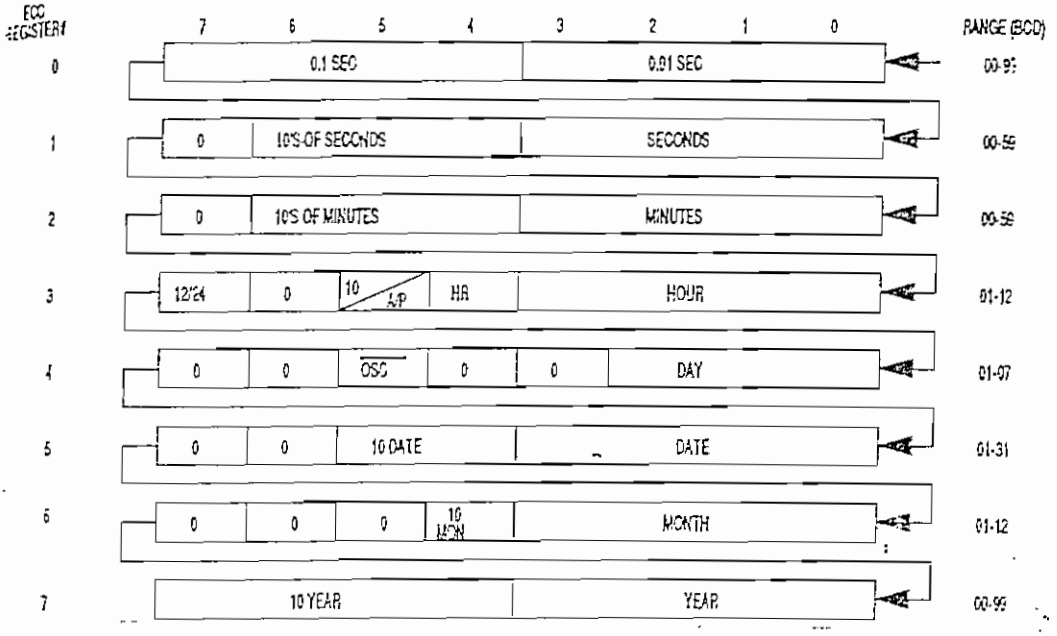
A flowchart is shown in Figure 3 which summarizes how to access the ECC for time retrieval and modification. Also, Appendix 1 lists a program called DEMODS5T which contains sample subroutines for communicating with the ECC that can be downloaded into the DS5000T (via the DS5000K for example). When DEMODS5T is run, a dumb terminal can retrieve the time from the DS5000T through the serial port (RXD,TXD). Please contact the Marketing Department for any assistance.

ECC REGISTER ENTRY FLOWCHART Figure 3



* To guarantee that the pattern recognition circuit is reset to the first bit of the sequence it is highly recommended that 65 read operations be performed. This is in case the DS5000T has been interrupted or reset while the ECC was open.

ECC TIME REGISTERS DESCRIPTION Figure 4



ECC REGISTERS

The time information in the ECC is contained in eight registers that are each 8 bits long. After the 64 bit recognition pattern has been received, data in these registers is accessed one bit at a time which is shown conceptually in Figure 4. It is recommended that data written to or read from the ECC include all 64 bits of the 8 time registers.

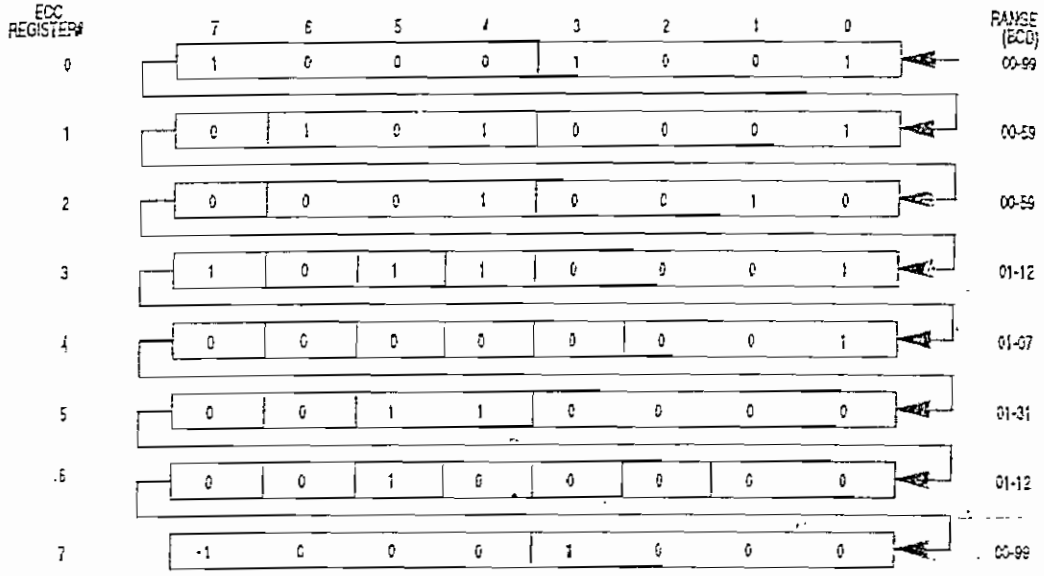
Register data is always in the BCD format except for the hours register (Reg 3) whose format changes depending upon the state of bit 7. If bit 7 is a one, the 12 hour mode is selected and bit 5 of the hours register becomes an AM/PM indicator; PM is indicated by a logic 1 while AM is a logic zero. If bit 7 is a zero, the 24 hour mode is selected and bit 5 becomes the second 10 hour bit (20-23 hours). Figure 5 contains examples that illustrate the content of these registers for different modes and times.

ECC SPECIAL BITS

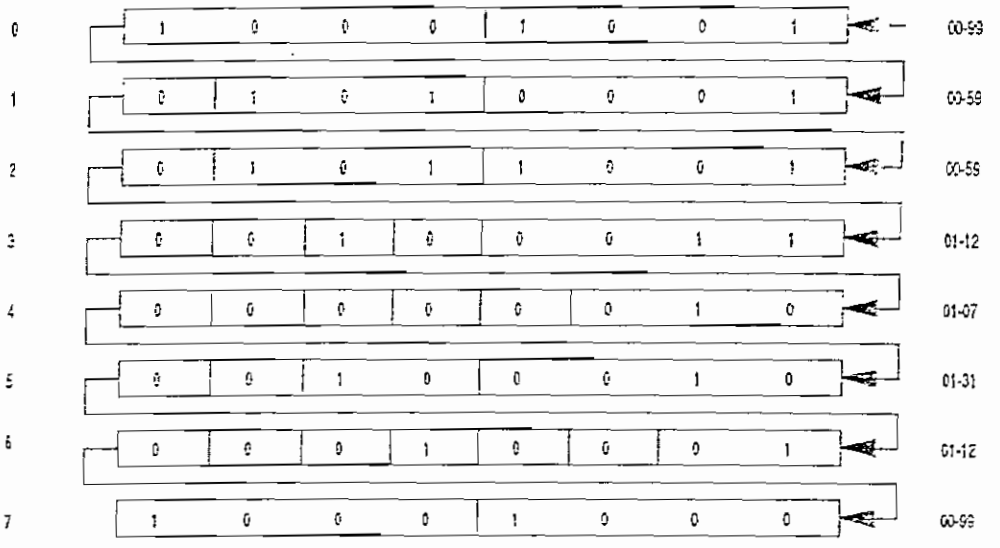
Bit 5 of the days register (REG 4) is the control bit for the ECC micro-power oscillator and it operates similiarly to that in the DS1215. Clearing bit 5 to a logic 0 enables the oscillator for normal operation; setting bit 5 to a logic 1 disables the oscillator and halts the ECC timekeeping. It is recommended that bit 5 always be cleared to 0.

Bit 4 of the days register does not affect ECC operation and can be set to any state. Register locations shown as logic 0's in Figure 6 will always return a 0 when being read. Write operations to these bit locations are ignored by the ECC and have no effect on its operation.

ECC TIME REGISTER EXAMPLES Figure 5



The time indicated is 11 o'clock PM, 12 minutes, 51.69 seconds.
The date indicated is Sunday, October 30th, 1988.



The time indicated is 2300 hour, 59 minutes, 51.69 seconds
The date indicated is Monday, November 22th, 1988.

APPENDIX 1.
DS5000T DEMONSTRATION SOFTWARE

```
;   Program DEMODS5T
;
;   This program responds to commands received over the serial
;   I/O port to send or receive the date/time information between
;   the Embedded Clock/calendar (ECC) in the DS5000T and the
;   serial I/O port.  This allows an external program access to the
;   date/time information.
;
;   The program first sets up the serial port for transmission at
;   9600 baud with eight data bits, no parity, and one stop bit.
;
;   Next, the program begins execution of a loop waiting for an
;   instruction from the serial port.  Two valid instructions, R
;   and W, are recognized:
;
;       Receipt of the R character causes the DEMODS5T program
;       to read eight bytes of date/time information from
;       the DS1215 and send them out over the serial port.
;
;       Receipt of the W character causes the DEMODS5T program
;       to wait for eight bytes of date/time information
;       from the serial port and write them to the ECC.
;
;   Any other byte received from the serial port is incremented
;   and then sent back out to the serial port.
;
PCON    EQU     87H
MCON    EQU     0C6H
TA      EQU     0C7H
;
        CSEG    at    0
        SJMP    START
        CSEG    at    30H
START:                                  ; Initialization.
        MOV     TA,   #0AAH             ; Timed
        MOV     TA,   #55H              ;   access.
        MOV     PCON, #0                ; Reset watchdog timer.
```

```
        MOV     MCON, #0F8H     ; Turn off CE2 for memory access.
        LCALL   CLOSE           ; Close date/time registers.

        MOV     IE,   #0
        MOV     TMOD, #20H      ; Initialize the
        MOV     TH1,  #0FAH     ;   serial port
        MOV     TL1,  #0FAH     ;     for 9600
        ORL     PCON, #80H      ;       baud.
        MOV     SCON, #52H
        MOV     TCON, #40H
L:
        JNB     RI,   L         ; Wait for character.
        CLR     RI              ; Clear the receiver.
        MOV     A,    SBUF      ; Load in the character.
        CJNE    A,#'R',H        ; Skip if not a read.
        LCALL   OPEN            ; Set up to read date/time.
        MOV     B,    #8        ; Set up to send 8 bytes.
F:
        LCALL   RBYTE           ; Read a byte of date/time.
G:
        JNB     TI,   G         ; Wait for end of previous send.
        CLR     TI              ; Clear transmitter.
        MOV     SBUF,           ; Send out the byte.
        DJNZ    B,    F         ; Loop for 8 bytes.
        SJMP    L               ; Return to main loop.
H:
        CJNE    A,#'W',J        ; Skip if not a write.
        LCALL   OPEN            ; Set up to read date/time.
        MOV     B,    #8        ; Set to receive 8 bytes.
I:
        JNB     RI,   I         ; Wait to receive a byte.
        CLR     RI              ; Clear the receiver.
        MOV     A,    SBUF      ; Bring in the byte.
        LCALL   WBYTE           ; Write a byte of date/time.
        DJNZ    B,    I         ; Loop for 8 bytes.
        SJMP    L               ; Return to main loop.
J:
        JNB     TI,   J         ; If it is neither an R nor a W,
        CLR     TI              ;   increment the character,
        INC     A               ;   send it back out to the
        MOV     SBUF,           ;   serial port, and then
        SJMP    L               ;   return to the main loop.
```

```
;*********************************************************
;** SUBROUTINE TO OPEN THE EMBEDDED CLOCK/CALENDAR (ECC)**
;*********************************************************

; This subroutine executes the sequence of reads and writes which
; is required in order to open communication with the timekeeper.
; The subroutine returns with the timekeeper opened for data
; access and with both the accumulator and B register modified.

OPEN:   LCALL   CLOSE           ; Make sure it is closed.
        MOV     B,#4            ; Set pattern period count.
        MOV     A,#0C5H         ; Load first byte of pattern.
OPENA:  LCALL   WBYTE           ; Send out the byte.
        XRL     A,#0FFH         ; Generate next pattern byte.
        LCALL   WBYTE           ; Send out the byte.
        SWAP    A               ; Generate next pattern byte.
        DJNZ    B,OPENA         ; Repeat until 8 bytes sent.
        RET                     ; Return.

;*****************************************
;**** SUBROUTINE TO CLOSE ECC **
;*****************************************

; This subroutine insures that the registers of the timekeeper
; are closed by executing 9 successive reads of the date and time
; registers. The subroutine returns with both the accumulator
; and the B register modified.

CLOSE:  MOV     B,#9            ; Set up to read 9 bytes.
CLOSEA: LCALL   RBYTE           ; Read a byte;
        DJNZ    B,CLOSEA        ; Loop for 9 byte reads.
        RET                     ; Return.

;*************************************************
;**** SUBROUTINE TO READ A DATA BYTE****
;*************************************************

; This subroutine performs a "context switch" to the CE2 data
; space and then reads one byte from the timekeeping device.
; Then it switches back to the CE1 data space and returns
; the byte read in the accumulator, with all other registers
```

```
; unchanged.
;
RBYTE:  PUSH   DPL            ; Save the data
        PUSH   DPH            ;   pointer on stack.
        PUSH   MCON           ; Save MCON register.
        ORL    MCON,#4        ; Switch to CE2.
        PUSH   B              ; Save the B register.
        MOV    DPL ,#4        ; Set up for data input.
        MOV    DPH, #0        ; Set high address byte.
        MOV    B,  #8         ; Set the bit count.
LI:     PUSH   ACC            ; Save the accumulator.
        MOVX   A,  @DPTR      ; Input the data bit.
        RLC    A              ; Move it to carry.
        POP    ACC            ; Get the accumulator.
        RRC    A              ; Save the data bit.
        DJNZ   B,  LI         ; Loop for a whole byte.
        POP    B              ; Restore the B register.
        POP    MCON           ; Restore MCON register.
        POP    DPH            ; Restore the data
        POP    DPL            ;   pointer from stack.
        RET                   ; Return.
;
;-----------------------------------------------------
;**** SUBROUTINE TO WRITE A DATA BYTE *****
;-----------------------------------------------------
;
; This subroutine performs a "context switch" to the CE2 data
; space and then writes one byte from the accumulator to the
; timekeeping device.  Then it switches back to the CE1 data
; space and returns with all registers unchanged.
;
WBYTE:  PUSH   DPL            ; Save the data
        PUSH   DPH            ;   pointer on stack.
        PUSH   MCON           ; Save MCON register.
        ORL    MCON,#4        ; Switch to CE2.
        PUSH   B              ; Save the B register.
        MOV    DPH, #0        ; Set high address byte.
        MOV    B,  #8         ; Set the bit count.
LO:     PUSH   ACC            ; Save the accumulator.
        ANL    A,  #1         ; Set up bit for output.
        MOV    DPL, A         ; Set address to write bit.
        MOVX   A,  @DPTR      ; Output the data bit.
```

```
        POP     ACC             ; Restore the accumulator.
        RR      A               ; Position next bit.
        DJNZ    B,  LO          ; Loop for a whole byte.
        POP     B               ; Restore the B register.
        POP     MCON            ; Restore MCON register.
        POP     DPH             ; Restore the data
        POP     DPL             ;   pointer from stack.
        RET                     ; Return.

;*****************************
;***** END OF PROGRAM ********
;*****************************

        END                     ; End of program.
```

SELECTED ELECTRICAL CHARACTERISTICS

The following are selected electrical operating characteristics of the DS5000T. A full set of electrical characterics is available in the DS5000 User's Guide.

ABSOLUTE MAXIMUM RATINGS*

Voltage on any pin relative to ground	- 0.1 to 7.0V
Operating Temperature	- 0° to 70° C
Storage Temperature	- 40°C to +70° C
Soldering Temperature	- 260°C for 10 sec.

* This is a stress rating only and functional operation of the device at these or any other conditions outside of those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

D.C. CHARACTERISTICS

(T_A = 0°C to 70°C; V_{CC} = 5V ± 10%)

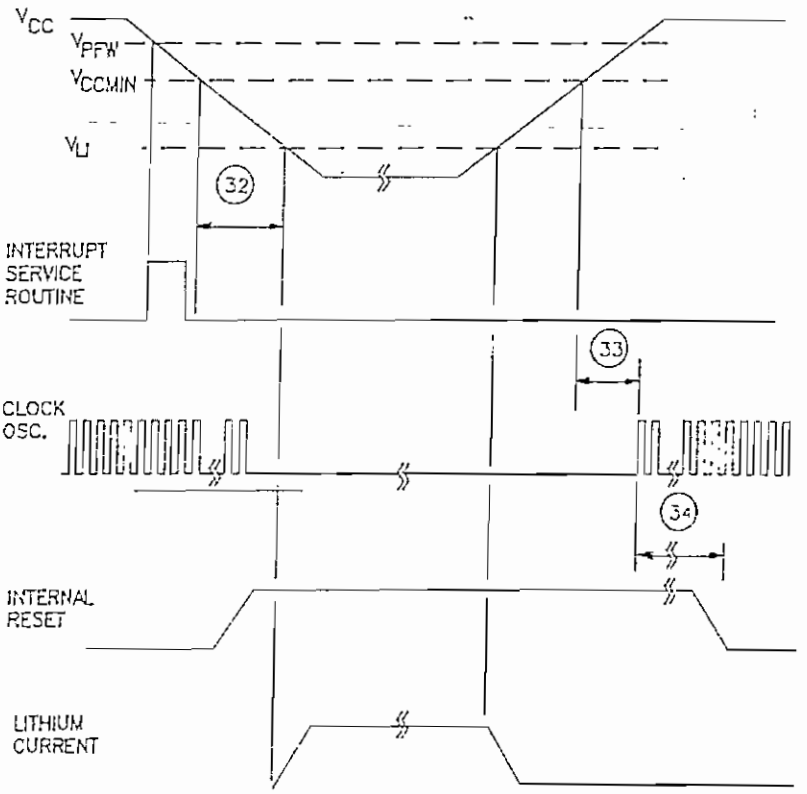
PARAMETER	SYM.	MIN.	TYP.	MAX.	UNITS	NOTES
Stop Mode Current	I_{SM}		45	80	uA	4
Power Fail Warning Voltage	V_{PFW}	4.15	4.6	4.75	V	
Minimum Operating Voltage	V_{CCmin}	4.05	4.5	4.65	V	
Lithium Supply Voltage	V_{LI}			3.3	V	
Programming Supply Voltage (Parallel Program Mode)	V_{PP}	12.5		13.0	V	
Program Supply Current	I_{PP}		9.2	15	mA	
Operating Current DS5000 8 (@12 MHz) DS5000 32	I_{CC}		20 25	43.2 48.2	mA	2
Idle Mode Current (@12 MHz)	I_{CC}			6.2	mA	3

A.C. CHARACTERISTICS
POWER CYCLING TIMING

(T_A = 0°C to 70°C; V_{CC} = 5V ±10%)

#	PARAMETER	SYMBOL	MIN.	MAX.	UNITS
32	Slew rate from V_{CCmin} to V_{LImax}	t_F	40		µs
33	Crystal start up time	t_{CSU}	(note 5)		
34	Power On Reset Delay	t_{POR}	2150^*4t_{CLK}		µs

POWER CYCLING TIMING DIAGRAM Figure 6

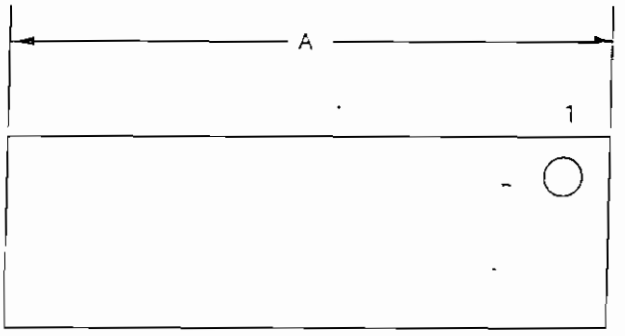


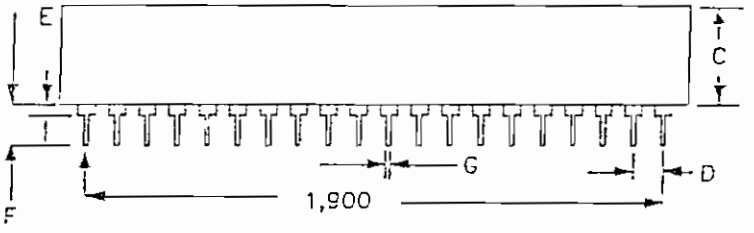
NOTES:

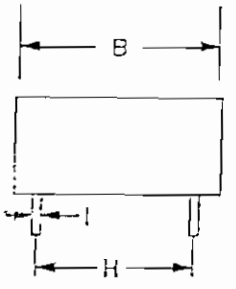
1. All voltages are referenced to ground.
2. Maximum operating ICC is measured with all output pins disconnected; XTAL1 driven with a clock source at 12 MHz with tCLKR, tCLKF = 10 ns, VIL =0.5V, VIH = 4.5V; XTAL2 disconnected; /EA=RST=PORT0=VCC.
3. Idle Mode ICC is measured with all output pins disconnected; XTAL1 driven with tCLKR, tCLKF = 10 ns, VIL = 0.5V, VIH = 4.5V; XTAL2 disconnected; EA=RST= PORT0=VCC.
4. Stop Mode ICC is measured with all output pins disconnected; EA=PORT0=VCC; XTAL2 not connected; RST = VSS.
5. Crystal start up time is the time required to get the mass of the crystal into vibrational motion from the time that power is first applied to the circuit until the first clock pulse is produced by the on-chip oscillator. The user should check with the crystal vendor for a worst case spec on this time.

DS5000T
Time Microcontroller

DIM.	INCHES	
	MIN.	MAX.
A	2.080	2.100
B	.680	.700
C	.290	.310
D	.090	.110
E	.040	.060
F	.165	.185
G	.016	.020
H	.590	.610
I	.009	.012







MAXIM

+5V Powered RS-232 Drivers/Receivers

General Description

Maxim's family of line drivers/receivers are intended for all RS-232 and V.28/V.24 communications interfaces, and in particular, for those applications where ±12V is not available. The MAX230, MAX236, MAX240 and MAX241 are particularly useful in battery powered systems since their low power shutdown mode reduces power dissipation to less than 5µW. The MAX233 and MAX235 use no external components and are recommended for applications where printed circuit board space is critical.

All members of the family except the MAX231 and MAX239 need only a single +5V supply for operation. The RS-232 drivers/receivers have on-board charge pump voltage converters which convert the +5V input power to the ±10V needed to generate the RS-232 output levels. The MAX231 and MAX239, designed to operate from +5V and +12V, contain a +12V to −12V charge pump voltage converter.

Since nearly all RS-232 applications need both line drivers and receivers, the family includes both receivers and drivers in one package. The wide variety of RS-232 applications require differing numbers of drivers and receivers. Maxim offers a wide selection of RS-232 driver/receiver combinations in order to minimize the package count (see table below).

Both the receivers and the line drivers (transmitters) meet all EIA RS-232C and CCITT V.28 specifications.

Features

- Operates from Single 5V Power Supply (+5V and +12V — MAX231 and MAX239)
- Meets All RS-232C and V.28 Specifications
- Multiple Drivers and Receivers
- Onboard DC-DC Converters
- ±9V Output Swing with +5V Supply
- Low Power Shutdown — <1µA (typ)
- 3-State TTL/CMOS Receiver Outputs
- ±30V Receiver Input Levels

Applications

Computers
Peripherals
Modems
Printers
Instruments

Selection Table

Part Number	Power Supply Voltage	No. of RS-232 Drivers	No. of RS-232 Receivers	External Components	Low Power Shutdown /TTL 3-State	No. of Pins
MAX230	+5V	5	0	4 capacitors	Yes/No	20
MAX231	+5V and +7.5V to 13.2V	2	2	2 capacitors	No/No	14
MAX232	+5V	2	2	4 capacitors	No/No	16
MAX233	+5V	2	2	None	No/No	20
MAX234	+5V	4	0	4 capacitors	No/No	16
MAX235	+5V	5	5	None	Yes/Yes	24
MAX236	+5V	4	3	4 capacitors	Yes/Yes	24
MAX237	+5V	5	3	4 capacitors	No/No	24
MAX238	+5V	4	4	4 capacitors	No/No	24
MAX239	+5V and +7.5V to 13.2V	3	5	2 capacitors	No/Yes	24
MAX240	+5V	5	5	4 capacitors	Yes/Yes	44 (Flatpak)
MAX241	+5V	4	5	4 capacitors	Yes/Yes	28 (Small Outline)

*Patent Pending

+5V Powered RS-232 Drivers/Receivers

ABSOLUTE MAXIMUM RATINGS

V_{CC} ..... -0.3V to +6V
V^+ ..... (V_{CC} - 0.3V) to +14V
V^- ..... +0.3V to -14V
Input Voltages
T_{IN} ..... -0.3 to (V_{CC} + 0.3V)
R_{IN} ..... ±30V
Output Voltages
T_{OUT} ..... (V^+ + 0.3V) to (V^- - 0.3V)
R_{OUT} .....

Short Circuit Duration
T_{OUT} ..... continuous
Power Dissipation
CERDIP 675mW
(derate 9.5mW/°C above +70°C)
Plastic DIP 375mW
(derate 7mW/°C above +70°C)
Small Outline (SO) 375mW
(derate 7mW/°C above +70°C)
Storage Temperature .....

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions above those indicated in the operational sections of the specifications is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

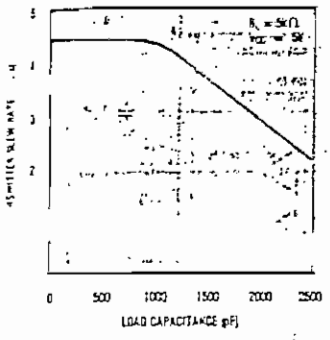
(MAX232, 234, 236, 237, 241, ... ±10%; MAX233, 235 V_{CC} = 5V ± 5%; MAX231, 239 V_{CC} = 5V ± 10%, V^+ = 7.5V to 13.2V, T_A = Operating Temperature Range, Figure ...)

PARAMETER	CONDITIONS		MIN.	TYP.	MAX.	UNITS
Output Voltage Swing	All Transmitter Outputs loaded with 3kΩ to Ground		±5	±9		V
V_{CC} Power Supply Current	No load, T_A = +25°C			5	10	mA
	MAX231, MAX239			0.4	1	
V^+ Power Supply Current	No load, MAX231 and MAX239 only	MAX231		1.8	5	mA
		MAX239		5	15	
Shutdown Supply Current	Figure 1, T_A = +25°C			1	10	µA
Input Logic Threshold Low	T_{IN}, $\overline{EN}$, Shutdown				0.8	V
Input Logic Threshold High	T_{IN}		2.0			V
	$\overline{EN}$, Shutdown		2.4			
Logic Pullup Current	T_{IN} = 0V			15	200	µA
RS-232 Input Voltage Operating Range			-30		+30	V
RS-232 Input Threshold Low	V_{CC} = 5V, T_A = +25°C (MAX231, 239 V^+ = 0V)		0.8	1.2		V
RS-232 Input Threshold High	V_{CC} = 5V T_A = +25°C (MAX231, 239 V^+ = 12V)			1.7	2.4	V
RS-232 Input Hysteresis	V_{CC} = 5V		0.2	0.5	1.0	V
RS-232 Input Resistance	T_A = +25°C, V_{CC} = 5V		3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 1.6mA (MAX231-233, I_{OUT} = 3.2mA)				0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5			V
TTL/CMOS Output Leakage Current	$\overline{EN}$ = V_{CC}, 0V ≤ R_{OUT} ≤ V_{CC}			0.05	±10	µA
Output Enable Time (Figure 2)	MAX235, MAX236, MAX239, MAX240, 241			400		ns
Output Disable Time (Figure 2)	MAX235, MAX236, MAX239, MAX240, 241			250		ns
Propagation Delay	RS-232 to TTL			0.5		µs
Instantaneous Slew Rate	C_L = 10pF, R_L = 3-7kΩ T_A = +25°C (Note 1)				30	V/µs
Transition Region Slew Rate	R_L = 3kΩ, C_L = 2500pF Measured from +3V to -3V or -3V to +3V			3		V/µs
Output Resistance	V_{CC} = V^+ = V^- = 0V, V_{OUT} = ±2V		300			Ω
RS-232 Output Short Circuit Current				±10		mA

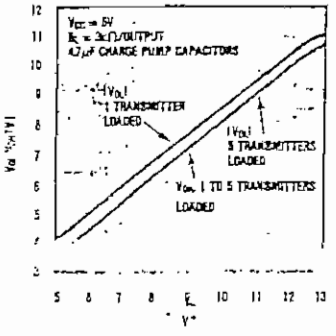
Note 1: Sample tested.

Typical Operating Characteristics

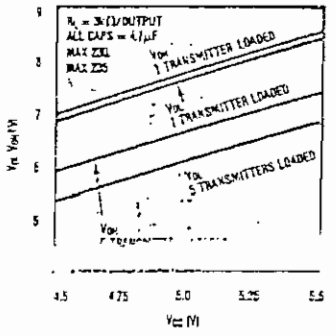
TRANSMITTER SLEW RATE
vs. LOAD CAPACITANCE



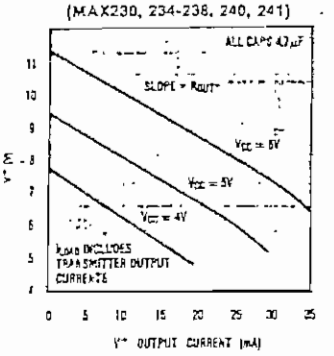
MAX239 TRANSMITTER OUTPUT VOLTAGE
vs. V+ VOLTAGE



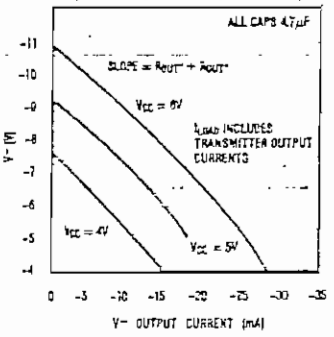
TRANSMITTER OUTPUT VOLTAGE
vs. V_{CC} VOLTAGE



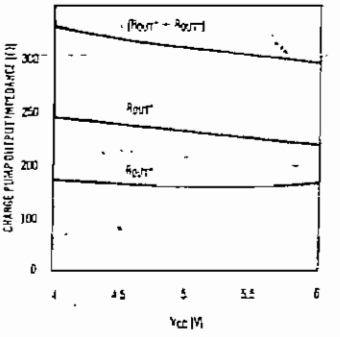
V+ SUPPLY VOLTAGE
vs. LOAD CURRENT
(MAX230, 234-238, 240, 241)



V− SUPPLY VOLTAGE
vs. LOAD CURRENT
(MAX230, 234-238, 240, 241)



CHARGE PUMP OUTPUT IMPEDANCE
vs. V_{CC}
(MAX230, 234-238, 240, 241)



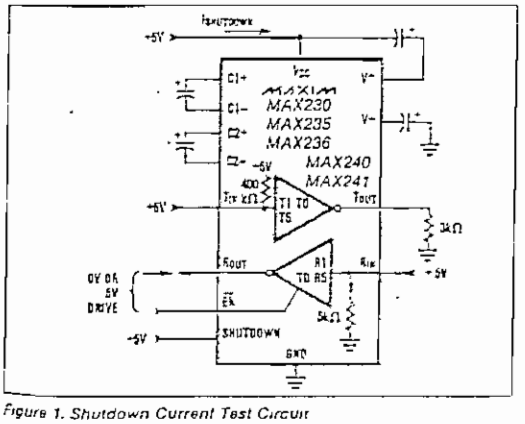


Figure 1. Shutdown Current Test Circuit

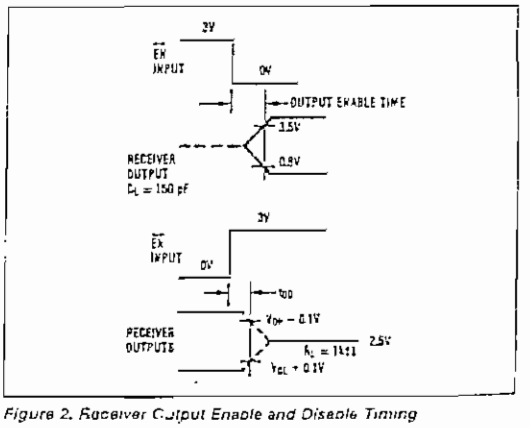
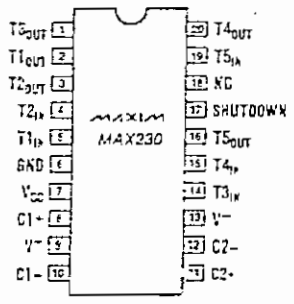


Figure 2. Receiver Output Enable and Disable Timing



20 Lead Small Outline
also available.

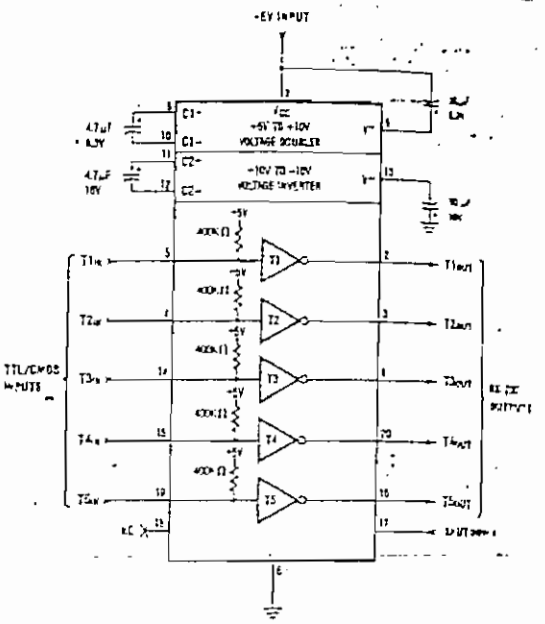
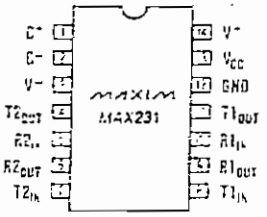
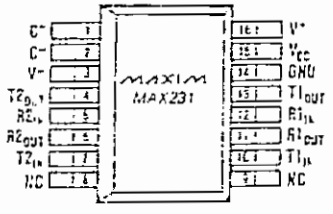


Figure 3. MAX230 Typical Operating Circuit



14 Lead Plastic DIP



16 Lead Small Outline

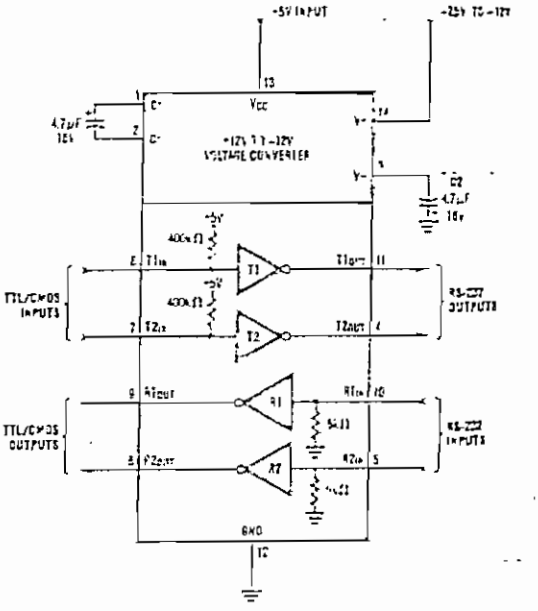
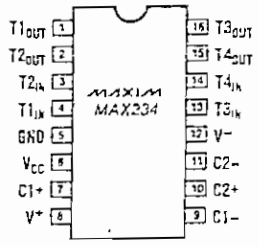


Figure 4. MAX231 Typical Operating Circuit

+5V Powered RS-232 Drivers/Receivers



16 Lead Small Outline also available.

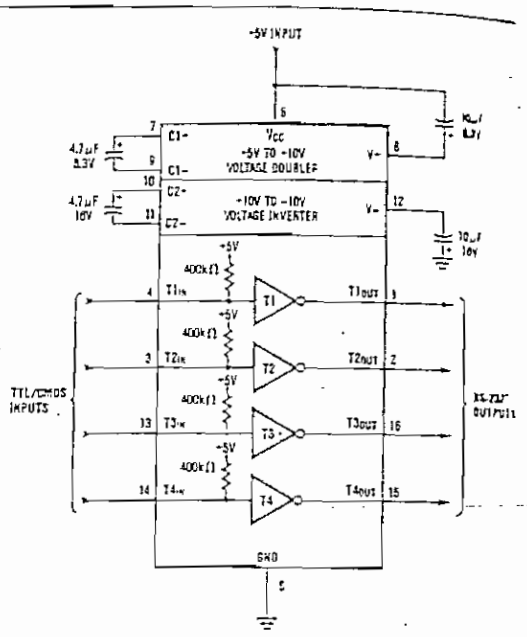
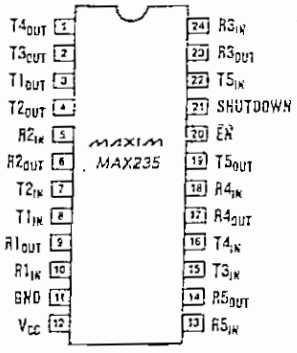


Figure 7. MAX234 Typical Operating Circuit



0.600" Wide Package Only

Small Outline Not Available

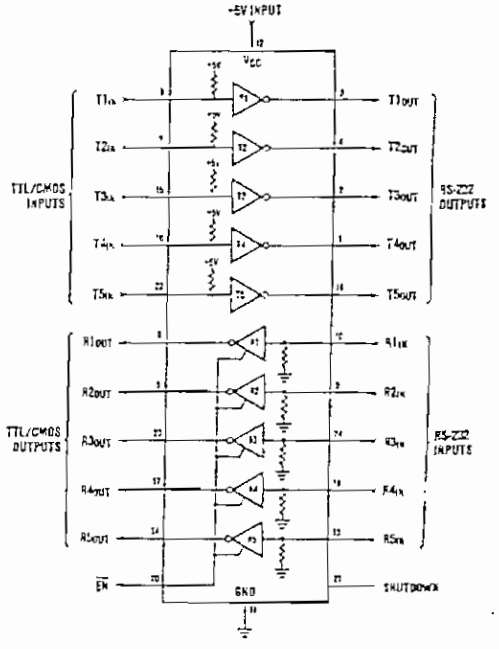
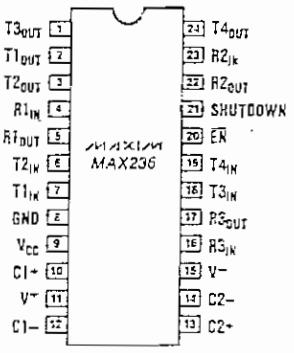
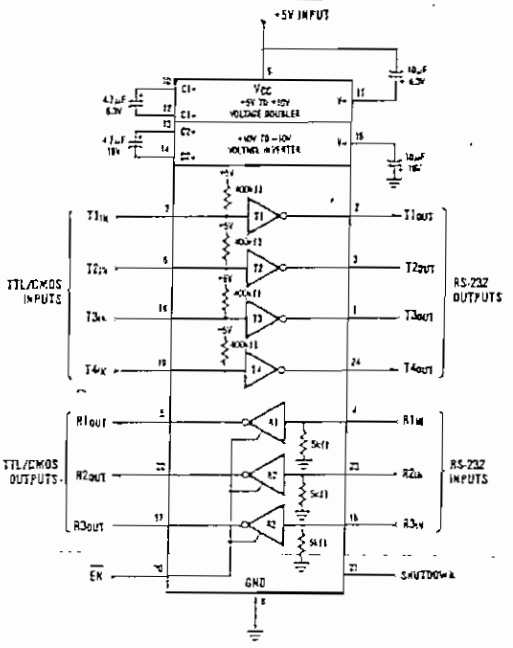


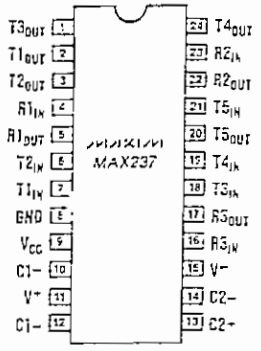
Figure 8. MAX235 Typical Operating Circuit



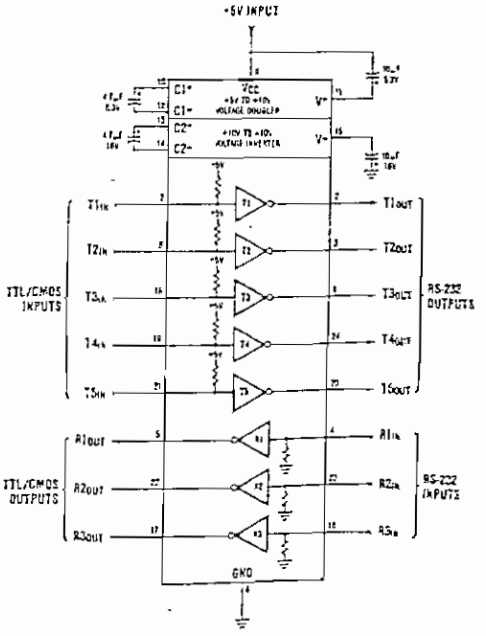
24 Lead Small Outline
also available.



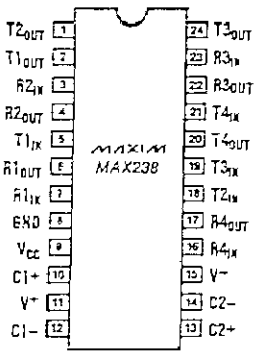
ure 9. MAX236 Typical Operating Circuit



24 Lead Small Outline
also available.



ure 10. MAX237 Typical Operat g Circuit



24 Lead Small Outline
also available.

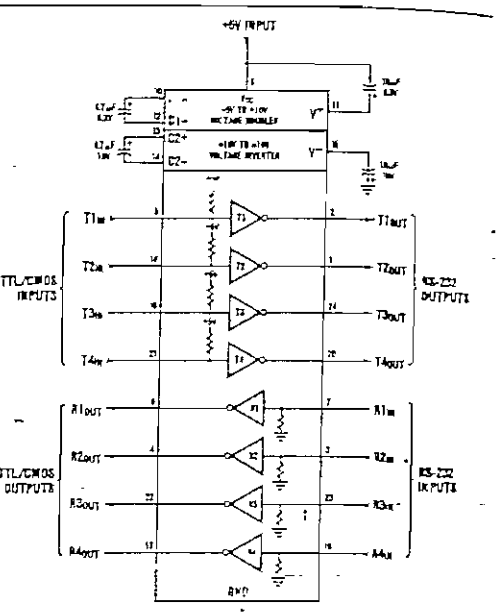
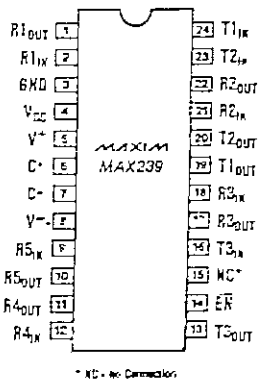


Figure 11. MAX238 Typical Operating Circuit



24 Lead Small Outline
also available.

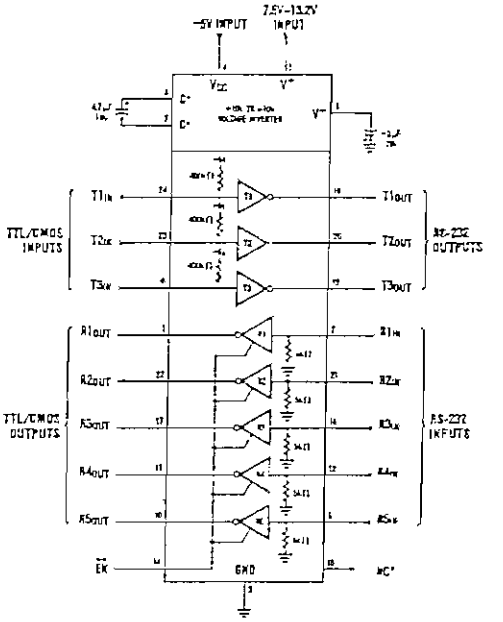
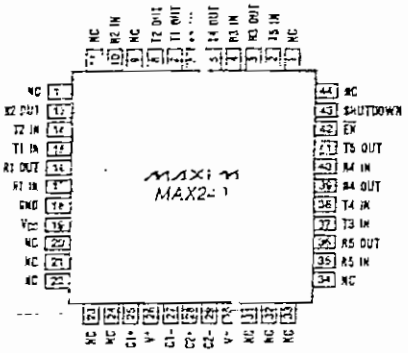


Figure 12. MAX239 Typical Operating Circuit



44 Lead Plastic Flatpak Only

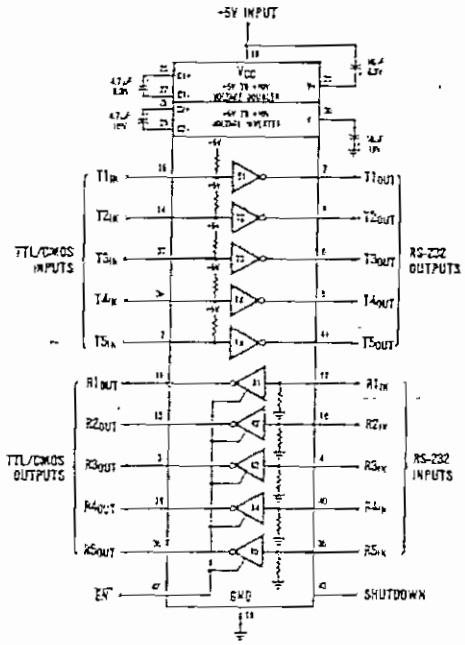
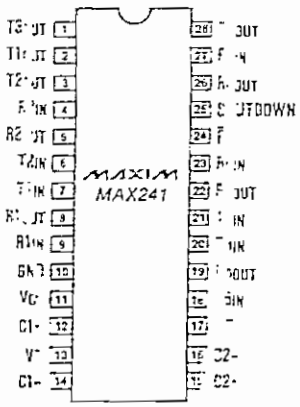


Figure 13. MAX240 Typical Operating Circuit.



28 Lead Wide Small Outline Only

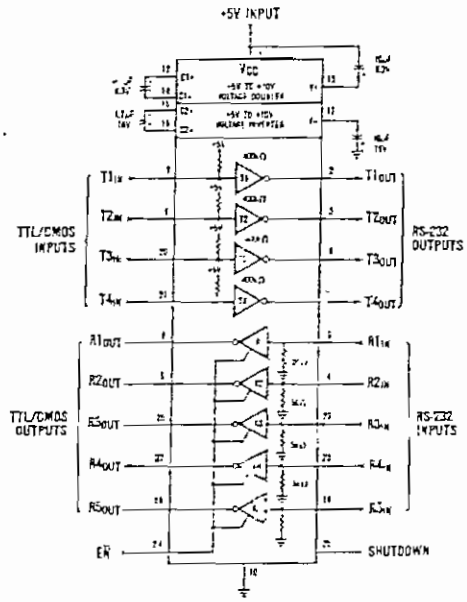


Figure 14. MAX241 Typical Operating Circuit.

BIBLIOGRAFIA

- DALLAS SEMICONDUCTOR, DS5000 Soft Microcontroller User's Guide, Dallas Semiconductor Corporation, Dallas, 1989.

- DALLAS SEMICONDUCTOR, Product Data Book, Dallas Semiconductor Corporation, Dallas, 1989.

- MAXIM, Integrated Circuits Data Book, Maxim Integrated Products, Inc., California, 1989.

- ORDOÑEZ, I., ORTEGA, F., Depurador de Programas para el Microprocesador i8086, Quito, 1989.

- LEIBSON, Steve, The Handbook of Microcomputer Interfacing, Tab Books, Inc., California, 1985.

- RODRIGUEZ-ROSELLO, Miguel, 8088 - 8086/8087 Programación Ensamblador en entorno MSDOS, Ediciones Anaya Multimedia, Madrid, 1987.

- NORTON, P., Socha, J., Peter Norton's Assembly Lanquaje Book, Prentice Hall, New York, 1986.

- NORTON, Peter, Guia del Programador para el IBM PC, Ediciones Anaya Multimedia, Madrid, 1987