TARJETA EDUCACIONAL BASADA EN 80186

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ciencias y Humanidades

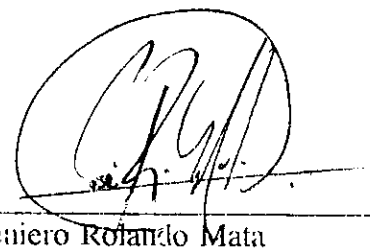
TARJETA EDUCACIONAL BASADA EN 80186

JONATHAN ROBERTO CUKIER ALCAHE

Trabajo de graduación presentado para optar
al grado académico de
Licenciatura en Ingeniería Electrónica

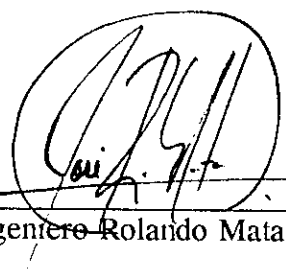
Guatemala
1993

Vo. Bo. :

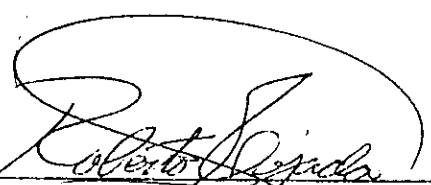


(f)_____
Ingeniero Rolando Mata
Asesor

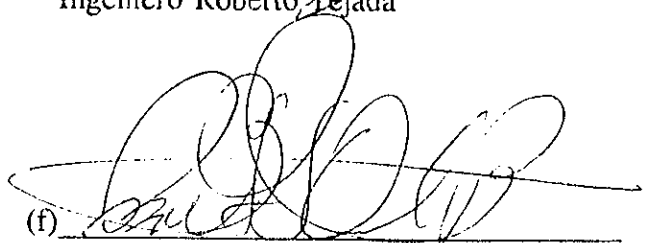
Tribunal:



(f)_____
Ingeniero Rolando Mata



(f)_____
Ingeniero Roberto Tejada



(f)_____
Ingeniero Robert Duke

Fecha de aprobación: Julio 23 de 1993.

"Scientific research consists in seeing what everyone else has seen, but thinking what no one else has thought"

A. Szent-Gyorgi (b. 1893)

RESUMEN

El presente trabajo describe el diseño de una tarjeta educacional, útil para el estudio de microprocesadores y sus aplicaciones, basada en el procesador INTEL 80186.

El INTEL 80186 es un procesador integrado de 16 bits, que combina de 15 a 20 de los componentes más comunes de un sistema con microprocesador en un solo chip.

La tarjeta se ha diseñado para ser conectada vía un enlace serial asíncrono RS232 a una terminal serial o computadora personal con un puerto serial disponible. Para ello se utiliza el controlador serial multiprotocolo 8274, también fabricado por INTEL.

El sistema cuenta con un procesador 80186, 128 Kbytes de ROM, 128 Kbytes de RAM, y un controlador serial multiprotocolo 8274, además de todos los chips de apoyo, como compuertas lógicas, latches, transcievers y buffers.

El software se elaboró utilizando principalmente lenguaje C, y auxiliándose de rutinas en lenguaje ensamblador.

CONTENIDO

Páginas

LISTA DE FIGURAS

LISTA DE TABLAS

I. INTRODUCCION

Debido al equipo con que se cuenta actualmente en la Universidad del Valle de Guatemala para las prácticas de los cursos de Arquitectura Digital, que es basado en microprocesadores de 8 bits, surge la necesidad de un nuevo sistema para que el estudiante pueda trabajar con arquitecturas, técnicas y microprocesadores más avanzados. Aun cuando en el curso teórico se estudian dichos temas, es de suma importancia que el estudiante se familiarice con ellos, y no hay mejor forma para esto que trabajando y haciendo proyectos con ellos.

En la actualidad, muchas de las aplicaciones para microprocesadores requieren de la tecnología de los microprocesadores integrados (embedded processors), por lo que trabajar con el INTEL 80186, proporciona al estudiante una experiencia didáctica y valiosa para aplicarla en el futuro de su carrera profesional.

Este trabajo describe el diseño de un sistema educacional basado en el procesador INTEL 80186, un microprocesador integrado de 16 bits. Este sistema fue diseñado para comunicarse con una terminal a través de un enlace serial asíncrono RS-232.

El diseño del sistema básico cuenta con un procesador INTEL 80186, 128 Kbytes de ROM, 128 Kbytes de RAM estática, y un controlador serial multiprotocolo INTEL

8274. La frecuencia del reloj del sistema es de 8 MHz.

El bus de datos y direcciones, multiplexado en el tiempo, sería demultiplexado usando estabilizadores (latches) de dirección (74LS373) y transceptores (transcievers) de bus (74LS245), creando un bus de direcciones de 20 bits, y un bus de datos de 16 bits. Se han utilizado en el diseño dos EEPROMS 27C512 para tener 128 Kbytes de memoria ROM, y cuatro chips 60256 de RAM estática para 128 Kbytes de memoria RAM.

El programa monitor residiría en la memoria de sólo lectura ROM, y se implementa conectando rutinas elaboradas en assembler y lenguaje C. El sistema diseñado tiene la capacidad de conectarse a una terminal vía un enlace RS232, desde la cual se opera el sistema. Si se utiliza una computadora personal para emular la terminal serial, pueden aprovecharse las unidades de almacenamiento de ésta (disk drives y disco duro) para transferir archivos hacia el sistema, utilizando el protocolo XMODEM.

Los siguientes capítulos describen el desarrollo del proyecto. El capítulo 2 provee los fundamentos teóricos y características principales de los componentes utilizados en el diseño.

El capítulo 3 describe el diseño del hardware del sistema, presentándose los diagramas correspondientes. El capítulo 4 describe el diseño del software del sistema,

incluyéndose los listados respectivos de las rutinas en assembler y lenguaje C.

El capítulo 5 presenta un manual del usuario, donde se explica la operación del sistema. Se cubre la preparación del hardware, y el formato y ejemplos de cada uno de los comandos permitidos.

El capítulo 6 presenta las conclusiones del trabajo y recomendaciones para su ampliación y aplicación.

II. FUNDAMENTOS TEORICOS

Este capítulo proporciona una breve descripción de las características más importantes del microprocesador 80186, así como del controlador serial multiprotocolo utilizado (8274). Las características de ambos que se discuten no son todas las que poseen, pero sí las de mayor relevancia para el presente proyecto. Para una descripción más completa de estas y otras características, así como de su uso, es conveniente referirse a los manuales técnicos que proporciona el fabricante (Intel Corporation).

A. El microprocesador 80186

El Intel 80186 es un microprocesador de alta integración. Este microprocesador combina efectivamente de 15 a 20 de los componentes más comunes que encontraríamos en un sistema basado en el 8086. Además el 80186 proporciona el doble de la performancia del 8086 de 5 MHz. El 80186 es compatible con el software del 8086 y del 8088, pero además añade 10 nuevos tipos de instrucciones al set ya existente.

1. **Historia.** El microprocesador 8086 fue introducido por primera vez en 1978, y obtuvo un rápido soporte como la parte integral de las computadoras personales. Actualmente hay en el mundo millones de sistemas basados en 8086/8088.

El 8086, requiere docenas de chips como soporte para poder implementar un sistema aun moderadamente complejo. Intel reconoció entonces la necesidad de integrar los periféricos más comúnmente utilizados en la misma pastilla del CPU. En 1982 Intel introdujo la familia 80186/80188 de microprocesadores integrados (embedded microprocessors). Además de la adición de los nuevos periféricos integrados, el CPU fue mejorado con nuevas instrucciones y el tiempo requerido para llevar acabo todas las instrucciones de acceso a memoria fue reducido.

2. **Arquitectura base.** Las familias 8086,8088, 80186, 80188 y 80286, poseen todas el mismo set básico de registros, instrucciones y modos de direccionamiento.

REGISTROS:

La figura 2.1 muestra el set de registros del 80186.

El 80186 tiene ocho registros generales de 16 bits, que pueden ser usados para operaciones lógicas y aritméticas, los cuales son subdivididos en dos sets de cuatro registros cada uno. Los primeros cuatro son los registros de datos, y los otros cuatro son los registros punteros y de índice.

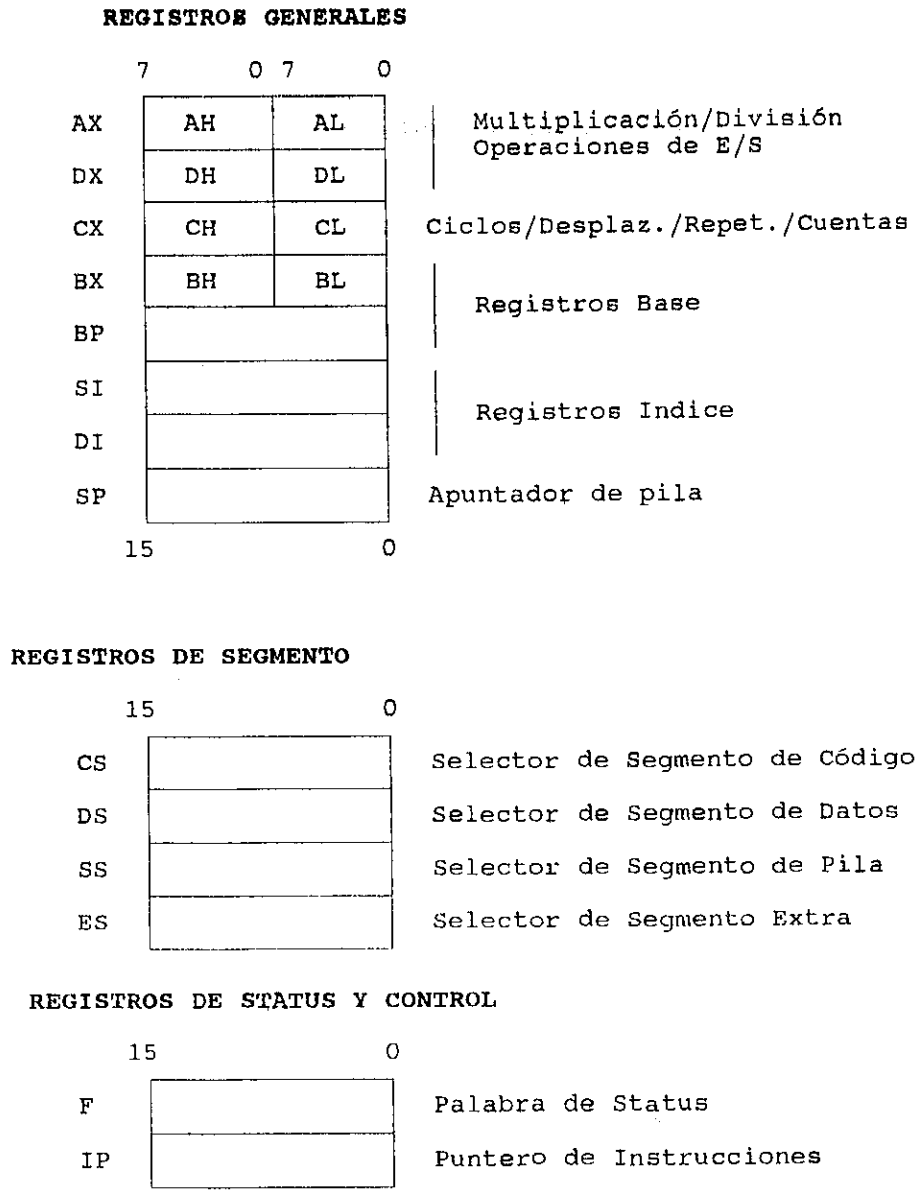


Figura 2.1 Set de registros del 80186

Los registros de datos son únicos, en el sentido que sus mitades superior e inferior son direccionables independientemente. Por ello, cada uno de estos registros (AX, BX, CX y DX), pueden ser utilizados flexiblemente como un registro de 16 bits (e.g. AX), o dos registros de 8 bits cada uno (e.g. AH y AL)

El espacio de memoria del 80186 es dividido en segmentos lógicos de 64 Kb cada uno para su direccionamiento. El CPU tiene acceso directo a cuatro segmentos a la vez. Existen cuatro registros de 16 bits, que son utilizados para almacenar la dirección base de cada uno de los cuatro segmentos de memoria: código, datos, pila y segmento extra. Las instrucciones son leídas del segmento de código. Las operaciones de pila son realizadas en el espacio designado para el segmento de pila. El segmento de datos, generalmente contiene variables del programa. El cuarto segmento, el segmento extra, generalmente se utiliza también para almacenamiento de datos.

Dos registros de 16 bits alteran ciertos aspectos del estado del 80186. Estos son el del puntero de instrucciones, y el de la palabra de status. El registro del puntero de instrucciones contiene el desplazamiento (en bytes) de la siguiente instrucción a ejecutar a partir de la dirección base del segmento de código. La palabra de status anota características específicas del resultado de operaciones lógicas y aritméticas, y controla la operación del 80186 dentro de un modo de operación dado.

SET DE INSTRUCCIONES:

El set de instrucciones es dividido en siete categorías: transferencia de datos, aritmética, corrimiento/rotación/lógica, manipulación de cadenas, transferencia de control, instrucciones de alto nivel y control del procesador.

Una instruccion puede referenciar desde ninguno hasta varios operandos. Un operando puede residir en un registro, en la instrucción misma, o en algún lugar de la memoria. Casi cualquier instrucción puede operar en datos de 8 ó de 16 bits.

ORGANIZACION DE LA MEMORIA:

La memoria es organizada en sets de segmentos. Cada segmento es una secuencia lineal contigua de hasta 64 Kb. La memoria es accesada utilizando una dirección con dos componentes (puntero), que consiste de un segmento de base de 16 bits, y un segmento de desplazamiento, también de 16 bits. Los valores de base están contenidos en cada uno de los cuatro registros internos de segmento.

Todos los segmentos comienzan en fronteras de memoria de 16 bits. Los segmentos pueden ser adyacentes, separados, parcialmente sobrepuestos, o totalmente sobrepuestos. Una dirección física puede resultar mapeada (localizada) en más de un segmento a la vez. La figura 2.2 muestra la generación de la dirección física a partir de estos dos componentes.

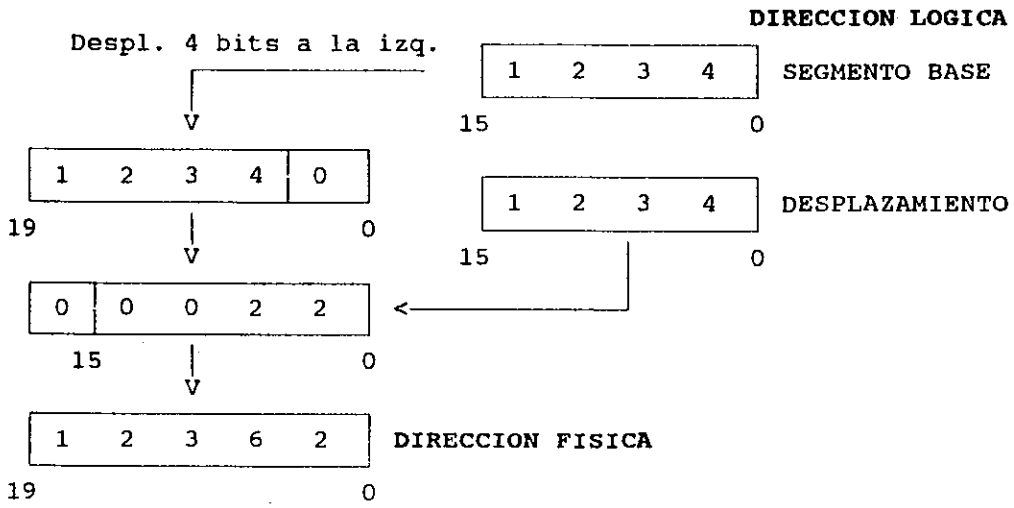


Figura 2.2 Generación de la dirección física

ESPACIO DE ENTRADA/SALIDA:

El espacio de E/S consiste de 64 Kb de puertos de 8 bits ó 32 Kb de puertos de 16 bits. Instrucciones separadas accesan el espacio de E/S, ya sea con una dirección de puerto de 8 bits, que es especificada en la instrucción, o con una dirección de puerto de 16 bits contenida en el registro DX.

INICIALIZACION Y RESET DEL PROCESADOR:

La inicialización o "startup" del procesador es lograda inyectando una señal baja en la pata RES. RES forza al 80186 a terminar toda ejecución y actividad de bus local.

Ninguna instrucción ni actividad del bus será ejecutada mientras RES está activo. Luego que RES se vuelve inactivo, y pasa un intervalo interno de proceso, el 80186 comienza su ejecución con la instrucción en la dirección física FFFF0h. RES da valores predefinidos a ciertos registros, tal como se muestra en la siguiente tabla.

Tabla 2.1

Valores predefinidos en reset del 80186

REGISTRO	VALOR PREDEFINIDO
Palabra de Status	F002h
Puntero de Instrucción (IP)	0000h
Segmento de Código (CS)	FFFFh
Segmento de Datos (DS)	0000h
Segmento Extra (ES)	0000h
Segmento de Pila (SS)	0000h
Registro de Relocación	20FFh
Sel. de Chip de Memoria Alta (UMCS)	FFFBh

3. Generador de reloj. El 80186 proporciona un generador de reloj y un oscilador de cristal. El oscilador de cristal puede ser utilizado con un cristal paralelo resonante a dos veces la velocidad de reloj de CPU deseada. La salida del oscilador es dividida

internamente por dos para proporcionar una señal de reloj de CPU, con un ciclo de trabajo del 50 %, en la cual se basa toda la temporización del sistema 80186.

4. **Lógica de reset.** El 80186 proporciona una pata de entrada de reset (reinicio) y una pata de salida sincronizada de reset, para uso con otros compenentes del sistema. La pata de entrada tiene histéresis para así facilitar la generación del reset al encender el sistema, utilizando una red RC.

La figura 2.3 muestra un diagrama de tiempo con las señales del reloj generado y el reset.

5. **Controlador de bus local.** El 80186 provee un controlador de bus local que genera las señanes necesarias para esta labor. Además proporciona señales que pueden ser utilizadas en el protocolo HOLD/HLDA para entregar el mando del bus local otros procesadores o periféricos. Provee también señales de salida que pueden ser utilizadas para activar buffers externos y para dirigir el flujo de datos desde y hacia el bus local.

CONTROL DE MEMORIA Y PERIFERICOS:

El 80186 provee señales de ALE (Address Latch Enable), RD (Read) y WR (Write) que son usadas para el control del bus. Las señanes de RD y WR son usadas para hacer un strobe de datos desde la memoria o los periféricos de E/S hacia el 80186

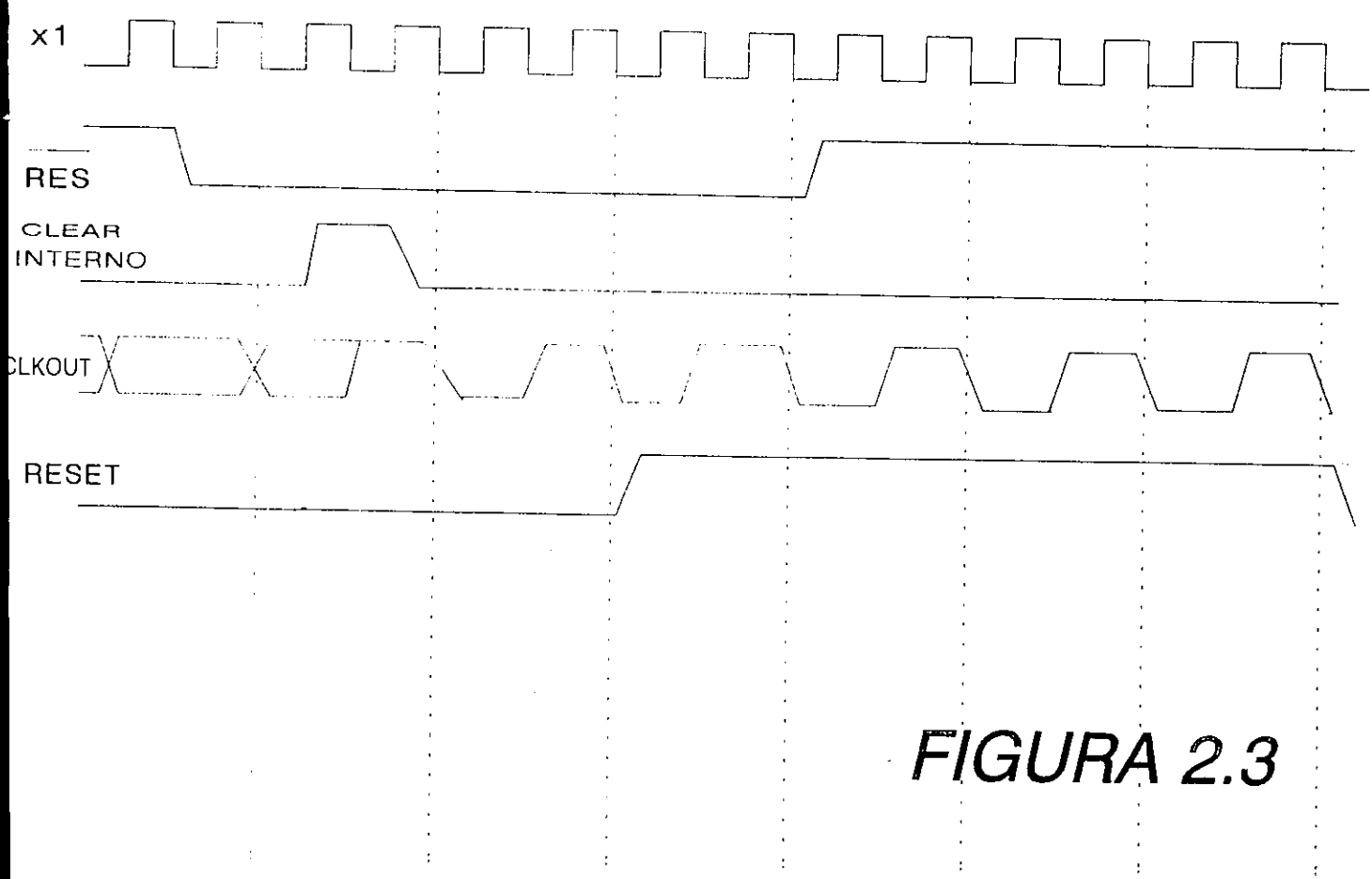


FIGURA 2.3

o viceversa. La línea de ALE proporciona un strobe para hacer un latch a la dirección cuando ésta es válida. El controlador de bus local del 80186 no proporciona una señal de memoria/periféricos de E/S.

CONTROL DE TRANSCIEVERS:

El 80186 genera dos señales de control para conectar a chips transcievers. Esta capacidad permite que se incluyan los transcievers para tener buffers sin necesidad de agregar lógica externa..

Estas líneas de control, DT/R (Data Transmit/Recieve), y DEN (Data Enble) son generadas para controlar el flujo de datos a través de los transcievers.

La figura 2.4 muestra los diagramas de tiempo para los ciclos de escritura y lectura del 80186.

6. Generación de señales de selección de chip (chip select). El 80186 contiene lógica que permite generar señales de selección de chip programables, tanto para memoria como para periféricos. Además puede ser programado para generar señales de READY ó WAIT STATES. Tiene la capacidad de proveer señales estabilizadas de los bits A_1 y A_2, para ser utilizadas con algunos periféricos que así lo requieren.

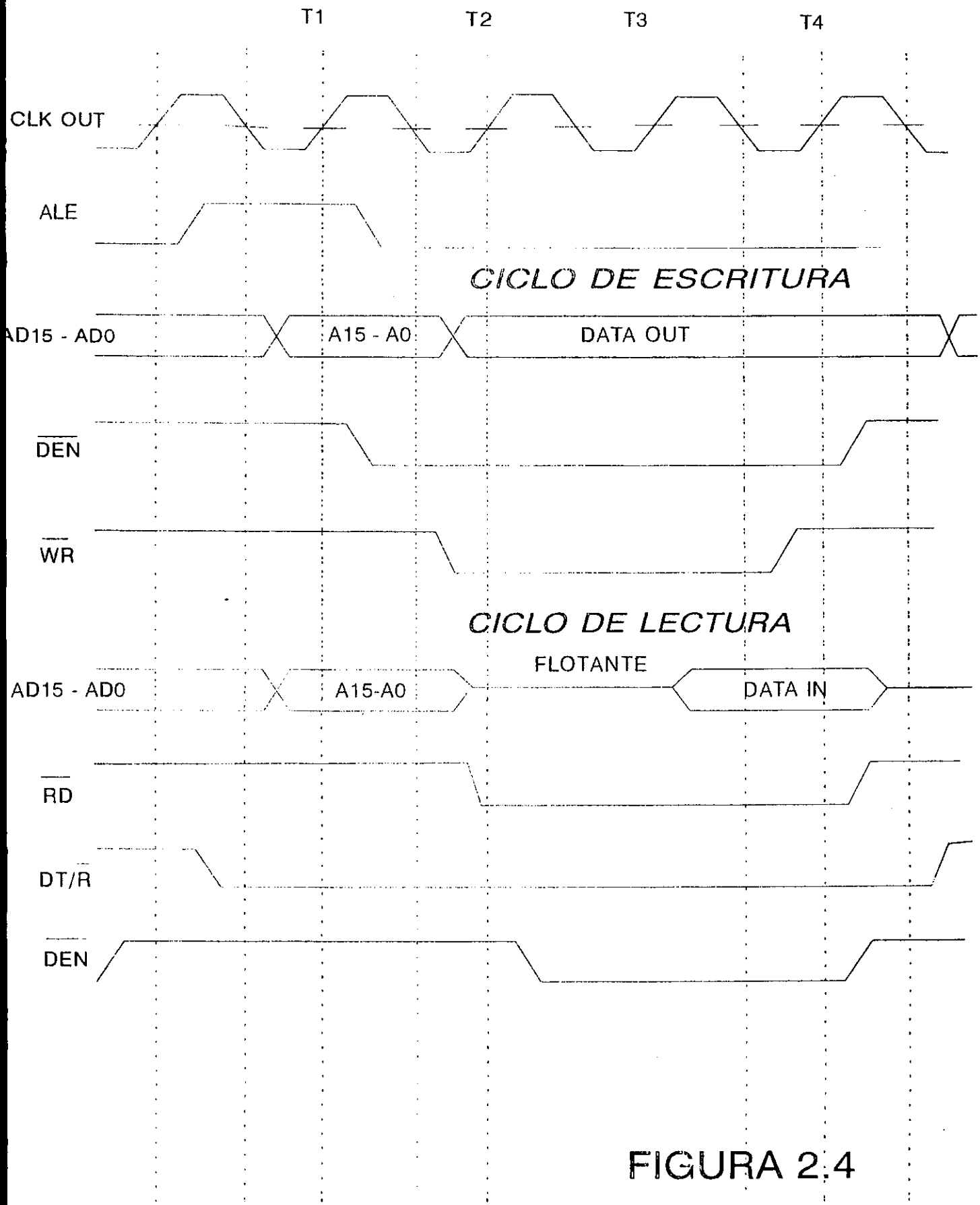

T1
T2
T3
T4
CLK OUT
ALE
CICLO DE ESCRITURA
AD15 - AD0
A15 - A0
DATA OUT
DEN
WR
CICLO DE LECTURA
FLOTANTE
AD15 - AD0
A15-A0
DATA IN
RD
DT/R
DEN

FIGURA 2.4

SELECCION PARA CHIPS DE MEMORIA:

El 80186 proporciona 6 señales de salida para conectar componentes de memoria en el sistema. Estas señales se agrupan en tres áreas de direccionamiento. Memoria alta, memoria baja y memoria media. Se proporciona una señal para la memoria alta y una para la memoria baja, mientras que para la memoria media existen cuatro señales.

El rango para cada señal es programable, y puede ser de 2, 4, 8, 16, 32, 64 o 128 K (además de 1 K y 256 K para las señales de selección de memoria alta y memoria baja). Además el comienzo de la dirección base para la memoria media puede ser programado.

La figura 2.5 muestra una posible utilización de estas señales para habilitar las tres áreas de memoria direccionables por el 80186.

El límite superior para la señal de selección de memoria alta (UCS) y el límite inferior para la señal de selección de memoria baja (LCS) son fijos, y se encuentran en FFFFFh y 00000h respectivamente. Para la memoria media, se programa la dirección base y el tamaño del bloque de memoria que se usará. La única limitación es que la dirección base debe ser un multiplo entero del tamaño del bloque. Por ejemplo si el tamaño del bloque es de 128 Kbytes, la dirección base podría ser 0 ó 20000h, pero no 10000h.

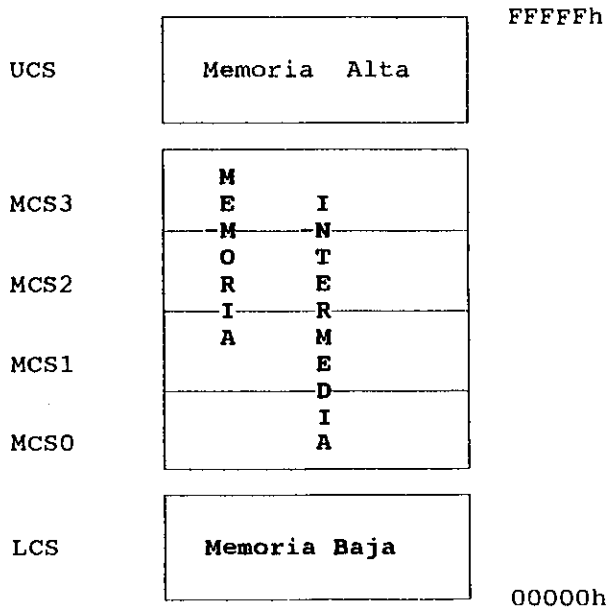


Figura 2.5 Utilización de las señales de selección de chip para memoria.

SELECCION PARA CHIPS DE PERIFERICOS:

El 80186 puede generar señales de selección de chip hasta para siete dispositivos periféricos. Estas señales son activas para siete bloques continuos de 128 bytes arriba de una dirección base programable. La dirección base puede estar localizada, ya sea en espacio de memoria o en espacio de E/S. La dirección base es programable por el usuario, pero sólo puede ser un múltiplo de 1 Kbytes, i.e. los 10 bits menos significativos de la dirección inicial son siempre 0.

Estas señales son controladas por dos registros en el bloque interno de control

de periféricos (ver tabla 2.2). Estos registros contienen la dirección base de los periféricos y direccionan a los periféricos en espacio de memoria o de E/S. Ambos registros deben ser accesados antes que cualquiera de las señales de selección se torne activa.

Tabla 2.2

Registros de control para señales de selección de chip

Reg.	Formato				Off.
UMCS	Tamaño de la memoria alta			Bits de ready de memoria alta	A0h
LMCS	Tamaño de la memoria baja			Bits de ready de memoria baja	A2h
PACS	Direcc. de selecc. para periféricos			Bits de ready PCS0-PCS3	A4h
MMCS	Direcc. de memoria intermedia			Bits de ready de mem. media	A6h
MPCS	Tamaño mem. intermedia	EX	MS	Bits de ready PCS4-PCS6	A8h

MS: 1=Periférico activo en espacio de memoria 0 = Periférico activo en espacio de E/S
EX: 1= 7 líneas de PCS 0=PCS5=A_1, PCS6=A_2

Un bit del registro MPCS permite que PCS5 y PCS6 sean salidas estabilizadas de las señales A_1 y A_2 durante un ciclo de bus. Esto permite una selección de periféricos externos en un sistema en que las señales de direcciones no están estabilizadas. Luego de un reset, estas líneas estarán en ALTO (1 lógico), y sólo reflejarán A_1 y A_2 cuando

los registros PACS y MPCS, hayan sido accesados.

7. **Temporizadores.** El 80186 incluye una unidad de temporizadores, que contiene tres temporizadores/contadores independientes de 16 bits. Dos de estos temporizadores pueden ser usados para contar eventos externos y proporcionar formas de onda derivadas, ya sea del reloj de CPU, de un reloj externo o para interrumpir al CPU luego de un número específico de eventos. El tercer temporizador cuenta sólo ciclos del reloj del CPU, y puede ser utilizado para interrumpir al CPU después de un número programable de pulsos de reloj, para dar un pulso de conteo para uno o ambos de los otros dos temporizadores luego de un número programable de pulsos de reloj del CPU, o para dar un pulso de requerimiento a la unidad integrada de DMA (Acceso Directo a Memoria) después de un número programable de pulsos de reloj del CPU.

B. El controlador serial multiprotocolo (MPSC) 8274

El Intel 8274, controlador serial multiprotocolo es un sofisticado controlador de comunicación de dos canales, que interfasa sistemas de microprocesadores con enlaces seriales de datos a altas velocidades utilizando protocolos síncronos o asíncronos. El 8274 se puede conectar fácilmente a los microprocesadores Intel más conocidos (e.g. 8048, 8051, 8085, 8086, 8088, 80186 y 80188), a controladores de ADM, tales como el 8237 y 8257 y al controlador de E/S 8089. Los dos canales de comunicación en el

CSMP 8274 son completamente independientes y pueden operar en un modo de comunicación full-duplex (transmisión y recepción simultánea de datos).

El 8274 puede realizar muchas funciones orientadas a comunicaciones, entre las cuales se incluyen:

- Convertir bytes de datos del sistema a un flujo serial de bits para transmisión por el enlace hasta el sistema receptor.
- Recibir flujos seriales de bits y reconvertir estos datos a bytes de datos en paralelo que pueden ser procesados por el sistema del microprocesador.
- Realizar chequeo de errores durante las transferencias de datos. Las funciones de chequeo de error incluyen calcular/transmitir códigos de error (tales como bits de paridad) y usar estos códigos para chequear la validez de los datos recibidos.
- Operar independientemente del procesador del sistema, en una manera diseñada para reducir el tiempo involucrado en las transferencias de datos.

1. **Interfase con el sistema.** El interfase del MPSC con el sistema es extremadamente flexible, y suporta los siguientes modos de transferencia de datos:

i. Modo de escrutinio (poll). El procesador del sistema periódicamente lee (escrutina) un registro de status del 8274 para determinar cuándo un caracter ha sido recibido, cuándo

es necesario hacer disponible un caracter para transmisión, y si se han detectado errores de transmisión.

ii. Modo de interrupciones. El MPSC interrumpe al procesador del sistema cuando un caracter ha sido recibido, cuando un caracter se necesita para transmisión, y cuando ocurren errores de transmisión.

iii. Modo de DMA. El MPSC automáticamente solicita transferencias de datos desde la memoria del sistema para las funciones de transmisión y recepción, mediante dos señales de solicitud de DMA por cada canal serial.

iv. Modo de espera (wait). La señal de ready del MPSC es utilizada para sincronizar las transferencias de datos del procesador, forzando al procesador a entrar en wait states hasta que el 8274 esté listo para otro byte de datos.

El 8274 se conecta con el procesador del sistema a través de un bus de datos de 8 bits. Cada canal de E/S serial responde a dos direcciones de memoria, tal como se muestra en la tabla 2.3. El 8274 puede ser configurado para operación en espacio de memoria o en espacio de E/S.

Tabla 2.3

Direccionamiento del 8274

CS	A_1	A_2	Operación de lectura	Operación de escritura
0	0	0	Lectura datos Ch. A	Escritura datos Ch. A
0	1	0	Lectura status Ch. A	Escritura status Ch. A
0	0	1	Lectura datos Ch. B	Escritura datos Ch. B
0	1	1	Lectura status Ch. B	Escritura status Ch. B
1	X	X	Alta impedancia	Alta impedancia

La configuración de hardware del MPSC depende del modo de operación que se haya seleccionado (escrutinio, interrupciones, DMA, o espera).Todas las conversiones serial-a-paralelo, paralelo-a-serial y los chequeos de paridad, requeridos durante transmisiones de E/S seriales y asíncronas, son realizados automáticamente por el 8274.

Cada canal serial de E/S en el 8274 se interfasa a dos lineas de datos, una para transmitir y una para recibir. Durante la transmisión, los caracteres son convertidos de formato de datos paralelo (que es como los proporciona el procesador del sistema, o el dispositivo de DMA) a un tren serial de bits (con bits de comienzo y de fin) y son enviados por la pata de salida TxD. Durante la recepción, un tren serial de bits se recibe por la pata RxD, los bits de enmarcado (comienzo y fin) son eliminados, y el caracter resultante es convertido a formato de datos paralelo y es puesto a disposición del

procesador del sistema o del dispositivo de DMA.

La frecuencia de transmisión o recepción de datos de las líneas seriales es controlada por el reloj del MPSC, en conjunto con el divisor de reloj programado (en el registro WR4). El 8274 está diseñado para permitr que las cuatro líneas seriales (TxD y RxD para cada canal) operen a diferentes frecuencias de transmisión (o recepción). Cuatro entradas de reloj (TxC y RxC para cada canal) están disponibles para esta función.

La información de comandos, parámetros y status se almacena en 21 registros dentro del MPSC (8 registros de escritura y 2 registros de lectura por cada canal, más el registro de vector de interrupciones). Todos estos registros son accesados por medio de los puertos de comando/status para cada canal. Un registro puntero interno selecciona cuáles de los registros de comando o status serán leídos o escritos durante un acceso de comando/status a un canal del MPSC. En la discusión que sigue, los registros de escritura se denominan WR0 a WR7, y los registros de lectura se denominan RR0 a RR2.

Los tres bits menos significativos de WR0 son automáticamente cargados en el puntero de registro cada vez que se escribe a WR0. Luego de un reset, WR0 tiene el valor de cero, para que el primer acceso de escritura a un registro de comando, haga que

los datos sean cargados en WR0 (dándole de esa manera un valor al puntero de registro). Luego de que WR0 se escribe, la siguiente operación de lectura o escritura se hace al registro que selecciona el puntero de registro. El puntero se inicializa luego de que la operación previa de lectura o escritura se completa. De esta manera, leer o escribir a un registro arbitrario del MPSC requiere dos accesos de E/S. El primer acceso es siempre un comando de escritura. Este comando de escritura se utiliza para darle el valor necesario al registro de puntero. El segundo acceso es un comando de lectura o escritura. Nótese que leer RR0 o escribir WR0 no requiere la inicialización del puntero de registro.

2. **Reset.** Cuando la línea de RESET del 8274 es activada, ambos canales del MPSC entran en un estado dormante. Las líneas de salida son forzadas al estado de marca (ALTO), y las señales de interfase de modem (RTS y DTR) son forzadas también a un estado ALTO. Además el puntero de registro asume el valor de cero.

3. **Reporte de errores.** Tres condiciones de error pueden ser encontradas durante la recepción de datos en el modo de transmisión asíncrono.

i. Paridad. Si los bits de paridad son computados y transmitidos con cada caracter y el MPSC está programado para chequear la paridad. un error de paridad ocurrirá cuando el número de bits "1" dentro del caracter no corresponda a la información de par/impar de la bandera de chequeo de paridad.

ii. Enmarcaje (framing). Un error de enmarcaje ocurre si no se detecta un bit de parada inmediatamente después del bit de paridad (si el chequeo de paridad está activado), o inmediatamente después del bit más significativo de los datos (si el chequeo de paridad está desactivado).

iii. Desborde (overrun). Si un caracter de entrada ha sido ensamblado, pero los buffers del receptor están llenos (porque los caracteres previamente recibidos no han sido leídos por el procesador del sistema), un error de desborde ocurrirá. Cuando esto ocurre, el caracter de entrada que se acaba de recibir, sobre-escribirá el caracter inmediatamente anterior a él.

III. DISEÑO DEL HARDWARE DEL SISTEMA

Este capítulo se encarga de describir la estructura de lo que se refiere al diseño del hardware del sistema. Describe en detalle las partes más importantes e interesantes del sistema, y datos de especificaciones, tales como las direcciones disponibles de memoria ROM y RAM, y direcciones de los puertos de E/S.

A. Descripción general del sistema

Un diagrama de bloques del sistema completo se puede apreciar en la figura 3.1.

El sistema diseñado está basado en el microprocesador 80186. Este microprocesador fue escogido, tal como se discutió en el capítulo 2 por su alta integración, lo cual simplifica considerablemente el diseño del sistema al evitar el uso de muchos de los periféricos que trae ya integrados.

El sistema se ha diseñado con 128 Kb de ROM y 128 Kb de RAM. Se escogió RAM estática en vez de RAM dinámica para evitar el uso de un controlador de memoria dinámica. El reloj del sistema correría a 8 MHz.

El sistema se opera desde una terminal serial, la cual se podría conectar al sistema

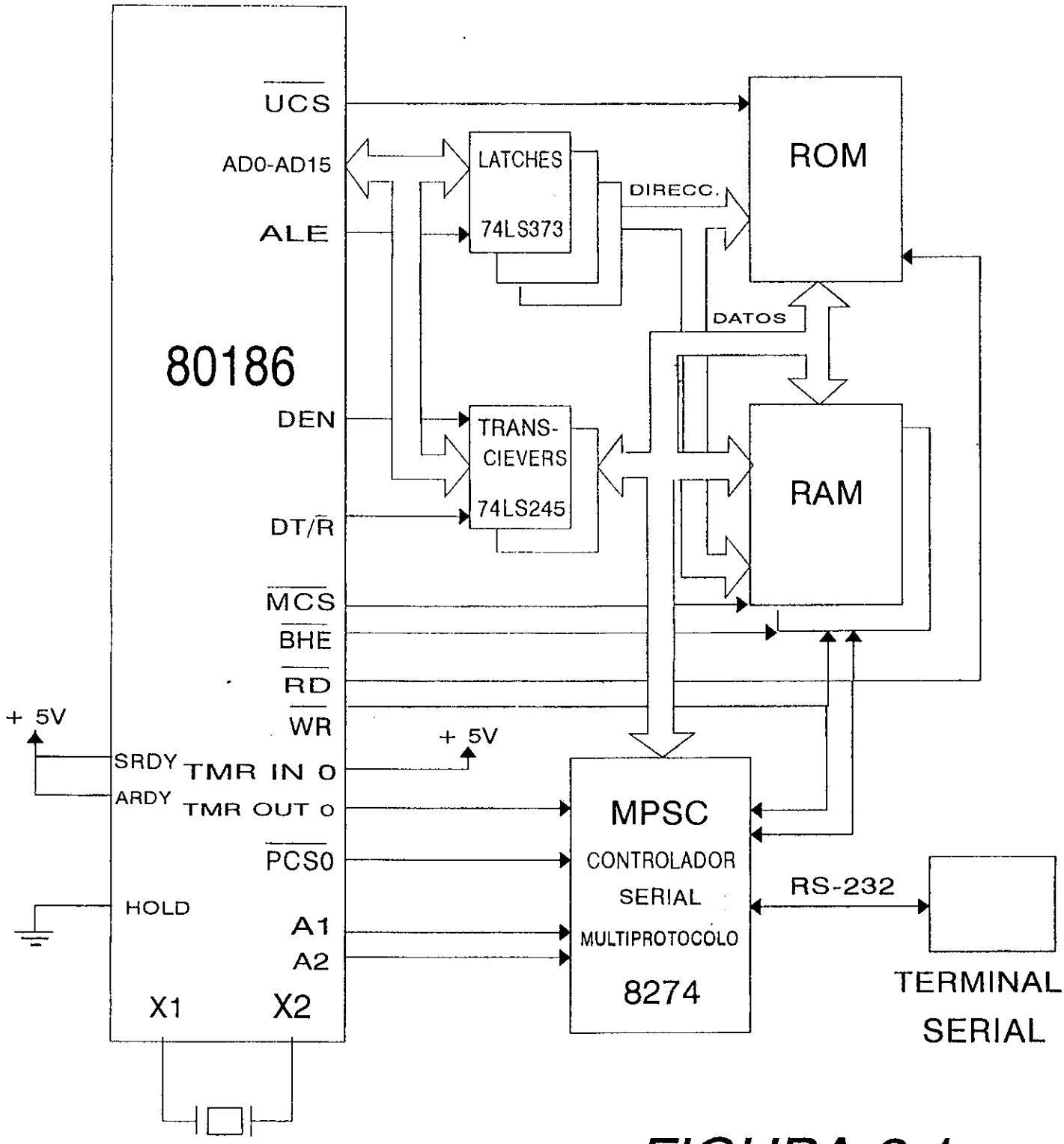


FIGURA 3.1

DIAGRAMA DE BLOQUES

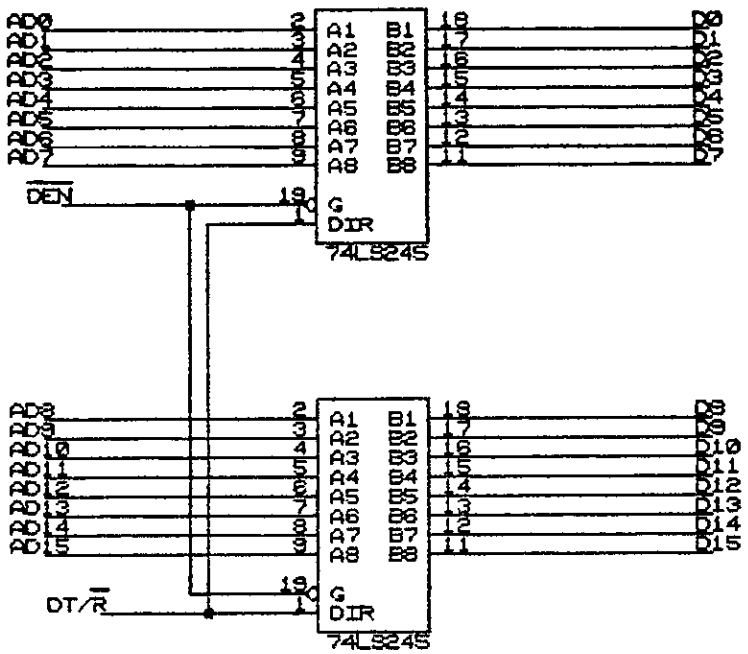
utilizando el protocolo RS-232. Para ello se utilizó un controlador de comunicación serial multi-protocolo: el Intel 8274.

B. Demultiplexación de los buses de datos y direcciones del 80186

Las señales de los buses de datos y direcciones del 80186 están multiplexadas en el tiempo. Estas señales vienen en los pines AD_0 - AD_{15} del microprocesador, siendo estos pines de entrada o salida, y de lógica positiva o activos en ALTO. Cuatro pines más A_{16}/S_3, A_{17}/S_4, A_{18}/S_5 y A_{19}/S_6, contienen otras cuatro señales que también forman parte del bus de direcciones. Estas últimas cuatro señales está multiplexadas en tiempo con las señales de status del ciclo de bus, y son exclusivamente de salida; también son activas en ALTO.

Para la demultiplexación de los buses se utilizaron transcievers bidireccionales, así como latches de 8 bits.

Dos transcievers bidireccionales, con capacidad de 8 bits (74LS245) controlarían lo que corresponde al flujo del bus de datos. Un diagrama de la conexión de éstos se encuentra en la figura 3.2. La dirección del flujo a través de los transcievers puede ser controlada con la señal DT/R del 80186. Esta señal está en BAJO cuando se transfieren datos hacia el 80186, y en ALTO cuando el 80186 envía datos al bus. La señal de



CONEXION DE LOS TRANSCIEVERS BIDIRECCIONALES		
Title: FIGURA 3.2		
Size: A	Document Number: 1	REV
Date: May 15, 1992	Sheet of 1	

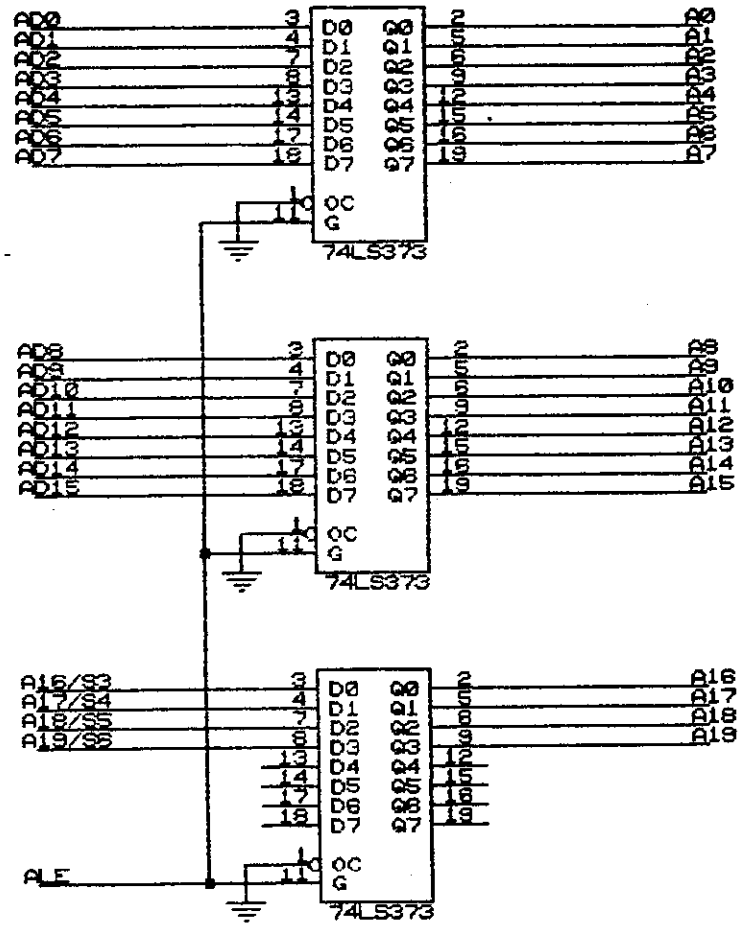
habilitación de los chips (G) es generada por la señal de Data ENable (DEN), que proviene del 80186.

Para el control de las señales de direcciones se emplearon en el presente diseño tres latches de 8 bits (74LS373), cuyas conexiones se muestran en la figura 3.3. La señal de Address Latch Enable (ALE) proveniente del 80186 se utilizó para dar un strobe a los latches en el momento en que las direcciones son válidas.

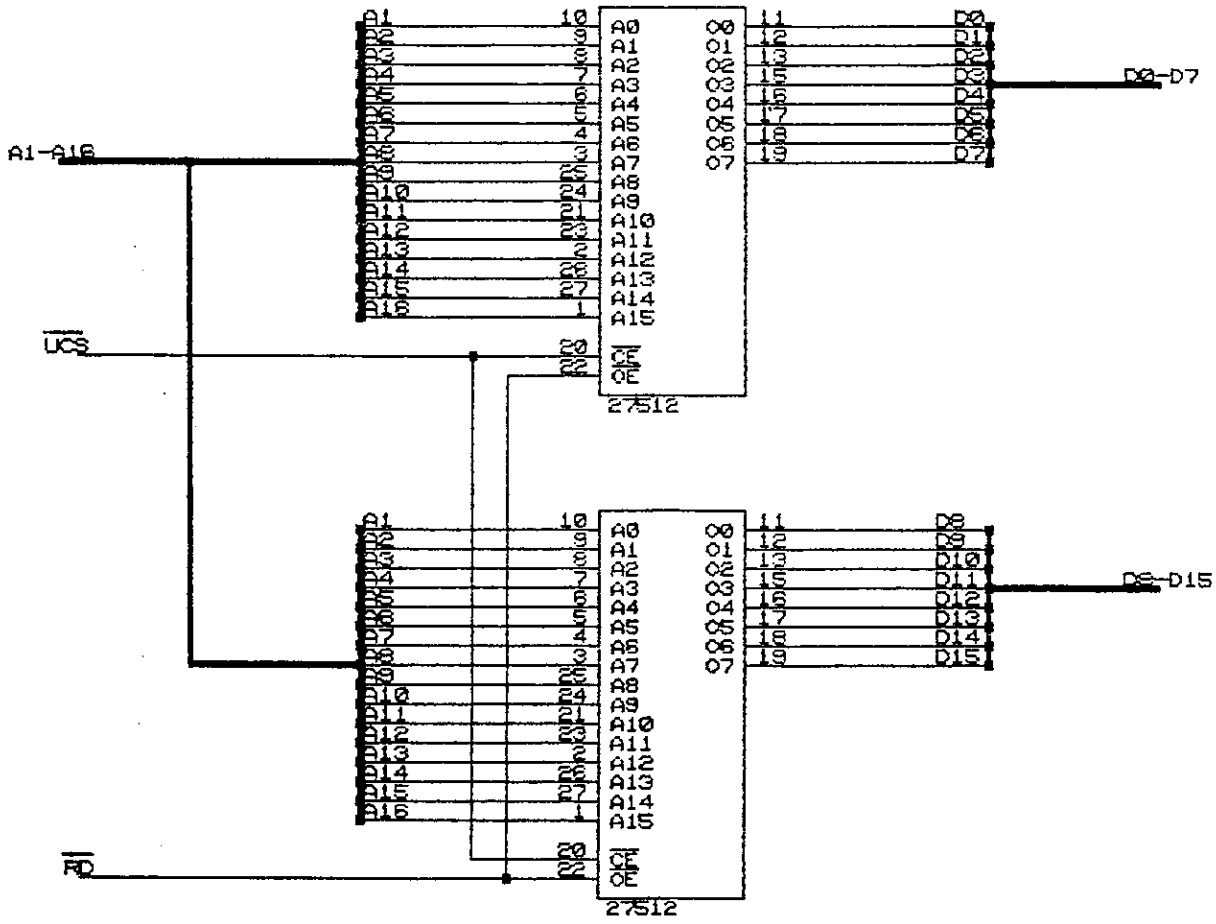
C. Conexión del los chips de ROM

Se habilitaron en el sistema diseñado 128 Kb de memoria de sólo lectura (ROM), utilizándose para ello dos chips EPROMs 27C512 (64 Kb * 8 cada uno).

La conexión de los EPROMs se muestra en la figura 3.4. Nótese que la pata A_0 de cada EPROM está conectada a la línea A_1 de direcciones, y no a la línea A_0. Es importante recordar que A_0 sólo señaliza una transferencia de datos en los 8 bits menos significativos del bus de datos de 16 bits. A la señal de habilitación de salida de los EPROMs (OE) se encuentra conectada la señal de lectura (RD) del 80186. La habilitación de los chips (CE) se conecta a la salida de Upper Memory Chip Select (UCS) proveniente del 80186, la cual se activa en un rango de direcciones programable mediante software. Es importante notar también que durante un ciclo de lectura a esta



CONEXION DE LOS LATCHES		
Title	FIGURA 3.3	
Size A	Document Number	REV
Date: May 22, 1992	Sheet 1 of 1	



CONEXION DE EEPROMS		
Title: FIGURA 3.4		
Size: A	Document Number: 1	REV
Date: May 15, 1992	Sheet of 1	

memoria, ambos chips estarán habilitados, independientemente de si la transferencia que está ocurriendo es de un byte o de una palabra.

D. Conexión de los chips de RAM

En el sistema diseñado se habilitaron 128 Kb de memoria RAM, utilizando 4 chips de memoria estática 60256 (32 Kb * 8 c/u).

Las conexiones de los chips de RAM se muestran en la figura 3.5. La habilitación de los chips se hace en pares, ya que el bus de datos es de 16 bits. El primer par de chips es habilitado por la señal MCS_0, y el segundo por MCS_1. Estas señales de Midrange Memory Chip Select provienen del 80186, y sus rangos de activación son programables mediante software. Tal como se puede notar en el diagrama mostrado (fig. 3.5), las señales de habilitación de chip son combinadas con las señales de BHE (Bus High Enable) y A_0 para determinar si los datos deben ser habilitados en los 8 bits mas significativos, en los 8 menos significativos, o en todos los 16 bits del bus. Estas dos señales, que provienen del 80186, se utilizan como se muestra en la tabla 3.1.

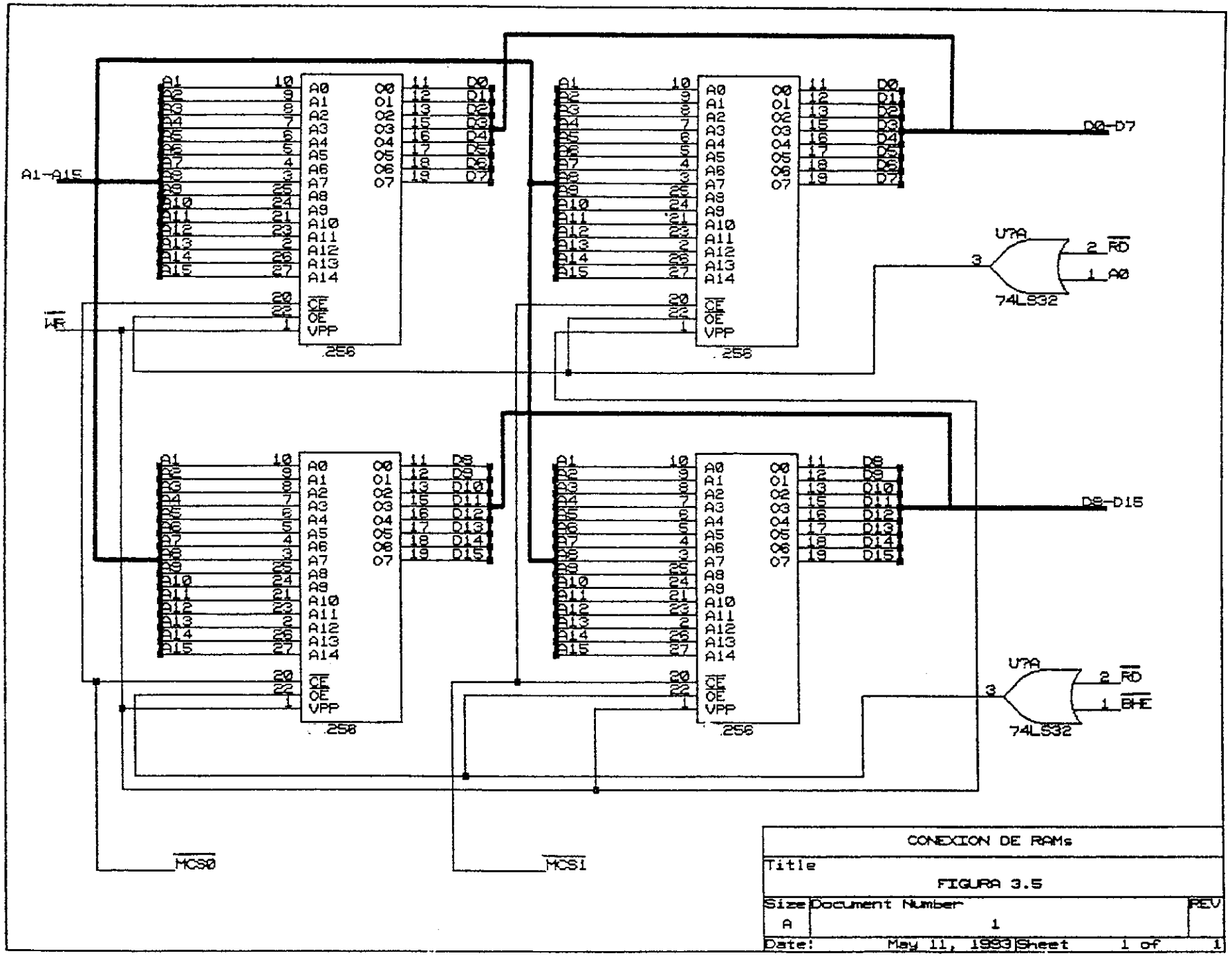

A1-A15
WR
D0-D7
D8-D15
U?A
74LS32
RD
BHE
A0
MCS0
MCS1
256
CONEXION DE RAMs
Title
FIGURA 3.5
Size
A
Document Number
1
REV
Date:
May 11, 1993
Sheet
1 of
1

Tabla 3.1

Utilización de las señales BHE y A_0

Valor de BHE	Valor de A_0	Función
0	0	Transferencia de Palabra
0	1	Transferencia de byte en parte alta (D_{15}-D_8)
1	0	Transferencia de byte en parte baja (D_7-D_0)
1	1	Reservado

Conviene estudiar esto un poco más a fondo. Toda la memoria es accesable byte por byte. Todos los bytes con direcciones pares ($A_0 = 0$) residen en los 8 bits más bajos del bus de datos, mientras que todos los bytes con direcciones impares ($A_0 = 1$), residen en los 8 bits que conforman la parte alta del bus de datos. Cuando se accesa sólo el byte par, A_0 se torna BAJO, y BHE se torna ALTO, y la transferencia de datos ocurre en D_0 - D_7 en el bus de datos. Cuando se accesa sólo el byte impar, BHE se torna BAJO, A_0 se torna ALTO, y la transferencia ocurre en D_8 - D_{15} del bus de datos. Finalmente, si se accesa una palabra en una dirección par, tanto A_0 como BHE se tornan BAJOS y la transferencia de datos ocurre en D_0 - D_{15} en el bus de datos.

Los accesos de palabra son realizados al byte direccionado y al siguiente byte superior. Si accesa una palabra en una dirección impar, deben hacerse dos accesos de byte, el primero para accesar el byte impar en la primera palabra (en D_8 - D_{15}), y el

segundo para accesar el byte par en la siguiente dirección secuencial de palabra (en D_0 - D_7). Por lo anterior, los accesos a palabras en direcciones impares requieren dos ciclos de bus. Es por ello que todos los datos de 16 bits, deben ser localizados en direcciones pares para incrementar la performancia del procesador.

El 80186 siempre carga (hace un "fetch") el flujo de instrucciones en palabras desde direcciones pares, excepto que en la primera carga luego de una transferencia de un programa a una dirección impar obtenga un byte. El procesador desensambla el flujo de instrucciones dentro del mismo procesador, de modo que el alineamiento o localización de las instrucciones no afectará materialmente la performancia del procesador.

E. Direccionamiento de memoria y periféricos de E/S

El 80186 puede soportar directamente hasta 1 Megabyte de memoria en el sistema. Un mapa de direccionamiento del espacio de memoria, así como uno del direccionamiento del espacio de entrada y salida (E/S), se muestra en la figura 3.6.

1. **Direccionamiento de memoria.** Existen dos áreas específicas en el espacio de memoria que son reservadas por el 80186. Estas son:

- El área desde 0h, hasta 3FFh en memoria baja, la cual está reservada para los vectores de interrupción.
- El área desde 0FFFF0h hasta 0FFFFFh en memoria alta, la cual está reservada para el código de inicio, ya que el procesador empieza su ejecución en la dirección 0FFFF0h.

Tomando en cuenta estas áreas, se diseñó el mapeo de memoria a utilizar en el sistema, el cual, como ya se dijo en el capítulo 2, es totalmente programable por software.

La memoria ROM del sistema se encuentra mapeada en los 128 Kb más altos de la memoria, i.e. desde la dirección 0E0000h, hasta 0FFFFFh. Esta es una localización conveniente para la ROM del sistema, ya que incluye la porción donde debe residir el código de inicialización, para reconfigurar todo lo necesario en el sistema en el momento en que éste arranca.

La memoria RAM del sistema, se ha mapeado desde la dirección 0h, hasta la dirección 01FFFFh.

2. **Direccionamiento de periféricos.** El 80186 puede manejar dispositivos periféricos, ya sea utilizando instrucciones de E/S, o instrucciones de memoria (periféricos de E/S direccionados en espacio de memoria). Las instrucciones de E/S permiten que los

dispositivos periféricos puedan residir en un espacio separado de E/S, mientras que los periféricos que están mapeados en espacio de memoria tienen todo el poder del set completo de instrucciones para las operaciones de E/S.

Se pueden definir hasta 64 Kbytes de espacio direccionable para E/S. Para el usuario, este espacio es solamente accesable vía los comandos IN y OUT.

En el espacio de E/S existen también dos áreas reservadas para uso del 80186:

- El área desde 0h hasta 255h, en la cual se encuentra el bloque de control de periféricos,que controla los periféricos integrados en el 80186.
- El área desde 0F8h, hasta 0FFh, la cual se reserva para comunicación con otros productos de INTEL. En la familia del 80186, se utilizan como puertos de E/S para la extensión del coprocesador matemático el 80187.

En el mapa de espacio de E/S para el sistema, presentado en la figura 3.6, podemos notar la localización de las áreas reservadas, así como el área donde se ha mapeado el controlador serial multiprotocolo (MPSC) utilizado en el diseño, el Intel 8274.

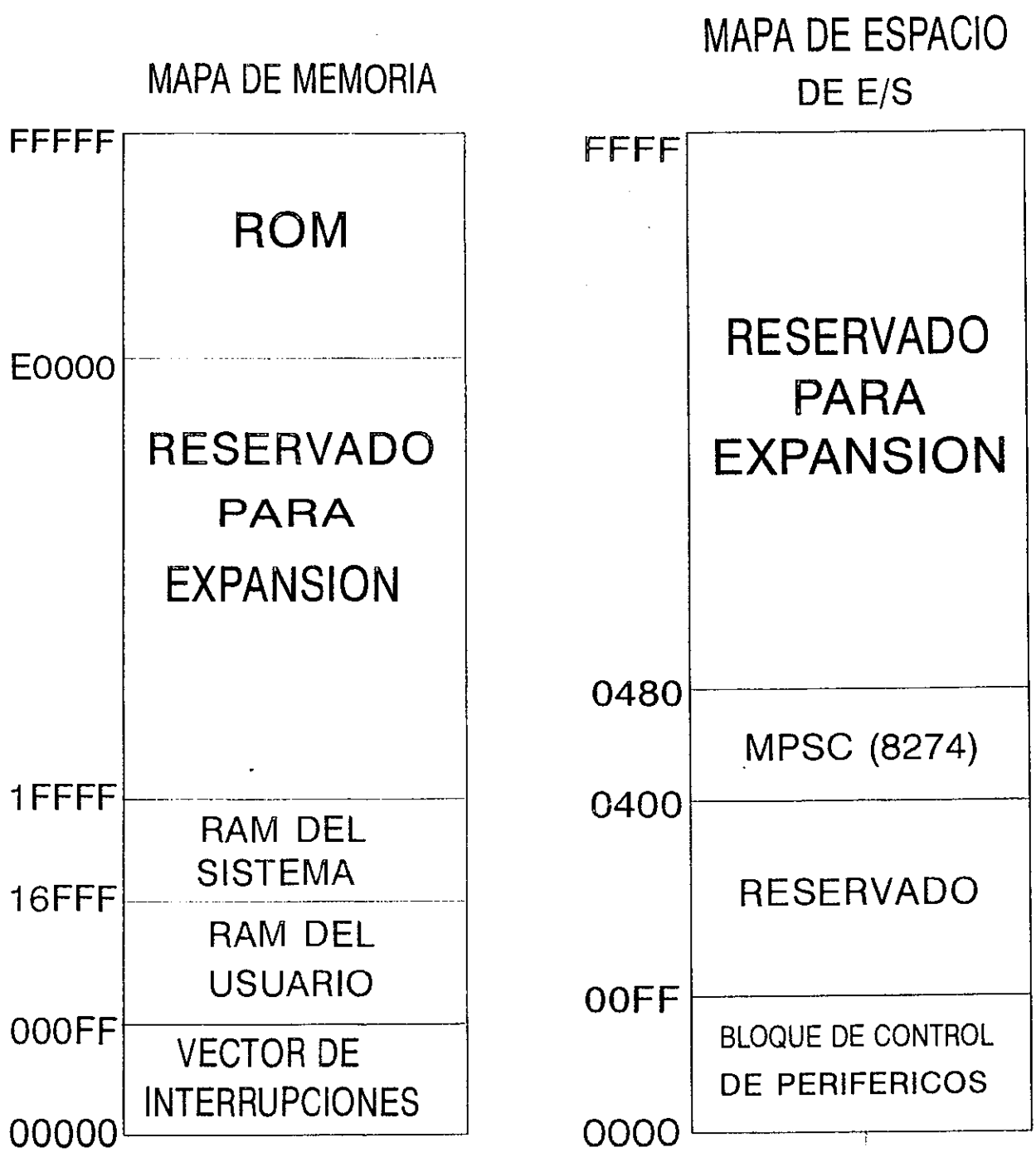


FIGURA 3.6

MAPEO DE MEMORIA Y PERIFERICOS

F. Circuito generador de reloj

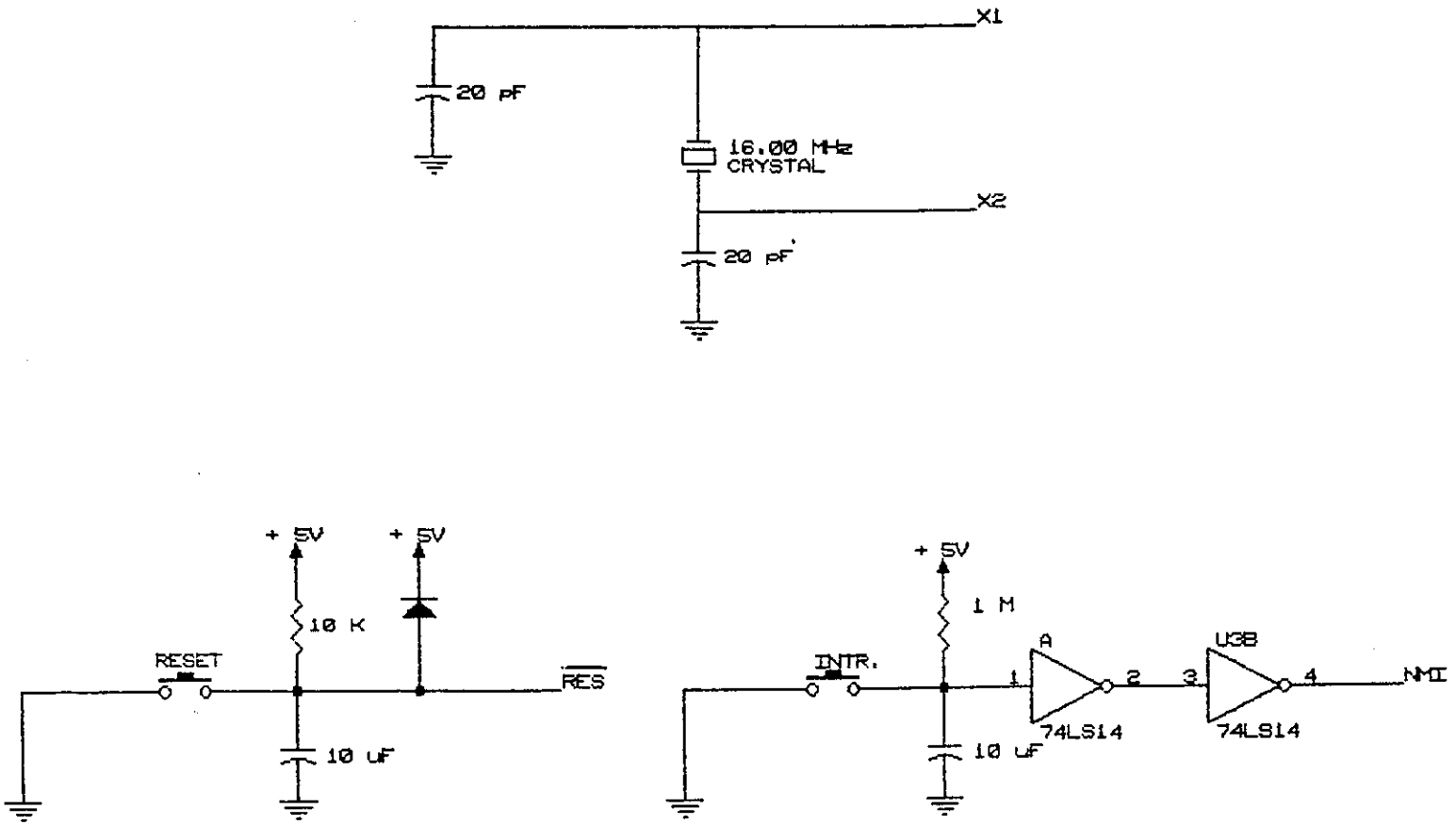
Tal como se mencionó en el capítulo 2, el 80186 cuenta con un circuito interno utilizado para la generación del reloj del sistema. El circuito utilizado en este diseño, se muestra en la figura 3.7.

Todas las referencias de tiempo del procesador son basadas en este reloj, el cual está disponible externamente en el pin CLKOUT, con lo cual podemos sincronizar los otros dispositivos del sistema.

G. Circuito de reset

Otro aspecto ya mencionado en el capítulo 2 es la existencia de una señal de RESET sincronizada en el sistema. Esta señal se genera basada en la entrada RES al 80186. El generador de reloj se encarga de sincronizar estas señales con la señal de CLKOUT.

Un Schmitt trigger en la entrada RES del procesador asegura que una diferencia de voltaje separa los niveles de cambio para los estados lógicos 0 y 1. La histéresis con la que cuenta el procesador es aproximadamente de 600 mV.



CIRCUITO GENERADOR DE RELOJ,
CIRCUITO DE RESET Y CIRCUITO DE NMI

Title
FIGURA 3.7

Size	Document Number	REV
A		

Date: May 8, 1983 Sheet 1 of

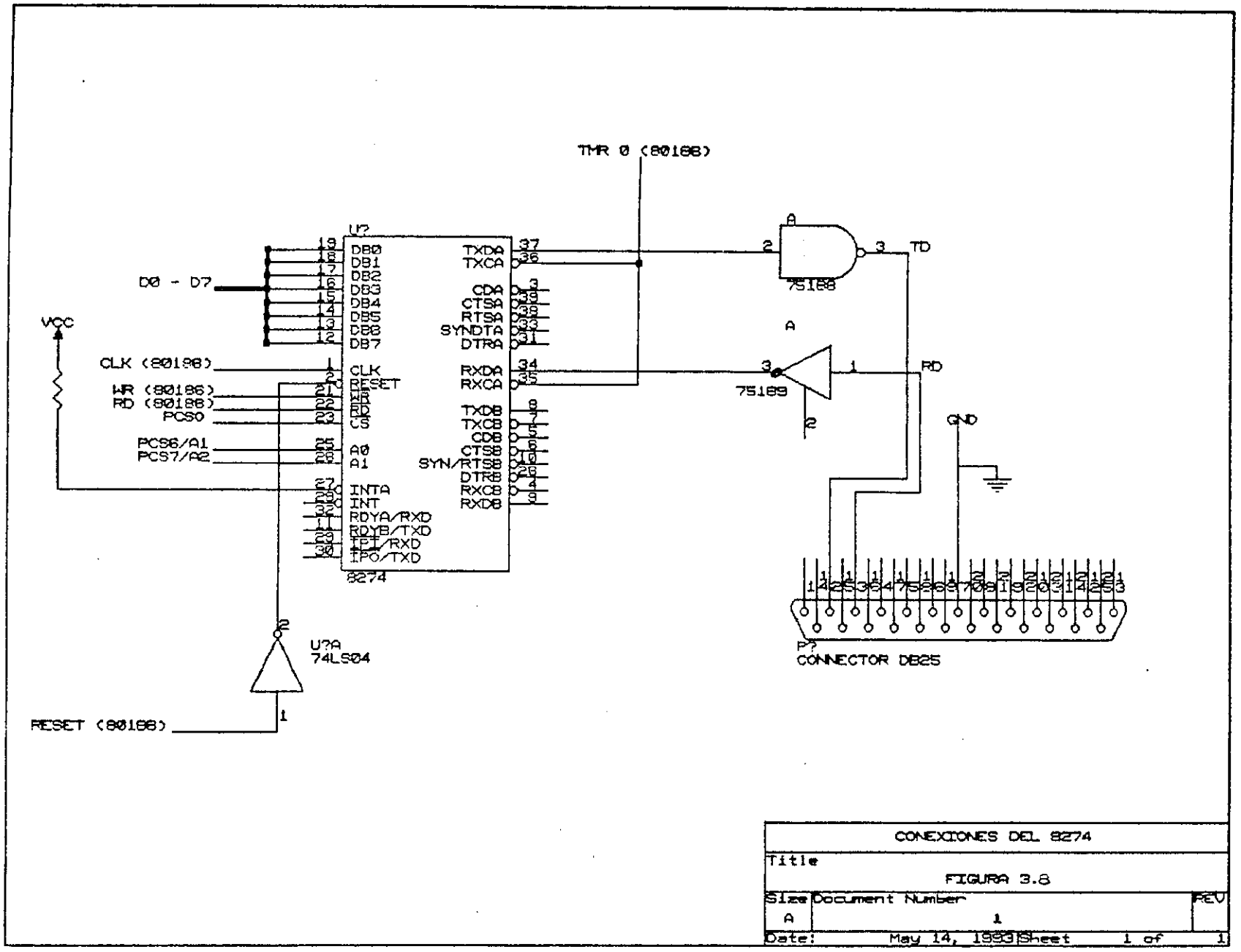
El 80186 debe permanecer en estado de RESET por lo menos cuatro ciclos de CLKOUT, luego que Vcc y CLKOUT se estabilizan. La histéresis permite que la entrada de RES pueda manejarse con un simple circuito RC, tal como se diseñó en el sistema, y se muestra en la figura 3.7.

H. Conexión del 8274 al sistema

El controlador serial multiprotocolo 8274 realiza la comunicación serial asíncrona entre el sistema y un dispositivo RS232, tal como una terminal serial. El sistema transmitiría y recibiría en modo full-duplex (transmisión y recepción simultánea) a una velocidad de 9600 baudios. Se utilizan caracteres de 8 bits de datos con 1 bit de parada, y paridad impar.

La figura 3.8 muestra un diagrama de las conexiones del 8274 y el sistema.

El 8274 es un periférico de 8 bits, por lo que se utilizan en este diseño nada más los 8 bits menos significativos del bus. Se utilizan señales provenientes de los transcievers (D_0-D_7), para tomar el bus de datos ya demultiplexado de las señales provenientes del 80186.


TMR 0 (80188)
U?
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
D0 - D7
CLK (80198)
WR (80186)
RD (80186)
PCS0
PCS6/A1
PCS7/A2
CLK
RESET
WR
RD
CS
A0
A1
INTA
INT
RDYA/RXD
RDYB/TXD
TXDA
TXCA
CDA
CTSA
RTSA
SYNDTA
DTRA
RXDA
RXCA
TXDB
TXCB
CDB
CTSB
SYN/RTSB
DTRB
RXCB
RXDB
8274
VCC
U?A
74LS04
RESET (80188)
A
75188
TD
75189
RD
GND
P?
CONNECTOR DB25
CONEXIONES DEL 8274
Title
FIGURA 3.8
Size
A
Document Number
1
REV
Date: May 14, 1993
Sheet 1 of 1

El controlador serial ha sido mapeado en espacio de E/S (MS en el registro MPCS tiene el valor de 0), tal como se muestra en la figura 3.6, y se encuentra en las direcciones 400h a 479h. La señal de selección de chip para este dispositivo que se utiliza es la PCS0, pues se ha programado la dirección 400h como dirección base para todos los periféricos. Se utilizan PCS5 y PCS6 como señales estabilizadas de A_1 y A_2 (EX en el registro MPCS tiene el valor de 0), tal como se indica en la figura 3.8. En realidad esto no es totalmente necesario, pues tenemos las señales de direcciones estabilizadas a las salidas de los latches, pero se ha hecho por simplicidad.

Para lograr una señal de reloj para las entradas de reloj de transmisión y recepción, TxC y RxC, se utiliza en este diseño uno de los contadores internos del 80186, el timer 0 (ver figura 3.1). El software necesario para generar una velocidad de 9600 baudios, será presentado en el capítulo 4, cuando se discuta el software del sistema. Utilizar los contadores o temporizadores del 80186 de esta forma, nos provee además la flexibilidad de cambiar la velocidad de interface del 8274 con el sistema fácilmente, vía software.

Se utiliza la señal de salida RESET que genera el 80186, para el reset del 8274, pero por ser esta última de lógica negativa, es necesario hacer pasar la señal de RESET por un inversor. Las señales de lectura y escritura (RD y WR) provenientes del 80186 son conectadas a las señales de entrada con los mismos nombres del 8274.

La señal de entrada de acuso de interrupción, INTA (Interrupt Acknowledge), se conecta mediante una resistencia de "pull-up" a Vcc, para hacerla inactiva, pues no se hará trabajar nunca el controlador serial en modo de interrrupción.

Las señales de salida del 8274 al exterior, utilizan el protocolo de comunicación RS232, y por haberse escogido comunicación asíncrona, sólo se conectan 3 señales al sistema exterior. Estas tres señales son transmisión de datos, recepción de datos y tierra común.

1. Convertidores TTL-RS232 y RS232-TTL

El sistema diseñado tiene la capacidad de conectarse a una terminal serial RS232 o al puerto serial de una computadora personal. El puente entre el ésta y el sistema es el controlador serial multiprotocolo 8274. Todo el lado del sistema (y el 8274) trabaja con niveles de voltaje TTL (0 y +5 V). El tren de salida serial del 8274 debe ser convertido a los niveles standard RS232 (0, +12V y -12V), para que puedan ser interpretados correctamente en el otro punto de comunicación. Además, el tren de entrada, proveniente del exterior al 8274, debe ser convertido de niveles RS232 a niveles TTL para ser interpretado correctamente en el sistema.

Por lo anterior, se hace necesario incorporar al diseño convertidores TTL-RS232

y RS232-TTL. Los chips ECG 75188 y ECG 75189, respectivamente realizan estas funciones de una forma bastante sencilla. La conexión de éstos al sistema, se muestra en la figura 3.8, junto con el resto de las conexiones para el 8274.

Es importante ver que la inclusión (necesaria) de estos chips convertidores, especialmente el ECG 75188 nos obliga a incluir dos niveles diferentes de voltaje más. Por lo que el sistema necesita los siguientes niveles de voltaje para su alimentación: 0 V (GND), +5 V, +12 V, -12 V. Utilizando una fuente de poder de una computadora personal, con potencia de 200 W, se puede resolver este problema de una forma sencilla y eficiente.

IV. DISEÑO DEL SOFTWARE DEL SISTEMA

Este capítulo describe el software del sistema, dando una descripción detallada de cada una de las partes del mismo. Esta descripción es útil para comprender los aspectos de operación del sistema. Es recomendable referirse a los manuales publicados por Intel Corporation para una descripción más profunda de las palabras de status y comandos de configuración utilizados.

A. Descripción del programa monitor

El programa monitor, tal como ya fue discutido anteriormente, residiría en los EPROMs del sistema, y está diseñado para operar con una terminal serial o una computadora personal con un puerto serial disponible. Cuando se enciende el poder del sistema, o cuando se presiona el switch de RESET, el 80186 empieza la ejecución del programa, que empieza en la dirección 0E0000h. Las primeras 256 direcciones de memoria RAM (direcciones 0h a 0FFh) han sido reservadas para la tabla de vectores de interrupción del sistema, y el área de pila para el usuario es de 17000h a 1BFFFh. La porción de memoria entre las direcciones 1C000h y 1FFFFh está reservada para ser usada por el programa monitor, por lo que el usuario puede utilizar el rango de memoria de 00100h a 16FFFFh. Tres programas deben ser linkeados para formar el programa monitor. Uno inicializa el 80186 y el 8274,6 y está hecho en lenguaje ensamblador

(assembler), otro contiene las rutinas básicas de E/S para comunicarse con el dispositivo serial (terminal o computadora personal), así como otras rutinas necesarias, y también fue escrito en lenguaje ensamblador. El tercero es el programa principal, que contiene la mayoría de las funciones del programa monitor, y fue hecho en lenguaje C.

El programa puede ser descrito en forma general según el algoritmo que se presenta a continuación:

ALGORITMO:

```
Inicializar el 80186 y el 8274.
Inicializar el vector de interrupciones, los registros de usuarios y breakpoints.
main(){
        while (true) {                          /* loop infinito */
                capte comando();
                interprete comando();
                ejecute comando();
        }
}

capte comando() {
        while ( el caracter de entrada no es carriage return ) {
                chequee si el caracter de entrada es Ctrl-C.
                        si es, ingnore todos los caracteres ingresados y salte
                        al principio del ciclo de while para captar un nuevo comando.
                chequee si el caracter de entrada es back-space,
                        si es, decremente el contador de caracteres, y envíe una
                        señal de back-space.
                guarde el caracter en el buffer.
        }
}
```

```
interprete comando() {
        compare el buffer de caracteres con la tabla para decidir qué comando
        debe ejecutarse.
}

ejecute comando() {
        ejecute el comando.
}
```

B. Inicialización

Esta parte del programa se encarga del código de inicialización de hardware y software del sistema. En esta sección se inicializan el 80186, el 8274 y algunos registros y variables internas.

1. **Inicialización del 80186**. Las señales en el 80186 para la selección de chip, tanto para memoria como para periféricos, son programables vía software, tal como se discutiera en el capítulo 2. Los valores iniciales asignados a los registros internos para programar estas señales son discutidos a continuación:

La señal de selección de chip de memoria alta (UCS), se utiliza para proveer la selección y habilitación de los EPROMs (que contienen el programa monitor). El límite superior para este bloque de memoria es siempre 0FFFFFh, y el límite inferior y el número de WAIT STATES se definen en el registro UMCS. El registro UMCS es asignado con el valor 0E03Ch, lo que nos

programa un bloque de memoria de 128 Kb (de E0000h a FFFFFh), 0 wait states e ignora las señales externas de ready.

El 80186 provee cuatro líneas de MCS (selección de chip para memoria intermedia) que son activas dentro de un bloque de memoria definido. Cada una de estas cuatro líneas de selección de chip es activa por una de cuatro divisiones iguales contiguas del bloque total. El tamaño máximo del bloque es de 128 Kb. En el sistema se utilizan dos bloques de 64 Kb, y se usan dos de las líneas de MCS. La dirección base de los bloques de memoria intermedia se define en el registro MMCS, y en este caso el valor asignado a ese registro es 01FCh. Esto hace que la dirección base de la memoria RAM que se usará sea 0000h, y el rango sea de 00000h a 1FFFFh.

Debe además programarse el registro MPCS, en el cual se indicará el tamaño del bloque de memoria intermedia. La programación de este registro involucra también parámetros para el mapeo de periféricos, tal como se describe en los párrafos siguientes.

El 80186 puede generar señales de selección de chip hasta para siete dispositivos periféricos. Estas señales son activas para siete bloques contiguos de 128 bytes arriba de una dirección base programable.

El registro PACS es programado con el valor 07Ch, lo que provoca que la dirección base

para las señales de selección de periféricos sea 0400h, con 0 wait states, e ignorando las señales externas de ready. El registro MPCS es asignado con el valor 0C03Ch, lo que provoca que el tamaño de los bloques de selección en memoria intermedia sea de 128 Kbytes, los periféricos sean mapeados en espacio de E/S, y se tengan 5 líneas de selección de periféricos (PCS), y las señales estabilizadas A_1 y A_2, tal como se explicó en el capítulo 2.

El registro de relocación es asignado con el valor 0000h, localizando su dirección base en 0000h en espacio de E/S.

Tal como se planteara en el capítulo 3, se utiliza uno de los temporizadores del 80186 como generador de frecuencia de baudios para proveer de señales de reloj de transmisión y recepción al controlador serial utilizado. Para ello se programa el timer 0 para generar continuamente pulsos con un período de 6.5 μs para usar con el controlador serial a 9600 baudios y un divisor de reloj de 16 (en realidad el período necesario es de 6.51 μs), y teniendo el 80186 corriendo a 8 MHz. Para ello se le asigna al registro de cuenta máxima del timer 0 el valor de 13 (4 ciclos / 8 MHz = 500 ns; 500 ns*13 = 6.5 μs). Luego se le asigna el valor 0C001h al registro de control del timer 0, para habilitar el timer, hacer que cuente continuamente y utilice sólo un registro de cuenta máxima.

El listado 1, incluido en el apéndice B, muestra el código en assembler necesario para realizar la inicialización del 80186 que hemos descrito.

2. **Inicialización del 8274**. Para poder operar en modo de transmisión asíncrona, cada canal del 8274 debe ser inicializado de acuerdo con la secuencia que a continuación se presenta. En nuestro caso sólo se hace la inicialización para el canal A, pues es el único que se utiliza.

Primero debe hacerse un reset al canal A, por lo que se escribe 018h en WR0. Luego, deben programarse la el factor de frecuencia de transmisión/recepción, el número de bits de parada, y el tipo de paridad; x16, 1 e impar, respectivamente, en nuestro caso. Para ello se escribe la palabra de control 045h al registro WR4.

Luego, debe programarse el número de bits de los caracteres de datos de recepción (8, en nuestro caso), y debe activarse el receptor del canal. Esto se hace escribiendo 0C1h a WR3.

Debe después programarse el número de bits de los caracteres de datos de transmisión (8), y activarse el transmisor del canal. Para ello se escribe 068h a WR5.

Durante la inicialización es deseable garantizar que las señales estabilizadas de external/status reflejen la información más reciente. Ya que por lo menos dos cambios de estado son almacenados internamente en el 8274, al menos dos comandos de inicialización de señales externas/status deben ser dados. Deben darse, además, comandos de inicialización de señales de error. Este procedimiento es de forma fácil implementado simplemente dando estos comandos cuando se utiliza WR0 para proporcionarle valores a los punteros de qué registro se escribirá

en la próxima operación.

El listado 1, incluido en el apéndice B, muestra el código en assembler necesario para la inicialización del 8274.

C. Rutinas básicas de E/S

Se desarrollaron tres rutinas básicas de soporte para operaciones de Entrada/Salida. Estas tres funciones impr(), pongac() y captec(), son utilizadas para leer o escribir datos a la terminal serial a través del enlace RS232. Los datos que se transmiten deben ser convertidos de números hexadecimales a código ASCII, y los que se reciben deben ser convertidos de código ASCII a números hexadecimales.

La función impr() se utiliza para desplegar una cadena de caracteres (string) en la terminal. El siguiente ejemplo muestra el uso de esta rutina para desplegar un mensaje en la terminal:

```
char *c="Este es un mensaje de prueba";
impr (c);
```

La función pongac() se utiliza para desplegar un caracter en la terminal. Es útil para

caracteres como carriage return, espacio, back-space, etc. El siguiente ejemplo muestra el uso de esta rutina para desplegar un caracter de carriage return (hexadecimal 13) en la terminal:

```
char cr='\13';
pongac(cr);
```

La función captec() se utiliza para leer un caracter de la terminal. Tal como se indicó anteriormente, estos caracteres son recibidos en formato ASCII.

El listado 2, incluido en el apéndice B, muestra el código en assembler utilizado para la implementación de estas rutinas.

D. Comandos del sistema

Existen varios comandos dentro del sistema que son capaces de realizar diversas funciones. Estos serán invocados por el interpretador de comandos. Las rutinas necesarias para la realización de estos comandos fueron escritas en lenguaje C, al igual que el programa principal, que incluye, tal como se describió antes, la captura, interpretación y ejecución de comandos.

Las rutinas que realizan los comandos serán discutidas en detalle en las secciones

siguientes, presentándose donde sea apropiado, el algoritmo de las mismas. El algoritmo no se presenta en algunas de las rutinas, pues por su simplicidad éste es trivial.

1. **Despliegue del contenido de los registros - dr.** Esta función simplemente despliega el contenido de cada uno de los registros. Los registros son almacenados en el vector reg[].

2. **Modificación del contenido de los registros - mr.** Esta función permite al usuario modificar secuencialmente el contenido de los registros.

ALGORITMO:

```
while (m< 14) {
        despliegue el nombre y el contenido del registro[m];
        imprima " > " y espera a que el usuario ingrese el valor
        capte el valor y un caracter para indicar si debe seguir hacia adelante, ir hacia atrás o
terminar la función;
        si el caracter == "^"                   /* ir hacia atrás */
                si m=0 {                        /* mensaje de error */
                        imprima el mensaje de error;
                        continue;
                }
                decremente m;
        else si el caracter == "cr"     /* ir hacia adelante */
                incrementar m;
        else si el caracter = "="       /* quedarse en el mismo reg. */
                continue;
        else si el caracter = "."               /* terminar el loop */
                break;
}
```

3. **Despliegue del contenido de memoria - dm.** Esta función despliega el contenido de memoria en las direcciones especificadas por el usuario. Este debe indicar forzosamente la dirección inicial, siendo la dirección final, opcional. Si no se indica una dirección final, se desplegarán 16 bytes de datos. La dirección final, debe ser mayor o igual que la dirección inicial.

ALGORITMO:

pedir las direcciones inicial y final;
obtener la dirección incial;
si se especificó dirección final,
    obtener la dirección final, e indicar en la bandera que se ha especificado dirección final;
si sólo se especificó dirección inicial,
    desplegar 16 bytes de datos;
si se han especificado dirección inicial y dirección final,
    despliegue los datos entre la dirección inicial y la dirección final.

4. **Modificación del contenido de memoria - mm.** Esta función únicamente necesita una dirección de memoria inicial, en la cual se empezará a modificar los contenidos de la memoria. Esta dirección es la dirección física de la memoria (hasta 5 dígitos hexadecimales en nuestro caso para 20 bits), que combina el valor de segmento y desplazamiento.

ALGORITMO

```
obtener la dirección de inicio;
while (true){                                   /*loop infinito, hasta presionar "." */
        despliegue el contenido de esa dirección de memoria;
        despliegue el prompt " > " y espere a que el usuario ingrese el nuevo valor;
        capte el nuevo valor, y un caracter para indicar si se debe seguir hacia adelante, ir hacia
        atrás, o terminar la función;
        si el caracter == "^"                           /*ir hacia atrás*/
                despliegue la dirección previa;
        si el caracter == "carriage return (cr)"        /*seguir hacia adelante*/
                despliegue la dirección siguiente;
        si el caracter == "="                   /*modificar la misma dirección*/
                despliegue la misma dirección;
        si el caracter == "."                           /*terminar*/
                break;                                  /*sale del loop*/
}
```

5. Colocación de break points - cbp. Esta función utiliza dos vectores, uno de ellos de dos dimensiones, y el otro de una. El primero tablabp[][] guarda la dirección de memoria en donde hay un break point, y el segundo vector, datosbp[], almacena el byte de datos que estaba originalmente en esta dirección. La dirección de los break points se almacena de la siguiente manera: tablabp[i][0] almacena el segmento, y tablabp[i][1] almacena el desplazamiento del i-ésimo break point.

Cuando el usuario coloca el break point, el programa guarda la dirección en tablabp tal como se indicó, y el dato original en datosbp, y sustituye el dato en esa dirección de memoria por 0CCh para INT3.

Se permite un número máximo de break points: cinco. Se utiliza una variable global, b, que almacena el número de break points colocados.

Después de colocar los break points, la rutina se encarga de desplegar los break points colocados al momento en la terminal, en orden ascendente de direcciones.

ALGORITMO

```
si el usuario sólo escribió "cbp", desplegar los break points colocados en ese momento;
si el usuario especifica direcciones para colocar break points {
        chequear si ya hay 5 break points colocados{
                si ya hay 5, se llegó al número máximo permitido: desplegar mensaje y salir de
                la rutina.
        }
        while (hayan caracteres en el buffer){
                while (el caracter no sea espacio ó return){
                        cambie el caracter a binario;
                        combine los números binarios a enteros;
                }
                guarde la dirección del break point en tablabp;
                guarde el dato original de esa dirección en datosbp;
                escriba 0CCh en esa dirección de memoria;
                incremente b.
        }
        ordene todos los break points en orden ascendente de direcciones;
        despliegue todos los break points colocados.
}
```

6. **Remoción de break points - rbp.** Esta función se utiliza para remover break points colocados anteriormente. Remueve las direcciones del vector tablabp y escribe en esas direcciones de memoria, los datos originales, almacenados en el vector datosbp. La variable b, que almacena el número de break points colocados es actualizada.

ALGORITMO

```
si el usuario sólo escribió "rbp", se remueven todos los break points;
si el usuario especifica direcciones para remover break points {
        while (hayan caracteres en el buffer){
                while (el caracter no sea espacio ó return){
                        cambie el caracter a binario;
                        combine los números binarios a enteros;
                }
                elimine la dirección del break point en tablabp;
                regrese el dato original a esa dirección de datosbp;
                decremente b.
        }
        ordene todos los break points en orden ascendente de direcciones;
        despliegue todos los break points colocados.
}
```

7. **Carga de un programa a memoria - load.** Esta función utiliza el protocolo XMODEM para transferir un archivo desde la terminal serial (o computadora personal) al sistema. Una descripción del protocolo XMODEM se encuentra en el apéndice A. La función implementada es la función de recepción de acuerdo a ese protocolo. La función de transmisión la lleva a cabo la terminal o computadora personal.

ALGORITMO:

```
captar la dirección a dónde cargar el programa;
enviar un NAK
while (true)                              /* loop infinito */
        capte el primer campo;
        si es EOT {
                enviar un NAK;
                enviar mensaje que terminó bien;
                terminar;
        }
        si no es EOT ni SOH {
                mensaje de error;
                terminar;
        }
        capte el segundo campo;
        capte tercer campo;
        if (segundo campo XOR complemento de 2 del tercer campo != 0) {
                enviar un NAK;
                continúe;                         /* empezar el loop de nuevo */
        }
        else {
                if (la secuencia está correcta)
                        captar los 128 bytes de datos;
                else if (la secuencia es la correcta menos 1){
                        enviar un NAK;
                        continúe;                 /* empezar el loop de nuevo */
                        }
                else {
                        enviar un CAN;
                        enviar mensaje de error;
                        terminar;
                        }
                if (el checksum es correcto){
                        incrementar la secuencia;
                        enviar un ACK;
                }
```

```
            else {
                enviar un NAK;
                continúe;                    /* empezar el loop de nuevo */
            }
        }
}
```

8. Ejecución de un programa - go. El usuario puede especificar la dirección donde comienza el código que desea ejecutar o utilizar los valores actuales de CS e IP (Segmento de Código y Puntero de Instrucciones, respectivamente), para ejecutar un programa.

Cuando el usuario desea continuar con la ejecución de un programa, luego que éste se ha detenido por un break point colocado en esa dirección, el programa monitor debe restaurar el dato original en esa dirección para ejecutar la instrucción. Luego de ejecutar la instrucción, el programa monitor coloca nuevamente 0CCh en la dirección para mantener el break point, y continúa ejecutando el programa hasta que termina o encuentra otro break point.

9. Ejecución paso a paso (trace) de un programa - t. Esta función se vale del step flag (TF) en la palabra de status para implementar la ejecución paso a paso. Cuando esta bandera está en estado de set, una interrupción INT1 ocurre luego que cada instrucción es ejecutada. La subrutina de interrupción para INT1 guarda los contenidos de todos los registros, y regresa a la función main() para esperar el nuevo comando. Esta función permite al usuario especificar cuántos pasos o cuántas instrucciones quiere ejecutar a la vez.

ALGORITMO:

```
si el usuario especificó número de pasos
        capte el número de pasos (instrucciones) a ejecutar;
si no se especificó el número de pasos
        se ejecuta sólo un paso;
for (el número de pasos a ejecutar) {
        ponga el step flag (TF) en estado "set" (1);
        ejecute la instrucción siguiente;
        si la siguiente dirección tiene un break point
                regrésele su valor (dato) original;
        despliegue el contenido de los registros CS e IP, y el dato en
        esta dirección.
        despliegue todos los registros.
}
```

10. Escritura "manual" a periféricos - iow. Esta función recibe como parámetros la dirección del periférico y el dato a escribir. Simplemente escribe a la dirección dada en espacio de E/S el dato proporcionado.

11. Lectura "manual" a periféricos - ior. Análogamente a la función anterior (escritura), esta función recibe como parámetro la dirección del periférico a leer. Se lee el dato presente en el periférico localizado en la dirección especificada de espacio de E/S.

12. Despliegue de pantalla de ayuda - ?. Esta función proporciona el formato y una explicación muy breve de cada uno de los comandos permitidos.

E. Otras funciones y rutinas

Durante el desarrollo del programa fue necesario implementar muchas rutinas útiles para realizar ciertas funciones y procedimiento. Algunas fueron realizadas en assembler, mientras que la mayoría se implementaron en C. Estas funciones son:

asc2bin() - conversión ascii - binario

bin2asc() - conversión binario - ascii

despl_byte() - desplegar un byte de memoria

desp_int() - despliegue de integers

despl_bp() - despliegue de los break points existentes

ordene_bp() - ordena los break points existentes

ponga_datos() - coloca un byte en la dirección especificada

obtenga_datos() - lee los datos de la dirección especificada

ejecute() - ejecuta el código a partir de la dirección dada

iord() - escribe a un puerto de E/S

iowt() - lee el contenido de un puerto de E/S

F. Subrutinas de interrupciones

El programa monitor utiliza tres subrutinas de interrupción: INT 1, INT 3 y NMI. El

vector de interrupciones se inicializa al principio del programa principal, ejecutándose entonces al encender el sistema o cuando se presiona el switch de RESET.

La rutina INT 1 provee servicio a la interrupción de ejecución paso a paso y comienza en la dirección 0F4000h. Esta rutina hace que todos los registros del 80186 sean salvados y luego regresa al programa principal (main()).

La rutina INT 3 provee servicio a la interrupción de break points, y comienza en la dirección 0F4250h. Esta subrutina es llamada cada vez que el programa se detiene en la ejecución de una instrucción porque hay un break point en ella. Esta rutina salva todos los registros del 80186, y regresa al programa principal.

La rutina NMI empieza en la dirección 0F4300h, y se ejecuta cada vez que se presiona el switch de interrupción INTR. Luego que se ha presionado el switch, el circuito conectado a él provee una señal activa al pin NMI del 80186, generando la interrupción. La rutina hace que todos los registros del 80186 sean salvados para regresar el control luego al programa principal.

V. MANUAL DEL USUARIO DEL SISTEMA

Este capítulo provee la información necesaria para operar el sistema diseñado. Esta información incluye la preparación y conexiones del hardware, configuración necesaria de la terminal (o emulador de terminal) RS-232 conectada al sistema, formato de los comandos con ejemplos y algunos otros procedimientos de operación.

A. Preparación y conexiones del hardware

Para utilizar el sistema, primero debe revisarse que la alimentación de poder al mismo esté conectada. Para encender el sistema utilice el switch maestro de la fuente de poder. Esta fuente de poder provee al sistema con la alimentación de +12V, -12V y +5V. Al encender el sistema, aparecerá el mensaje "** Bienvenido al sistema 80186 **", y aparecerá en la siguiente línea el prompt del sistema ">", listo para aceptar comandos del usuario.

Es necesario además, que el cable de transmisión esté conectado a la terminal serial que será utilizada con el sistema. Este cable debe ser conectado al conector DB-25 que posee el sistema.

El sistema cuenta con un switch de reset y un switch de interrupción. El switch

de RESET permite al usuario terminar cualquier actividad que esté realizando, para retornar el sistema a un estado inicializado (i.e. igual al que se tiene cuando se enciende). Cuando se presiona, al igual que al encender, aparece el mensaje de bienvenida al sistema, y aparece el prompt, listo para aceptar comandos.

El switch de interrupción (INTR.) es usado para generar una interrupción no enmascarable tipo 2 (NMI). Esta rutina interrumpe la actividad que se está realizando en ese momento, guarda los registros del 80186 y regresa el control al programa monitor, el que permitirá al usuario ejecutar los comandos que desee.

B. Configuración de la terminal RS-232 conectada al sistema

La terminal serial se conecta utilizando un enlace RS232 para comunicarse con el sistema. Las conexiones del cable a utilizar corresponden a transmisión asíncrona, y son como se muestran en la figura 5.1.

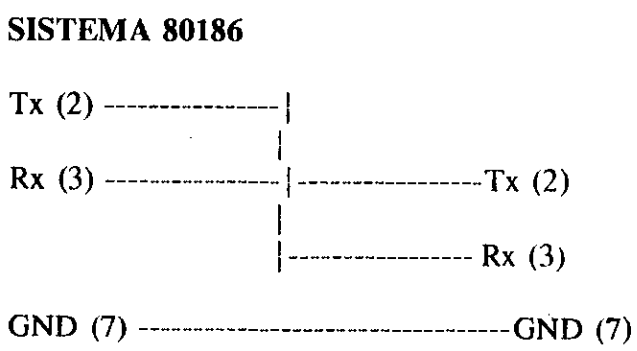


Figura 5.1. Conexión del cable serial

El sistema transmite utilizando ciertos parámetros y configuración que se han escogido para este diseño. Estos parámetros podrían ser cambiados modificando el programa monitor. La tabla 5.1 muestra estos parámetros y sus correspondientes asignaciones.

Es importante que la terminal o el emulador de terminal que se utilice, sea configurado de acuerdo a los parámetros presentados en la tabla 5.1 para la correcta operación del sistema.

Tabla 5.1

Parámetros de comunicación serial del sistema

Parámetro	Valor
Velocidad	9600 baudios
Bits de datos	8
Bits de parada (stop bits)	1
paridad	impar (odd)
modo de comunicación	full duplex

C. Formato y ejemplos de los comandos del sistema

El formato de los comandos utilizados en el sistema, en su forma más general, es como sigue:

> <comando> [<parámetros>](RETURN)

donde:

> es el prompt del sistema

<comando> es el mnemonico del comando

<parámetros> son los parámetros (direcciones u otros)

(RETURN) la tecla de RETURN o ENTER del teclado, presionada para terminar el comando

Es importante señalar que los parámetros en algunos casos pueden ser opcionales, por lo que se usará la notación más común encontrada en manuales técnicos de encerrar en corchetes ([ ]) las partes del comando que sean opcionales.

Todos los parámetros numéricos que se ingresen, con la única excepción del parámetro en el comando de ejecución paso a paso o trace (t), son interpretados en formato hexadecimal.

Todos los comandos deben ser ingresados completamente en letras minúsculas para poder ser interpretados correctamente por el sistema. Los errores cometidos al estar ingresando el comando (siempre y cuando no se haya presionado ENTER todavía), pueden corregirse utilizando la tecla de backspace, que regresa un espacio, o abortando totalmente esa línea con CTRL-C.

En lo que se refiere a la notación de las direcciones, el sistema utiliza segmentos y desplazamientos (offsets), tal como lo hace el microprocesador 80186. Para simplificar el ingreso de los comandos y parámetros, y para una visualización más cómoda de las direcciones, todas deben ingresarse y serán tratadas como direcciones físicas (5 dígitos hexadecimales, para el caso de direcciones de memoria, y 4 para direcciones de E/S). El programa monitor se encargará de hacer la separación de la dirección física en su

correspondiente segmento y desplazamiento, de la manera siguiente:

Si la dirección ingresada es AAAA, entonces el segmento se asume como 0000, y el desplazamiento es tomado como AAAA.

Si la dirección ingresada es BAAAA, el segmento se asume como B000, y el desplazamiento como AAAA.

Los comandos permitidos en el programa monitor se encuentran resumidos en la tabla 5.2, y su uso, junto con ejemplos, será discutido en las siguientes secciones.

Tabla 5.2

Comandos del sistema

Sintaxis	Comando
dr	despliegue del cont. de los registros
mr	modificación del cont. de los registros
dm <dir. inicial> [<dir. final>]	despliegue del contenido de memoria
mm <dir. inicial>	modificación del contenido de memoria
cbp [<dir. 1> <dir. 2> ...]	colocación de break points
rbp [<dir. 1> <dir. 2> ...]	remoción de break points
load <dirección>	cargar un programa
go [<dirección>]	ejecutar un programa
t [<# de pasos>]	ejecutar paso a paso un programa
iow <dir.> <dato>	escribir "manualmente" a perifericos"
ior <dir.>	leer "manualmente" a perifericos"
?	desplegar pantalla de ayuda

1. **Despliegue del contenido de los registros.**

Formato: **dr**

El comando dr despliega el contenido de los registros del microprocesador. El siguiente ejemplo muestra el uso del comando.

Ejemplo:

```
> dr
AX=0000  BX=0000  CX=0000  DX=0000  SP=0000  BP=FFFF  SI=FFFF
DI=0110  DS=0000  ES=0000  SS=1000  IP=0100  FLAG=1004  .....P.
```

2. Modificación del contenido de los registros.

Formato: **mr**

El comando mr es utilizado para modificar el contenido de los registros del microprocesador. El comando empieza desplegando el contenido del registro AX y permite que se modifique el valor. El resto de los registros continúa secuencialmente este mismo proceso en el siguiente orden: AX,BX,CX,DX,SP,BP,SI,DI,DS,ES,SS,CS,IP,FLAG.

Ya dentro del comando, se pueden utilizar ciertos caracteres de control para moverse dentro de la estructura de registros discutida. Estos caracteres son:

(return)	modifica el registro y pasa al siguiente
^	modifica el registro y regresa al anterior
=	modifica el registro y se queda en el mismo

modifica el registro y termina

Si el usuario ingresa más de 4 dígitos para el contenido de un registro, el programa desplegará el mensaje: "*** No ingrese más de 4 dígitos ***".

Ejemplo:

> mr	descripción
AX=0000>12=	cambia AX y se queda en ese mismo
AX=0012>(cr)	pasa al siguiente (BX)
BX=0000>6000(cr)	cambia BX y pasa al siguiente (CX)
CX=0000>5000^	cambia CX y regresa al anterior
BX=6000>(cr)	pasa al siguiente
CX=5000>.	termina
>	

3. Despliegue del contenido de memoria.

Formato: **dm <dirección inicial> [<dirección final>]**

El comando dm es utilizado para desplegar el contenido de un bloque de memoria, que comienza en la dirección inicial y termina en la dirección final. La

dirección final es opcional, y si no se ingresa, se despliega un bloque de 16 bytes de datos, que comienza en la dirección inicial.

La dirección final debe ser mayor que la dirección inicial. Si esta condición no se cumple, el programa desplegará el mensaje : "<*> ERROR: Dir. final debe ser mayor que dir. inic. <*>". Si se ingresan más de 5 dígitos para cualquiera de las direcciones, el programa desplegará el mensaje "<*> ERROR: Dirección inválida. Sólo 0h - FFFFFh <*>".

Ejemplo:

```
> dm 20000 20020
20000   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20010   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20020   FF
> dm 1000
01000   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
>
```

4. Modificación del contenido de memoria.

Formato: **mm <dirección inicial>**

El comando mm permite al usuario modificar los contenidos de memoria en las direcciones especificadas, de una manera interactiva. Existen distintas formas de pasar de una dirección a otra, y son las que se listan a continuación:

(return)	actualizar y pasar a la siguiente dirección
^	actualizar y regresar a la dirección previa
=	actualizar y quedarse en la misma dirección
.	actualizar y terminar

Una vez que se entra al comando, se puede seguir modificando la memoria indefinidamente, hasta que se presiona ".".

Para cada dirección, pueden ingresarse únicamente dos dígitos hexadecimales, de lo contrario, el programa desplegará el mensaje: "** Ingrese exactamente dos dígitos **". Además, si está posicionado en el lugar de memoria 0h, el usuario intenta regresar a la dirección previa ("^"), el programa desplegará el mensaje: "** 00000 es la dirección más baja posible **".

Ejemplo:

	descripción
> mm 20000	empieza a modificar en la dirección 20000h

20000=FF>18= modifica la direcc. 20000h y se queda allí

20000=18>(cr) no modifica esa dirección y pasa a la siguiente

20001=FF>26^ modifica esa dirección y regresa a la direcc. previa

20000=18>(cr) no modifica esa dirección y pasa a la siguiente

20001=26>(cr) no modifica esa dirección y pasa a la siguiente

20002=FF>. no modifica esa dirección y sale del loop, terminando

>

5. Colocación de break points.

Formato: **cbp [<dirección1> <dirección2> ...]**

El comando cbp agrega la o las direcciones dadas a la tabla de break points. Durante la ejecución del programa, una interrupción de break point ocurre inmediatamente antes de la ejecución de cualquier instrucción localizada en una dirección que sea elemento de la tabla de break points. Se permite un máximo de 5 break points. Si el usuario ingresa más de 5 break points, el programa desplegará el mensaje "<*> ERROR: Ya hay 5 break points definidos <*>".

Si el usuario intenta definir un break point que ya había sido definido antes, el programa desplegará el mensaje "<*> ERROR: Ya hay break point en esa dirección

<*>".

Luego de agregar la o las direcciones especificadas como parámetros de este comando, el programa despliega en la pantalla, los break points existentes en ese momento.

Si no se especifican parámetros, el comando simplemente despliega los break points existentes en ese momento.

Ejemplo:

> cbp 01000

breakpoints=01000

> cbp 02000 03000

breakpoints=01000 02000 03000

> cbp

breakpoints=01000 02000 03000

6. Remoción de break points.

Formato: **rbp [<dirección1> <dirección2> ...]**

Este comando es utilizado para remover uno o más break points de la tabla de break points. El uso de este comando es opuesto al del comando de colocación de break points (cbp).

Si se especifica una dirección en la cual no hay break point definido, el programa desplegará en la pantalla el mensaje "<*> ERROR: No hay break point en esa dirección <*>".

Luego de remover el o los break points indicados como parámetros de este comando, el programa despliega en la pantalla, los break points existentes en ese momento.

Ejemplo:

```
> cbp
Breakpoints=01000 02000 03000
> rbp 02000
Breakpoints=01000 03000
> rbp
Breakpoints=
>
```

7. Carga de un programa a memoria.

Formato: **load <dirección>**

Este comando permite al usuario transferir un archivo desde la terminal serial (o computadora personal) a la memoria del sistema, utilizando el protocolo de transmisión de archivos XMODEM.

Para efectuar la transmisión, en la terminal (o computadora personal), luego de invocarse el comando en el sistema, se producirá el mensaje "Listo para hacer el upload ...", entonces debe invocarse la función de transmisión de archivos utlizando el protocolo XMODEM, indicarse la localización del archivo que se desea enviar y proceder con la transmisión.

Si la transmisión se efectúa sin algún problema, el programa desplegará en la pantalla el mensaje "Transferencia finalizada", y regresará al prompt normal", pero si se aborta o se produce algún error, se desplegará el mensaje "Transferencia abortada ... Errores", y regresará al prompt. Si sucede esto último, deberá re-intentarse la transmisión.

Ejemplo:

> load 01000

Listo para hacer el upload ...

[invocar el comando de Tx con XMODEM en la terminal, o paquete de emulación utilizado]

Transferencia finalizada

>

8. Ejecución de un programa.

Formato: **go [< dirección >]**

Este comando permite al usuario correr un programa que empieza en la dirección de memoria dada como parámetro. Si no se especifica una dirección como parámetro, la ejecución del programa empieza con los valores actuales de CS:IP. La ejecución del programa continúa mientras no encuentre un break point, se presione el switch de interrupción (INTR.), se presione el switch de RESET o se apague el sistema.

Ejemplo:

> g 1000

9. Ejecución paso a paso de un programa.

Formato: **t [<pasos>]**

El comando t, o ejecución paso a paso permite al usuario ejecutar un programa instrucción a instrucción. Utilizando un parámetro (que es un número decimal) opcional, también se pueden ejecutar varias instrucciones de una vez. La ejecución comienza en la dirección que apunten CS e IP. Los valores de CS:IP son desplegadados junto con el valor que se encuentra almacenado en ese momento en esa posición de memoria.

Ejemplo:

```
> t
0000:1000 34
AX=0000  BX=0000  CX=0000  DX=0FFF  SP=6FFF  BP=0000  SI=0000
DI=0000  CS=0000  SS=1000  ES=0000  IP=1000  FLAG=F002  .....
>t 2
0000:1010 45
AX=0000  BX=0000  CX=0000  DX=0FFF  SP=6FFF  BP=0000  SI=0000
DI=0000  CS=0000  SS=1000  ES=0000  IP=1013  FLAG=F002  .....
0000:1013 F4
```

AX=0000 BX=0000 CX=0000 DX=0FFF SP=6FFF BP=0000 SI=0000

DI=0000 CS=0000 SS=1000 ES=0000 IP=1014 FLAG=F002 .....

>

10. Escritura "manual" a periféricos.

Formato: **iow <dirección> <dato>**

El comando iow permite enviar un dato de una forma "manual" a una dirección de E/S. Debe especificarse la dirección del periférico y el dato que se desea enviar como parámetros.

Ejemplo:

El siguiente comando envía en valor 0A2h al puerto 0405h

> iow 405 A2

>

11. Lectura "manual" a periféricos.

Formato: **ior <dirección>**

El comando ior permite al usuario leer "manualmente" el valor presente en ese momento en alguna dirección de E/S. Se provee como parámetro la dirección del puerto que se desea leer, y el programa imprimirá entonces el valor del dato presente en ese puerto en ese momento.

Ejemplo:

```
>ior 400
0400: A0
>
```

12. Despliegue de pantalla de ayuda.

Formato: ?

Este comando provee al usuario de información útil para la operación del sistema. Le da una lista de todos los comandos disponibles y la sintaxis correcta de cada uno.

Ejemplo:

```
> ?
dr - desplegar el contenido de los registros
mr - modificar el contenido de los registros
dm <dir. inic.> [<dir. final>] - desplegar el contenido de memoria
mm <dir. inic.> - modificar el contenido de memoria
cbp [<dir.1> <dir.2> ...] - colocar break points (5 maximo)
rbp [<dir.1> <dir.2> ...] - remover break points
load <dir.> - cargar un programa
go [<dir.>] - ejecutar un programa
t [<# de pasos>] - ejecutar un programa paso a paso
iow <dir.> <dato> - escribir manualmente a un periférico
ior <dir.> - leer manualmente a un periférico
? - despliega esta pantalla de ayuda
>
```

VI. CONCLUSIONES Y RECOMENDACIONES

A. Conclusiones

Luego de concluido el presente proyecto, podemos afirmar que el diseño realizado cumple con las especificaciones iniciales, teniendo integrado un microprocesador de 16 bits con 128 Kb de ROM y 128 Kb de RAM, así como una comunicación funcional con una terminal serial.

El sistema diseñado integra el uso de un microprocesador de 16 bits, memoria ROM, memoria RAM y un controlador de comunicaciones seriales, proveyendo al usuario opciones útiles para el aprendizaje y experimentación del uso del microprocesador INTEL 80186 y toda la familia de microprocesadores y periféricos afines a él.

El sistema integra eficientemente, además, el hardware diseñado con el software desarrollado, demostrando el poder que puede alcanzarse al manejar ambos aspectos en un proyecto de electrónica.

El programa monitor cuenta con opciones que facilitan las lecturas y modificaciones a los datos almacenados en memoria y puertos, y permite la ejecución del

código almacenado en memoria tanto en corrida libre como paso a paso. Permite además al usuario transferir archivos desde una computadora personal con un puerto serial. Todo esto facilita la experimentación con el sistema, y permite que se incremente el ritmo de desarrollo del software a usarse con el mismo.

Uno de los objetivos trazados al iniciar el proyecto fue el de realizar un prototipo de la tarjeta diseñada, sin embargo, esto no pudo llevarse a cabo, debido a la falta de equipo adecuado para detección de errores, tal como un analizador lógico, cuyo costo es excesivamente alto. Este tipo de aparatos no forman parte del equipo con que se cuenta actualmente en los laboratorios de la universidad, y fue imposible conseguirlo en el medio.

B. Recomendaciones

El presente trabajo puede servir como un punto de partida para otros proyectos, tanto en el área de hardware como de software, que pueden venir a complementarlo, utilizando otras funciones y capacidades del 80186, o tomar algunas características de él y desarrollar un sistema distinto.

Existen también aspectos funcionales interesantes del 80186 que no fueron explotados en el presente proyecto, tales como la unidad programable de ADM (Acceso

Directo a Memoria), algunas funciones de la unidad de interfase de buses, y algunas características de los temporizadores, que sería interesante explotar en futuros proyectos basados en este microprocesador.

Una ampliación interesante es agregar al sistema otro procesador, que podría ser un coprocesador matemático 8087, lo que obligaría a la inclusión de un controlador/interfase de buses como el 82188, y al manejo de autoridad en los buses.

Otro aspecto que podría modificarse es el utilizar memoria RAM dinámica en vez de memoria RAM estática, lo que obligaría a agregar un controlador de memoria dinámica, como el 8208, y al manejo de memoria dinámica, que requiere un poco más de control que la memoria estática.

Un proyecto interesante en el área de software, es un ensamblador de código para 80186, con todos los chequeos de sintaxis y uso de instrucciones necesarios, que proporcione un listado con el código ensamblado a lenguaje de máquina (hexadecimal), y que además, genere un archivo con todas las características necesarias para que pueda ser transferido directamente a la memoria del sistema utilizando los comandos que éste permite.

Sería de mucho beneficio para el departamento de Ingeniería Electrónica, que la

universidad contara dentro de sus equipos de laboratorio, por lo menos con un analizador lógico, con el cual se hubiera podido desarrollar el prototipo de la tarjeta diseñada en este trabajo. Se sugiere un analizador lógico marca Hewlett Packard modelo HP 1650A ó HP 1650B, por sus características idóneas para el caso, como 80 canales y un amplio soporte para microprocesadores (especialmente el 80186), buses e interfaces.

VII. BIBLIOGRAFIA

16-/32- bit embedded processors. Intel Corporation.
1990 Illinois. 1042 pp.

80C186EB/80C188EB user's manual. Intel Corporation.
1990 Illinois. 302 pp.

Campbell, J. C programmer's guide to serial
1987 communications. Indiana, Howard W. Sams &
Company. 655 pp.

Memory handbook. Intel Corporation. Illinois. 1021 pp.
1990

Microcommunications handbook volume I. Intel
1990 Corporation. Illinois. 889 pp.

Microcommunications handbook volume II; applications.
1990 Intel Corporation. Illinois. 962 pp.

Microprocessors and peripheral handbook volume II:
1989 peripheral. Intel Corporation. Illinois.
1112 pp.

Murray, W.; C. Pappas. 80386/80286 programación en
1987 lenguaje ensamblador. Madrid, Osborne/McGraw-
Hill. 546 pp.

Siegel, C. Teach yourself... C. Oregon, Management
1989 Information Source, Inc. 357 pp.

Smith, J. Advanced turbo C. New York, McGraw-Hill
1989 Publishing Company. 477 pp.

Turbo assembler 2.0 user's guide. Borland International
1988 Inc. California. 503 pp.

Using the 80186/188/C186/C188 application note. Intel
1988 Corporation. California. 89 pp.

APENDICE A

El protocolo XMODEM

Los protocolos para transmisión de bloques de datos son utilizados para una amplia variedad de aplicaciones en comunicaciones seriales, y son empleados en la transmisión asíncrona de archivos enteros. En este contexto, se conocen como protocolos de transmisión de archivos ó "File Transfer Protocols" (FTP).

En casi todos los protocolos asíncronos de transmisión de archivos usados en microcomputadoras, la unidad básica de transferencia es el paquete, que agrupa varios elementos de longitud de 1 byte, llamados campos. Sólo uno de esos campos contiene los datos del archivo, mientras que los otros, conocidos como campos de servicio, contienen la información requerida para el chequeo de errores y verificación de la integridad del paquete. El número y propósito de los campos varía de protocolo a protocolo.

El protocolo de transmisión de archivos XMODEM fué creado en 1977 por Ward Christensen, un programador que ha contribuído con muchos programas útiles para dominio público. Este protocolo es tan ampliamente conocido, que rara vez un programa de comunicaciones es lanzado al mercado para microcomputadoras sin soporte para XMODEM.

Descripción Técnica:

XMODEM es un protocolo simple que utiliza un campo de datos con largo fijo. El valor de chequeo es una suma aritmética de un byte. La figura A.1. muestra la composición de un paquete en este protocolo. Todos los campos, excepto DATA son de 1 byte de largo.

Figura A.1
Composición de un paquete de XMODEM

SOH	Número de secuencia	Complemento del número de secuencia	DATA 128 bytes	Chequeo

Con referencia a la Figura A.1.:

SOH - Es el byte de inicio (Start Of Header), anunciando el primer byte del paquete.

Número de secuencia - Es el número secuencial del paquete correspondiente. El número del primer paquete es 1.

Complemento del número de secuencia - Es el complemento del núumero de secuencia del campo previo.

DATA - El largo del campo de datos es fijo, y es de 128 bytes.

Chequeo - Es una suma aritmética de 1 byte de largo de los contenidos del campo de DATA únicamente.

Transmisión en XMODEM:

FASE INICIAL:

La primera cosa que cualquier protocolo debe hacer es establecer contacto entre el transmisor y el receptor. Un lado siempre asume el papel dominante en esta fase. El protocolo XMODEM es manejado por el receptor; i.e. el receptor es responsable por estimular y mantener el flujo de paquetes. Concordantemente, el papel del transmisor en6 la fase de inicio o sincronización de la transferencia consiste en pacientemente esperar a recibir un caracter ASCII NAK del receptor. Cuando el primer NAK llega, el transmisor lo interpreta como un mensaje de "envíe el primer paquete". La fase inicial concluye al recibirse el primer NAK.

FASE INTERMEDIA:

Una vez que el transmisor ha recibido el primer NAK de inicio, prepara un bloque de 128 bytes de datos, lo transmite, y espera la confirmación de parte del receptor sobre ese paquete. Un caracter ASCII ACK del receptor indica que el paquete fue recibido sin errores, y es interpretado como una solicitud implícita para enviar el próximo paquete; si se recibe un NAK, se interpreta como una solicitud de retransmisión del mismo paquete, y la recepción de un ASCII CAN, termina incondicionalmente la transferencia. Cuando ya no hay más datos que envíar, la fase intermedia de la transmisión se considera completa.

FASE FINAL:

Si la fase intermedia termina normalmente, el transmisor envía un ASCII EOT

para informarle al receptor que no hay más datos que transmitir. El receptor confirma la recepción de este mensaje con un ACK. Si la fase intermedia termina anormalmente, con un CAN del receptor, entonces, no se envía ningún EOT, pues el proceso fue abortado.

Recepción en XMODEM:

El receptor, además de recibir el paquete, debe verificar que el paquete recibido sea el esperado, y que no contiene errores. Esto lo hace basado en la información contenida en los campos de servicio.

FASE INICIAL:

Esta fase consiste completamente de enviar un NAK para anunciar al transmisor que el receptor está listo para recibir datos.

FASE INTERMEDIA:

El receptor ha entrado ahora en el ciclo de recepción propiamente dicho, esperando por los datos durante cierto tiempo, y enviado un NAK nuevamente si no recibe nada. La llegada de un ASCII SOH, simboliza la llegada de un paquete, que el receptor evalúa de la siguiente forma:

1. Un paquete es formalmente identificado comenzando con SOH, pero un EOT en su lugar, es interpretado como signo que no hay más paquetes y que la fase intermedia de la transferencia ha terminado.

2. La integridad del número de secuencia del paquete es chequeada, asegurándose que

los campos segundo y tercero no estén corruptos. Típicamente esto se logra complementando uno de los dos campos, y haciendo un XOR con el otro. Un resultado de 0 indica que ninguno de los dos campos está dañado. Si no son idénticos, el receptor envía un NAK para solicitar la retransmisión de ese paquete.

3. Luego, el receptor se asegura que el número de secuencia sea el que se espera. En general si el número de secuencia es incorrecto, hay algún error. Ya que no hay forma para el receptor de recuperarse de un error de secuencia, un CAN es envíado para abortar la transmisión y la transferencia termina. Se debe hacer una excepción para lo anterior, cuando el número de secuencia es el del paquete anterior. En este caso, el receptor asume que el transmisor nunca recibió el ACK de ese paquete, ignorando el paquete redundante, y enviando otro ACK para solicitar al transmisor que envíe el siguiente paquete.

4. Finalmente, el receptor calcula una suma aritmética del campo de datos únicamente, y lo compara contra el campo de chequeo del paquete. Si los dos concuerdan, el receptor envía un ACK, si no concuerdan el receptor envía un NAK.

FASE FINAL:

Si la transferencia termina correctamente, se envía un ACK para confirmar la recepción del EOT del transmisor, y la transferencia se considera concluída.

APENDICE B

Listados de rutinas en assembler

```
.80186
;********* LISTADO 1: INICIALIZACION DEL 80186

UMCS_REG        EQU     0FFA0h
MMCS_REG        EQU     0FFA6h
PACS_REG        EQU     0FFA4h
MPCS_REG        EQU     0FFA8h
TMR0_CONT       EQU     0FF5Eh
TMR0_MAXCNT     EQU     0FF5Ah
RELOC_REG       EQU     0FFFEh

UMCS_VAL        EQU     0E03Ch
MMCS_VAL        EQU     01FCh
PACS_VAL        EQU     07Ch
MPCS_VAL        EQU     0A3Ch
TMR0_CVAL       EQU     13d
TMR0_MVAL       EQU     0C001h
RELOC_VAL       EQU     00h

MPSC_CONT       EQU     0404h

PUBLIC          main

        .ORG 0FFFF0h

RESTART         JMP     INICIO

        .ORG 0FFF00h

INICIO  MOV     DX,UMCS_REG     ;prog. memoria alta
        MOV     AX,UMCS_VAL     ;128K, 0 WS
        OUT     DX,AX

        MOV     DX,MMCS_REG     ;prog. memoria media
        MOV     AX,MMCS_VAL     ;dir. base = 0h, 0 WS
        OUT     DX,AX           ;bloques de 64K

        MOV     DX,PACS_REG     ;prog. perifericos
        MOV     AX,PACS_VAL     ;dir. base = 0400h, 0 WS
        OUT     DX,AX

        MOV     DX,MPCS_REG     ;perifericos en espacio
        MOV     AX,MPCS_VAL     ;de E/S, A0 y A1 estabi-
        OUT     DX,AX           ;lizadas, 0 WS

        MOV     DX,TMR0_MAXCNT  ;inicializacion y progra-
        MOV     AX,TMR0_MVAL    ;macion del timer 0 para
        OUT     DX,AX           ;utilizarlo como genera-
        MOV     DX,TMR0_CONT    ;dor de baudios (9600)
        MOV     AX,TMR0_CVAL
```

```
            OUT     DX,AX

            MOV     DX,RELOC_REG    ;prog. reg. de relocacion
            MOV     AX,RELOC_VAL    ;dir. base 00h en espacio
            OUT     DX,AX           ;de E/S.

            JMP     MONITOR         ;ejecuta el resto del
                                    ;programa monitor


            .ORG    0E0000h

MONITOR     MOV     DX,MPSC_CONT    ;** INICIALIZACION DEL 8274
            MOV     AX,018h         ;channel reset
            OUT     DX,AX

            MOV     AX,014h         ;reset ext. status
            OUT     DX,AX           ;apunta a WR4

            MOV     AX,045h         ;X16 clock, 1 stop bit,
            OUT     DX,AX           ;odd parity, enable parity

            MOV     AX,013h         ;apunta a WR3
            OUT     DX,AX

            MOV     AX,0C1h         ;Rx 8 bits/char, Rx enable
            OUT     DX,AX

            MOV     AX,015h         ;apunta a WR5
            OUT     DX,AX

            MOV     AX,068h         ;Tx 8 bits/char, Tx enable
            OUT     DX,AX

            MOV     AX,30           ;error reset
            OUT     DX,AX

            JMP     FAR PTR main    ; programa monitor
```

```
.80186
;********* LISTADO 2: RUTINAS DE ENTRADA/SALIDA

EXTERNAL impr
EXTERNAL pongac
EXTERNAL captec
EXTERNAL ponga_datos
EXTERNAL obtenga_datos
EXTERNAL ejecute
EXTERNAL iord
EXTERNAL iowt
EXTERNAL exit_main

PUBLIC   int3main
PUBLIC   main

EXTRN    i_int3  BYTE
EXTRN    i_nmi   BYTE

MPSC_DATA        EQU     0400h
MPSC_CONT        EQU     0404h

                 .ORG    01C000h

USERS_OFF        DWS 1
MAINS_OFF        DWS 1
MAIN_DS          DWS 1
MAIN_ES          DWS 1

                 .ORG    01D000h

REG_AX           DWS 1
REG_BX           DWS 1
REG_CX           DWS 1
REG_DX           DWS 1
REG_SP           DWS 1
REG_BP           DWS 1
REG_SI           DWS 1
REG_DI           DWS 1
REG_CS           DWS 1
REG_DS           DWS 1
REG_SS           DWS 1
REG_ES           DWS 1
REG_IP           DWS 1
REG_FL           DWS 1

;*************** Rutina impr ************************************
;******* Imprime una cadena de caracteres almacenada en memoria,
;******* y terminada con 00h

                 .ORG    0E1000h

                 PROC    FAR
```

```
impr
                PUSH    BP
                MOV     BP,SP
                CLD                             ;Obtener los datos
                MOV     SI,SS:WORD PTR [BP+06h]  ; del stack
IMPR1           MOV     DX,MPSC_CONT            ;Chequear la palabra de
IMPR2           IN      AL,DX                   ;status del 8274,chequear
                AND     AL,04h                  ;Tx Buffer Empty. Si esta
                JZ      IMPR2                   ;vacio se va al loop

                LODSB                           ;Carga el caracter en AL
                CMP     AL,00                   ;Si es 00h, fin.
                JE      FIMPR

                MOV     DX,MPSC_DATA
                OUT     DX,AL                   ;Transmite el caracter
                JMP     IMPR1                   ;Continua con el loop

FIMPR           POP     BP
                RET

;*************** Rutina pongac ********************************
;******* Imprime un caracter enviado como parametro en la
;******* terminal

                PROC    FAR
pongac
                PUSH    BP
                MOV     BP,SP
                MOV     CX,SS:WORD PTR [BP+06h]  ;Obtener los
                                                 ;datos del stack
                MOV     DX,MPSC_CONT            ;Chequear TBE en la
PONGAC1         IN      AL,DX                   ;palabra de status del
                AND     AL,04h                  ;8274. Si esta vacio se
                JZ      PONGAC1                 ;va al loop

                MOV     AL,CL
                MOV     DX,MPSC_DATA            ;Se envia el caracter
                OUT     DX,AL

FPONGAC         POP     BP
                RET

;*************** Rutina captec ********************************
;******* Lee un caracter del teclado, y lo almacena en AL

                PROC    FAR
captec
                PUSH    BP
                MOV     BP,SP

                MOV     DX,MPSC_CONT            ;Chequear Rx Char Avail
```

```
CAPTEC1         IN      AL,DX           ;en la palabra de status
                AND     AL,01h          ;del 8274. Si no hay se
                JZ      CAPTEC1         ;va al loop
                                        ;Luego se imprime el
caracter
                                        ;captado en la terminal
                MOV     DX,MPSC_DATA
                IN      AL,DX           ;se capta el caracter
                MOV     AH,0h
                MOV     BX,AX

                MOV     DX,MPSC_CONT    ;Chequear TBE en la
CAPTEC2         IN      AL,DX           ;palabra de status del
                AND     AL,04h          ;8274. Si esta vacio se
                JZ      CAPTEC2         ;va al loop

                MOV     BX,AX
                MOV     DX,MPSC_DATA    ;Se envia el caracter
                OUT     DX,AL

FCAPTEC         POP     BP
                RET


;*************** Rutina ponga_datos ************************
;******* Coloca un byte de datos en la direccion (Segmento y
;******* Desplazamiento dados

                PROC    FAR
ponga_datos
                PUSH    BP
                MOV     BP,SP

                MOV     DI,SS:WORD PTR [BP+08h] ;Desplazamiento
                MOV     AX,SS:WORD PTR [BP+06h] ;Segmento
                MOV     ES,AX
                MOV     AX,SS:WORD PTR [BP+0Ah] ;Dato
                MOV     ES:[DI],AX

                POP     BP
                RET


;*************** Rutina obtenga_datos ***************************
;******* Obtiene un byte de datos de la direccion (Segmento y
;******* Desplazamiento dados

                PROC    FAR
obtenga_datos
                PUSH    BP
                MOV     BP,SP

                MOV     AX,SS:WORD PTR [BP+08h] ;Segmento
                MOV     CL,0Ch
```

```
                SHL     AX,CL
                MOV     ES,AX
                MOV     DI,SS:WORD PTR [BP+06h] ;Desplaz. en DI
                MOV     BX,ES:[DI]      ;Lee el Dato en esa
direccion

                POP     BP
                RET

;*************** Rutina ejecute ****************************
;******* Ejecuta las instrucciones a partir de la direccion:
;*******  (Segmento y Desplazamiento dados)

                PROC    FAR
ejecute
                PUSH    BP
                MOV     BP,SP

                MOV     SS:MAINS_OFF,SP         ;Almacena el
;valor de SP
                MOV     SS:MAIN_ES,ES
                MOV     SS:MAIN_DS,DS

                MOV     CX,00
                MOV     ES,CX
                MOV     SI,0018h
                LEA     AX,EJEC
                MOV     ES:[SI],AX
                MOV     BX,0F000h
                MOV     ES:[SI+2],BX

                MOV     AX,SS:REG_AX
                MOV     BX,SS:REG_BX
                MOV     CX,SS:REG_CX
                MOV     DX,SS:REG_DX
                MOV     SP,SS:REG_SP
                MOV     BP,SS:REG_BP
                MOV     SI,SS:REG_SI
                MOV     DI,SS:REG_DI
                MOV     DS,SS:REG_DS
                MOV     ES,SS:REG_ES
                PUSH    SS:REG_FL
                PUSH    SS:REG_CS
                PUSH    SS:REG_IP

                IRET

EJEC:           MOV     SS:REG_AX,AX
                MOV     SS:REG_BX,BX
                MOV     SS:REG_CX,CX
                MOV     SS:REG_DX,DX
                MOV     SS:REG_SP,SP
                MOV     SS:REG_BP,BP
```

```
                MOV     SS:REG_SI,SI
                MOV     SS:REG_DI,DI
                MOV     SS:REG_DS,DS
                MOV     SS:REG_ES,ES
                POP     SS:REG_IP
                POP     SS:REG_CS
                POP     SS:REG_FL

                MOV     SP,SS:MAINS_OFF
                POP     BP
                MOV     ES,SS:MAIN_ES
                MOV     DS,SS:MAIN_DS

                RET


;*************** Rutina iord ************************
;******* Lee el contenido del periferico cuya direccion se
;******* especifica

                PROC    FAR
iord
                PUSH    BP
                MOV     BP,SP

                MOV     DX,SS:WORD PTR[BP+06h]
                IN      AL,DX
                MOV     AH,00
                MOV     BX,AX

                POP     BP

                RET


;*************** Rutina iowt ************************
;******* Escribe al periferico se especifica el dato dado como
;******* parametro

                PROC    FAR
iowt
                PUSH    BP
                MOV     BP,SP

                MOV     DX,SS:WORD PTR[BP+08h]
                MOV     AX,SS:WORD PTR[BP+06h]
                OUT     DX,AL

                POP     BP

                RET
```

```
;*************** Rutina exit_main
;*********************************
;******* Salta a la rutina main()

exit_main       PROC    FAR
                JMP     FAR PTR main
                RET


;*************** Rutina INT 1 *********************************
;******* Interrupcion de ejecucion paso a paso (Single Step
;******* Interrupt)

                .ORG 0F4000h

INT1            PROC    FAR

                MOV     SS:REG_AX,AX
                MOV     SS:REG_BX,BX
                MOV     SS:REG_CX,CX
                MOV     SS:REG_DX,DX
                MOV     SS:REG_SP,SP
                ADD     SS:REG_SP,6
                MOV     SS:REG_BP,BP
                MOV     SS:REG_SI,SI
                MOV     SS:REG_DI,DI
                MOV     SS:REG_DS,DS
                MOV     SS:REG_ES,ES

                POP     SS:REG_IP
                POP     SS:REG_CS
                POP     SS:REG_FL

                MOV     ES,SS:MAIN_ES
                MOV     DS,SS:MAIN_DS
                MOV     SP,SS:MAINS_OFF

                POP     BP
                RET


;*************** Rutina INT3 ***************************
;******* Interrupcion de breakpoints (Breakpoints Interrupt)

                .ORG 0F4250h

INT3            PROC    FAR

                MOV     SS:REG_AX,AX
                MOV     SS:REG_BX,BX
                MOV     SS:REG_CX,CX
                MOV     SS:REG_DX,DX
                MOV     SS:REG_SP,SP
```

```
                ADD     SS:REG_SP,#6
                MOV     SS:REG_BP,BP
                MOV     SS:REG_SI,SI
                MOV     SS:REG_DI,DI
                MOV     SS:REG_DS,DS
                MOV     SS:REG_ES,ES

                POP     AX
                DEC     AX
                MOV     SS:REG_IP,AX
                POP     SS:REG_CS
                POP     SS:REG_FL

                MOV     SP,SS:MAINS_OFF
                MOV     ES,SS:MAIN_ES
                MOV     DS,SS:MAIN_DS
                CALL    FAR PTR int3main
                MOV     SS:i_int3,#1

                JMP     FAR PTR  main


;*************** Rutina NMI ************************
;******* Interrupcion de NMI (Boton de interrupcion)

                .ORG 0F43000h

NMI             PROC    FAR

                MOV     SS:REG_AX,AX
                MOV     SS:REG_BX,BX
                MOV     SS:REG_CX,CX
                MOV     SS:REG_DX,DX
                MOV     SS:REG_SP,SP
                ADD     SS:REG_SP,6
                MOV     SS:REG_BP,BP
                MOV     SS:REG_SI,SI
                MOV     SS:REG_DI,DI
                MOV     SS:REG_DS,DS
                MOV     SS:REG_ES,ES

                POP     SS:REG_IP
                POP     SS:REG_CS
                POP     SS:REG_FL

                MOV     SS:i_nmi,#01
                MOV     SP,SS:MAINS_OFF
                MOV     AX,17000h
                MOV     ES,AX
                MOV     SS,AX
                MOV     AX,0F000h
                MOV     DS,AX
                JMP     FAR PTR main
```

APENDICE C

Listados del programa monitor en C

```
/* LISTADO3.C */
/* PROGRAMA MONITOR REALIZADO EN LENGUAJE C */


extern inireg(),                    /* inicializacion de registros */
inibp(),                            /* inicializacion de break points */
iniint(),                           /* inicializacion de interrupciones */
capte_comando(),                    /* captacion de comandos */
interprete_comando(),               /* interpretacion de comandos */
ejecute_comando(),                  /* ejecucion de comandos */
error(),                            /* despliegue de errores */
dm(),                               /* despl. del contenido de memoria */
mm(),                               /* modif. del contenido de memoria */
cbp(),                              /* colocacion de break points */
rbp(),                              /* remocion de break points */
load(),                             /* cargar un programa */
go(),                               /* ejecucion de un programa */
t(),                               /* ejecucion paso a paso de un prog. */
iow(),                              /* escritura manual a perifericos */
ior(),                              /* lectura manual de perifericos */
int3main(),                         /* rutina para la interrupcion INT3 */
help(),                             /* despliega pantalla de ayuda */
despl_byte(),                       /* desplegar un byte de memoria */
despl_bp(),                        /* desplegar los break p. existentes */
ordene_bp(),                        /* ordenar los break p. existentes */
desp_int(),                         /* despliegue de integers */
asc2bin(),                          /* conversion ascii - binario */
bin2asc(),                          /* conversion binario - ascii */
bin2int(),                          /* armado binarios - integer */
impr(),                            /* imprimir una cadena en la terminal */
pongac(),                          /* imprim. un caracter en la terminal */
captec(),                           /* capta un caracter de la terminal */
ponga_datos(),                      /* coloca un byte en direccion dada */
obtenga_datos(),                    /* lee los datos en direccion dada */
ejecute(),                          /* ejecuta el codigo */
iord(),                             /* escribe a un puerto de E/S */
iowt(),                             /* lee de un puerto de E/S */
exit_main();                       /* sale directamente a la rutina main */

char
cr='\13',lf='\10',prompt='\62',ctrl_c='\3',bs='\8',sp='\32',ast='\42',

cero='\48',dos_ptos='\58',igual='\61',punto='\46',soh='\1',eot='\4',
     ack='\6',nak='\21',can='\24';

char com[12][6]={"dr","mr","dm","mm","cbp","rbp","load","go","t",
     "iow","ior","?"};

$ORG 01C700h

short i_error,              /* indice de error. Es 1 si hay error. */
      i_int3,               /* indice de INT3. 1 si ocurrio INT3.  */
      i_nmi,                /* indice de NMI. 1 si ocurrio NMI.    */
      i_pap;                /* indice de ejecucion paso a paso     */
```

```
$END_ORG

$ORG 01D000h

int reg[14];                /* vector de registros del usuario      */
                            /* AX,BX,CX,DX,SP,BP,SI,DI,CS,DS,SS,ES,IP,FL *

int d[5];                   /* buffer para datos o direcciones */

short c[40];                /* buffer de comandos */

int b,                      /* numero de break points               */
    bn,                     /* indice auxiliar para break points    */
    bt,                     /* indice auxiliar para break points    */
    cnt,                    /* contador de caracteres de comando    */
    i,j;                    /* contadores de uso general            */

short datosbp[5];           /* vector de datos de break points      */

unsigned int tablabp[5][2]; /* tabla de break points                */

$END_ORG



main()                              /* programa principal */
{
        int f;
        char *c="** Bienvenido al sistema 80186 **";
        if (i_error == 0 && i_int3 == 0 && i_nmi == 0) {
                inireg();           /* inicializa los registros */
                inibp();            /* inicializa los break points */
                iniint();           /* inic. vectores de interrupcion */
                impr(c);            /* despliega el mensaje */
        }
        pongac(cr);                 /* carriage return */
        pongac(lf);                 /* line feed */
        pongac(prompt);             /* despliega el prompt ">"  */
        i_error=0;                  /* inicializacion de indices */
        i_nmi=0;
        while (1) {                 /* loop infinito */
                capte_comando();
                f=interprete_comando();
                ejecute_comando(f);
        }
}


inireg()                            /* inicializacion de registros */
{
        int l;
        for (l=0;l<14;l++)          /* inicializa todos los registros a 0
                reg[l]=0;           /* con excepcion de SS y SP */
```

```
        reg[4]=0BFFFh;          /* inicializa SP */
        reg[10]=01000h;         /* inicializa SS */
}

inibp()                         /* inicializacion de break points */
{
        int l;
        b=0;                    /* inicializacion de b */
        for (l=0;l<5;l++) {     /* inicializa los vectores */
                tablabp[l][0]=0;
                tablabp[l][1]=0;
                datosbp[l]=0;
        }
}

iniint()                /* inicializac. del vector de interrupciones */
{
        ponga_datos(04000h,04h,0h);     /* inic. vector para INT1 */
        ponga_datos(0F000h,06h,0h);
        ponga_datos(04250h,0Ch,0h);     /* inic. vector para INT3 */
        ponga_datos(0F000h,0Eh,0h);
        ponga_datos(04300h,08h,0h);     /* inic. vector para NMI */
        ponga_datos(0F000h,0Ah,0h);
}

capte_comando()                 /* captacion de comandos */
{
        char ca;
        cnt=0;                  /* inicizliza el contador */
        while ((ca=captec()) != cr || (ca == cr && cnt == 0)) {
                if (ca == ctrl_c) {     /* si es ctrl-c, empezar */
                        pongac(cr);   /* de nuevo */
                        pongac(lf);
                        pongac(prompt);
                        cnt=0;
                        continue;
                }
                else if (ca == bs) {    /* si es back space, */
                        cnt--;          /* regresar un caracter */
                        pongac(bs);
                        if (cnt<0) cnt=0;
                }
                else if (ca == cr && cnt == 0) {
                        pongac(cr);
                        pongac(lf);
                        pongac(prompt);
                }
                else {
                        c[cnt]=ca;      /* se guarda el caracter */
                        pongac(ca);     /* en el buffer, y se des- */
                        cnt++;          /* pliega en la terminal */
                }
        }
        c[cnt]=ca;                      /* se guarda el ultimo */
```

```
}

interprete_comando()                    /* interpretacion de comandos *
{
        for (i=0,j=0;j<=13;j++) {     /* barre el vector de comandos *
                while (c[i] == com[j][i]) {
                        i++;
                        if((c[i] == cr || c[i] == sp) &&
                          com[j][i] == '\0')
                                return(j);
                }
                i=0;        /* inic. para el prox. comando */
        }
        return ('\42');  /* "*" para comando no existente */
}

ejecute_comando(f)                       /* ejecucion de comandos */
int f;
{
        pongac(cr);
        pongac(lf);
        switch(f) {
                case 0:
                        dr();           /* despl. cont. de registros *
                        break;
                case 1:
                        mr();           /* modificacion de registros *
                        break;
                case 2:
                        dm();            /* despl. cont. de memoria */
                        break;
                case 3:
                        mm();            /* modif. contenido memoria *
                        break;
                case 4:
                        cbp();           /* colocar break points */
                        break;
                case 5:
                        rbp();           /* remover break points */
                        break;
                case 6:
                        load();          /* carga de un prog. a mem. *
                        break;
                case 7:
                        go();            /* ejecucion de un programa *
                        break;
                case 8:
                        t();             /* ejecucion paso a paso */
                        break;
                case 9:
                        iow();          /* escritura manual a perif. *
                        break;
```

```
                case 10:
                        ior();          /* lectura manual a perif. */
                        break;
                case 11:
                        help();         /* pantalla de ayuda */
                        break;
                default:
                        error(1);       /* mensaje de error */
        }
        pongac(prompt);
}


error(x)                                /* despliegue de errores */
int x;
{
        char *e1 = "<*> ERROR: Comando Inexistente <*>",
             *e2 = "<*> ERROR: Ya hay 5 break points definidos <*>",
             *e3 = "<*> ERROR: Caracter invalido. Solo 0-9 y A-F <*>",
             *e4 = "<*> ERROR: Ya hay break point en esa direccion <*>",
             *e5 = "<*> ERROR: No hay break point en esa direccion <*>",
        *e6 = "<*> ERROR: Direccion invalida. Solo 0h - FFFFFh <*>",
        *e7 = "<*> ERROR: Dir. final debe ser mayor que dir. inic. <*>";

        pongac(cr);
        pongac(lf);
        i_error=1;
        switch(x) {
                case 1:
                        imprim(e1);
                        break;
                case 2:
                        imprim(e2);
                        break;
                case 3:
                        imprim(e3);
                        break;
                case 4:
                        imprim(e4);
                        break;
                case 5:
                        imprim(e5);
                        break;
                case 6:
                        imprim(e6);
                        break;
                case 7:
                        imprim(e7);
                        break;
        }
        exit_main();
}
```

```
dr()                                        /* despliegue de registros */
{
        static char *pp[14]={"AX=","BX=","CX=","DX=","SP=","BP=","SI"
                             "DI=","CS=","DS=","SS=","ES=","IP=","FLAG="};
        static char *po[12]={"C"," ","P"," ","A"," ","Z","S","T","I",
                             "D","O"};
        char *spp="   ";
        int d1,l,k;
        for (l=0;l<14;l++) {                /* desplegar los valores */
                imprim(pp[l]);
                d1=reg[l];
                desp_int(d1);
                if (l == 7) {               /* otra linea */
                        pongac(cr);
                        pongac(lf);
                }
                else imprim(spp);
        }
        d1=reg[13];                         /* FLAG */
        for (l=11;l>=0;l--) {
                if (l !=5 && l!=3 && l!=1) {
                        k=((d1>>l) & 01);
                        if (k == 1) imprim(po[l]);
                        else pongac(punto);
                }
                else pongac(sp);
        }
        pongac(cr);
        pongac(lf);
}

mr()                                        /* modificacion de registros
{
        static char *pp[14]={"AX=","BX=","CX=","DX=","SP=","BP=","SI"
                             "DI=","CS=","DS=","SS=","ES=","IP=","FLAG="};
        char *dig_4="*** No ingrese mas de 4 digitos ***",
             *mal_c="*** Comando erroneo ***",
             *prim="** AX es el primero, no puede ir mas atras **";
        int d1,v;
        int h,k,l,dd[5];
        l=0;
        while (l<14) {
                imprimp(pp[l]);
                d1=reg[l];
                desp_int(d1);     /* da el valor actual del registro
                pongac(prompt);
                k=0;
                dd[k]=captec();
                while (dd[k]!=cr && dd[k]!=punto && dd[k]!=94 &&
                   dd[k]!=igual) {
                        if (dd[k] == bs) {
                                pongac(bs);
                                k--;
```

```
        }
        else {
                pongac(dd[k]);
                k++;
        }
        dd[k]=captec();
}
pongac(dd[k]);
if (k > 4) {
        pongac(cr);pongac(lf);
        imprim(dig_4);    /* error: mas de 4 digitos */
        goto mr_prox;
}
if (dd[0] == punto) goto mr_fin;  /* finalizar */
if (dd[0] == cr) {              /* siguiente registro */
        if (l == 13) goto mr_fin;
        l++;
        goto mr_prox;
}
if (dd[0] == 94) {              /* "^" reg. anterior */
        if (l == 0) {
                pongac(cr);pongac(lf);
                imprim(prim);   /* error: esta en AX */
                goto mr_prox;
        }
        l--;
        goto mr_prox;
}
for (h=0;h<k;h++) {              /* conv. a binario */
        d[h]=asc2bin(dd[h]);
        if (d[h] == ast)
                error(3);       /* caracter invalido */
}
reg[l]=bin2int(k);
switch(dd[k]) {
        case 13:                 /* return */
                l++;
                break;
        case 46:                 /* "." */
                goto mr_fin;
        case 94:                 /* "^" */
                if (l == 0) {
                        pongac(cr);
                        pongac(lf);
                        imprim(prim);
                        break;
                }
                l--;
                break;
        case 61:                 /* "=" */
                break;
        default:
                imprim(mal_c);
}
```

```
mr_prox:        pongac(cr);pongac(lf);
        }
mr_fin: pongac(cr);pongac(lf);
}


dm()                                        /* despliegue de memoria */
{
        short s,*pml,*pm2;
        int a,g,f,k=3,l,dl,d2,flag=0,m,n,msbl,msb2,sr;
        while (c[k] == sp) k++;             /* ignorar los espacios */
        for (l=0;c[k]!=sp && c[k]!=cr;l++,k++) {
                d[l]=asc2bin(c[k]);
                if (d[l] == ast)            /* caracter invalido */
                        error(3);
        }
        if (l > 5) error(6);                /* direccion invalida */
        m=5-l;
        for (n=0;n<m;n++) pongac(cero);
        for (n=0,k=3;n<l;n++,k++) pongac(c[k]);
        pongac(dos_ptos);
        pml=bin2int(l);                     /* direccion inicial */
        if (l == 5 & d[0] != 0) msbl=d[0];  /* esta en otro segmento */
        else msbl=0;
        if (c[k] == sp) {                   /* si hay direccion final, */
                k++;                        /* obtengala */
                flag=l;
                while (c[k] == sp) k++; /* ignorar los espacios */
                for (l=0;c[k]!=cr;l++,k++)
                        d[l]=asc2bin(c[k]);
                if (l == 5 & d[0] != 0) msb2=d[0]; /* otro segmento */
                else msb2=0;
                pm2=bin2int(l);
        }
        if (flag == 0) {                /* solo se especifico direcc. */
                for (f=0;f<16;f++) {    /* inicial */
                        despl_byte(pml,msbl);
                        pml++;                  /* siguiente direcc. */
                        if (pml == 0) msbl++;
                }
        }
        if (flag == 1) {        /* especificada tambien direcc. final */
                m=(pm2-pml)/16;
                n=(pm2-pml)%16 + 1;
                sr=msb2-msbl;
                if (sr < 0 || (sr == 0 && m < 0) || (sr == 0 &&
                   m == 0 && n <= 0))
                        error (7); /* direcc. inicial > direcc. final */
                else if ((m == 0) && (n > 0) && (sr == 0))
                        for (f=0;f<n;f++) {     /* rango menor de 16 */
                                despl_byte(pml,msbl);
                                pml++;
                                if (pml == 0) msbl++;
                        }
```

```
                else {
                        if (m == 0) m=sr*01000h;
                        for (k=0;k<m;k++) {     /* primeros 16 */
                                for (f=0;f<16;f++) {
                                despl_byte(pml,msbl);
                                pml++;
                                if (pml == 0) msbl++;
                                }
                        if (k < m || n != 0) {  /* si hay mas datos */
                                pongac(cr);   /* direcc. de comienzo */
                                pongac(lf);
                                s=bin2asc(msbl);
                                pongac(s);
                                desp_int(pml);
                                pongac(dos_ptos);
                                }
                        }
                if (n != 0)                          /* el resto */
                        for (f=0;f<n;f++) {
                                despl_byte(pml,msbl);
                                pml++;
                                if (pml == 0) msbl++;
                        }
                }
        }
        pongac(cr);
        pongac(lf);
}


mm()                                        /* modificacion de memoria */
{
        short s,mdat,*pm;
        int dd[6],l,k=3,mmsb;
        int a,g,m,n;
        char *me1="** Entrada incorrecta **",
             *me2="** 00000 es la direccion mas baja posible **",
             *me3="** Ingrese exactamente dos digitos **";
        for (l=0;c[k]!=cr;k++,l++) {    /* obtener direcc. de inic. */
                d[l]=asc2bin(c[k]);
                if (d[l] == ast) error(3);      /* caracter invalido */
        }
        if (l > 5) error (6);                    /* mas de 5 digitos */
        m=5-l;
        for (n=0;n<m;n++) pongac(cero);
        for (n=0,k=3;n<l;n++,k++)
                pongac(c[k]);
        pm=bin2int(l);
        if (l == 5 && d[0] != 0) mmsb=d[0];
        else mmsb=0;
        while(l) {                  /* loop infinito hasta presionar . */
                pongac(igual);
                despl_byte(pm,mmsb);    /* dato actual */
```

```
pongac(prompt);
l=0;
dd[l]=captec();
while(dd[l] != cr && dd[l] != punto && dd[l] != 94 &&
  dd[l] != igual) {
        if (dd[l] == bs) {      /* back space */
                pongac(bs);
                l--;
        }
        else {
                pongac(dd[l]); /* despl. en pantalla
                l++;
        }
        dd[l]=captec();          /* proximo caracter
}
pongac(dd[l]);                  /* despl. en pantalla
if (l == 1 || l >= 3) {         /* mas de 2 digitos o
        pongac(cr);              /* uno solo */
        pongac(lf);
        impr(me3);             /* impr. mensaje y deja
        goto sigm;              /* ingresarlo de nuevo
}
if (dd[0] == punto) goto finm;
if (dd[0] == cr) {
        pm++;
        if (pm == 0) mmsb++;
        goto sigm;
}
if (dd[2] == 94) {                      /* "^" */
        if (pm == 0) {
                if (mmsb == 0) {
                        pongac(cr);
                        pongac(lf);
                        impr(me2);
                        goto sigm;
                }
                mmsb--;
        }
        pm--;
        goto sigm;
}
d[0]=asc2bin(dd[0]);
d[1]=asc2bin(dd[1]);
mdat=bin2int(2);
a=mmsb<<12;
ponga_datos(mdat,pm,a);    /* guardar el nuevo valo
if(dd[2] == 13) {
        pm++;
        if (pm == 0) mmsb++;
}
else if (dd[2] == 46)                   /* "." */
        goto finm;
else if (dd[2] == 94) {
        if (pm == 0) {
```

```
                                if (mmsb == 0) {
                                        pongac(cr);
                                        pongac(lf);
                                        impr(me2);
                                        goto sigm;
                                }
                                mmsb--;
                        }
                        pm--;
                }
                else if (dd[2] == 61)                   /* "=" */
                        goto sigm;
sigm:           pongac(cr);
                pongac(lf);
                s=bin2asc(s);
                desp_int(pm);
        }
finm:   pongac(cr);
        pongac(lf);
}


cbp()                                   /* colocar break points */
{
        int x,y,z;
        if (i == cnt)           /* si no hay parametros, desplegar */
                despl_bp();     /* los break points actuales */
        else {
                if (b >= 5)     /* ya hay 5 break points */
                        error (2);
                x=4;
                while (x <= cnt) {
                        if (b >= 5)         /* ya hay 5 break points */
                                error (2);
                        while (c[x] == sp)
                                x++;        /* ignorar los espacios */
                        if (x == cnt)       /* se llego al final */
                                goto cbp_fin;
                                            /* obtener las direcciones */
                        for (y=0;c[x]!=sp && c[x]!=cr;y++,x++) {
                                if (c[x] == cr) break;
                                d[y]=asc2bin(c[x]);
                                if (d[y] == ast) /* caracter invalido */
                                        error(3);
                        }
                        tablabp[b][1]=bin2int(y);
                        if (y == 5 && d[0] != 0) /* byte mas signif. */
                                tablabp[b][0]=(d[0] << 12);
                        else tablabp[b][0]=0;
                        for (z=0;z<b;z++)
                                if (tablabp[z][0] == tablabp[b][0] &&
                                  tablabp[z][1] == tablabp[b][1])
                                     error(4); /* ese b.p. ya existe */
```

```
                                        datosbp[b]=obtenga_datos(tablabp[b][0]
                                        tablabp[b][1]); /* guarda datos orig. */
                                        ponga_datos(0CCh,tablabp[b][1]
                                          tablabp[b][0]); /* pone 0CCh */
                                        b++;    /* incr. el contador de b.p. */
                                        x++;    /* proximo caracter */
                        }
                        ordene_bp();
                        despl_bp();
                }
cbp_fin:        pongac(cr);
                pongac(lf);
}


rbp()                                           /* remover break points */
{
        int a,a1,a2,a3,ah,k=4,l,n,flag;
        char *p="breakpoints=";
        if (i == cnt) {                         /* si no hay parametros, */
                for (l=0;l<b;l++) {             /* remover todos los b.p. */
                        a1=datosbp[l];
                        a2=tablabp[l][1];
                        a3=tablabp[l][0];
                     ponga_datos(a1,a2,a3); /* regresa el dato orig. */
                }
                b=0;
                imprim(p);
        )
        else {                                  /* hay parametros especificados */
                while (k <= cnt) {
                        while (c[k] == sp)
                                k++;    /* ignorar los espacios */
                        for (l=0;c[k]!=sp && c[k]!=cr;l++,k++) {
                                d[l]=asc2bin(c[k]);
                                if (d[l] == ast) /* caracter invalido */
                                        error(3);
                        }
                     if (l == 0 & c[k] == cr) break; /* era el ultimo */
                        a=bin2int(l);
                        if (l == 5 && d[0] != 0) /* byte mas signif. */
                                ah=(d[0] << 12);
                        else ah=0;
                        flag=0;
                        for(n=0;n<b;n++)
                                if (tablabp[n][0] == ah &&
                                tablabp[n][1] == a) { /* si coinciden */
                                        ponga_datos(datosbp[n],a,ah);
                                        tablabp[n][0]=0F000h;
                                        tablabp[n][1]=0FFFFh;
                                        flag=1;
                                        ordene_bp();
                                      b--; /* dism. contador de b.p. */
                                }
```

```
                        if (flag == 0)    /* no hay bp en la direcc. */
                                error(5);     /* dada como parametro */
                        k++;                     /* proximo caracter */
                }
                despl_bp();
        }
        pongac(cr);
        pongac(lf);
}


load()                                      /* cargar un programa */
{
        short *lp;
        int seq,chks,cin,cins,com,sok,k,l,m,inic,sinic,flag_c;
        char *listo="Listo para hacer el upload ...",
             *termino="Transferencia finalizada ",
             *notok="Transferencia abortada ... Errores ";
        while (c[k] == sp) k++;
        for (l=0,k=5;c[k]!=cr;k++,l++) { /* obtener direcc. de inicio */
            d[l]=asc2bin(c[k]);
            if (d[l] == ast)          /* caracter invalido */
                    error(3);
        }
        inic=bin2int(l);
        if (l == 5 & d[0] != 0 ) sinic=d[0];   /* segmento */
        else sinic=0;
        imprim(listo);
        pongac(cr);pongac(lf);
        seq=1;                          /* inicializa el # de secuencia */
        lp=inic;
l_ini:  pongac(nak);
        while (1) {                                 /* loop infinito */
             cin=captec();                          /* CAMPO 1 */
             if (cin == eot) {                      /* fin de la Tx */
                    pongac(ack);
                    goto fin_ok;
             }
             else {
                if (cin == soh) {
                        cins=captec();           /* CAMPO 2 */
                        com=captec();            /* CAMPO 3 */
                        sok=(cins ^ ~(com));    /* XOR con complemento */
                        if (sok != 0) {
                                pongac(can);
                                goto fin_mal;
                        }
                        chks=0;
                        flag_c=0;
                        for (m=0;m<128;m++) {
                              cin=captec(); /* CAMPO DATA (128 bytes) */
                                lp++;
                                if (lp == 0) {
                                    sinic++;
```

```
                                        flag_c=1;
                                }
                                ponga_datos(cin,sinic,lp);
                                chks=chks+cin;
                        }
                        if (cins != seq && cins != seq-1) {
                                pongac(can);
                                goto fin_mal;
                        }
                        else if (cins == seq-1) {
                                pongac(nak);
                                lp=lp-128;
                                if (flag_c == 1) sinic--;
                                continue;
                        }
                        cin=captec();           /* CAMPO 5 */
                        if (cin == chks) {
                                pongac(ack);
                                seq++;
                        }
                        else {
                                pongac(nak);
                                lp=lp-128;
                                if (flag_c == 1) sinic--;
                        }
                }
                else goto l_ini;
            }
        }
fin_mal: pongac(cr);pongac(lf);
        imprim(notok);
        goto fin_fin;
fin_ok: pongac(cr);pongac(lf);
        imprim(termino);
fin_fin: pongac(cr);pongac(lf);
}


go()                                    /* ejecutar un programa */
{
        int l,k=3;
        if (cnt != 2) {
                inireg();
                while (c[k] == sp) k++;          /* ignorar espacios */
                for (l=0;c[k]!=cr;l++,k++) {   /* direcc. de comienzo */
                        d[l]=asc2bin(c[k]);
                        if (d[l] == ast)
                                error(3);       /* caracter invalido */
                }
                reg[12]=bin2int(l);
                if (l == 5 && d[0] != 0)
                        reg[8]=(d[0] << 12);
                else reg[8]=0;
        }
```

```
        else {
                if (b != 0) {
                        if (i_int3 == 1) {
                                l=cnt;
                                reg[13]=(reg[13] | 0100h);
                                ejecute();
                                reg[13]=( reg[13] & 0FEFFh);
                                ponga_datos(0CCh,tablabp[bn][1],
                                  tablabp[bn][0]);
                                i_int3=0;
                        }
                }
        }
        ejecute();
        if (i_pap == 1) {
                ponga_datos(0CCh,tablabp[bt][1],tablabp[bt][0]);
                i_pap=0;
        }
}


t()                                     /* ejecucion paso a paso */
{
        int k=2,l,n,t_num,a1,a2;
        char *bp="** Breakpoint **";
        if (i == cnt) t_num=1;           /* ejecutar solo un paso */
        else if (i != cnt) {            /* el usuario dio # de pasos */
                while (c[k] == sp) k++; /* ignorar espacios */
                for (l=0;c[k]!=sp && c[k]!=cr;l++,k++) {
                        if (c[k] == cr) break;
                        d[l]=asc2bin(c[k]);
                      if (d[l] > 9 || d[l] == ast) /* solo decimales */
                                error(3);       /* caracter invalido */
                }
        switch(l) {                     /* convertir a un numero decimal */
                case 1:
                        t_num=d[0];
                        break;
                case 2:
                        t_num=d[0]*10+d[1];
                        break;
                case 3:
                        t_num=d[0]*100+d[1]*10+d[2];
                        break;
                case 4:
                        t_num=d[0]*1000+d[1]*100+d[2]*10+d[3];
                        break;
                case 5:
                       t_num=d[0]*10000+d[1]*1000+d[2]*100+d[3]*10+d[4];
                        break;
        }
        for (l=0;l<t_num;l++) {
              reg[13]=(reg[13] | 0100h); /* TF = 1 en pal. de status */
                ejecute();
```

```
                if (i_pap == 1) {
                        ponga_datos(0CCh,tablabp[bt][1],tablabp[bt][0])
                         i_pap=0;
                }
                for (bt=0;bt<b;bt++)
           if (tablabp[bt][1] == reg[12] && tablabp[bt][0] == reg[8])
                                ponga_datos(datosbp[bp],reg[12],reg[8])
                                 i_pap=1;
                                 break;
                        }
                if (j == 13) {
                        al=reg[8];
                        a2=reg[12];
                        desp_int(a1);           /* despliega CS */
                        pongac(dos_ptos);
                        desp_int(a2);           /* despliega IP */
                        pongac(sp);
                        pongac(sp);
                        al=al>>12;
                        despl_byte(a2,al);      /* despliega el dato */
                        pongac(cr);             /* en esa direccion */
                        pongac(1f);
                        dr();                   /* despl. los regs. */
                }
                pongac(cr);
                pongac(1f);
    }   )
}


iow()                           /* escribir a perifericos manualmente */
{
        int k=4,l,m,n,dat,dir;
        while (c[k] == sp) k++;         /* ignorar espacios */
        for (l=0;c[k]!=sp && c[k]!=cr;l++,k++) { /* obtiene la dir. */
                d[l]=asc2bin(c[k]);
                if (d[l] == ast)
                        error(3);       /* caracter invalido */
        }
        dir=bin2int(l);
        while (c[k] == sp) k++;         /* ignorar espacios */
        for (l=0;c[k]!=cr;l++,k++)      /* obtiene el dato */
                d[l]=asc2bin(c[k]);
        dat=bin2int(l);
        iowt(dir,dat);
}


ior()                                   /* leer de perifericos manualmente */
{
        int k=4,l,m,n;
        unsigned short dd,d1,d2;
        while (c[k] == sp) k++;         /* ignorar espacios */
        for (l=0;c[k]!=sp && c[k]!=cr;l++,k++) { /* obtiene direccion */
```

```
                d[l]=asc2bin(c[k]);
                if (d[l] == ast)
                        error(3);       /* caracter invalido */
        }
        if (l > 4) error (6);           /* direccion invalida */
        m=5-l;
        for (n=0;n<m;n++)               /* despliegue en pantalla */
                pongac(cero);
        for (n=0,k=4;n<l;n++,k++)
                pongac(c[k]);
        pongac(dos_ptos);
        m=bin2asc(l);
        dd=iord(m);
        dl=bin2asc(dd/16);              /* primer digito */
        pongac(dl);
        d2=bin2asc(dd%16);              /* segundo digito */
        pongac(d2);
        pongac(cr);
        pongac(lf);
}


help()                                  /* despliega pantalla de ayuda */
{
        char *lin1="dr - desplegar el contenido de los registros",
  *lin2="mr - modificar el contenido de los registros",
  *lin3="dm <dir. inic.> [<dir. final>] - desplegar el contenido de
memoria",
  *lin4="mm <dir. inic.> - modificar el contenido de memoria",
  *lin5="cbp [<dir.1> <dir.2> ...] - colocar break points (5 maximo)",
  *lin6="rbp [<dir.1> <dir.2> ...] - remover break points",
  *lin7="load <dir.> - cargar un programa ",
  *lin8="go [<dir.>] - ejecutar un programa",
  *lin9="t [<# pasos>] - ejecutar un programa paso a paso",
  *lin10="iow <dir.> <dato> - escribir manualmente a un periferico ",
  *lin11="ior <dir.> - leer manualmente de un periferico ",
  *lin12="? - desplegar esta pantalla de ayuda";
  imprim(lin1);pongac(cr);pongac(lf);imprim(lin2);pongac(cr);pongac(lf);
  imprim(lin3);pongac(cr);pongac(lf);imprim(lin4);pongac(cr);pongac(lf);
  imprim(lin5);pongac(cr);pongac(lf);imprim(lin6);pongac(cr);pongac(lf);
  imprim(lin7);pongac(cr);pongac(lf);imprim(lin8);pongac(cr);pongac(lf);
imprim(lin9);pongac(cr);pongac(lf);imprim(lin10);pongac(cr);pongac(lf);
  imprim(lin11);pongac(cr);pongac(lf);imprim(lin12);pongac(cr);pongac(lf);
}


despl_byte(spl,msb)                     /* despliega un byte de memoria */
short *spl;
{
        unsigned short dat,dl,d2;
        pongac(sp);
        dat=obtenga_datos(spl,msb);     /* leer el dato de la direcc. */
        dl=bin2asc(dat/16);              /* primer digito */
```

```
        pongac(d1);
        d2=bin2asc(dat%16);                    /* segundo digito */
        pongac(d2);
}


despl_bp()                                     /* desplegar break points */
{
        short u;
        char *p="breakpoints=";
        int m,n,o;
        imprim(p);
        for (o=0;o<b;o++) {
                m=tablabp[o][0];
                n=bin2asc((short)((m & 0F000h)>>12));
                pongac(n);
                desp_int(tablabp[1][1]);
                pongac(sp);
                pongac(sp);
        }
}

int3main()                                     /* rutina para interrupc. INT3 */
{
        int k,a1,a2;
        for (bn=0;bn<b;bn++) /* ver si CS:IP actual tiene algun br. p.*/
          if (tablabp[bn][1] == reg[12] && tablabp[bn][0] == reg[8]) {
                ponga_datos(datosbp[bn],tablabp[bn][1],tablabp[bn][0]);
                   break;
                }
                a1=reg[8];
                a2=reg[12];
                desp_int(a1);                  /* despliega CS */
                pongac(dos_ptos);
                desp_int(a2);                  /* despliega IP */
                pongac(sp);
                pongac(sp);
                a1=a1>>12;
                despl_byte(a2,a1);             /* despliega el dato */
                pongac(cr);                    /* en esa direccion */
                pongac(lf);
                dr();                          /* despliega los registros */
}

ordene_bp()                                    /* ordena los break points */
{
        int a,temp,l,k,n;
        for (l=0;l<b-1;++l)                    /* ordena de menor a mayor */
                for (k=b-1;l<k;--k) {
                        n=k-1;
                        if ((tablabp[n][0] > tablabp[k][0]) ||
                          (tablabp[n][0] == tablabp[k][0] &&
```

```
                        tablabp[n][1] > tablabp[k][1])) {
                                temp=datosbp[n];
                                datosbp[n]=datosbp[k];
                                datosbp[k]=temp;

                                temp=tablabp[n][0];
                                tablabp[n][0]=tablabp[k][0];
                                tablabp[k][0]=temp;

                                temp=tablabp[n][1];
                                tablabp[n][1]=tablabp[k][1];
                                tablabp[k][1]=temp;
                        }
                }
}



desp_int(v)                             /* despliegue de integers */
int v;
{
        short s;
        int w;
        for (w=12;w>=0;w-=4) {
                s=bin2asc((short)((V>>1) & 0Fh));
                pongac(s);
        }
}

asc2bin(w)                              /* conversion ASCII - binario */
int w;
{
        if (w>=48 && w<=57)               /* caracteres del 0 al 9 */
                return(w-48);
        else if (w>=65 && w<=70)          /* caracteres A - F */
                return(w-55);
        else if (w>=97 && w<=102)         /* caracteres a - f */
                return(w-87);
        else return ('\*');               /* caracteres invalidos */
}

bin2asc(w)                              /* conversion binario - ASCII */
int w;
{
        if (w>=0 && w<=9)                 /* 0 - 9 */
                return(w+48);
        else if (w>=10 && w<=15)          /* A - F */
                return(w+55);
        else return ('\*');               /* caracter invalido */
}
```

```
bin2int(v)                                  /* armado binarios - integer *
int v;
{
        short *tmp;
        if (v == 5) {                        /* en otro segmento */
                tmp=d[1]*4096+d[2]*256+d[3]*16+d[4];
                return (tmp);
        }
        else if (v == 4) {
                tmp=d[0]*4096+d[1]*256+d[2]+16+d[3];
                return (tmp);
        }
        else if (v == 3) {
                tmp=d[0]*256+d[1]*16+d[2];
                return (tmp);
        }
        else if (v == 2) {
                tmp=d[0]*16+d[1];
                return (tmp);
        }
        else if ( v == 1)
                return (d[0]);
}
```